Robotica (1998) volume 16, pp. 433–444. Printed in the United Kingdom © 1998 Cambridge University Press

Manipulator Inverse Kinematics using an Adaptive Back-propagation Algorithm and Radial Basis Function with a Lookup Table

A. S. Morris and M. A. Mansor

Department of Automatic Control and Systems Engineering, University of Sheffield, Mappin St., Sheffield S1 3JD (UK)

(Received in final form: September 5, 1997)

SUMMARY

This is an extension of previous work which used an artificial neural network with a back-propagation algorithm and a lookup table to find the inverse kinematics for a manipulator arm moving along pre-defined trajectories. The work now described shows that the performance of this technique can be improved if the back-propagation is made to be adaptive. Also, further improvement is obtained by using the whole workspace to train the neural network rather than just a pre-defined path. For the inverse kinematics of the whole workspace, a comparison has also been done between the adaptive back-propagation algorithm and radial basis function.

KEYWORDS: Artificial neural network; Radial basis function; Adaptive back-propagation algorithm; Inverse kinematics; Lookup table; Two link planar manipulator arm; Three link manipulator arm.

1. Introduction

Calculation of the inverse kinematics for manipulator arms is time consuming and prone to error due to the many variables involved. The equations are not complicated but much effort is involved in finding solutions, which are complicated by the existence of multiple solutions due to redundancy.

In a paper published in 1997[1] we reviewed the solutions previously proposed for inverse kinematics computation and proposed a new technique. This involved a hidden layer perceptron with back-propagation algorithm which used lookup tables to cater for the redundancies of the arm. The lookup tables contain neuron weights that represent the different configurations of the arm. The neural network was trained using patterns from a predefined trajectory.

This work can also be applied to finding the inverse kinematics solutions of flexible manipulators. Instead of using patterns from a rigid manipulator, the co-ordinates of the end-effector of the flexible manipulator arm and its corresponding joint angle value can be used as the pattern for training the artificial neural network. Using this method avoids applying complex mathematics[2,3] to solve the problem of inverse kinematics, and it may even be more accurate because the flexible manipulator could not be exactly modelled mathematically.

Section two describes the work with subsections detailing the manipulator arm with its kinematics equations and artificial neural networks, section three are the results obtained from the simulation run and section four is the discussion and conclusion of the report. The final section gives the references.

2. Outline of work

There are two parts in this work. The first is finding the solutions of the inverse kinematics for a three-link manipulator arm along a predefined trajectory in 3D workspace. A two hidden layer perceptron with adaptive back-propagation algorithm was used during the training session. As the path considered is the same as in previous work the same patterns were also used for the training of the neural network (Table I).

The second was done by training the artificial neural network on patterns taken from the workspace of the manipulator arm, i.e. the area that the end-effector can reach. For a two link planar manipulator arm this workspace will be a semicircle with its centre being at the base of the arm. For this work though, only half of the workspace was considered. Points were taken from the whole workspace rather than limiting it to just a prescribed path. This has the advantage that the use of the lookup tables are not limited to a single predefined path but rather it can be used for any points or paths in that workspace. Two types of artificial neural networks were used, a multi-layered perceptron with adaptive back-propagation algorithm and a radial basis function. For comparison, conventional calculation of the inverse kinematics was also applied.

2.1. Two link planar manipulator arm

Figure 2.1 shows the two link planar manipulator arm. It can move its end-effector within a semicircle work space as shown with a radius of 2 units. Joint 1 can rotates 180 degrees and joint 2 rotates 360 degrees. Both joints 2 and 3 turn about their own horizontal axes. Using

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

Table I. The normalised training data used for the three link manipulator

First configuration						Second configuration					
x	y	z	θ_1	θ_2	θ_3	x	y	z	θ_1	θ_2	θ_3
0.05	0.00	0.950	0.00000	0.93060	0.42227	0.05	0.00	0.950	0.00000	0.81852	0.42227
0.10	0.05	0.950	0.23182	0.91293	0.41850	0.10	0.05	0.950	0.46365	0.79874	0.41850
0.15	0.10	0.948	0.29400	0.89260	0.41223	0.15	0.10	0.948	0.58800	0.77791	0.41223
0.20	0.15	0.946	0.32175	0.87158	0.40499	0.20	0.15	0.946	0.64350	0.75712	0.40499
0.25	0.20	0.942	0.33737	0.85065	0.39840	0.25	0.20	0.942	0.67474	0.73580	0.39840
0.30	0.25	0.936	0.34737	0.83040	0.39408	0.30	0.25	0.936	0.69474	0.71312	0.39408
0.35	0.30	0.929	0.35431	0.81135	0.39361	0.35	0.30	0.929	0.70863	0.68819	0.39361
0.40	0.35	0.918	0.35941	0.79390	0.39839	0.40	0.35	0.918	0.71883	0.66014	0.39839
0.45	0.40	0.904	0.36332	0.77828	0.40943	0.45	0.40	0.904	0.72664	0.62827	0.40943
0.50	0.45	0.887	0.36641	0.76450	0.42726	0.50	0.45	0.887	0.73282	0.59207	0.42726
0.55	0.50	0.867	0.36891	0.75231	0.45184	0.55	0.50	0.867	0.73782	0.55124	0.45184
0.60	0.55	0.842	0.37097	0.74128	0.48265	0.60	0.55	0.842	0.74195	0.50572	0.48265
0.65	0.60	0.813	0.37271	0.73075	0.51881	0.65	0.60	0.813	0.74542	0.45552	0.51881
0.70	0.65	0.778	0.37419	0.71994	0.55918	0.70	0.65	0.778	0.74838	0.40074	0.55918
0.75	0.70	0.739	0.37546	0.70797	0.60249	0.75	0.70	0.739	0.75093	0.34146	0.60249
0.80	0.75	0.694	0.37658	0.69381	0.64729	0.80	0.75	0.694	0.75315	0.27779	0.64729
0.85	0.80	0.643	0.37755	0.67634	0.69197	0.85	0.80	0.643	0.75510	0.20982	0.69197
0.90	0.85	0.585	0.37842	0.65429	0.73462	0.90	0.85	0.585	0.75683	0.13777	0.73462
0.95	0.90	0.521	0.37919	0.62625	0.77301	0.95	0.90	0.521	0.75838	0.06199	0.77301

Denavit-Hartenberg notation and applying transformation matrix on the manipulator[4] gives:

$$^0T_2 = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & \cos(\theta_1+\theta_2)+\cos\theta_2 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & \sin(\theta_1+\theta_2)+\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

From equation (2.1), the forward kinematics of the manipulator arm is given as:

$$x = \cos(\theta_1+\theta_2)+\cos\theta_2 \quad \text{and} \quad y = \sin(\theta_1+\theta_2)+\sin\theta_2 \tag{2.2}$$

The inverse kinematics of the manipulator arm is then obtained by solving equation (2.2) giving:

$$\theta_1 = \text{atan}\,2(y,x) - \text{atan}\,2(k_2,k_1)$$
$$\text{and} \tag{2.3}$$
$$\theta_2 = \text{atan}\,2(\sin\theta_2, \cos\theta_2)$$

where $\cos\theta_3 = (x^2+y^2-2)/2$, $\sin\theta_3 = \pm\sqrt{1-\cos^2\theta_2}$, $k_1 = 1+\cos\theta_2$ and $k_2 = \sin\theta_2$.

2.2. *Three link manipulator arm*

Figure 2.2 shows the three link manipulator arm as defined for this work. It can move its end-effector in a 3D work space. The link lengths are assumed to be 1 unit and thus the radius of its work space is 2 units. Joint 1 can rotates 360 degrees about the vertical axis and joint 2 and 3 can rotate 180 degrees and 360 degrees respectively. Both joints 2 and joint 3 rotate about their own horizontal axes. The work space of the end-effector is thus a semi-hemisphere.

Joint *i*	a_i	α_i	d_i	θ_i
1	L_1=1	0	0	θ_1
2	L_2=1	0	0	θ_2

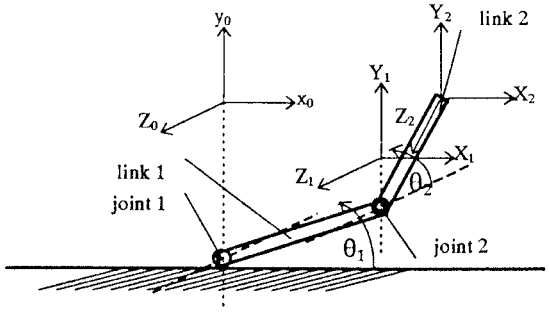

Fig. 2.1. A diagram of the two link planar manipulator arm.

Joint *i*	a_i	α_i	d_i	θ_i
1	0	90	0	θ_1
2	L_1=1	0	0	θ_2
3	L_2=1	0	0	θ_3

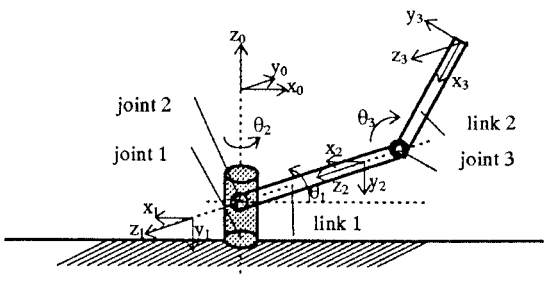

Fig. 2.2. A three link planar manipulator arm having a 3D work space.

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

The inverse kinematics of the three link manipulator arm is:

$$\theta_1 = \text{atan } 2(y, x)$$
$$\theta_2 = \text{atan } 2(-z, x/\cos\theta_1) - \text{atan } 2(k_2, k_1)$$
$$\theta_3 = \text{atan } 2(\sin\theta_3, \cos\theta_3) \quad (2.4)$$

where

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

and

$$k_1 = 1 + \cos\theta_3$$
$$k_2 = \sin\theta_3.$$

2.3. Artificial neural networks

In the previous work, 30,000 iterations were needed during the training session of a single hidden layer neural network with back-propagation algorithm on patterns described in Table I. Even when done off-line, the time taken is unreasonably long. It was decided to apply a neural network with adaptive back-propagation algorithm for the training session in order to try to reduce the training time. The algorithm changes its learning rate according to the error at the output layer. The next section describes the neural network more fully.

Apart from the time taken for training, the design time of a feed-forward perceptron with hidden layers is quite lengthy. This is due to a number of parameters that need to be adjusted, like the number of hidden layers, the number of neurons in each layer, the rate of learning, the momentum factor and the type of sigmoid functions to be applied. A radial basis function consists of an input layer, a hidden layer and an output layer. Training in radial basis function is equivalent to finding a surface in a multidimensional space that provides a best fit to the training data[5]. Thus a radial basis function tends to have many more neurons in its hidden layer than a comparable feed-forward network, but designing a radial basis function often takes much less time.

2.3.1. Two hidden layer neural network with adaptive back-propagation algorithm. A two hidden layer neural network with adaptive back-propagation algorithm (Figure 2.3) was used for the training of the three link manipulator arm. During the training session there are two passes of computation, namely the forward and backward pass. In the forward pass, the synaptic weights remain unaltered throughout the network. The function signal appearing at the output of neuron j is calculated as:

$$o_j = \varphi(\text{inp}_j) \quad \text{and} \quad \text{inp}_j = \left(\sum w_{ji} o_i\right) + {}^b w_i \quad (2.5)$$

where $\varphi(\cdot)$ is the activation function, w_{ji} is the weight associated with a neuron in layer i and a neuron in layer j, and ${}^b w_i$ is the weight of the bias neuron in layer i. The activation functions applied in this work were:

$$\varphi = \frac{1}{1 + e^{\text{inp}_j}} \quad \text{and} \quad \varphi = \frac{2}{1 + e^{(-2\text{xinp}_j)}} - 1 \quad (2.6)$$

which are the log-sigmoid function and tan-sigmoid function respectively.

The error, δ, at the output layer of a neuron in the output layer is calculated by comparing its output, o_{out}, to the desired output, d:

$$\delta = \varphi'(\text{inp}_{out})(d - o_{out}) \quad (2.7)$$

where inp_{out} is the net input into that neuron in the output layer. These error values are then used in the backward pass to adjust the weights from the output layer back to the input layer as:

$$\delta_j = \varphi'(\text{inp}_j) \sum_j w_i \delta \quad (2.8)$$

where w_j are weights between the output layer and the next layer before it. The error calculation is implemented on a neuron-by-neuron basis over the entire epoch of patterns. The weights are then adjusted according to:

$$w(\text{new}) = w(\text{old}) + \eta\delta o + \alpha(\Delta w(\text{old})) \quad (2.9)$$

where is the momentum factor and η is the adaptive learning rate.

The adaptive learning rate is applied heuristically. Depending on the error at the output layer, if it exceeds a predefined ratio (1.04), the new weights, biases, output and error are discarded. In addition the learning rate is decreased by multiplying by a predefined value (0.7). Otherwise, the new weights, etc. are kept. If the new error is less than the old error, the learning rate is increased (by multiplying by 1.05).

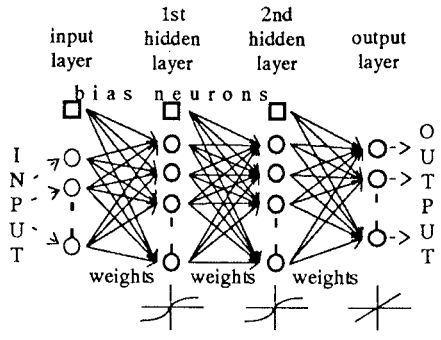

Fig. 2.3. Multi-layer perceptron with two hidden layers.

2.3.2. Radial basis function. The construction of a radial basis function network involves three entirely different layers. The input layer is made up of source nodes (sensory units), the second layer is a hidden layer of high enough dimensions, and the output layer supplies the response of the network to the activation patterns applied to the input layer. The transformation from the input space to the hidden-unit space is non-linear, whereas the transformation from the hidden-unit space to the output space is linear.

If an input vector is presented, each neuron in the hidden layer will output a value corresponding to how

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

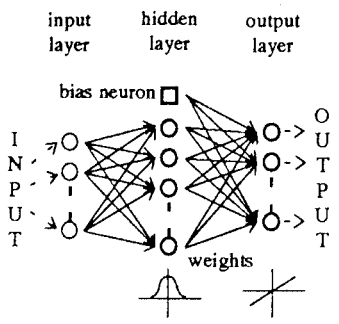

Fig. 2.4. Radial basis function network.

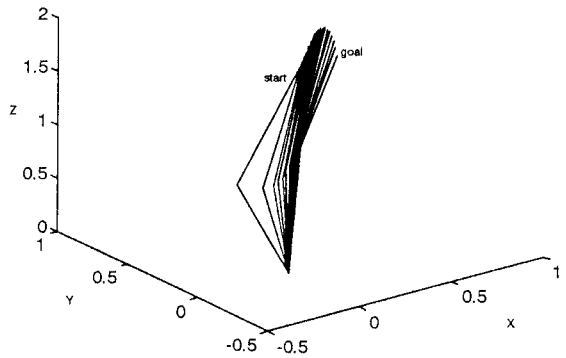

Fig. 3.1. The path traced by the arm in first configuration.

close the input vector is to each neuron's centre c_i. Each hidden neuron has an activation function of the form $\varphi(\|\vec{x} - \vec{c}\|)$. The output layer is linear, so:

$$o_{out} = \left(\sum_{i=1}^{n} w_i \, \varphi(\|\vec{x} - \vec{c}_i\|) \right) + {}^{b}w \qquad (2.10)$$

where n is the number of centres, w_i are the weights between corresponding neurons in the hidden and output layers and ${}^{b}w$ is the bias weight.

Figure 2.4 shows the structure of a radial basis function network. The input layer is connected to the hidden layer via unweighted connections whereas the hidden layer is connected to the output layer using a set of weighted connections.

The weights can be learned by using Hebb's activation rule, i.e. the change in the weight associated with a neuron is proportional to the error at the output and the activation of the neuron.

3.1. Training of the three link arm and results

The training patterns were points taken from a path traverse by the end-effector of the arm. This is a three dimensional path as shown in Figure 3.1 and Figure 3.2. Nineteen patterns (Table I) were used for the training of the neural network and eight other points (Table II) were used to test the network after it had been trained.

A two hidden layer perceptron with adaptive back-propagation algorithm was used for the training of normalised patterns in the three link manipulator arm. Both configurations were trained 8000 times using their respective patterns with 11 neurons for the first hidden layer and 13 neurons for the second hidden layer. A log-sigmoid transfer function (Eqn. 2.12) was used for all the neurons. The number of neurons for each layer was chosen after a number of trials comparing the sum squared error produced[6]. The momentum factor was fixed at 0.9 and the learning rate was initially set at 0.775. Even though the arm has three links, there are only two configurations that the arm can have to reach the same point in space. These configurations correspond to the two sets of weights obtained after the training of the network.

Figures 3.3 and 3.4 show the graph of the difference between the desired and actual output for each configuration. For configuration 1, the average percentage error for θ_1 is 0.109%, θ_2 is 0.149% and θ_3 is −0.113%. This compares well with the average percentage errors in the previous work, which were 0.7%, 0.01% and 0.002% where training with 30,000 iterations was needed. The average percentage error for

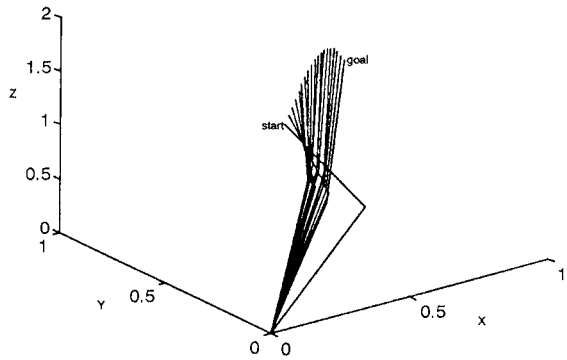

Fig. 3.2. The path traced by the arm in second configuration.

Table II. The normalised data used for the test session of the three link manipulator

First configuration						Second configuration					
x	y	z	θ_1	θ_2	θ_3	x	y	z	θ_1	θ_2	θ_3
0.05	0.00	0.950	0.00000	0.93060	0.42227	0.05	0.00	0.950	0.00000	0.81852	0.42227
0.12	0.07	0.949	0.26404	0.90495	0.41620	0.12	0.07	0.949	0.52807	0.79039	0.41620
0.23	0.18	0.944	0.33202	0.85898	0.40086	0.23	0.18	0.944	0.66405	0.74444	0.40086
0.36	0.31	0.927	0.35545	0.80772	0.39411	0.36	0.31	0.927	0.71091	0.68285	0.39411
0.52	0.47	0.880	0.36747	0.75945	0.43630	0.52	0.47	0.880	0.73494	0.57630	0.43630
0.64	0.59	0.819	0.37239	0.73285	0.51120	0.64	0.59	0.819	0.74477	0.46593	0.51120
0.67	0.62	0.800	0.37333	0.72650	0.53452	0.67	0.62	0.800	0.74666	0.43415	0.53452
0.83	0.78	0.664	0.37718	0.68381	0.67423	0.83	0.78	0.664	0.75435	0.23752	0.67423

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

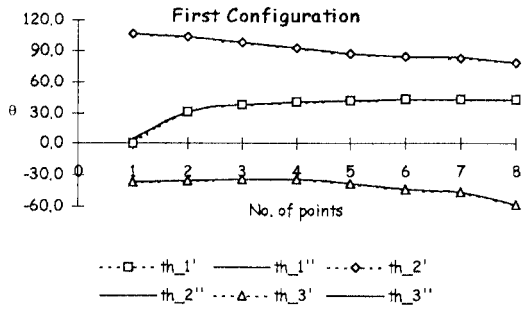

Fig. 3.3. A graph showing the differences between actual and desired output.

the second configuration is 0.213%, 0.152% and 0.025% for θ_1, θ_2 and θ_3 respectively. The corresponding results in the previous work[1] were 2.4%, 0.1% and 0.5%.

3.2. Training of the two link planar arm and results

The results above were based on a predefined trajectory of the manipulator arm. Unfortunately, a trajectory of the end-effector cannot be determined in advance, thus the need to have a lookup table of the workspace. Then for any path or trajectory taken, the solution can be found without having to train the neural network again.

The patterns chosen for the training of the neural networks in this work were taken from points in the workspace of the arm, i.e. the area that can be reached

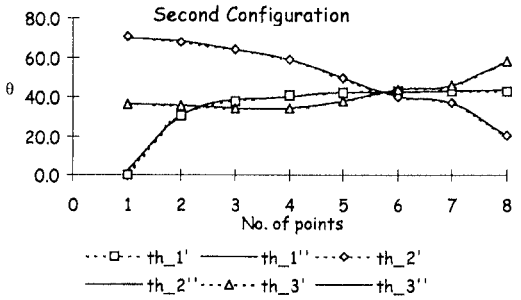

Fig. 3.4. A graph showing the differences between actual and desired output.

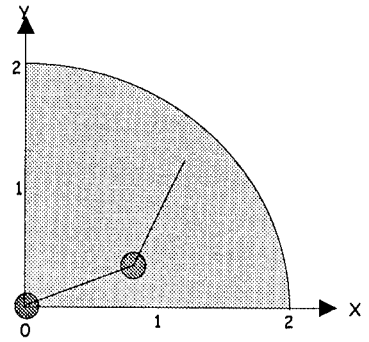

Fig. 3.5. The actual workspace area used in the training of the network.

by the end-effector of the arm (Figure 3.5). For this work though, only half of the workspace was considered. The number of patterns used for the training was 102, so that the networks can generalise the workspace well (Table IV and Figure 3.6). Fifteen points taken randomly from within the workspace were used to test the networks initially (Table III and Figure 3.7). As the arm has only

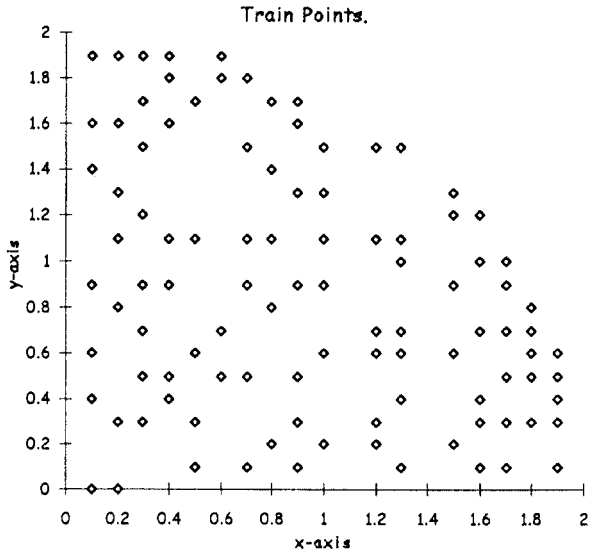

Fig. 3.6. The points used to train the networks.

Table III. Test patterns for the neural networks for the planar manipulator

End effector position in operational space		End effector position in joint space (deg). First configuration		Second configuration	
x	y	θ_1	θ_2	θ_1	θ_2
0	2	90.0	0.0	90.0	0.0
0.1	0.5	153.9	−150.5	3.5	150.5
0.1	1.4	131.3	−90.9	40.5	90.9
0.3	1.9	96.9	−31.8	65.1	31.8
0.4	0	78.5	−156.9	−78.5	156.9
0.4	1.9	92.0	−27.7	64.2	27.7
0.5	0.5	114.3	−138.6	−24.3	138.6
0.6	1.2	111.3	−95.7	15.6	95.7
0.8	1.7	84.8	−40.1	44.8	40.1
1	1.1	89.7	−84.0	5.7	84.0
1	1.5	82.0	−51.3	30.7	51.3
1.3	1.2	70.5	−55.6	14.9	55.6
1.3	1.5	56.1	−14.1	42.1	14.1
1.6	1.2	36.9	0.0	36.9	0.0
1.8	0.7	36.3	−30.1	6.2	30.1

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

Table IV. Training patterns for the neural networks for the planar manipulator

End effector position in operational space		End effector position in joint space (deg).				End effector position in operational space		End effector position in joint space (deg).			
		First configuration		Second configuration				First configuration		Second configuration	
x	y	θ_1	θ_2	θ_1	θ_2	x	y	θ_1	θ_2	θ_1	θ_2
0.1	0	87.1	−174.3	−87.1	174.3	0.9	0.5	88.1	−118.0	−30.0	118.0
0.1	0.4	154.1	−156.2	−2.1	156.2	0.9	0.9	95.5	−101.0	−5.5	101.0
0.1	0.6	152.8	−144.6	8.2	144.6	0.9	1.3	93.1	−75.5	17.5	75.5
0.1	0.9	146.7	−126.2	20.6	126.2	0.9	1.6	84.0	−46.8	37.3	46.8
0.1	1.4	131.3	−90.9	40.5	90.9	0.9	1.7	78.0	−31.8	46.2	31.8
0.1	1.6	123.1	−73.4	49.7	73.4	1	0.2	70.7	−118.7	−48.0	118.7
0.1	1.9	104.9	−35.9	69.0	35.9	1	0.6	85.3	−108.7	−23.4	108.7
0.2	0	84.3	−168.5	−84.3	168.5	1	0.9	89.7	−95.5	−5.7	95.5
0.2	0.3	135.9	−159.2	−23.3	159.2	1	1.1	89.7	−84.0	5.7	84.0
0.2	0.8	141.6	−131.3	10.3	131.3	1	1.3	87.3	−69.8	17.5	69.8
0.2	1.1	135.7	−112.0	23.7	112.0	1	1.5	82.0	−51.3	30.7	51.3
0.2	1.3	130.1	−97.8	32.4	97.8	1.2	0.2	62.0	−105.1	−43.1	105.1
0.2	1.6	119.1	−72.5	46.6	72.5	1.2	0.3	65.8	−103.6	−37.8	103.6
0.2	1.9	101.2	−34.4	66.8	34.4	1.2	0.6	74.4	−95.7	−21.3	95.7
0.3	0.5	132.1	−146.1	−14.0	146.1	1.2	0.7	76.3	−92.0	−15.7	92.0
0.3	0.3	122.8	−155.5	−32.8	155.5	1.2	1.1	78.0	−71.0	7.0	71.0
0.3	0.7	134.4	−135.2	−0.8	135.2	1.2	1.5	67.5	−32.3	35.2	32.3
0.3	0.9	133.2	−123.4	9.9	123.4	1.3	0.1	53.7	−98.6	−44.9	98.6
0.3	1.2	127.8	−103.6	24.2	103.6	1.3	0.4	64.3	−94.3	−30.0	94.3
0.3	1.5	118.8	−80.2	38.6	80.2	1.3	0.6	69.1	−88.6	−19.5	88.6
0.3	1.7	110.3	−60.7	49.7	60.7	1.3	0.7	70.7	−84.8	−14.1	84.8
0.3	1.9	96.9	−31.8	65.1	31.8	1.3	1	72.5	−69.8	2.7	69.8
0.4	0.4	118.6	−147.1	−28.6	147.1	1.3	1.1	71.9	−63.3	8.6	63.3
0.4	0.5	122.7	−142.7	−20.0	142.7	1.3	1.5	56.1	−14.1	42.1	14.1
0.4	0.9	126.5	−121.0	5.5	121.0	1.5	0.2	48.4	−81.7	−33.2	81.7
0.4	1.1	124.2	−108.4	15.8	108.4	1.5	0.6	57.9	−72.2	−14.3	72.2
0.4	1.6	110.4	−68.9	41.5	68.9	1.5	0.9	60.0	−58.0	2.0	58.0
0.4	1.8	100.3	−45.6	54.7	45.6	1.5	1.2	54.8	−32.3	22.5	32.3
0.4	1.9	92.0	−27.7	64.2	27.7	1.5	1.3	47.9	−14.1	33.9	14.1
0.5	0.1	86.5	−150.5	−63.9	150.5	1.6	0.1	40.3	−73.4	−33.1	73.4
0.5	0.3	104.0	−146.1	−42.1	146.1	1.6	0.3	46.1	−71.0	−24.9	71.0
0.5	0.6	117.2	−134.0	−16.8	134.0	1.6	0.4	48.5	−68.9	−20.4	68.9
0.5	1.1	118.4	−105.7	12.7	105.7	1.6	0.7	52.8	−58.3	−5.5	58.3
0.5	1.7	101.2	−55.2	46.0	55.2	1.6	1	51.4	−38.7	12.6	38.7
0.6	0.5	106.8	−134.0	−27.2	134.0	1.6	1.2	36.9	0.0	36.9	0.0
0.6	0.7	111.9	−125.1	−13.2	125.1	1.7	0.1	35.0	−63.3	−28.3	63.3
0.6	1.8	90.0	−36.9	53.1	36.9	1.7	0.3	40.3	−60.7	−20.3	60.7
0.6	1.9	77.4	−9.9	67.5	9.9	1.7	0.5	44.0	−55.2	−11.2	55.2
0.7	0.1	77.4	−138.6	−61.2	138.6	1.7	0.7	45.6	−46.4	−0.8	46.4
0.7	0.5	100.1	−129.1	−29.0	129.1	1.7	0.9	43.8	−31.8	12.0	31.8
0.7	0.9	107.4	−110.5	−3.1	110.5	1.7	1	40.0	−19.1	20.9	19.1
0.7	1.1	106.8	−98.6	8.2	98.6	1.8	0.3	33.6	−48.3	−14.7	48.3
0.7	1.5	99.1	−68.3	30.8	68.3	1.8	0.5	36.4	−41.8	−5.4	41.8
0.7	1.8	83.8	−30.1	53.7	30.1	1.8	0.6	36.9	−36.9	0.0	36.9
0.8	0.2	79.7	−131.3	−51.6	131.3	1.8	0.7	36.3	−30.1	6.2	30.1
0.8	0.8	100.6	−111.1	−10.6	111.1	1.8	0.8	33.9	−19.9	14.0	19.9
0.8	1.1	101.1	−94.3	6.8	94.3	1.9	0.1	21.0	−35.9	−14.9	35.9
0.8	1.4	96.5	−72.5	24.0	72.5	1.9	0.3	24.9	−31.8	−6.9	31.8
0.8	1.7	84.8	−40.1	44.8	40.1	1.9	0.4	25.8	−27.7	−2.0	27.7
0.9	0.1	69.4	−126.2	−56.7	126.2	1.9	0.5	25.5	−21.6	4.0	21.6
0.9	0.3	80.1	−123.4	−43.2	123.4	1.9	0.6	22.5	−9.9	12.6	9.9

two links, the arm can be in either of two poses or configurations to reach any point in space (Figure 3.8). Thus two sets of weights or lookup tables from the training session are needed to get a solution to the inverse kinematics, i.e. angles of the joints, for a point in the workspace.

For comparison, a conventional geometrical method, a perceptron network with two hidden layers and a radial basis function were applied. The conventional method was chosen to be the reference or standard for the comparison since the manipulator is assumed to be rigid. The reason for doing so is because even if a neural

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

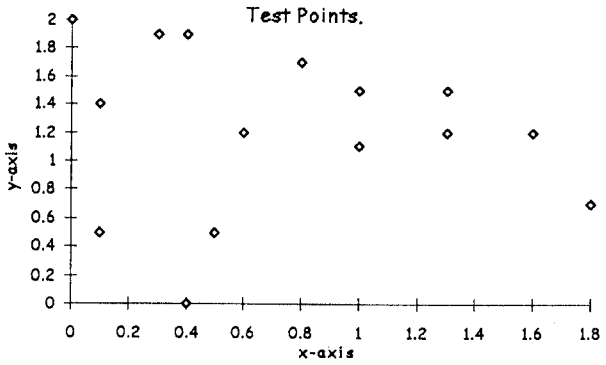

Fig. 3.7. The points used to test the networks.

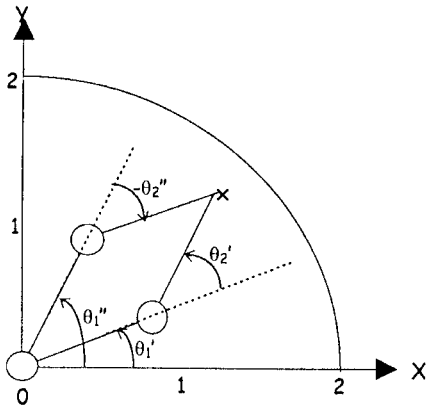

Fig. 3.8. The two poses possible to reach a point × in the workspace.

network have a small sum squared error during its training time, it may still not generalise well. A back-propagation algorithm with adaptive learning rate was used to teach the network. The momentum factor was set at 0.9 and the learning rate was initially set at 0.265. For the first configuration, the first hidden layer has 17 neurons and the second hidden layer has 23 neurons with each neuron having a log-sigmoid activation function. For the second configuration, due to its distribution being between −1 and +1, the first hidden layer has 23 neurons and the second has 27 neurons with a tan-sigmoid activation function. For both networks, a training of 20,000 times was done. Figure 3.9 and Figure 3.10 show the graphs of the training for both configuration 1 and 2 using the perceptron network. Figure 3.11 and Figure 3.12 show the graphs of the

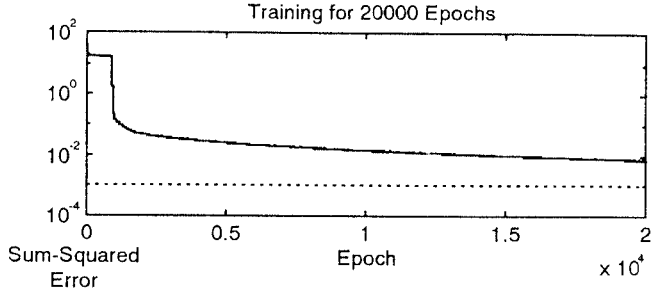

Fig. 3.9. A graph of the training for configuration 1 using two hidden layer perceptron.

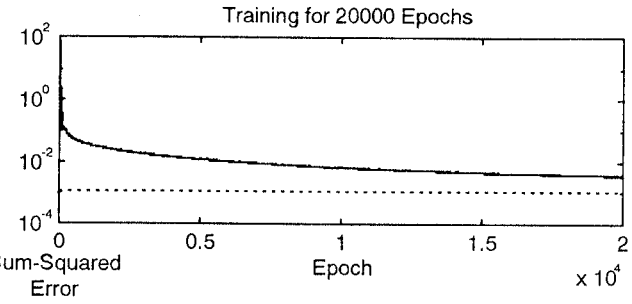

Fig. 3.10. A graph of the training for configuration 2 using two hidden layer perceptron.

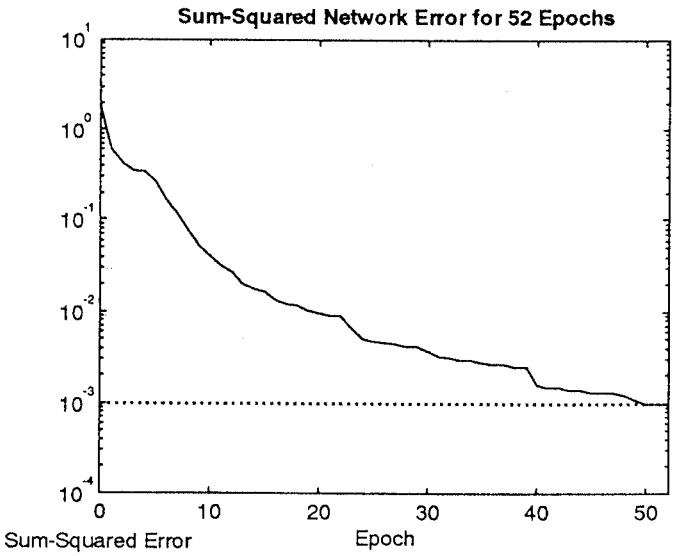

Fig. 3.11. A graph of the training for configuration 1 using radial basis function.

training for both configurations using the radial basis function.

Table V and Table VI show the resulting errors after the training sessions of the two hidden layer perceptron network and radial basis function respectively against the

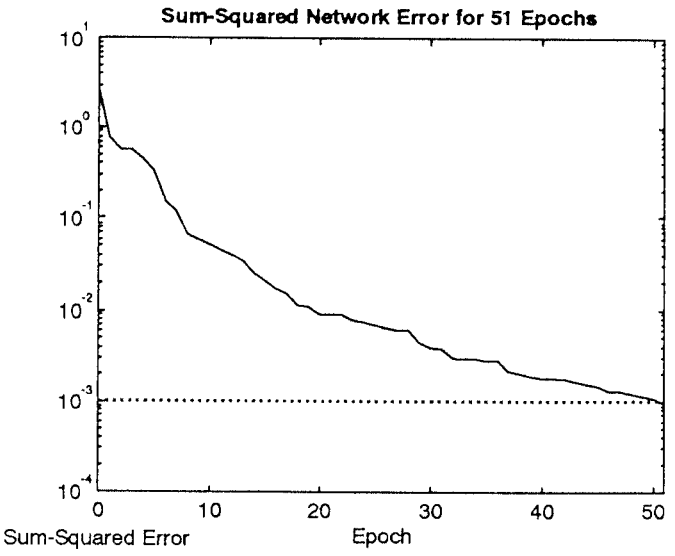

Fig. 3.12. A graph of the training for configuration 2 using radial basis function.

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

Table V. The resulting errors after training of the two hidden layer perceptron

Two hidden layer perceptron with backpropagation algorithm.
Errors after training for train2_1-test2_1 after de-normalising. (17/logsig-23/logsig-logsig)
{work2_1.mat *** †p = [2,000 20,000 0.001 0.265]}

$\theta_1^1 * 3$ (degrees)				$-\theta_2^1 * 4$ (degrees)		
target	actual	% error	n	target	actual	% error
90.0	99.5	10.543	1	0.0	−26.6	#DIV/OI
153.9	152.4	−0.979	2	−150.5	−150.4	−0.046
131.3	132.1	0.546	3	−90.9	−90.7	−0.164
96.9	96.8	−0.083	4	−31.8	−33.0	3.675
78.5	85.9	9.446	5	−156.9	−155.3	−1.054
92.0	89.9	−2.233	6	−27.7	−26.1	−5.761
114.3	113.9	−0.307	7	−138.6	−137.6	−0.730
111.3	112.7	1.260	8	−95.7	−94.6	−1.183
84.8	85.4	0.707	9	−40.1	−40.7	1.521
89.7	89.7	−0.063	10	−84.0	−82.7	−1.474
82.0	83.4	1.725	11	−51.3	−51.1	−0.365
70.5	68.6	−2.730	12	−55.6	−49.5	−10.961
56.1	54.7	−2.480	13	−14.1	−16.1	14.348
36.9	35.9	−2.704	14	0.0	−6.8	#DIV/OI
36.3	36.1	−0.586	15	−30.1	−27.7	−7.923
Average error		0.804		Average error		−0.778

Errors after training for train2_2-test2_2 after de-normalising. (23/tansig-27/tansig-tansig)
{work2_2mat *** †p = [2,000 20,000 0.001 0.265]}

$\theta_1^1 * 3$ (degrees)				$-\theta_2^1 * 3.5$ (degrees)		
target	actual	% error	n	target	actual	% error
90.0	75.1	−16.539	1	0.0	26.6	#DIV/O1
3.5	5.2	50.990	2	150.5	151.0	0.375
40.5	39.8	−1.711	3	90.9	91.7	0.952
65.1	65.4	0.414	4	31.8	31.0	−2.534
−78.5	−76.0	−3.128	5	156.9	156.8	−0.081
64.2	64.1	−0.165	6	27.7	26.8	−3.376
−24.3	−25.7	5.700	7	138.6	138.6	0.029
15.6	15.5	−0.393	8	95.7	94.5	−1.261
44.8	44.6	−0.445	9	40.1	39.9	−0.465
5.7	5.9	2.416	10	84.0	84.1	0.133
30.7	30.7	0.157	11	51.3	51.7	0.702
14.9	12.9	−13.425	12	55.6	57.4	3.302
42.1	41.4	−1.570	13	14.1	15.2	7.751
36.9	35.3	−4.336	14	0.0	4.1	#DIV/OI
6.2	7.3	17.426	15	30.1	28.8	−4.451
Average error		2.359		Average error		0.083

selected points (Table III). As can be seen, the average percentage errors were 0.8% and −0.8% for θ_1 and θ_2 for the first configuration, and 2.4% and 0.1% for θ_1 and θ_2 for the second configuration when trained using the perceptron network. Training by radial basis function gave average percentage errors of 0.7% and 0.8% for θ_1 and θ_2 for the first configuration, and 0.1% and 0.8% for θ_1 and θ_2 for the second configuration. Figure 3.13 and Figure 3.14 show the differences between the desired ('calculated') and actual values obtained by using the two hidden layer perceptron network and the radial basis function, respectively.

Three types of path were used to test the weights of the networks, as shown in Figure 3.15. The paths were chosen to represent the different paths that the end-effector might take from start to its goal in the workspace. Using the weights obtained from the training session of the perceptron network and radial basis function, the position of the end-effector was used to find its joint angles of θ_1 and θ_2 for each of the paths. Figure

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

Table VI. The resulting errors after training of the radial basis function

Radial basis function.
Errors after training for train2_1-test2_1 after de-normalising
{worb2_1.mat *** dp = [25,200 0.001 0.5]}

$\theta_1' * 3$ (degrees)				$-\theta_2' * 4$ (degrees)		
target	actual	% error	n	target	actual	% error
90.0	91.2	1.299	1	0.0	−7.1	#DIV/OI
153.9	154.1	0.149	2	−150.5	−150.2	−0.198
131.3	131.3	−0.030	3	−90.9	−90.7	−0.164
96.9	97.0	0.041	4	−31.8	−31.9	0.214
78.5	80.1	2.042	5	−156.9	−158.3	0.888
92.0	91.7	−0.308	6	−27.7	−27.2	−1.962
114.3	114.3	−0.021	7	−138.6	−138.7	0.047
111.3	111.2	−0.068	8	−95.7	−95.4	−0.345
84.8	85.1	0.261	9	−40.1	−40.6	1.293
89.7	89.7	−0.024	10	−84.0	−83.9	−0.082
82.0	82.0	0.026	11	−51.3	−51.3	0.037
70.5	71.6	1.536	12	−55.6	−57.9	4.085
56.1	56.5	0.644	13	−14.1	−14.9	5.552
36.9	38.4	4.056	14	0.0	−3.0	#DIV/OI
36.3	36.4	0.313	15	−30.1	−30.3	0.752
Average error		0.661		Average error		0.778

Errors after training for train2_2-test2_2 after de-normalising
{worb2_2mat *** dp = [25,200 0.001 0.5]}

$\theta_1' * 3$ (degrees)				$-\theta_2' * 4$ (degrees)		
target	actual	% error	n	target	actual	% error
90.0	75.6	−16.043	1	0.0	24.7	#DIV/OI
3.5	4.1	19.202	2	150.5	150.0	−0.278
40.5	40.7	0.497	3	90.9	90.5	−0.350
65.1	64.9	−0.351	4	31.8	32.2	1.440
−78.5	−83.4	6.270	5	156.9	164.5	4.826
64.2	64.6	0.558	6	27.7	26.9	−3.160
−24.3	−24.3	0.181	7	138.6	138.7	0.043
15.6	15.8	1.816	8	95.7	95.6	−0.171
44.8	44.6	−0.407	9	40.1	40.4	0.835
5.7	5.9	3.015	10	84.0	83.7	−0.345
30.7	30.6	−0.236	11	51.3	51.5	0.350
14.9	13.8	−7.431	12	55.6	57.7	3.770
42.1	41.7	−0.753	13	14.1	14.5	3.190
36.9	35.7	−3.264	14	0.0	2.4	#DIV/OI
6.2	6.1	−0.896	15	30.1	30.2	0.343
Average error		0.144		Average error		0.807

3.16, Figure 3.17 and Figure 3.18 show the resulting curves from applying conventional method, perceptron network and radial basis function for paths 1, 2 and 3, respectively. For path 1, the average percentage errors of θ_1 and θ_2 were 0.6% and −0.6% for the first configuration, and −3.7% and −1.4% for the second configuration when using the weights from the perceptron network. The radial basis function gave average percentage errors of −0.2% and −0.9% for the first configuration and −1.6% and −0.5% for the second configuration for θ_1 and θ_2 respectively. The error values given are relative to the performance of the conventional method. For the second path the average percentage errors of θ_1 and θ_2 were 0.1% and −0.2% for the first configuration, and 6.6% and −0.3% for the second configuration when using the weights from the perceptron network. The radial basis function gave average percentage errors of −0.1% and −0.04% for the first configuration and 3.1% and −0.1% for the second configuration for θ_1 and θ_2 respectively. And finally for

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

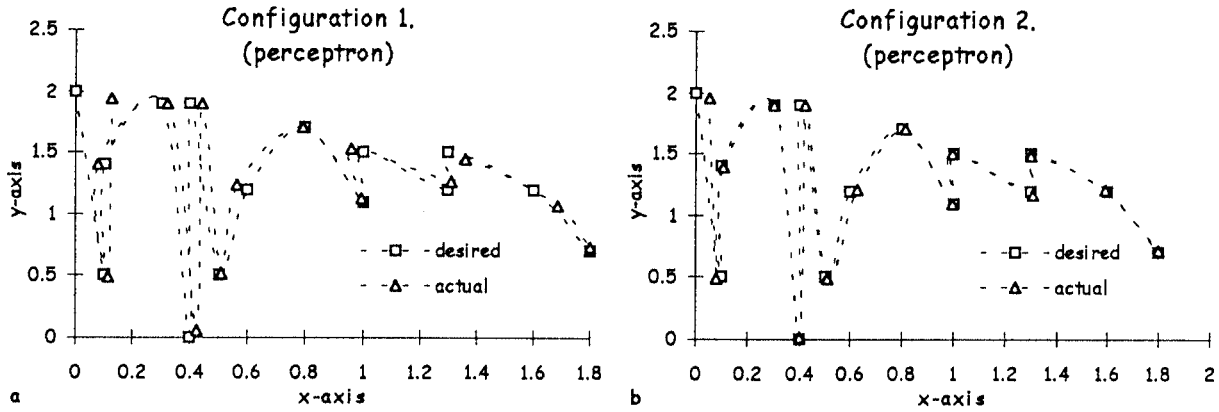

Fig. 3.13. The difference between the desired and actual co-ordinates for configurations 1 (a) and 2 (b) when using two hidden layer perceptron.

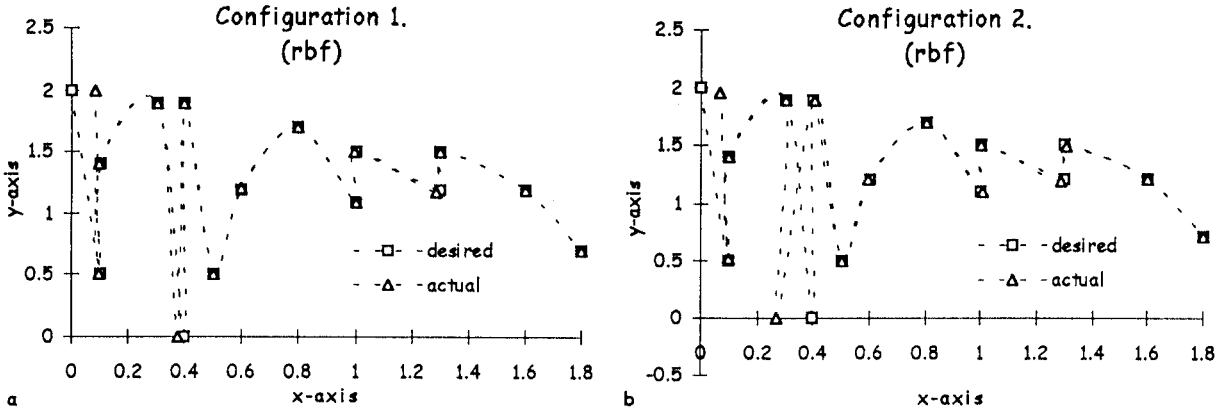

Fig. 3.14. The difference between the desired and actual co-ordinates for configurations 1 (a) and 2 (b) when using radial basis function.

path 3, the average percentage errors of θ_1 and θ_2 were 1.3% and 0.1% for the first configuration, and 2.1% and −0.3% for the second configuration when using the weights from the perceptron network. The radial basis function gave an average percentage errors of −0.1% and −0.4% for the first configuration and 4.3% and −0.4% for the second configuration for θ_1 and θ_2 respectively.

4. Discussion and conclusion

Part 1 of the work described here compared the results of training using a plain back-propagation algorithm with an adaptive back-propagation algorithm. As can be seen from the results, the adaptive back-propagation algorithm needs less training in order to generalise the patterns taken from the end-effector's path (8,000 iterations) and also has smaller percentage errors

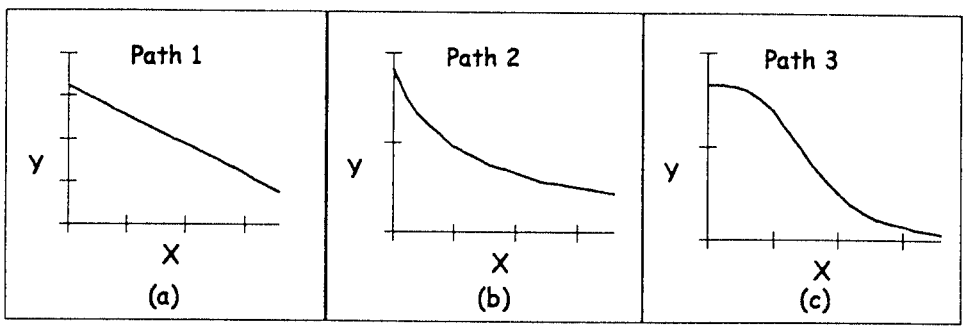

Fig. 3.15. Shown are three paths (a, b and c) as used to test the system.

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

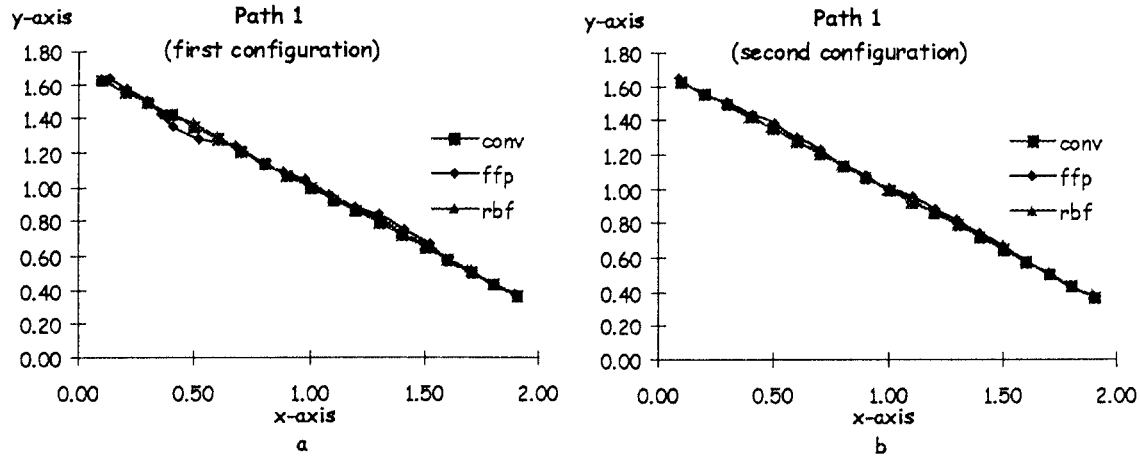

Fig. 3.16. The two graphs show the different results from the different methods applied for (a) first configuration and (b) second configuration.

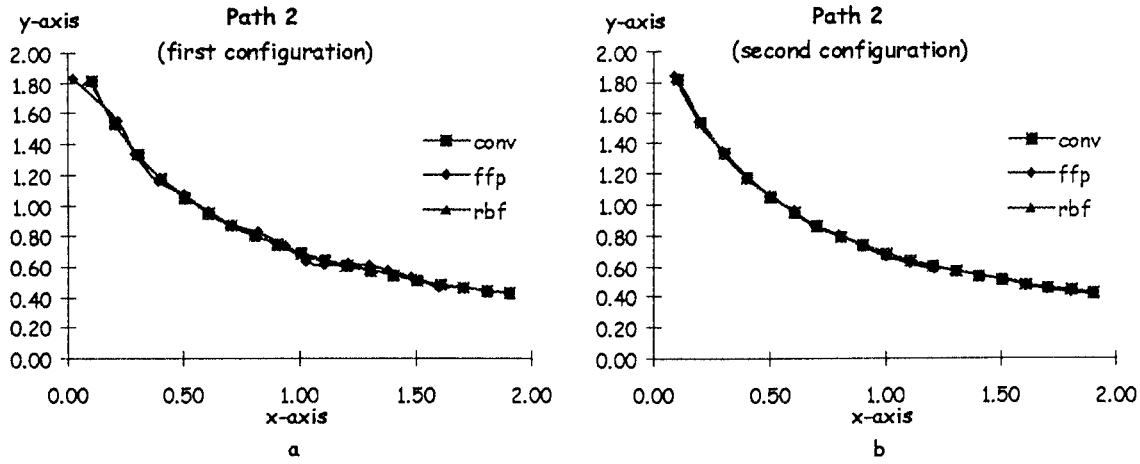

Fig. 3.17. The two graphs show the different results from the different methods applied for (a) first configuration and (b) second configuration.

because the learning rate was changing to minimise the errors. Unfortunately this is not sufficient as the path of the arm for most of the time cannot be determined well in advance. Furthermore the path might need to be changed to accommodate a new object. For this purpose, a training of the whole workspace is needed. This has the advantage of not limiting the end-effector to a predefined path and thus allowing any necessary changes. For comparison, 3 methods to finding the solution of the inverse kinematics were applied namely, conventional method, perceptron network and radial basis function. The conventional method was taken as a 'standard'

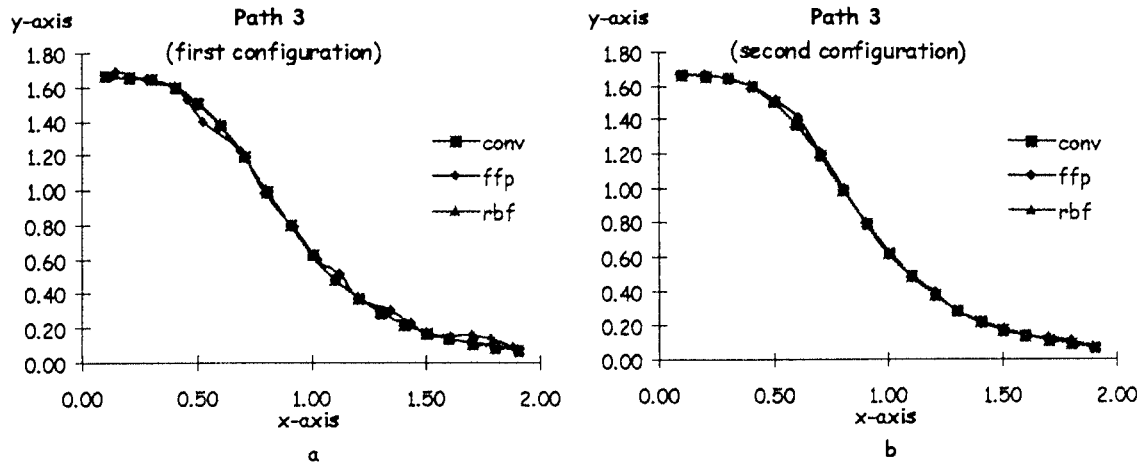

Fig. 3.18. The two graphs show the different results from the different methods applied for (a) first configuration and (b) second configuration.

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press

solution to this problem as a rigid manipulator were assumed. This might not be the case when applied to a flexible manipulator.

Quite a long training time was required for the adaptive back-propagation algorithm (20,000 iterations) while the radial basis function only required about 50 iterations before being able to determine the number of hidden neurons. This is due to the way each network transforms the input vectors through the hidden units and into the output vectors. During training, the perceptron network modified or changes its neurons' weights so as to minimise the sum squared error while the radial basis function was dependent on its design, i.e. the number of hidden neurons and their centres, to give a small sum squared error. Thus a well designed radial basis function can sometimes have a small number of hidden neurons while also giving a smaller error.

By training neural networks on the workspace of the manipulator, the inverse kinematics can be easily solved. As the workspace is fixed, this training can be done once and off-line. The same method can also be applied to a mobile manipulator as its workspace with respect to the mobile base is fixed. An appropriate transformation can be done in order to get its position with respect to the reference frame.

5. References

1. A.S. Morris and M.A. Mansor, *Robotica* **15,** *Part* 6, 617–625 (1997).
2. A.S. Morris and A. Madani, "Inclusion of shear deformation term to improve accuracy in flexible-link robot modelling", *Departmental Report* #583 (ACSE, University of Sheffield, UK, June 1995).
3. M.O. Tokhi, Z. Mohamed and A.K.M. Azad, "Finite difference and finite element simulation of a flexible manipulator", *Departmental Report* 617, *ACSE* (University of Sheffield, UK, Feb. 1996).
4. J.J. Craig, *Introduction to Robotics: Mechanics and Control*. 2nd edition (Addison-Wesley Publishing Co., Reading, Mass 1989).
5. S.S. Haykin, *Neural Networks, a comprehensive foundation* (Maxwell Mcmillan Int., New York, 1994).
6. R.C. Eberhart and R.W. Dobbins, *Neural Network PC Tools: A practical guide* (Academic Press Inc., San Diego, California, 1990).

https://doi.org/10.1017/S0263574798001064 Published online by Cambridge University Press