Trajectory Control of Robotic Manipulators Using Neural Networks

Tomochika Ozaki, Tatsuya Suzuki, *Member, IEEE,* Takeshi Furuhashi, *Member, IEEE,* Shigeru Okuma, *Member, IEEE,* and Yoshiki Uchikawa

Abstract—**This paper presents a nonlinear compensator using neural networks for trajectory control of robotic manipulators. The adaptive capability of the neural network controller to compensate unstructured uncertainties is clarified. A model learning scheme is also proposed in this paper. The learning procedure is effective and efficient in learning the manipulator dynamics, and error convergence with untrained trajectories is fast.**

I. Introduction

ROBOTIC manipulators have become increasingly important in the field of flexible automation. High-speed and high-precision trajectory tracking is one of the indispensable capabilities for versatile applications of the mechanical manipulators. Even in a well-structured setting for an industrial use, the manipulators are subjected to structured and/or unstructured uncertainties. Structured uncertainty is called to be the case of a correct dynamical model with parameter uncertainty due to imprecision on the manipulator link properties, unknown loads, inaccuracies on the torque constants of the actuators, and so on. Unstructured uncertainty corresponds to the case of unmodeled dynamics. Unmodeled dynamics results from the presence of the high-frequency mode of the manipulator, neglected time-delays, nonlinear friction, and so on. The computed torque method is an effective means for the trajectory control of robotic manipulators [1], but it has become widely recognized that the tracking performance of the method in high-speed operations is severely affected by the uncertainties mentioned above. This is especially true for direct-drive robots that have no gearing to reduce the dynamic effects. Adaptive approaches have been proposed to maintain the tracking performance of the robotic manipulators in the presence of the structured uncertainty [2], [3]. Although the adaptive controls are effective to compensate the influence of the structured uncertainty, it is not clear that the adaptive means can overcome the effects of the unstructured uncertainty.

Neural networks with a learning algorithm called back propagation are considered to be new approaches to adaptive controls [4]–[9]. In [6]–[9], a neural network was used in a feedforward loop with a conventional feedback PD controller for manipulator control. As learning went on, the feedforward/feedback compensation tended to move to the feedforward path with little feedback compensation. The inverse-dynamics model of the manipulator was considered to be obtained by the learned neural network. However, the neural network in [6]–[8] had many preprocessors computing various nonlinear transformations of input signals. The designer of the neural network controller had to know the fairly precise structure of the control object. The effectiveness of the neural network controller to compensate the unstructured uncertainties was not clear. In [9], a three-layered neural network without the preprocessor was used, and the designer of the controller was not required to know the structure of the robotic manipulator. The actual trajectory of the manipulator followed the desired one well after the learning was finished, but when the desired trajectory was changed to that not used in the training of the neural network controller, the error between the desired trajectory and the actual one became large, and more learning was necessary. This means that the inverse-dynamics model was not obtained by the learning of the neural network. Learning architectures for neural network controllers were proposed by Psaltis *et al.* [10]. The architectures seem to be more efficient in learning the nonlinear mechanical manipulators than that in [9]. However, for the general learning architecture proposed in [10], data of the inputs and outputs of the plant have to be taken by actually operating the plant in real time. The operation may take much time. Furthermore, although the neural network controllers in [9] and [10] were expected to be effective to compensate the unstructured uncertainties, no clear comparison of the tracking performances between the neural network controller and conventional adaptive schemes was made.

This paper presents a nonlinear compensator using neural networks, which incorporates the idea of the computed torque method. The neural networks are used to not learn inverse-dynamics but to compensate nonlinearities of robotic manipulators. A comparison of the performance of the proposed neural network controller with that of the adaptive controller proposed by Craig [2] is shown, and the effectiveness of the proposed neural network controller in compensating the unstructured uncertainties is clarified.

A learning scheme using a model of known dynamics of manipulators is also proposed. The model learning can be done off line and needs no data recording of actual manipulator operation. Therefore, the model learning is more efficient than the scheme in [10]. After the model learning is finished, the neural network controllers learn structured/unstructured uncertainties of the actual manipulator on line. The trained

Manuscript received January 8, 1990; revised August 8, 1990.
The authors are with the Department of Electronic Mechanical Engineering, School of Engineering, Nagoya University, Nagoya, Japan.
IEEE Log Number 9144910.

neural network is faster in error convergence than that in [9] in responding to an untrained trajectory.

II. Computed Torque Method

The robotic manipulator is modeled as a set of n rigid bodies connected in series with one end fixed to the ground and the other end free. The bodies are jointed together with revolute or prismatic joints. A torque actuator is acting at each joint. The dynamic equation of the manipulator is given by [2]

$$\tau = M(\theta)\ddot{\theta} + h(\theta,\dot{\theta}) + F \tag{1}$$

where

τ	$n \times 1$ vector of joint torques supplied by the actuators
M	$n \times n$ manipulator inertia matrix
h	$n \times 1$ vector representing centrifugal, Coriolis, and friction forces
θ	$n \times 1$ vector of joint positions
$\dot{\theta}$	$n \times 1$ vector for joint velocities
F	$n \times 1$ vector of unknown terms arising from unmodeled dynamics and external disturbances.

The control law of the computed torque method is expressed by the following equations:

$$\begin{aligned}\tau &= \hat{M}(\theta)u + \hat{h}(\theta,\dot{\theta})\\ u &= \ddot{\theta}_d + K_p(\theta_d - \theta) + K_v(\dot{\theta}_d - \dot{\theta})\end{aligned} \tag{2}$$

where $\hat{M}$ and $\hat{h}$ are the estimates of M and h, respectively. These estimates are used for nonlinear compensation of robotic manipulators. θ_d is the desired joint position. K_p and K_v are $n \times n$ diagonal matrices with each element on the diagonals being positive. The control system using (2) is shown in Fig. 1. The dynamics of error $e(= \theta_d - \theta)$ is given by substituting (2) into (1)

$$\hat{M}(\ddot{\theta}_d + K_p e + K_v\dot{e}) + \hat{h} = M\ddot{\theta} + h + F \tag{3}$$

then

$$\ddot{e} + K_v\dot{e} + K_p e = \hat{M}^{-1}((M - \hat{M})\ddot{\theta} + h - \hat{h} + F). \tag{4}$$

It is easy to see that if parameter estimation errors are not zero or unknown terms exist, asymptotic stability is not assured.

For $F = 0$ in (1), adaptive schemes of manipulator control have been proposed [2], [3]. The parameter modeling algorithms are globally asymptotically stable, but such an unstructured uncertainty as F may not be negligible in actual use, and the stability of the algorithms is not guaranteed.

III. Neural Network Controller

The proposed neural network controller is shown in Fig. 2. The robotic manipulator in Fig. 2 has the unmodeled term F. The neural networks NN1, NN2 are to identify the inertia matrix M and the term $h + F$ of the manipulator, respectively, for nonlinear compensation of the manipulator. The structure of the controller is simple and is easy to compare the performances with the adaptive controllers. Moreover,

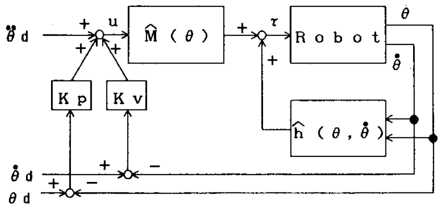


Fig. 1. Controller using computed torque method.

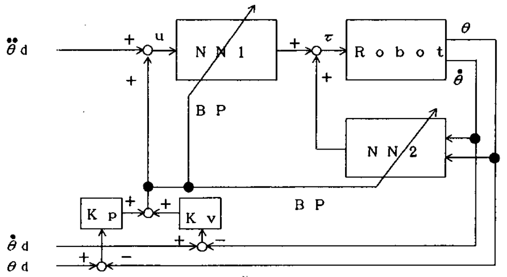


Fig. 2. Neural network controller.

the structure is suitable for the model learning, which will be discussed in Section IV.

We use a simple two-degree-of-freedom SCARA-type manipulator, as is shown in Fig. 3. The number n in (1) is two. The meaning of the symbols in Fig. 3 is listed on Table I. The unstructured uncertainty is assumed to be the Coulomb friction in this paper. The elements of the inertia matrix M_{ij} $(i, j = 1, 2)$ are

$$\begin{aligned}M_{11} &= \frac{1}{N_1}(m_1k_1^2 + m_2(L_1^2 + k_2^2\\ &\quad + 2L_1k_2\cos\theta_2 + I_1 + I_2 + J_{m1}N_1^2)\\ M_{12} &= \frac{1}{N_1}(m_2(k_2^2 + L_1k_2\cos\theta_2) + I_2)\\ M_{21} &= \frac{N_1}{N_2}M_{12}\\ M_{22} &= \frac{1}{N_2}(m_2k_2^2 + I_2 + J_{m2}N_2^2).\end{aligned} \tag{5}$$

The torques arising from centrifugal and Coriolis forces h_i $(i = 1, 2)$ are given by

$$\begin{aligned}h_1 &= \frac{1}{N_1}(-L_1k_2m_2\ \theta_2 \cdot \dot{\theta}_2^2 - 2L_1k_2m_2\\ &\quad \cdot\sin\theta_2 \cdot \dot{\theta}_1 \cdot \dot{\theta}_2 + D_{m1}N_1^2\dot{\theta}_1)\\ h_2 &= \frac{1}{N_2}(L_1k_2m_2\sin\theta_2 \cdot \dot{\theta}_1^2 + D_{m1}N_2^2\dot{\theta}_2).\end{aligned} \tag{6}$$

The elements of the Coulomb friction F_i $(i = 1, 2)$ are expressed as

$$\begin{aligned}F_1 &= T_1 \cdot \operatorname{sgn}(\dot{\theta}_1)\\ F_2 &= T_2 \cdot \operatorname{sgn}(\dot{\theta}_2).\end{aligned} \tag{7}$$

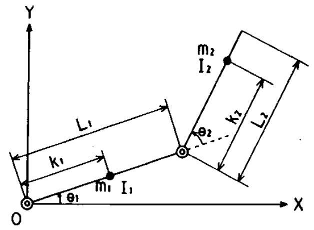


Fig. 3. Two-degree-of-freedom manipulator.

TABLE I
PARAMETERS OF MANIPULATOR

		Link 1	Link 2	Unit
Arm length	L	0.25	0.16	m
Link center of gravity	k	0.20	0.14	m
Mass	m	9.5	5.0	Kg
Inertia	I	4.3×10^{-3}	6.1×10^{-3}	$\mathrm{Kg \cdot m^2}$
Gear ratio	N	40	30	
Coulomb friction coefficient	T	0.10	0.10	$\mathrm{N \cdot m}$
Motor inertia	J_m	4.61×10^{-5}	2.65×10^{-5}	$\mathrm{Kg \cdot m^2}$
Motor damping coefficient	D_m	3.85×10^{-3}	1.39×10^{-3}	$\mathrm{N \cdot s \cdot m^{-1}}$

The structures of the neural networks NN1, NN2 are shown in Fig. 4. Both the networks are three-layered networks consisting of input, hidden, and output layers. The input signal of the input layer of NN1 is $\cos\theta_2$ because the elements of the inertia matrix M_{ij} $(i, j = 1, 2)$ have the form of $\alpha\cos\theta_2 + \beta$ (α, β: constant). Since the joint position of the link1 θ_1 does not appear in (6), the inputs to the neural network NN2 do not include θ_1. The joint torques τ applied to the manipulator is given by the following equation:

$$\tau = \tilde{M}u + \tilde{h} \tag{8}$$

where

$$\tilde{M} = \begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} \\ \tilde{M}_{21} & \tilde{M}_{22} \end{bmatrix} = \begin{bmatrix} K_1^1 O_1^1 & K_2^1 O_2^1 \\ K_3^1 O_3^1 & K_4^1 O_4^1 \end{bmatrix}$$

$$\tilde{h} = \begin{bmatrix} \tilde{h}_1 + \tilde{F}_1 \\ \tilde{h}_2 + \tilde{F}_2 \end{bmatrix} = \begin{bmatrix} K_1^2 O_1^2 \\ K_2^2 O_2^2 \end{bmatrix}.$$

O_j^l ($l = 1$ for NN1 and $l = 2$ for NN2) denotes the output signal of the output layer of each neural network. K_j^l is the matching gain for adjusting the outputs of the neural networks to the desirable torques. The units in each layer are connected through weights, and the weights are to be modified by the error back propagation [4]. The output of a unit in the output and hidden layers O_j is obtained by the following equations:

$$O_j = f(i_j) \tag{9}$$

$$i_j = \sum_i W_{ji} O_i \tag{10}$$

where i_j is the weighted sum of the outputs of the previous layer. W_{ji} denotes the weight. The function $f(\cdot)$ is called the sigmoid function and is expressed by

$$f(x) = \frac{2}{1 + \exp(-x)} - 1. \tag{11}$$

The units in the input layer of NN1 and NN2 just deliver their input signals to the units in the hidden layer. The connection strength W_{ji} is changed by an amount given by

$$\Delta W_{ji} = \eta \delta_j f'(O_j) \cdot i_i \tag{12}$$

where η is the learning rate, δ_j is given by (for the units in the output layer of NN1)

$$\delta_j = K_j^1 u_l \left(K_p(\theta_d - \theta) + K_v(\dot{\theta}_d - \dot{\theta})\right) \tag{13}$$
$$l = 1 \quad \text{when } j = 1, 2$$
$$l = 2 \quad \text{when } j = 3, 4$$

(for the units in the output layer of NN2)

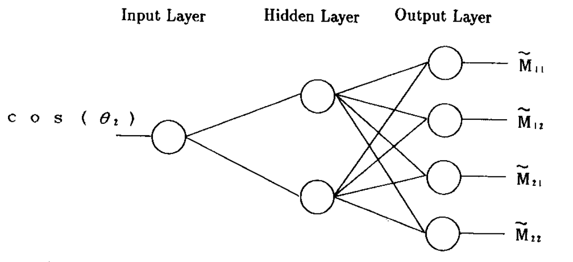
$$\delta_j = K_j^2 \left(K_p(\theta_d - \theta) + K_v(\dot{\theta}_d - \dot{\theta})\right) \tag{14}$$

(and for the units in the hidden layer of NN1 and NN2)

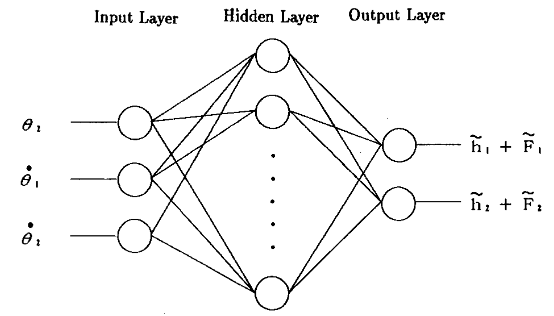
$$\delta_j = \sum_l W_{lj} \delta_l. \tag{15}$$

IV. SIMULATION

Simulations were done to verify the proposed neural network controller compensating unstructured uncertainties. The simulation conditions of the robotic manipulator are listed in Table I. The manipulator used for the simulation was a two-degree-of-freedom SCARA-type one. The matching gains are listed in Table II. The structured uncertainty in this paper was the Coulomb friction defined by (7). The link inertia and the link centroid position are the main parameters, which are difficult to measure. Here, the parameter uncertainty was



(a)



(b)

Fig. 4. Structure of neural networks: (a) Neural network (NN1); (b) neural network (NN2).

TABLE II
MATCHING GAINS

	Symbol	Gain
*NN*1	K_1^1	0.05
	K_2^1	0.005
	K_3^1	0.005
	K_4^1	0.005
*NN*2	K_1^2	0.5
	K_2^2	0.5

assumed to be the deviation of the link centroid position as follows:

$$\hat{k}_1 = 1.2 \cdot k_1$$
$$\hat{k}_2 = 1.2 \cdot k_2. \tag{16}$$

$\hat{k}_1$ and $\hat{k}_2$ were used for calculating $\hat{M}$ and $\hat{h}$ for the computed torque method. The Coulomb friction was not incorporated into the control law.

Fig. 5 shows the simulation results of the trajectory control by the neural network controller. The figure shows the first, 100th, and 200th trial of writing a circle on the X-Y Plane. It took about 3s for one trial, and the sampling period was 2 ms. The proportional gain K_p and the differential gain K_v were set at 20 and 5, respectively. At the outset of the simulation, the connection weights of the neural network W_{ji} were randomly initialized. As the learning of the neural network proceeded, the endpoint trajectory of the manipulator well followed the desired one. At the 200th trial, the tracking error converged to the value shown in Fig. 5(d). The tracking error E is defined as follows:

$$E = \sum_{i=1}^{N} (X_{di} - X_i)^2 + (Y_{di} - Y_i)^2 \tag{17}$$

where N is the sampling number in writing a circle. X_{di} and Y_{di} are the desired trajectories on the X-Y plane at the ith sampling period. X_i and Y_i are the actual trajectories.

For comparison, Fig. 6 shows the obtained endpoint trajectory by the computed torque method with the adaptive scheme [2]. The top figure is the trajectory on X-Y plane at the 100th trial. The bottom figure is the tracking error. Since the Coulomb friction is not incorporated in neither the model used for the computed torque method nor the adaptive scheme, the tracking error is approximately seven times greater than that of the neural network controller. The distinctive feature of the neural network controller over the conventional controller is that the neural network controller needs no information about the unstructured uncertainty, which was assumed to be F_i in (7) in this paper. The neural network controller can learn the unstructured uncertainty F_i as well as the inertia matrix M and the centrifugal, Coriolis forces h through the trial of following desired trajectories.

V. Model Learning

Psaltis *et al.* proposed a two-stage learning procedure [10]. The first stage is called generalized learning with its configuration shown in Fig. 7(a), and the second stage is called specialized learning with a different configuration shown in Fig. 7(b). For generalized learning, the robotic manipulator should actually be operated, and operating data should be recorded. Then, the neural network receives the obtained trajectories and is trained to yield the desired torque command. This training can be fulfilled off line. After the neural network is well trained, the neural network is installed in the feedforward loop of the manipulator controller, and specialized learning is used to fine tune the neural network on line.

This procedure is inefficient in generalized learning because of the following:

1) For recording the learning data, the robotic manipulator should actually be operated. This is time consuming.
2) It is difficult to obtain data that are uniformly distributed over the working space of the endpoint of the manipulator.

As far as robotic manipulators are concerned, we can obtain information of their dynamical models to a certain extent. We propose a learning scheme using the obtained dynamical model for the generalized learning of the neural network. Since no actual operation of the manipulator is necessary in generalized learning, mode learning is efficient. After model learning, the neural network can be trained to learn structured/unstructured uncertainties by actually operating the manipulator on line. Fig. 8 shows the configuration of the proposed model learning. The models used for the learning are $\hat{M}$ and $\hat{h}$ in (2). The unknown torque F is not incorporated in the model. The neural networks to be trained in this paper are those in Fig. 4. The inputs to the models and the neural networks are desired trajectories, and the outputs

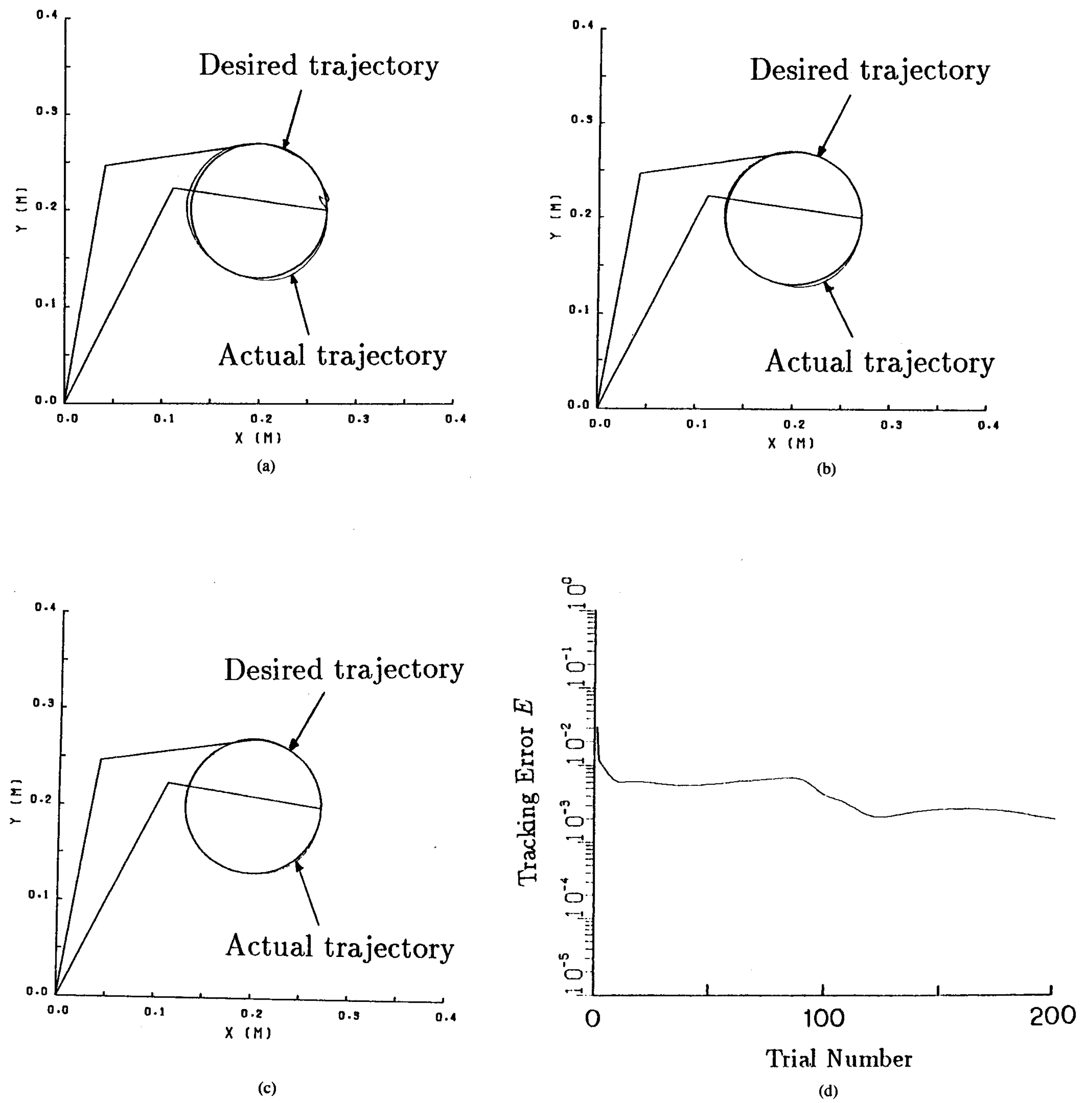
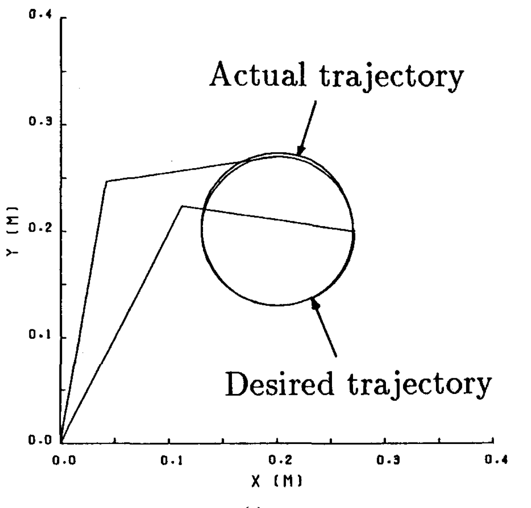


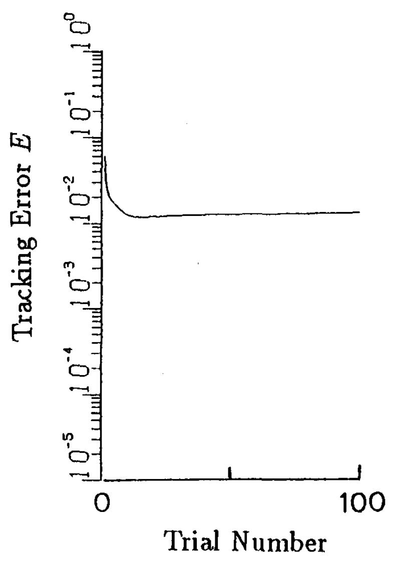
Fig. 5. Simulation results with neural network controller: (a) First trial; (b) 100th trial; (c) 200th trial; (d) tracking error convergence.

are corresponding torques. Since we can restrict the working space of the robotic manipulator, it is easy to define the ranges of the inputs, i.e., the link positions and velocities. The ranges of the inputs used for simulation are listed in Table III. By equally dividing the ranges by the numbers on the table, input data for the model learning are obtained. Since the outputs of the models are uniquely determined by the link positions and velocities, it is possible to randomly choose the combination of the inputs $\dot{\theta}_1, \theta_2, \dot{\theta}_2$. Thus, the neural network is generally trained. The neural network NN1 was trained to learn the inertia matrix $\hat{M}$, and NN2 was trained to learn $\hat{h}$. For NN2, the total number of the combination of the learning data was 25 × 25 × 25. One trial of the model learning is defined as such that every combination of the learning data is fed once to the neural networks.

After the 200th trial of the model learning, the neural networks are installed in the controller in Fig. 2. Then, the neural networks were trained to learn parameter deviations and such unmodeled effects as the Coulomb friction by actually controlling the manipulator to follow the straight line trajectory shown in Fig. 9. As for the parameter deviations, (16) was assumed. The manipulator wrote the straight line 100 times. Then, the desired trajectory was changed to a circle. Fig. 10(a) shows the trajectory at the first trial after

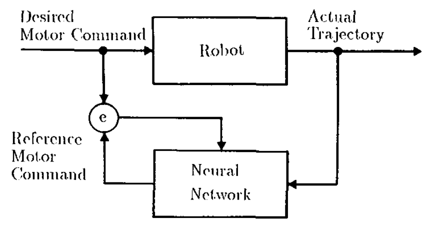


(a)

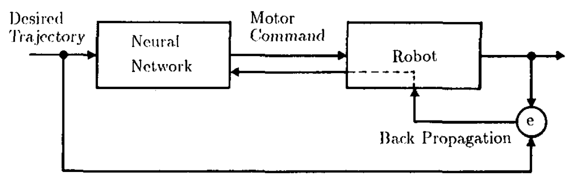


(b)

Fig. 6. Simulation results (computed torque method with adaptive scheme): (a) 100th trial; (b) tracking error convergence.



(a)



(b)

Fig. 7. Two stage learning procedure: (a) Generalized learning; (b) specialized learning.

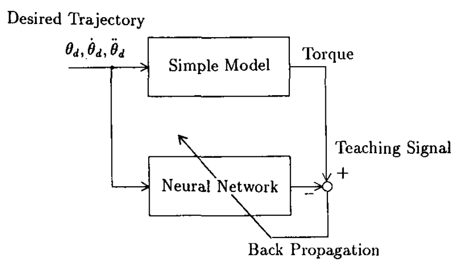


Fig. 8. Model learning.

TABLE III
INPUTS OF MODEL LEARNING

N.N.	Variables	Range	Number of Learning Data
*NN*1	θ_2	$-\frac{2}{3}\pi \sim \frac{2}{3}\pi$	200
	θ_2	$-\frac{2}{3}\pi \sim \frac{2}{3}\pi$	25
*NN*2	$\dot{\theta}_1$	$-1.5 \sim 1.5$	25
	$\dot{\theta}_2$	$-1.5 \sim 1.5$	25

the change. The neural network controller trained through the above training procedure followed the circle fairly well. Fig. 10(b) shows the case where model learning was not used. In this case, the neural network was directly trained by the system in Fig. 2 with the desired trajectory in Fig. 9. After the manipulator wrote the straight line 400 times, the tracking error converged to almost the same value as that of the model learning case. Then, the desired trajectory was changed to the same circle as that in Fig. 10(a). Fig. 10(b) shows the trajectory at the first trial after the change. The tracking error of the trajectory in Fig. 10(b) was about 30 times greater than that in Fig. 10(a). Fig. 11 shows the convergence of the tracking errors as learning goes on from the first trial in Fig. 10. The tracking error of the controller with the model learning converged fast, and the error at the 30th trial was about 1/25th that of the controller without the model learning at the 100th trial.

The neural networks that directly learned the nonlinearities of the robotic manipulator were trained to write the straight line. The networks did not learn the nonlinearities over the working space. Thus, when the desired trajectory was changed, the networks had to learn further. On the other hand, the neural networks that were trained with the simple model learned the nonlinearities of the manipulator over all of the working space. Therefore, what is necessary for the networks to learn further is only the structured/unstructured uncertainties. The simulation results show that the learning of these uncertainties takes less number of trials.

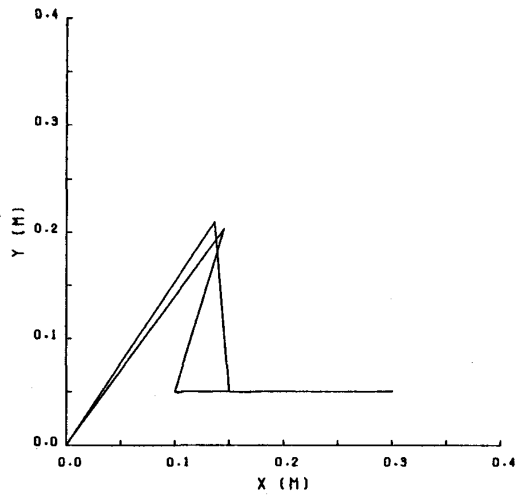
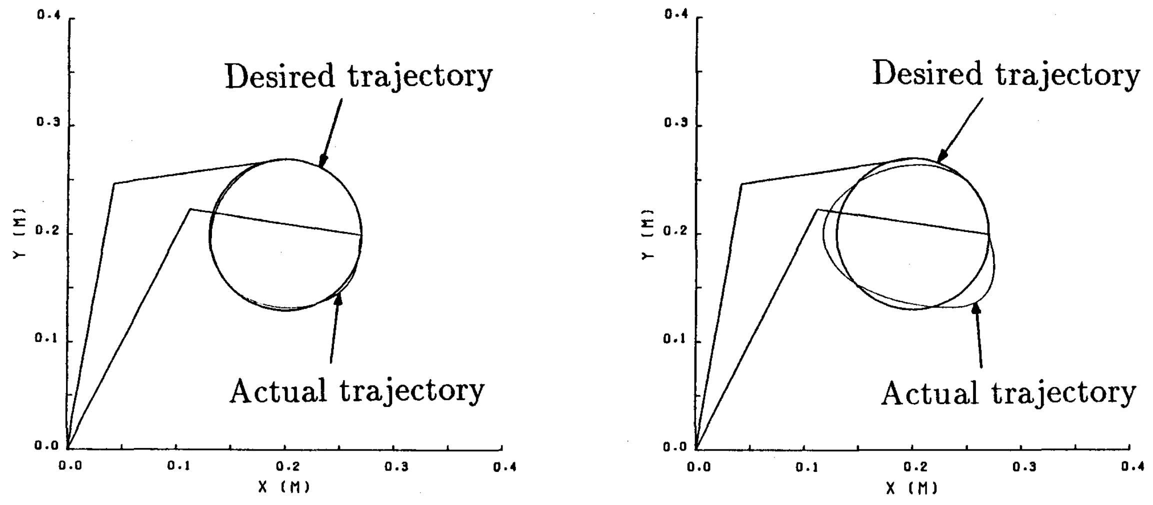


Fig. 9. Trajectory for fine tuning of the neural network.



(a) (b)

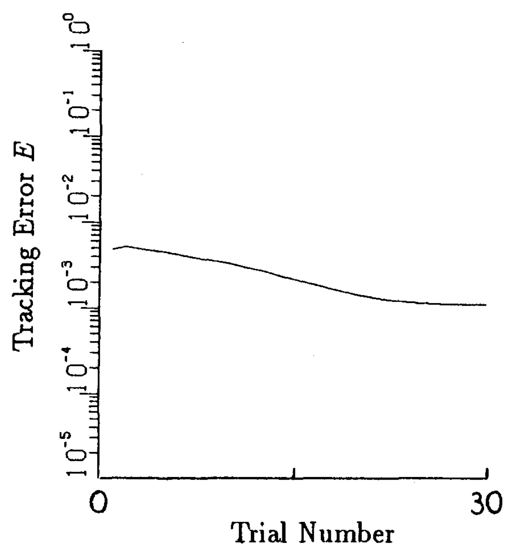
Fig. 10. Simulation results: (a) With model learning; (b) without model learning.

Link inertia and link centroid position are the most difficult to exactly measure. Approximately 20% error of the measurement is expected. Incorporating this amount of parameter deviation, the model learning scheme was confirmed to be effective. The model learning with simple models for elementary training of neural networks is an effective and efficient scheme.

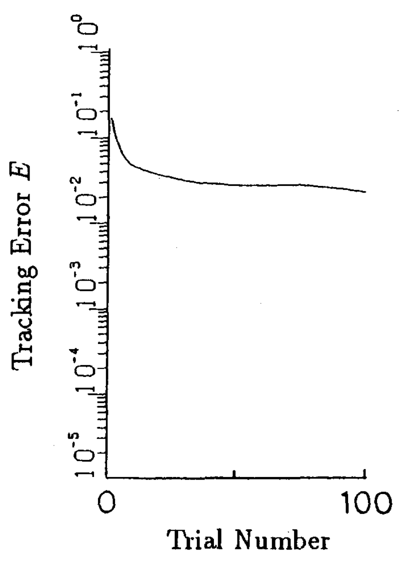
VI. Conclusions

This paper presented a nonlinear compensator using neural networks for trajectory control of robotic manipulators. A comparison of its performance with the conventional adaptive scheme in compensating the unmodeled effects was done. As a result, the adaptive capability of the neural network controller to the unstructured effects was clarified. On the contrary, the conventional scheme has no capability to overcome the unmodeled effects.

A model learning scheme was also proposed. The elementary training of the neural network using an obtained model can be fulfilled off line. After the model learning is finished, the neural network learns structured/unstructured uncertainties on line. This learning procedure is effective and efficient in learning the manipulator dynamics, and the error convergence rate with untrained trajectory is fast.



(a)



(b)

Fig. 11. Tracking errors: (a) With model learning; (b) without model learning.

References

[1] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *J. Dyn. Syst., Meas. Contr.*, vol. 102, pp. 69–76, June 1980.

[2] J. J. Craig, *Adaptive Control of Mechanical Manipulators*. Reading, MA: Addison-Wesley, 1988.

[3] J. E. Slotine and W. Li, "Adaptive manipulator control: A case study," *IEEE Trans. Automat. Contr.*, vol. 33, no. 11, pp. 995–1003, Nov. 1988.

[4] Rumelhart *et al.*, *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986.

[5] B. Bavarian, "Introduction to neural networks for intelligent control," *IEEE Contr. Syst. Mag.*, pp. 3–7, Apr. 1988.

[6] M. Kawato *et al.*, "Hierarchical neural network model for voluntary movement with application to robotic," *IEEE Contr. Syst. Mag.*, pp. 8–16, Apr. 1988.

[7] H. Miyamoto *et al.*, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, no. 3, 1988.

[8] M. Kawato, K. Furukawa, and R. Suzuki, "A hierarchical neural-network model for control and learning of voluntary movement," *Biol. Cybern.*, vol. 57, pp. 169–185, 1987.

[9] T. Setoyama, M. Kawato, and R. Suzuki, "Manipulator control by inverse-dynamic model learned in multi-layer neural network," in Japan IEICE Tech. Rep., vol. MBE87-135, pp. 249–256, 1987.

[10] D. Psaltis, A. Sideris, and A. Yamamura, "A multilayer neural network controller," *IEEE Contr. Syst. Mag.*, pp. 17–21, Apr. 1988.