'92 韓國自動制御學術會議論文集 1992. 10. 19~21

반복학습 제어를 사용한 신경회로망 제어기의 구현

최 종 호 장 태 정 백 석 찬

서울대학교 제어계측공학과, 자동화시스템공동연구소

Realization of a Neural Network controller by Using Iterative Learning Control

Chong-Ho Choi, Tae-Jeong Jang, and Seok-Chan Baek

Dept. of Control & Instrumentation Eng., Seoul National University,
Automation and Systems Research Institute

Abstract

We propose a method of generating data to train a neural network controller. The data can be prepared directly by an iterative learning technique which repeatedly adjusts the control input to improve the tracking quality of the desired trajectory. Instead of storing control input data in memory as in iterative learning control, the neural network stores the mapping between the control input and the desired output. We apply this concept to the trajectory control of a two link robot manipulator with a feedforward neural network controller and a feedback linear controller. Simulation results show good generalization of the neural network controller.

1. Introduction

Recently, many researchers have been interested in the application of neural networks to nonlinear system control. A number of architectures for neural network control systems have been presented[1–7]. In constructing a neural network control system, one important consideration is the training method of the neural network controller. Generally, the neural network is trained as a feedforward controller which learns the inverse dynamics of the plant. In this case, the neural network input is the desired output of the plant, and the neural network output is the control input of the plant which causes the desired plant output. Since it is difficult to know the plant input that causes the desired output, the neural network controller is trained by several indirect methods such as "error back propagation through the plant" [3] or "feedback error learning" [1,2], etc. We point out in this paper that the training data for the neural network controller can be prepared by the iterative learning technique[8–12], which enables us to obtain the control input that gives the desired output of the plant.

The main idea of iterative learning control is to repeatedly improve the quality of control by using information obtained from previous trials over a fixed time iteraval. The iterative learning control process begins by applying an arbitrary continuous input trajectory $u^0(\cdot) \in \Re^m \times [0,T]$ to the plant. Then $u^0(\cdot)$ is stored, along with the resulting error $e^0(\cdot) = y^d(\cdot) - y^0(\cdot) \in \Re^p \times [0,T]$, where $y^d(\cdot)$ is the desired output and $y^0(\cdot)$ is the actual output caused by the input $u^0(\cdot)$. Starting with the same initial state of the plant, the iterative learning process continues by applying to the plant a new input trajectory $u^1(\cdot)$ formed from $u^0(\cdot)$ and $e^0(\cdot)$, $u^1(\cdot)$ and the resulting error $e^1(\cdot) = y^d(\cdot) - y^1(\cdot)$ are then stored. By repeating this process, we can obtain the input trajectory $u^k(\cdot)$ arbitrarily close to the desired input trajectory $u^d(\cdot) \in \Re^m \times [0,T]$. Thus, as the number of iterations k approaches infinity, the output error $e^k(\cdot) = y^d(\cdot) - y^k(\cdot) \in \Re^p \times [0,T]$ vanishes, where $y^k(\cdot)$ is the actual output at k-th iteration.

In iterative learning, the learned control input is stored in memory and used for a particular output trajectory, but it cannot be used for different output trajectories. Thus, a lot of memory is required for various trajectories. Accomplishing almost perfect tracking of the desired output trajectory by iterative learning control means that the iterative learning control gives the desired plant input that causes the desired plant output. Accordingly, we can use the iterative learning technique to generate training data for the neural network controller. Combining neural network control and iterative learning control has some advantages. Training data for the neural network controller are prepared by the iterative learning technique, while the neural network controller stores the mapping between the desired output trajectories and the control input learned by the iterative learning.

We apply this method to the trajectory control of a two link robot manipulator with a feedforward neural network controller and a feedback linear controller. Learning the inverse dynamics of the robot manipulator is conceptually simpler than any other indirect method. Simulation results show good generalization of the neural network controller.

2. Iterative Learning Control for a Robot Manipulator

Two important considerations in using the iterative learning control are: (1) a method to generate the new input trajectory, namely the input update law which modifies the previous control input by errors between the actual and desired outputs, and (2) the convergence condition which guarantees that the output error either vanishes or reduces to within the given ϵ error bound. In this section, we illustrate the iterative learning procedure for controlling robot manipulators based on Ahn [12] for a class of nonlinear systems.

The dynamic equation of an n degree of freedom robot manipulator is expresed as the following Lagrange-Euler equation [13]:

$$\tau(t) = D(\theta(t))\ddot{\theta}(t) + h(\theta(t), \dot{\theta}(t)) + c(\theta(t)), \qquad (1)$$

where $\tau(t)$, $\theta(t)$, $D(\theta)$, $h(\theta, \dot{\theta})$, and $c(\theta)$ are the control torque, joint variables of the robot arm, inertial matrix, Coriolis and centrifugal force vector, and gravity loaded force vector, respectively.

Assume that the desired output trajectory $\theta^d(t)$, which each joint of the robot manipulator must follow, is given for a finite time interval $t \in [0, T]$. The control objective is to obtain the control input torque $\tau(t)$ which maintains the actual output error within an arbitrary small ϵ error bound, i.e.,

$$\|\theta^d(t) - \theta(t)\| < \epsilon, \quad \forall\, t \in [0, T], \qquad (2)$$

where the vector norm is defined as $\|x\| = max_{1 \le i \le n}|x_i|$.

We propose the input update law as the following:

$$\tau^{k+1}(t) = \tau^k(t) + \alpha \hat{D}(\theta^k(t)) \left(\ddot{\theta}^d(t) - \ddot{\theta}^k(t)\right), \; t \in [0, T], \qquad (3)$$

where $\hat{D}(\cdot)$ denotes the model of the inertial matrix and α ($0 < \alpha \le 1$) is a constant learning rate. The superscript k denotes the number of iterations, so τ^k and θ^k represent the control torque and link output of the k-th iteration, respectively.

The convergence condition which guarantees the accomplishment of the control object described in (2) is given as the following:

$$\|I - \alpha \hat{D}(\theta(t)) D^{-1}(\theta(t))\| < 1, \quad \forall\, t \in [0, T], \qquad (4)$$

where the matrix norm is defined as $\|G\| = \max_{1 \le i \le n}(\sum_{j=1,m} |g_{ij}|)$ for an $n \times m$ matrix $G = \{g_{ij}\}$. We can easily show that (4) is true for all $\alpha \in (0, 1]$, if the following is satisfied:

$$\|\Delta D(\theta(t)) D^{-1}(\theta(t))\| < 1, \quad \forall\, t \in [0, T], \qquad (5)$$

where $\Delta D(\theta) = D(\theta) - \hat{D}(\theta)$. In this case, we may choose α freely in $0 < \alpha \le 1$. If the modeling error $\Delta D(\theta)$ is too big to satisfy (5), the learning rate $\alpha \in (0, 1]$ should be chosen sufficiently small to satisfy (4). But choosing too small a value of α may slow down the learning process.

3. Neural Network Control by Iterative Learning for Robot Manipulators

The general architecture of neural network control system for robot manipulators is shown in Fig. 1. The control architecture is a conventional one which many researchers have adopted, especially in robotics. Similar types of control schemes may be found in the computed torque method [13], the robot control method using associative content-addressible memory (ACAM) [4], and neural network control by feedback error learning [1,2], etc. The only difference among these methods is how they construct the feedforward controller. We adopt the multilayered neural network as the feedforward controller and the PD controller as the feedback controller.

The trajectory planner generates the desired trajectories that each link must follow. The feedforward neural network controller is expected to perform like the inverse dynamics of the robot manipulator. Thus, the neural network controller is trained to generate the control torque that makes the robot manipulator follow the desired trajectories. The feedback linear controller compensates for the error between the desired and the actual trajectory. This error is induced by external disturbances or by the incomplete inversion of the robot dynamics in the feedforword controller. The feedback controller also has the role of stabilizing the overall control system.

The training procedure of the feedforward neural network controller is as follows:

(1) Pre-plan the desired trajectory of each joint with the consideration of the control object.

(2) Obtain the desired control torque sequences which give the desired output trajectories by the iterative learning technique. Then store the desired control torques and desired link outputs in memory.

(3) Train the multilayered neural network controller to simulate the inverse dynamics of the robot manipulator with the training data stored in memory. The desired link outputs are used as neural network inputs, and the desired control torques are used as neural network outputs.

Since the proposed training method can generate arbitrary training data, we can use trajectories which are good for the generalization of the neural network.

The training of the feedforward neural network controller is performed off-line. Thus, the trained neural network cannot accommodate time-varying effects such as payload changes, mechanical wear, aging, etc. The feedback controller compensates for these effects. Using an on-line training method like "feedback error learning" [1,2] may also alleviate time-varying effects.

4. Numerical Example

We consider the two link robot manipulator described in Fig. 2. The dynamic equation of motion is given as the following [13]:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{3}m_1\ell^2 + (\frac{4}{3} + cos\theta_2)m_2\ell^2 & (\frac{1}{3} + \frac{1}{2}cos\theta_2)m_2\ell^2 \\ (\frac{1}{3} + \frac{1}{2}cos\theta_2)m_2\ell^2 & \frac{1}{3}m_2\ell^2 \end{bmatrix} \times \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} -\frac{1}{2}m_2\ell^2\dot{\theta}_2^2 sin\theta_2 - m_2\ell^2\dot{\theta}_1\dot{\theta}_2 sin\theta_2 \\ \frac{1}{2}m_2\ell^2\dot{\theta}_1^2 sin\theta_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}m_1 g\ell cos\theta_1 + \frac{1}{2}m_2 g\ell cos(\theta_1 + \theta_2) + m_2 g\ell cos\theta_1 \\ \frac{1}{2}m_2 g\ell cos(\theta_1 + \theta_2) \end{bmatrix}, \quad (6)$$

where g is the gravitational acceleration. In this example, we will ignore the gravity loaded force term in the end of (6) and assume the robot manipulator is a SCARA type. We choose the link length and link masses to be $\ell = 0.5$m and $m_1 = m_2 = 2kg$, respectively. Then the dynamic equation can be rewriten as

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} \frac{5}{6} + \frac{1}{2}cos\theta_2 & \frac{1}{6} + \frac{1}{4}cos\theta_2 \\ \frac{1}{6} + \frac{1}{4}cos\theta_2 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} -\frac{1}{4}\dot{\theta}_2(2\dot{\theta}_1 + \dot{\theta}_2)sin\theta_2 \\ \frac{1}{4}\dot{\theta}_1^2 sin\theta_2 \end{bmatrix}. \quad (7)$$

Thus, the inertial matrix $D(\theta)$ is given by

$$D(\theta) = \begin{bmatrix} \frac{5}{6} + \frac{1}{2}cos\theta_2 & \frac{1}{6} + \frac{1}{4}cos\theta_2 \\ \frac{1}{6} + \frac{1}{4}cos\theta_2 & \frac{1}{6} \end{bmatrix}, \quad (8)$$

where $\theta = [\theta_1\ \theta_2]^T$. Assume that there are some errors in the model of the robot manipulator, so that we modeled the link length and link masses as $\ell = 0.6$m and $m_1 = m_2 = 1.8kg$, respectively. Then the modeled inertial matrix $\hat{D}(\theta)$ is given by

$$\hat{D}(\theta) = \begin{bmatrix} \frac{27}{25} + \frac{81}{125}cos\theta_2 & \frac{27}{125} + \frac{81}{250}cos\theta_2 \\ \frac{27}{125} + \frac{81}{250}cos\theta_2 & \frac{27}{125} \end{bmatrix}. \quad (9)$$

For the inertial matrix $D(\theta)$ and the model of the inertial matrix $\hat{D}(\theta)$ given in (8) and (9), respectively, we can show that (5) is true. Thus, the convergence condition (4) is also satisfied for all α in $0 < \alpha \leq 1$. Since we cannot know the true inertial matrix $D(\theta)$ in actual systems, we should estimate a possible region where the true inertial matrix $D(\theta)$ may exist. The learning rate α should be determined so that the convergence condition (4) is satisfied for all possible inertial matrix $D(\theta)$ in this region.

The simulsation was performed for a discretized system of (7) with sampling period $0.01sec$. The desired link output trajectories $\theta^d = [\theta_1^d\ \theta_2^d]^T$, which are also the input part of the neural network training data, have been pre-planned for 100 seconds of movement, generating 10,000 data samples. We confined the moving range of each link within $-1 < \theta_i < 1$, $i = 1, 2$, in radians. The desired link output trajectories used in the training of the neural network controller are given in Fig. 3. We generated the desired trajectories with the help of a pseudo random number generator so that the input of the neural network would be distributed as evenly as possible.

By the iterative learning procedure, we obtained the desired control torque sequence $\tau^d = [\tau_1^d\ \tau_2^d]^T$ which causes the desired output trajectory. Dividing one long time interval into several short time intervals, we could perform the iterative learning process more effectively. The learning rate α has been set to 0.9 in this example.

In this manner, the training data of the neural network controller have been prepared. The inputs for the neural network controller are θ^d, $\dot{\theta}^d$, and $\ddot{\theta}^d$, which are the desired link output trajectory and its first and second derivates, respectively. The derivatives are computed numerically. The outputs for the neural network controller should be the desired control torque τ^d learned by iterative learning. We have selected several neural network controller structures having one or two hidden layers. We have trained eath neural network 200 times for 10,000 given training data pairs by error back-propagation learning. The learning rate and the momentum were set to 0.05 and 0.8, respectively. Table 1 shows the root-mean-square error of the neural networks after training for the training data. All the results in Table 1 show little difference among various neural network structures. Complex structures do not decrease the error. We selected the 6-6-2 structure for the simulation.

The proportional gain K_p and the derivative gain K_d for the PD controller are

$$K_p = \begin{bmatrix} 20 & 0 \\ 0 & 10 \end{bmatrix}, \quad K_d = \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix}. \quad (10)$$

Then the control torque τ is given as

$$\tau = K_p(\theta^d - \theta) + K_d(\dot{\theta}^d - \dot{\theta}) + \tau_{ff} \quad (11)$$

where τ_{ff} is the output of the feedforward neural network controller. The simulation results for a part of the desired output trajectory used in neural network training is shown in Fig. 4 and Fig. 5. Though perfect tracking of the desired trajectories is not accomplished in the neural network control, the errors for the proposed control system are much smaller than the errors for the control system with only the PD controller. If we train the neural network for a trajectory with short time duration, we may make the error much smaller. But the generalization of the neural network may suffer. The simulation results for several different output trajectories which were not used in neural network training are shown in Fig. 6 and Fig. 7. They show that the generalization of the neural network controller is very good.

5. Conclusion

We pointed out in this paper that the training data for the neural network controller can be prepared directly by the iterative learning technique. Instead of storing control input data obtained by the iterative learning in memory, the neural network can store the mapping between the control input and the desired output. We applied this concept to the trajectory control of a two link robot manipulator with a feedforward neural network controller and a feedback linear controller. Good generalization of the neural network controller was seen in the simulations. The proposed technique can also be applied to a class of nonlinear systems which satisfy a certain conditions for the convergence of iterative learning.

References

[1] H. Miyamoto, M. Kawato, T. Stoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Networks*, vol. 1, pp. 251–265, 1988.

[2] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki, "Hierarchical neural network model for voluntary movement with application to robitics," *IEEE Control Systems Magazine*, vol. 8, no. 3, pp. 8–16, Apr. 1988.

[3] D. Psaltis, A. Sideris, and A. A. Yamamura, "A multilayered neural network controller," *IEEE Control Systems Magazine*, vol. 8, no. 3, pp. 17–21, Apr. 1988.

[4] C. G. Atkeson, and D. J. Reinkensmeyer, "Using associative content-addressable memories to control robots," *Neural Networks for Control*, MIT Press, Cambridge, 1990, pp. 255–286.

[5] A. Sideris, and K. Orita, "Structured learning in feedforward neural networks with application to robot trajectory control," *Inter. Joint Conf. Neural Networks*, Singapore, Nov. 1991.

[6] K. S. Narendra, and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, Mar. 1990.

[7] K. S. Narendra, and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 252–262, Mar. 1991.

[8] S. Arimoto, S. Kawamura, and F. Miyaxaki, "Bettering operation of robots by learning," *J. Robotic Systems*, vol. 1, no. 2, pp. 123–140, 1984.

[9] J. E. Hauser, "Learning control for a class of nonlinear systems," in *Proc. 26th IEEE Conf. Decision Contr.*, Los Angeles, CA, Dec. 1987, pp. 859–860.

[10] G. Heinzinger, D. Fenwick, B. Paden, and F. Miyazaki, "Stability of learning control with disturbances and uncertain initial conditions," *IEEE Trans. Automat. Contr.*, vol. 37, no. 1, pp. 110–114, 1992.

[11] H. S. Ahn and C. H. Choi, "Iterative learning controller for linear systems with periodic disturbances," *Electronics Letters*, vol. 26, no. 18, pp. 1542–1544, 1990.

[12] H. S. Ahn, "Iterative learning control for a class of nonlinear systems," Ph. D. dissertation., Dept. Control & Instrumentation Eng., Seoul National Univ., Feb. 1992.

[13] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, New York, 1987, pp. 98–102.

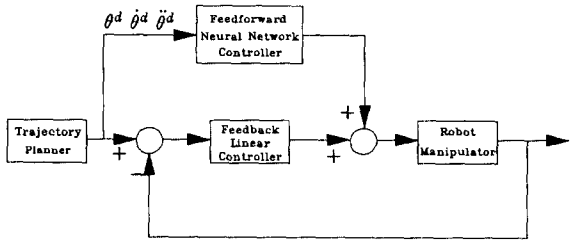

Fig. 1. Architecture of a neural network control system.

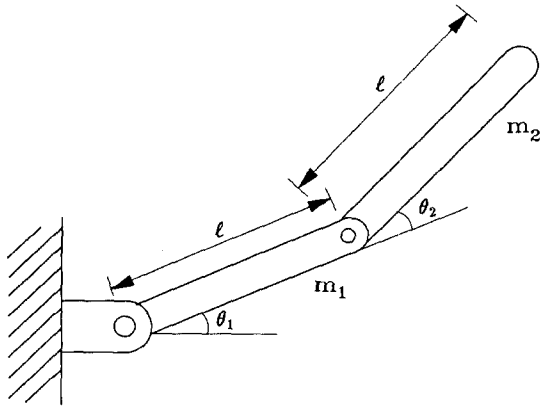

Fig. 2. A two-link robot manipulator.

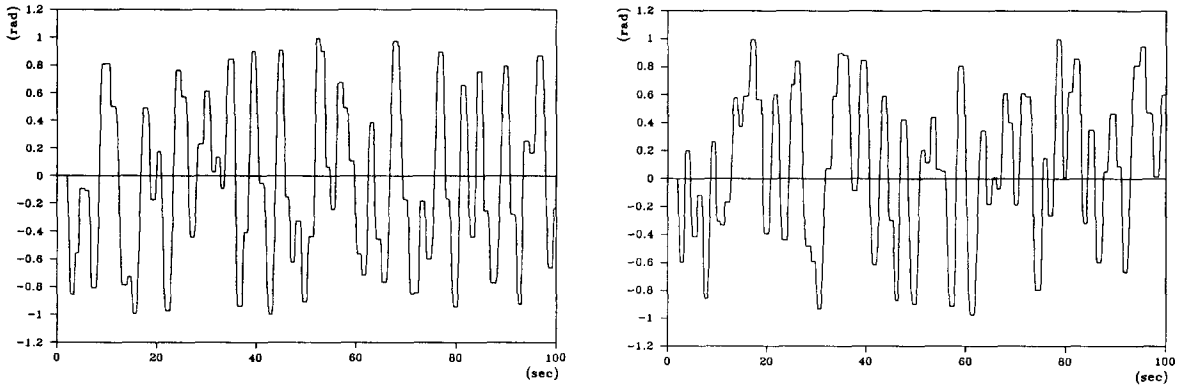

Fig. 3. Desired link output trajectories used in the training of the neural network controller: (a) link 1, (b) link 2.

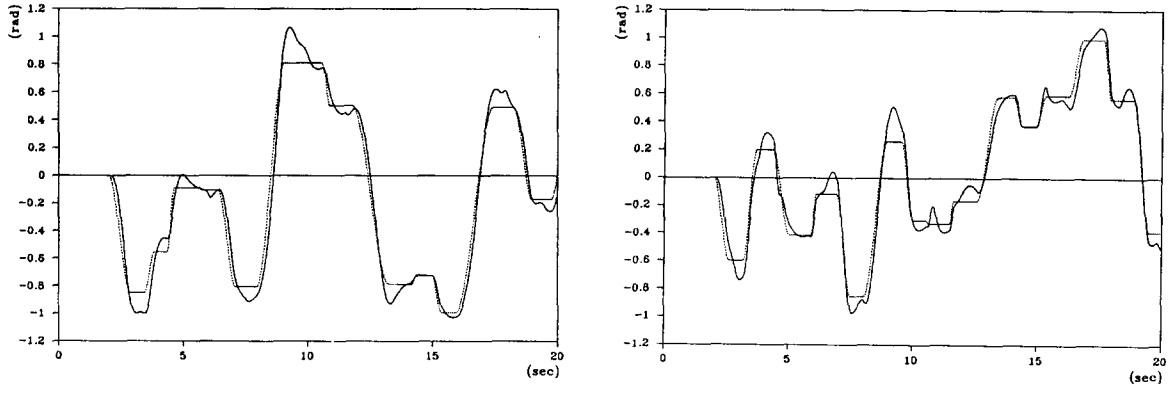

Fig. 4. Desired(dotted line) and actual outputs with the feedback PD controller for the input trajectory used in the training of the neural network controller: (a) link 1, (b) link 2.

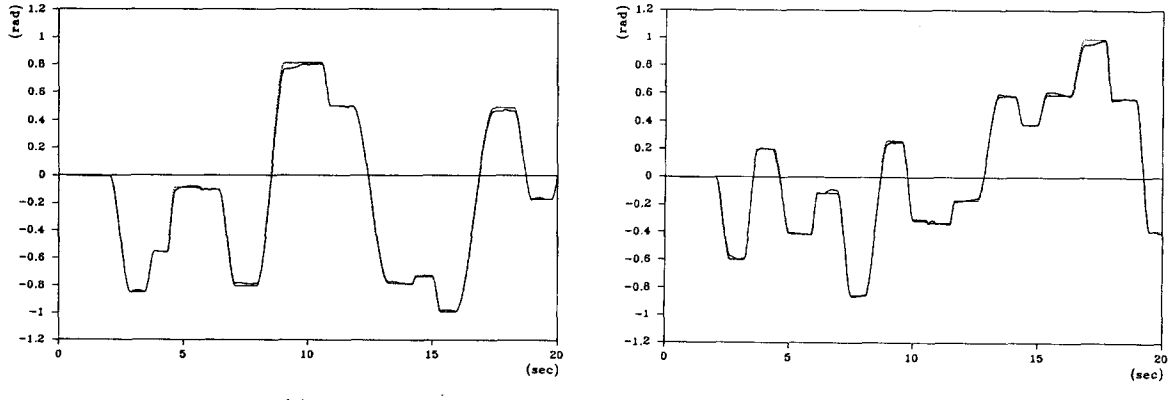

Fig. 5. Desired(dotted line) and actual outputs with the feedforward neural network controller and the feedback PD controller for the input trajectory used in the training of the neural network controller: (a) link 1, (b) link 2.

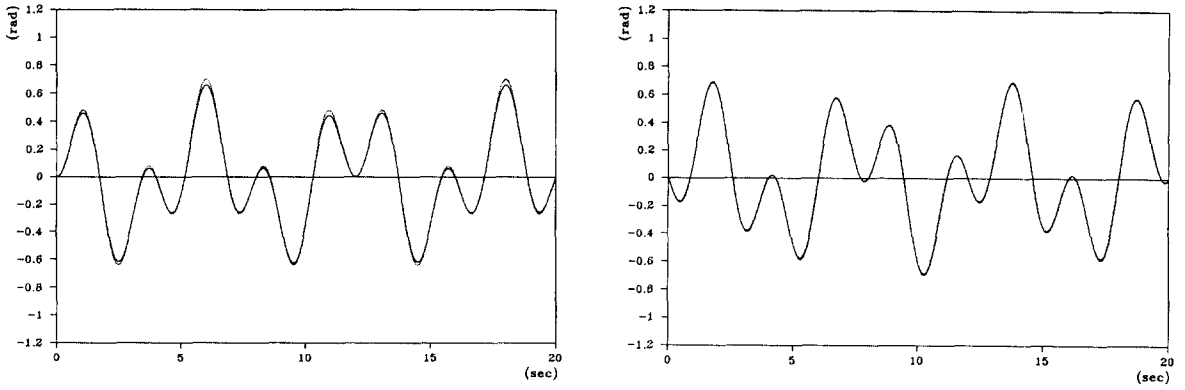

Fig. 6. Desired(dotted line) and actual outputs with the feedforward neural network controller and feedback PD controller for the test input 1: (a) link 1, (b) link 2.

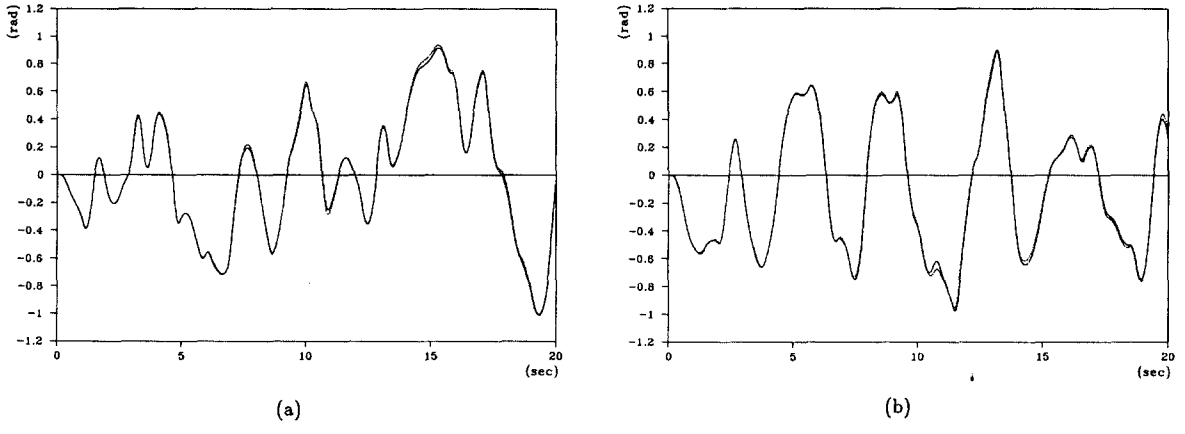

Fig. 7. Desired(dotted line) and actual outputs with the feedforward neural network controller and feedback PD controller for the test input 2: (a) link 1, (b) link 2.

Table 1. The root-mean-square (RMS) errors for various neural network structures.

Structure	6-20-10-2	6-10-5-2	6-6-3-2	6-10-2	6-6-2	6-4-2
RMS error (rad)	0.0394	0.0393	0.0395	0.0391	0.0390	0.0393