Direct Control and Coordination Using Neural Networks

Xianzhong Cui, *Member, IEEE,* and Kang G. Shin, *Fellow, IEEE*

Abstract— **The performance of an industrial process control system equipped with a conventional controller may be degraded severely by a long system—time delay, dead zone and/or saturation of actuator mechanisms, model and/or parameter uncertainties, and process noises. The coordinated control of multiple robots is another challenging problem. In a multiple-robot system, each robot is a stand-alone device equipped with commercially designed servo controllers. When such robots hold a solid object, failure of their effective coordination may damage the object and/or the robots. To overcome these problems, we propose to design a *direct* adaptive controller and a coordinator using neural networks. One of the key problems in designing such a controller/coordinator is to develop an efficient training algorithm. A neural network is usually trained using the output errors of the network, not controlled plant. However, when a neural network is used to directly control a plant, the output errors of the network are unknown, because the desired control actions are unknown. A simple training algorithm is proposed that enables the neural network to be trained with the output errors of the controlled plant. The only *a priori* knowledge of the controlled plant is the direction of its output response. A detailed analysis of the algorithm is presented and the associated theorems are proved. Due to its simple structure, algorithm and good performance, the proposed scheme has high potential for handling the difficult problems arising from industrial process control and multiple—system coordination.**

I. Introduction

THERE ARE MANY industrial control and coordination systems for which one may have difficulty in achieving high performance with conventional control designs. For example, the main problems in process control are negative effects such as a long system-response delay, dead zone and/or saturation of actuator mechanisms, and the nonlinear response of control valves. Process and measurement noises also degrade system performance. The dynamic property of a controlled plant may not be very complex, even though its detailed structure and parameters are unknown. However, when such a plant is put in operation, the control system is difficult to achieve high performance due mainly to the negative effects mentioned above.

Contemporary industrial process control systems dominantly rely on PID-type controllers, though the hardware to implement control algorithms has been improved significantly in recent years. Despite the difficulty in achieving high control quality, the fine tuning of controller parameters is a tedious task, requiring experts with knowledge both in control theory and process dynamics. Similarly, in coordinated control of multiple robots, each of the robot is a stand-alone device equipped with commercially designed servo controllers. When more than one robot must cooperate to accomplish a common goal, in addition to the good behavior of each individual robot, their effective coordination is crucial to achieve the desired level of overall performance. This coordination problem is usually organized hierarchically. The low level is the servo controllers that are designed independently of, and separately from, each other. Addition of a high-level coordinator should not require the internal structure and/or parameters of the low-level controllers to be altered. The main difficulties associated with this coordination problem come from nonlinear system dynamics, kinematic redundancy, multiple-input multiple-output (MIMO), inaccurate system parameter values, and so on. To cope with the above problems, we shall develop a new controller (coordinator) using neural networks (NNs). We shall focus on:

1) industrial process control in the presence of the nonlinearity of dead zone and saturation, and the negative effects of long response delays and process noises,
2) the coordinated control of two robots holding an object, in which each robot is equipped with commercially designed servo controllers.

The potential of NNs for control applications lies in that 1) NNs could be used to approximate any continuous mapping through learning, and 2) they can realize parallel processing and fault tolerance. One of the most popular NN architectures is a multilayer perceptron with the back propagation (BP) algorithm. It is proved that a four-layer (with two hidden layers) perceptron can be used to approximate any continuous function with the desired accuracy [4]. BP has been successfully used for pattern classification, though its original development placed more emphasis on control applications [18]. It is also proved that, in general, nonlinear control systems can be stabilized using four-layer networks [15].

A controller is usually connected serially to the controlled plant under consideration. For a multilayer perceptron, the weights of the network need to be updated using the network's output error. For an NN—controller, the NN's output is the control command to the system. However, when the NN is serially connected to a controlled plant, the network's output error is unknown, because the desired control action is unknown. This implies that the BP algorithm for training an NN cannot be applied to the control problems directly. Thus,

Manuscript received February 2, 1991; revised September 22, 1992. This work was supported in part by the National Science Foundation under Grant No. DMC-8721492.

X. Cui is with the Renaissance System Technology, Inc., 1135–1 Nielsen Ct., Ann Arber, MI 48105.

K. G. Shin is with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 9207514.

one of the key problems in designing a direct NN-controller is to develop an efficient training algorithm.

Several related schemes have been proposed. One of them is training an NN to learn the system's inverse, and then the desired system output is achieved using the control input produced by the system's inverse. Certainly, this requires the system to be invertible. Examples can be found from [3], [12], and [8]. In [3], the controlled plant was treated as an additional, unmodifiable layer, and the output error of the network was computed from the output error of the system. In [12], the system's output error was propagated back through the plant using its partial derivatives at an operating point. In [8], a set of actual system outputs are selected as training data and fed into the NN during its training period. By comparing the output of the NN with the desired system output, the network's output error is computed, which is then used to train the NN. After the NN becomes well-trained, the input of the NN is switched to the desired system output. Then, the NN acts as the inverse of the plant, and its output will drive the system to reach the desired value. However, in practice, even if the system is invertible, the inverse control scheme may be not acceptable. For example, if the system is a non-minimum phase system, then the resulting design is not internally stable. The invertibility of nonlinear systems was discussed in [5], and a sufficient-input criterion for designing an NN to learn a system's inverse was established.

Narendra and Parthasarathy [10] proposed a scheme of *indirect* adaptive control using a multilayer perceptron with the BP algorithm. The NN was trained first to attain the same dynamic behavior as the controlled plant. Then a controller was designed by using the NN's output to cancel the nonlinear part of the controlled plant and by including the same terms of a reference model. Other examples of NN-controllers are presented in [2], [6], which used reference models to train the NN. Kraft and Miller designed controllers using a structure similar to CMAC (cerebellar model articulation controller) [7], [9]. Five dominant system architectures with NNs for control applications were summarized in [18] and [19], and the importance and applications of NNs to control and system identification were also addressed there.

Most of the work mentioned above is in the form of *indirect* adaptive control or has complex training methods and system structures, and none of them was developed to coordinate multiple systems. This fact was summarized in [10] as: "At present, methods for directly adjusting the control parameters based on the output error (between the plant and the reference model output) are not available. This is because the unknown nonlinear plant lies between the controller and the output error." In contrast to the indirect adaptive control, we will develop a *direct* adaptive controller and a coordinator. A simple algorithm is proposed based on the BP for a class of nonlinear systems typified by industrial process control applications, and for a multiple-robot coordination problem. The proposed NN-controller (coordinator) is trained by using the system's output errors directly with little *a priori* knowledge of the controlled plant.

In Section II, the control problem using NNs is stated formally, and the basic structure of the proposed NN-controller (coordinator) is analyzed. The training algorithm is developed in Section III, and the corresponding theorems are proved. Section IV presents the procedures of designing the NN-controller and addresses the problems related to its implementation. Section V summarizes the simulation results of a temperature control system in a thermal power plant to test the proposed NN-controller. This is a typical system with a long response delay, nonlinearity of dead zone and saturation, and process noise. In Section VI, the coordinated control of two robots holding an object is presented, including the dynamics of the coordinated systems, specification of the desired forces, force error analysis, and the system structure with an NN-coordinator. The proposed NN-coordinator is evaluated for two 2-link robots holding an object via simulation, and the results are presented in Section VII. The paper concludes with Section VIII.

II. Problem Statement and the NN-Controller

A controlled plant can be viewed as a mapping from the control input to the system output:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{t})$$
$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{t})$$

where $\boldsymbol{x} \in \boldsymbol{R}^m$, $\boldsymbol{y} \in \boldsymbol{R}^n$, and $\boldsymbol{u} \in \boldsymbol{R}^{N_2}$ are the state, system output and input, respectively. If a controller exists for such a system, then the controller can also be represented as a mapping from system feedback and/or feedforward to the control input:

$$\boldsymbol{u} = \boldsymbol{c}(\boldsymbol{y}, \boldsymbol{y}_d, \boldsymbol{t}) \tag{1}$$

where $\boldsymbol{y}_d$ is the desired system output. As is usually the case, only the system output is assumed to be measured.

We want to design an NN-controller as the replacement of a conventional controller. In other words, the NN-controller is cascaded with the controlled plant as shown in Fig. 1, and trained to learn the mapping in (1). The desired control input $\boldsymbol{u}_d(t)$ is required to yield the desired output $\boldsymbol{y}_d(t)$. The *system-output error* and the *control-input error* are then defined, respectively, by

$$\boldsymbol{e}_y(t) = \boldsymbol{y}_d(t) - \boldsymbol{y}(t)$$

and

$$\boldsymbol{e}_u(t) = \boldsymbol{u}_d(t) - \boldsymbol{u}(t).$$

The control-input error $\boldsymbol{e}_u(t)$ is also called the *network-output error*, since $\boldsymbol{u}(t)$ is the output of the NN-controller. An NN is usually trained by minimizing the network-output error $\boldsymbol{e}_u(t)$. However, when the NN controller is cascaded in series with the controlled plant as shown in Fig. 1, $\boldsymbol{e}_u(t)$ is not known, since the desired control input $\boldsymbol{u}_d(t)$ is unknown. So, the immediate problem in designing such an NN—controller is how to train the NN.

One of the well-developed NNs is a multilayer perceptron with BP [13], [17]. The basic structure of a three-layer perceptron is shown in Fig. 2. The BP algorithm is based on the gradient algorithm to minimize the network-output error,

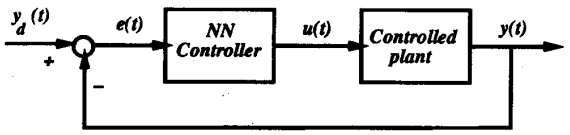


Fig. 1. A control system with an NN controller.

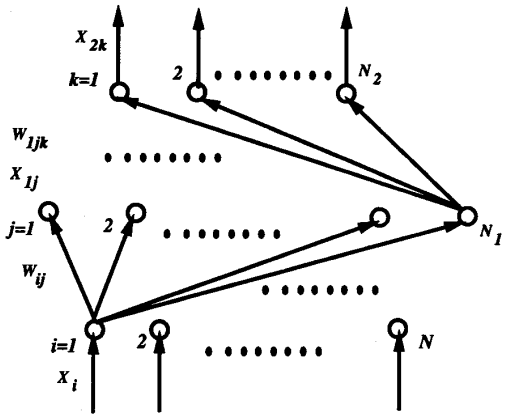


Fig. 2. Basic structure of a multilayer perceptron.

and derived from the special structure of the networks. Let θ_{1j} and θ_{2k} be the thresholds at the HIDDEN and the OUTPUT layer, respectively, where $1 \le j \le N_1$ and $1 \le k \le N_2$. Using the structure in Fig. 2, computing the NN output and updating the NN weights are summarized in the following five steps.

1) *Compute the output of the HIDDEN layer—X_{1j}:*

$$X_{1j}(t) = \frac{1}{1 + \exp(-O_{1j} - \theta_{1j})}$$

where

$$O_{1j} = \sum_{i=1}^{N} W_{ij} X_i(t), \quad j = 1, 2, \cdots, N_1.$$

2) *Compute the output of the OUTPUT layer—X_{2k}:*

$$X_{2k}(t) = \frac{1}{1 + \exp(-O_{2k} - \theta_{2k})}$$

where

$$O_{2k} = \sum_{j=1}^{N_1} W_{1jk} X_{1j}(t), \quad k = 1, 2, \cdots, N_2.$$

3) *Update the weights from the HIDDEN to the OUTPUT layer—W_{1jk}:*

$$W_{1jk}(t + \Delta t) = W_{1jk}(t) + \Delta W_{1jk} \tag{2}$$

where

$$\Delta W_{1jk} = \eta_1 \delta_{1k} X_{1j}(t) \tag{3}$$

$$\delta_{1k} = (X_{2kd}(t) - X_{2k}(t)) X_{2k}(t)(1 - X_{2k}(t)) \tag{4}$$

and X_{2kd} is the desired value of X_{2k}.

4) *Update the weights from the INPUT to the HIDDEN layer—W_{ij}:*

$$W_{ij}(t + \Delta t) = W_{ij}(t) + \Delta W_{ij}$$

where

$$\Delta W_{ij} = \eta \delta_j X_i(t)$$

$$\delta_j = \left[\sum_{k=1}^{N_2} \delta_{1k} W_{1jk}(t + \Delta t) \right] X_{1j}(t)(1 - X_{1j}(t)).$$

5) *Update the thresholds*: θ_{2k} and θ_{1j}.

$$\theta_{2k}(t + \Delta t) = \theta_{2k}(t) + \eta_{1\theta} \delta_{1k} \tag{5}$$
$$\theta_{1j}(t + \Delta t) = \theta_{1j}(t) + \eta_\theta \delta_j$$

where η, η_1, η_θ, and $\eta_{1\theta} > 0$ are the gain factors.

In any control system design, it is desired to specify the system performance in terms of system-output errors $\boldsymbol{e}_y(t) = \boldsymbol{y}_d(t) - \boldsymbol{y}(t)$, rather than the unknown network-output error $\boldsymbol{e}_u(t)$. To design such a controller with NNs, we adopt the basic principle of multilayer perceptron with BP, because of its ability of universal approximation and its convergent property based on the gradient algorithm [16]. The major obstacle in designing such an NN-controller is to train the NN with the system-output errors $\boldsymbol{e}_y(t)$, rather than the network-output errors $\boldsymbol{e}_u(t)$. The next section presents a solution to this problem.

III. Training an NN-Controller with System-Output Errors

To derive the BP algorithm, the cost function of the network is defined as:

$$E_u(t) = \frac{1}{2} \sum_{k=1}^{N_2} (e_{uk}(t))^2$$

where $e_{uk}(t) = u_{kd}(t) - u_k(t)$ is the network-output error at the kth node of the OUTPUT layer. As mentioned earlier, $E_u(t)$ is not available since $u_{kd}(t)$ is unknown for all k. Let the lth component of the system-output error be defined by

$$e_{yl}(t) = y_{ld}(t) - y_l(t), \quad l = 1, \cdots, n.$$

Then, the cost function in terms of the system-output error is defined as:

$$E_y(t) = \frac{1}{2} \sum_{l=1}^{n} (e_{yl}(t))^2 = \frac{1}{2} \sum_{l=1}^{n} (y_{ld}(t) - y_l(t))^2$$
$$= \frac{1}{2} \sum_{l=1}^{n} (\boldsymbol{G}_l(\boldsymbol{u}_d) - \boldsymbol{G}_l(\boldsymbol{u}))^2, \tag{6}$$

where $\boldsymbol{G}_l(\boldsymbol{u})$ is the lth component of the dynamic system $\boldsymbol{y}(t) = \boldsymbol{G}(\boldsymbol{u}(t))$, $\boldsymbol{y}(t) = [y_1(t), \cdots, y_n(t)]^T$, and $\boldsymbol{u}(t) = [u_1(t), \cdots, u_{N_2(t)}]^T$. Equation (6) is computable from the measurement of the system output. In other words, we know a function of the network-output error, though the detailed structure and parameters of the mapping $\boldsymbol{G}(.)$ may not be known. We want to train the NN by minimizing the cost function (6).

Using the gradient algorithm, the weights from the HIDDEN to the OUTPUT layer are modified by

$$W_{1jk}(t+\Delta t) = W_{1jk}(t) + \Delta W_{1jk}, \tag{7}$$

$$\text{and setting } \Delta W_{1jk} \propto -\frac{\partial E_y(t)}{\partial W_{1jk}(t)}. \tag{8}$$

Noting that $u_k(t) = X_{2k}(t)$ in the NN-controller,[1] we get

$$\frac{\partial E_y(t)}{\partial W_{1jk}(t)} = -\sum_{l=1}^{n}(y_{ld}(t) - y_l(t))\frac{\partial y_l(t)}{\partial u_k(t)}\frac{\partial X_{2k}(t)}{\partial O_{2k}}\frac{\partial O_{2k}}{\partial W_{1jk}(t)}. \tag{9}$$

Because $\partial X_{2k}(t)/\partial O_{2k} = X_{2k}(t)(1 - X_{2k}(t))$, and $\partial O_{2k}/\partial W_{1jk}(t) = X_{1j}(t)$, (9) becomes

$$\frac{\partial E_y(t)}{\partial W_{1jk}(t)} = -\sum_{l=1}^{n}(y_{ld}(t) - y_l(t))\frac{\partial y_l(t)}{\partial u_k(t)} \cdot X_{2k}(t)(1 - X_{2k}(t))X_{1j}(t). \tag{10}$$

Substituting (10) into (8), one can get

$$\Delta W_{1jk}(t) = \eta_1^y \delta_{1k}^y X_{1j}(t) \tag{11}$$

where

$$\delta_{1k}^y = \sum_{l=1}^{n}(y_{ld}(t) - y_l(t))\frac{\partial y_l(t)}{\partial u_k(t)}X_{2k}(t)(1 - X_{2k}(t)) \tag{12}$$

where $\eta_1^y > 0$ is a gain factor. The only unknown in (12) is $\partial y_l(t)/\partial u_k(t)$, the (l,k)th component of the Jacobian matrix of the controlled plant.

Recall that the network-output error at the kth node of the OUTPUT layer is defined as

$$e_{uk}(t) = u_{kd}(t) - u_k(t). \tag{13}$$

Referring to (12), the component of system-output error contributed by the kth control input is defined by

$$e_{sk}(t) = \sum_{l=1}^{n}(y_{ld}(t) - y_l(t))\frac{\partial y_l(t)}{\partial u_k(t)}. \tag{14}$$

To apply the gradient algorithm, we have the following theorem.

Theorem 1: Suppose the system response delay corresponding to the kth control input is d. To train the NN using the system-output error and ensure the convergence of the training algorithm, the necessary and sufficient condition is

$$\text{sign}\,(e_{sk}(t)) = \text{sign}\,(e_{uk}(t-d)). \tag{15}$$

[1]In fact, $X_{2k}(t)$ is the scaled value of $u_k(t)$. At this stage, it is assumed that the value of $u_k(t)$ is within the range of $(0,1)$. The scaling problem will be discussed later.

Proof: In the gradient algorithm, the solution converges to a minimum of the cost function if and only if the search is made along the negative direction of the gradient of the cost function. BP is based on the gradient algorithm and listed in (2) to (5). Because $u_{kd}(t) - u_k(t) = X_{2kd}(t) - X_{2k}(t)$, (4) becomes

$$\delta_{1k} = e_{uk}(t)X_{2k}(t)(1 - X_{2k}(t)) \tag{16}$$

where X_{2kd} is the desired value of X_{2k}. Substituting (14) into (12), we get

$$\delta_{1k}^y = e_{sk}(t)X_{2k}(t)(1 - X_{2k}(t)). \tag{17}$$

Because both (16) and (17) are derived by applying the gradient algorithm, in order to ensure the convergence of the training algorithm given in (7) and (11), the necessary and sufficient condition is (15), when the system response delay is accounted for. □

The accurate value of $|\partial y_l(t)/\partial u_k(t)|$ is not important, because the step size can be adjusted by setting $\eta_s \equiv \eta_1^y|\partial y_l(t)/\partial u_k(t)|$. Certainly, this requires $|\partial y_l(t)/\partial u_k(t)| < \infty, \forall t$. Therefore, if the sign of $\partial y_l(t)/\partial u_k(t)$ at each instant is known, then we get a simple algorithm to train the NN by using the system output error instead of the network output error. However, for general nonlinear systems, it is not easy to determine the sign of $\partial y_l(t)/\partial u_k(t)$ at each instant. Therefore, in what follows, we shall develop a training algorithm for a class of systems in which the sign of output response is known and $|\partial y_l(t)/\partial u_k(t)| < \infty, \forall t$. Specifically, in the next section, an NN-controller is designed for a class of SISO (single input, single output) systems, and MIMO systems are treated in Sections VI and VII.

IV. Design of the NN-Controller

For a SISO system, the training algorithm presented in the previous section can be simplified by using the following definition of *system direction.*

Definition 1: If the system output monotonically increases (decreases) as the control input to the controlled plant increases, then the system is said to be *positive-responded* (*negative- responded*). Both positive-responded and negative-responded systems are said to be *monotone-responded.*

Definition 2: For a SISO system $y(t) = G(u(t))$, if the system is positive-responded (negative-responded), then the *system direction* is written as $D(G) = 1$ ($D(G) = -1$).

Definition 1 characterizes a class of systems. For example, a linear system is cascaded with an element of pure response delay, dead zone and/or saturation. Fortunately, there are many industrial process control systems that possess the property of monotone-response. To train an NN-controller for such a class of systems, we have the following theorem.

Theorem 2: For a SISO monotone-responded system, in order to train the NN-controller in Fig. 2 using the system-output error, the weights on the arcs from the HIDDEN to the OUTPUT layer are updated by

$$W_{1j1}(t+\Delta t) = W_{1j1}(t) + \Delta W_{1j1} \tag{18}$$

where

$$\Delta W_{1j1} = \eta_1^y \delta_{11}^y X_{1j}(t)$$
$$\delta_{11}^y = (y_d(t) - y(t))D(G)X_{21}(t)(1 - X_{21}(t)).$$

Proof: For a SISO system, (13) and (14) are simplified to $e_u(t) = u_d(t) - u(t)$ and $e_s(t) = (y_d(t) - y(t))\partial y(t)/\partial u(t)$. From (15), we get the condition of convergence: sign $(e_s(t))$ = sign $(e_u(t-d))$. If the system response delay is d, then for a positive-responded system we have

$$\text{sign } (u_d(t-d) - u(t-d)) = \text{sign } (y_d(t) - y(t)). \quad (19)$$

Similarly, for a negative-responded system, we have

$$\text{sign } (u_d(t-d) - u(t-d)) = -\text{sign } (y_d(t) - y(t)). \quad (20)$$

From (19) and (20), we conclude that the condition for convergence is

$$\text{sign } (u_d(t-d) - u(t-d)) = \text{sign } (y_d(t) - y(t))D(G). \quad (21)$$

Equation (21) then implies that the corresponding training algorithm be based on (18). □

Figs. 1 and 2 show the basic structures of the system and the NN-controller, respectively. For a SISO system, there is one node at the OUTPUT layer, that is, $N_2 = 1$. The choice of the NN's inputs should reflect the desired and actual status of the controlled system. So, the inputs of the NN-controller are usually the system's desired & actual outputs, and tracking errors:

$$y_d(t), y_d(t-\Delta t), \cdots, y_d(t - m_1\Delta t)$$
$$y(t), y(t-\Delta t), \cdots, y(t - m_2\Delta t)$$
$$e_y(t), e_y(t-\Delta t), \cdots, e_y(t - m_3\Delta t)$$

where m_1, m_2 and $m_3 > 0$ are integer constants, and $e_y(t) = y_d(t) - y(t)$. The number of the HIDDEN nodes depends on the controlled plant under consideration. However, selection of a suitable number may require extensive experiments.

Based on Theorem 2, the formulas for updating the weights from the INPUT to the HIDDEN layer and the thresholds are derived using the same procedure given in Section III. The computation of the NN-controller for a SISO system is then summarized as follows.

A) *Compute the output of the HIDDEN layer*: $X_{1j}(t)$.

$$X_{1j}(t) = \frac{1}{1 + \exp(-O_{1j} - \theta_{1j})},$$
$$\text{where } O_{1j} = \sum_{i=1}^{N} W_{ij} X_i(t), j = 1, 2, \cdots, N_1.$$

B) *Compute the output of OUTPUT layer*: $X_{21}(t)$.

$$X_{21}(t) = \frac{1}{1 + \exp(-O_{21} - \theta_{21})},$$
$$\text{where } O_{21} = \sum_{j=1}^{N_1} W_{1j} X_{1j}(t).$$

C) *Update the weights from HIDDEN to OUTPUT layer*: $W_{1j1}(t)$.

$$W_{1j1}(t + \Delta t) = W_{1j1}(t) + \Delta W_{1j1},$$
$$\text{where } \Delta W_{1j1} = \eta_1^y \delta_{11}^y X_{1j}(t),$$
$$\delta_{11}^y = (y_d(t) - y(t))D(G)X_{21}(t)(1 - X_{21}(t)).$$

D) *Update the weights from INPUT to HIDDEN layer*: $W_{ij}(t)$.

$$W_{ij}(t + \Delta t) = W_{ij}(t) + \Delta W_{ij},$$
$$\text{where } \Delta W_{ij} = \eta^y \delta_j^y X_i(t),$$
$$\delta_j^y = \delta_{11}^y W_{1j1} X_{1j}(t)(1 - X_{1j}(t))$$

where η_1^y and $\eta^y > 0$ are the gain factors.

E) *Update the thresholds* : θ_{21} and θ_{1j}.

$$\theta_{21}(t + \Delta t) = \theta_{21}(t) + \eta_{1\theta}^y \delta_{11}^y,$$
$$\theta_{1j}(t + \Delta t) = \theta_{1j}(t) + \eta_\theta^y \delta_j^y,$$

where $\eta_{1\theta}^y$ and $\eta_\theta^y > 0$ are the gain factors of the thresholds at the OUTPUT and the HIDDEN layer, respectively.

Another problem in designing such an NN-controller is the choice of scaling factors. The sigmoid function in NN computation forces the NN outputs to be within the range of $(0, 1)$, although the control input $u(t)$ is limited by the range of actuators, $(U_{\min}, U_{\max})$. Therefore, the NN outputs should coincide with, or little narrower than, the range of the actuators' limits. The output of the NN-controller is then computed by

$$u(t) = X_{21}(t)(U_{\max} - U_{\min}) + U_{\min}.$$

Generally, an NN works in the mode of *train-first-then-operate*. In other words, an NN is put in operation only after it is "well-trained." By "well-trained," we mean that the weights of the NN need not be modified any more. However, for a time-varying system, it is meaningless to say that an NN is "well-trained," since the system always changes with time. Thus, not updating the weights for a time-varying system may result in the system going out of control. It is therefore necessary to always update the weights of the NN-controller. In other words, the weights of the NN-controller should be updated instead of the train-first-then-operate mode, though the weights may not have to be updated during every sampling interval.

V. Simulation Results of a Temperature Control System

Many industrial process control systems are characterized by a linear system cascaded with a nonlinear element as a result of dead zone and actuator limits, and/or a pure time delay (due to transport and system response delays). To test the capability of the proposed NN-controller, we conducted simulations while emphasizing the ability to overcome the negative effects of dead zone, saturation, long response delay, and process noise. The simulated system is a simplified temperature control system of a once-through boiler in a thermal power plant. The input is the variation of feedwater

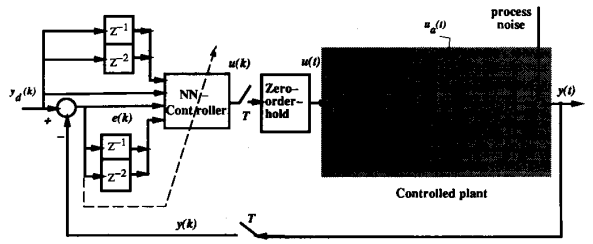


Fig. 3. The structure of NN–based control system.

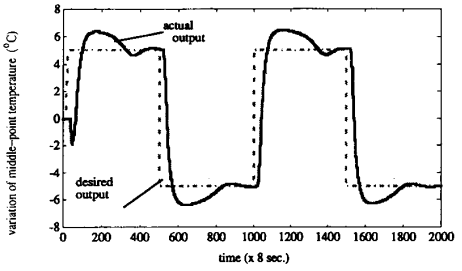


Fig. 4. System response with NN-controller when dead_zone = 5.0, and $R = 0.0$.

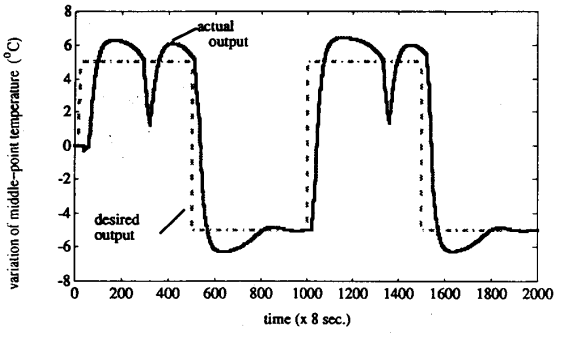


Fig. 5. System response with NN-controller when dead_zone = 7.0, and $R = 0.0$.

flow rate. The output is the variation of the temperature at the middle point where water becomes steam. The system is represented by an ARMAX model:

$$\boldsymbol{A}(z^{-1})y(k) = \boldsymbol{B}(z^{-1})u(k-d) + \boldsymbol{C}(z^{-1})\xi(k) \quad (22)$$

$$\text{where } \boldsymbol{A}(z^{-1}) = 1 - 0.45181z^{-1} - 0.47546z^{-2},$$

$$\boldsymbol{B}(z^{-1}) = -0.04560z^{-1} - 0.00404z^{-2},$$

$$\boldsymbol{C}(z^{-1}) = 1 - 0.35740z^{-1} - 0.03392z^{-2},$$

$$d = 18 \text{ sampling intervals.}$$

Here the sampling interval is chosen to be 8 seconds, $y(k)$ and $u(k)$ are the system output and control input at a discrete time k, respectively, and $\xi(k)$ is an uncorrelated random sequence with zero mean and variance R that represents the process noise. Note that this model is only for the pupose of simulation. The NN-controller has no knowledge about this system except its response direction.

To reflect the status of the controlled system, the inputs of the NN-controller are chosen as the desired system outputs and the output errors:

$$y_d(k), \quad y_d(k-1), \quad y_d(k-2), \quad y_d(k) - y(k),$$
$$y_d(k-1) - y(k-1), \quad y_d(k-2) - y(k-2).$$

That is, there are six inputs at the INPUT layer of the NN-controller ($N = 6$). The number of the HIDDEN nodes is selected to be three, that is, $N_1 = 3$. A nonlinear element of dead zone and saturation is cascaded with the system (22) to model an actuator, which is described by

$$u_a(t) = \begin{cases} 0 & \text{if } |u(t)| < \text{dead\_zone} \\ u(t) & \text{if dead\_zone } \le |u(t)| < U_{\max} \\ U_{\max} & \text{if } |u(t)| \ge U_{\max}. \end{cases}$$

The overall system structure is given in Fig. 3. The dead zone and saturation are treated as unknown properties of the controlled plant. We want to show that the NN-controller will overcome their negative effects by NN's learning ability. Actually, since system response direction will not be changed by dead zone and saturation, the proposed algorithm should still work. Moreover, no special consideration for process noise is given to the design of the NN-controller, like other deterministic controller designs, though the controllers must be tested for the ability of noise rejection.

The main simulation results are summarized below.

1) When dead_zone $= 5.0$, $U_{\max} = 10.0$, and no process noise ($R = 0.0$), the results are in Fig. 4. The initial weights of the NN are selected randomly, and the NN weights converge within 150 sampling intervals.
2) When dead_zone $= 7.0$, $U_{\max} = 10.0$, and $R = 0.0$, Figs. 5 and 6 show the system response and the corresponding control input, respectively. Obviously, a large dead zone affects the system performance severely, but the NN-controller still works well.
3) When dead_zone $= 5.0$, $U_{\max} = 10.0$, and $R = 0.5$, we evaluated the ability of noise rejection, and the desired and actual system output responses are plotted in Fig. 7. The corresponding control input and the process noise are shown in Fig. 8, where

$$n(k) = \xi(k) - 0.35740\xi(k-1) - 0.03392\xi(k-2).$$

4) Fig. 9 shows the results using $N_1 = 6$, dead_zone $= 5.0$, $U_{\max} = 10.0$ and $R = 0.0$. Comparing these results with Fig. 4, one can see that adding more HIDDEN nodes may not improve the system performance.

From the aforementioned simulation results, we conclude that the proposed NN-controller performs well for this class of nonlinear systems. In the NN-controller, the system-output error is computed from the measurements. As *a priori* knowledge, the system response direction is easily determined from either a step response experiment or the physical property of the controlled plant. To test the need of (18), $-D(G)$ is used in the training algorithm, which instantly leads to the NN's divergence.

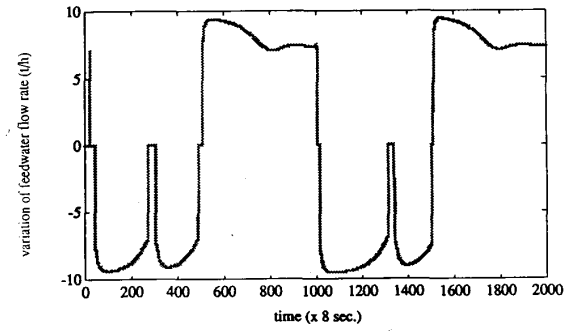


Fig. 6. System control input with NN-controller when dead_zone = 7.0, and $R = 0.0$.

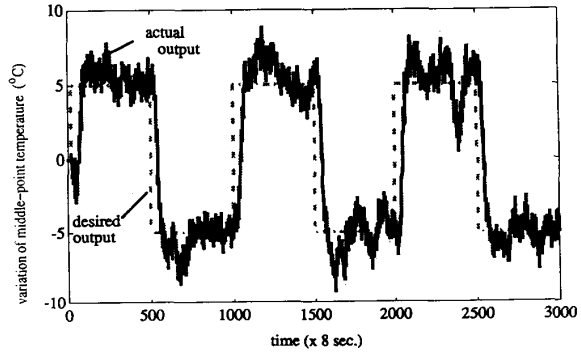


Fig. 7. System response with NN–controller when dead_zone = 5.0, and $R = 0.5$.

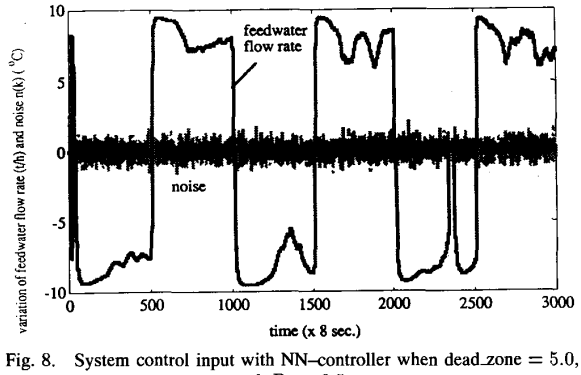


Fig. 8. System control input with NN–controller when dead_zone = 5.0, and $R = 0.5$.

VI. Design of the NN-Coordinator for Two Robots Holding an Object

The proposed algorithm is also tested for multiple-system coordination. As an example, in this section, an NN-coordinator is designed to deal with the problem of coordinating two 2-link robots holding an object. The purpose of this example is to investigate the suitability of the proposed method for MIMO systems, though controlling two 2-link robots may be not a real problem in industrial applications. With the NN-controller, the system forms a hierarchical

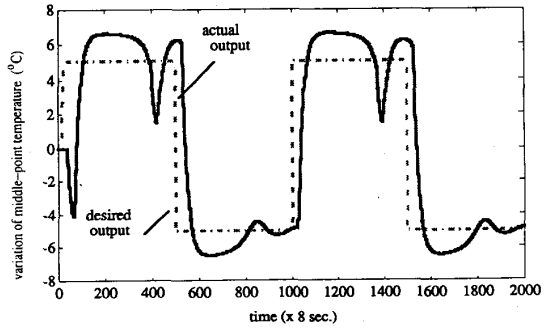


Fig. 9. System response with NN–controller for six hidden nodes.

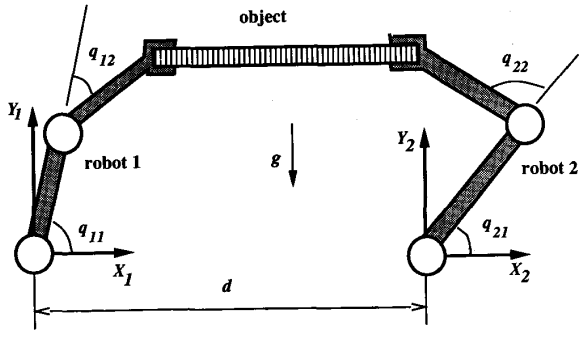


Fig. 10. Two 2–link robots holding an object.

structure: the high level is the NN-coordinator, and the low-level subsystems include two robots each with a separately designed servo controllers.

A. Dynamics of the Coordinated Systems and Problem Statement

The basic configuration of this example is given in Fig. 10. Let $\boldsymbol{f}_i = [f_{ix}, f_{iy}]^T$ be the vector of forces and torques[2] exerted by end-effector i on the object in Cartesian space, $i = 1, 2$. Then the motion of the object is described by

$$m\ddot{\boldsymbol{P}} + m\boldsymbol{g} = \boldsymbol{f}, \quad \boldsymbol{f} = \boldsymbol{WF} \equiv [\boldsymbol{I}_2, \boldsymbol{I}_2]\begin{bmatrix}\boldsymbol{f}_1 \\ \boldsymbol{f}_2\end{bmatrix} \tag{23}$$

where m is the mass of the object, $\boldsymbol{P}$ the position of the object in Cartesian space, $\boldsymbol{g}$ the gravitational acceleration, $\boldsymbol{f}$ the external force exerted on the object by the two robots, and $\boldsymbol{I}_2$ is a 2×2 unit matrix. From (23), one can see that, to achieve the object's specified acceleration, the combination of forces shared by the two robots is not unique.

Suppose two robots have an identical mechanical configuration, then the force-constrained dynamic equation of robot i in joint space is given by:

$$\boldsymbol{H}\ddot{\boldsymbol{q}}_i + \boldsymbol{h}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i) + \boldsymbol{J}_i^T \boldsymbol{f}_i = \boldsymbol{\tau}_i, i = 1, 2 \tag{24}$$

where $\boldsymbol{q}_i$ is the vector of the robot's joint positions; $\boldsymbol{H}$ the inertia matrix; $\boldsymbol{h}$ the centrifugal, Coriolis, and gravitational

[2]The term "force" will henceforth mean "force and torque," and "position" will mean "position and orientation," unless stated otherwise.

forces; J_i the Jacobian matrix; and τ_i the vector of joint torques []. In the proposed NN-coordinator, the controlled joint torques consist of two parts:

$$\tau_i = \tau_{ip} + \tau_{ic}.$$

where τ_{ic} is contributed by the NN-coordinator and τ_{ip} is given by a position controller

$$\tau_{ip} = \hat{H}(\ddot{q}_{id} - K_D(\dot{q}_i - \dot{q}_{id}) - K_p(q_i - q_{id})) + \hat{h} \quad (25)$$

where $\hat{H}$ and $\hat{h}$ are the estimated values of H and h, q_{id} is the desired value of q_i, K_D and K_p are the controllers' gains.

Suppose the object is a rigid body and there is no relative motion between the end-effectors and the object. For (23), let f_d and F_d be the desired values of f and F, respectively. Then, we have

$$F_d = F_{Md} + F_{Id} \equiv W^* f_d + (I_4 - W^* W) y_0, \quad (26)$$

where $W^* \in R^{4\times 2}$ is the pseudo-inverse of W, I_4 is a 4×4 unit matrix, and $y_0 \in R^4$ an arbitrary vector in the null space of W. Therefore, the forces exerted by the end-effectors consist of two parts: $F_{Md} = \begin{bmatrix} F_{M1d} \\ F_{M2d} \end{bmatrix} \in R^4$ is the force to move the object and $F_{Id} = \begin{bmatrix} F_{I1d} \\ F_{I2d} \end{bmatrix} \in R^4$ is the internal force. The following two problems arise: (1) sharing the moving force by the two robots, and (2) changing the internal force so as to satisfy a set of constraints, such as joint torque limits or energy capacity.

In (26), f_d can be specified by the desired trajectory. F_{Id} is given as the desired internal force, for example, $F_{Id} = 0$ for the least energy consumption. Because W^* is a constant matrix and both f_d and F_{Id} are specified, the desired force F_d is determined uniquely. However, this ideal situation of load sharing may not be achieved due to force and trajectory tracking errors. These errors may be caused by modeling/parameter errors, control performance tradeoff, and/or disturbances. It is, therefore, necessary to share the load by, or to reassign the load to, each robot dynamically. Our goal is to design an NN-coordinator for coordinate the two robots moving the object while minimizing the internal force.

Let the desired force sharing of the two robots be

$$f_{1d} = \sigma f_d + f_b, \text{ and } f_{2d} = (I_2 - \sigma) f_d - f_b, \quad (27)$$

where σ is a selection matrix, $\sigma = \text{diag}[\sigma_1, \sigma_2]$, $0 \le \sigma_j \le 1$, $j = 1, 2$; f_b a bias force. Then, the desired external and internal forces are

$$f_d = f_{1d} + f_{2d} \text{ and } f_{Id} = \frac{1}{2}(f_{1d} - f_{2d}) = f_b + \frac{1}{2}(2\sigma - I_2) f_d.$$

Therefore, the desired external force f_d, the selection matrix σ and the desired bias force f_b should be specified to compute the desired force exerted by each robot: f_{1d} and f_{2d}.

Suppose the measured forces are f_1 and f_2, then the actual external and internal forces exerted on the object are

$$f = f_1 + f_2 \text{ and } f_I = 1/2(f_1 - f_2).$$

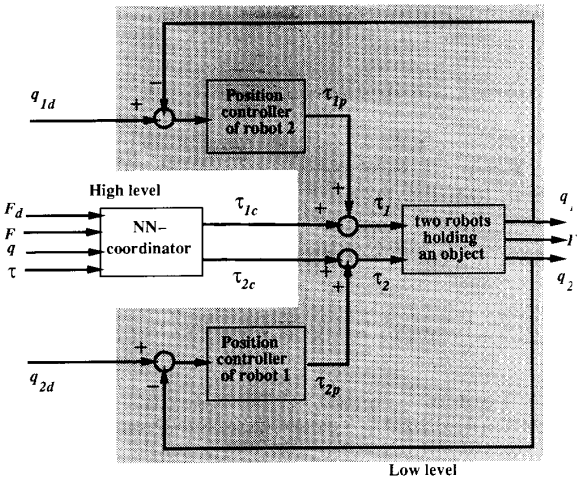


Fig. 11. The basic structure of the NN–coordinator.

Then, the force errors are

$$f_{1e} = f_{1d} - f_1 = \sigma f_d + f_b - f + f_2, \text{ and}$$
$$f_{2e} = f_{2d} - f_2 = -(\sigma f_d + f_b - f_d + f_2).$$

If the external force reached its desired value, then $f_{1e} + f_{2e} = 0$. So, these force errors do not contribute to moving the object, that is, they result from internal force errors. This implies that if we design a controller based on f_{1e} to regulate the internal force and if the controller is a linear controller with control output f_1', then the control action acted on robot 2 should be $f_2' = -f_1'$ [11]. Note that the force on the end-effector and joint torques are related by $\tau_i = J_i^T f_i$. Therefore, when the control action is transformed into the joint space, we usually have $\tau_1 \neq -\tau_2$ due to different Jacobian matrices.

Referring to (24) and (25), if $\hat{H} = H$ and $\hat{h} = h$, then the closed-loop system can be written as

$$f_i = (J_i^T)^{-1} H((\ddot{q}_{id} - \ddot{q}_i) + K_D(\dot{q}_{id} - \dot{q}_i) + K_p(q_{id} - q_i)) + (J_i^T)^{-1}\tau_{ic}.$$

Since the desired external force is specified according to the desired trajectory, the desired external force can be achieved by a well-designed position controller. We can therefore design a coordinator so as to regulate the internal force by changing τ_{ic}.

B. Design of the NN-Coordinator

The basic structure of a two-robot system equipped with an NN-coordinator is shown in Fig. 11. From the coordinator's point of view, the controlled plant is a mapping from the input torque $\tau_c = [\tau_{1c}^T, \tau_{2c}^T]^T$ to the forces exerted on the object, F. This is a time-varying MIMO mapping $F = G(\tau_c)$. We want to design an NN-coordinator to directly control such a system using the results developed in Sections III and IV. For such an MIMO system, we define the direction matrix as follows.

Definition 3: The direction matrix of an MIMO system $\boldsymbol{F} = \boldsymbol{G}(\boldsymbol{\tau}_c)$ is defined by

$$\boldsymbol{D}(\boldsymbol{G}) \equiv sign(\partial \boldsymbol{F}/\partial \boldsymbol{\tau}_c)$$

where the sign of a matrix $\boldsymbol{M}$ is defined as the matrix formed with the sign of the corresponding elements of $\boldsymbol{M}$.

When such conditions as the range of joint motion are imposed on the system, it is possible to determine this matrix as shown in Section VII. From the force analysis, the following variables may be used as the inputs of the NN-coordinator: the measured forces $\boldsymbol{F}$ exerted on the object, its desired value $\boldsymbol{F}_d$, the measured joint positions of the two robots $\boldsymbol{q} = [\boldsymbol{q}_1^T, \boldsymbol{q}_2^T]^T$, and the actual torques $\boldsymbol{\tau}$ exerted on each joint of the two robots. Obviously, all the inputs and outputs of this NN-coordinator are vectors. For such a vector-structured multilayer perceptron, a vector form of the BP algorithm has been derived in [14]. The main steps of the NN-coordinator are summarized below.

All inputs and outputs of this NN are vectors, $\boldsymbol{X}_i \in \boldsymbol{R}^n, \boldsymbol{X}_{1j} \in \boldsymbol{R}^m$, and $\boldsymbol{X}_{21} \in \boldsymbol{R}^p$ are the output of INPUT, HIDDEN and OUTPUT layer, respectively, with i nodes at the INPUT layer, $1 \le i \le N$; j nodes at the HIDDEN layer, $1 \le j \le N_1$; and one node at the OUTPUT layer. The computation includes five steps:

A) *Compute the output of the HIDDEN layer* $\boldsymbol{X}_{1j}$:

$$\boldsymbol{X}_{1j} = \boldsymbol{f}_j(\boldsymbol{O}_{1j})$$

where

$$\boldsymbol{O}_{1j} = \sum_{i=1}^{N} \boldsymbol{W}_{ij}\boldsymbol{X}_i, \quad j = 1, 2, \cdots, N_1$$

and

$$\boldsymbol{f}_j(\boldsymbol{O}_{1j}) \equiv [x_{1j1}, \cdots, x_{1jm}]^T = \left[\frac{1}{1+\exp(-o_{1j1}-\theta_{1j1})}, \cdots, \frac{1}{1+\exp(-o_{1jm}-\theta_{1jm})}\right]^T.$$

where $\boldsymbol{W}_{ij} \in \boldsymbol{R}^{m\times n}$ is the weights from the INPUT to the HIDDEN layer, $\Theta_{1j} = [\theta_{1j1}, \cdots, \theta_{1jm}]^T$ the threshold at the HIDDEN layer.

B) *Compute the output of the OUTPUT layer* $\boldsymbol{X}_{21}$:

$$\boldsymbol{X}_{21} = \boldsymbol{f}_1(\boldsymbol{O}_{21})$$

where

$$\boldsymbol{O}_{21} = \sum_{j=1}^{N_1} \boldsymbol{W}_{1j1}\boldsymbol{X}_{1j}$$

and

$$\boldsymbol{f}_1(\boldsymbol{O}_{21}) \equiv [x_{211}, \cdots, x_{21p}]^T = \left[\frac{1}{1+\exp(-o_{211}-\theta_{211})}, \cdots, \frac{1}{1+\exp(-o_{21p}-\theta_{21p})}\right]^T.$$

where $\boldsymbol{W}_{1j1} \in \boldsymbol{R}^{p\times m}$ is the weights from the HIDDEN to the OUTPUT layer, $\Theta_{21} = [\theta_{211}, \cdots, \theta_{21p}]^T$ the threshold at the OUTPUT layer.

C) *Update the weights from the HIDDEN to the OUTPUT layer* $\boldsymbol{W}_{1j1}$:

$$\boldsymbol{W}_{1j1}(t+\Delta t) = \boldsymbol{W}_{1j1}(t) + \Delta \boldsymbol{W}_{1j1}$$

where

$$\Delta \boldsymbol{W}_{1j1} = \eta_1[\boldsymbol{\delta}_{11}\boldsymbol{T}_1]^T,$$
$$\boldsymbol{\delta}_{11} = \left[\boldsymbol{X}_{21}^d - \boldsymbol{X}_{21}\right]^T \boldsymbol{D}(\boldsymbol{G}) diag[x_{211}(1-x_{211}), x_{212}(1-x_{212}), \cdots, x_{21p}(1-x_{21p})],$$

and $\boldsymbol{T}_1$ is a $p \times m \times p$ tensor with the lth matrix as

$$\boldsymbol{T}_{1l} = \begin{bmatrix} \mathbf{o} \\ \vdots \\ (\boldsymbol{X}_{1j})^T \\ \mathbf{o} \\ \vdots \end{bmatrix} \longleftarrow \text{ at the } l\text{th row}, l = 1, 2, \cdots, p.$$

D) *Update the weights from the INPUT to the HIDDEN layer* $\boldsymbol{W}_{ij}$:

$$\boldsymbol{W}_{ij}(t+\Delta t) = \boldsymbol{W}_{ij}(t) + \Delta \boldsymbol{W}_{ij}$$

where

$$\Delta \boldsymbol{W}_{ij} = \eta[\boldsymbol{\delta}_j \boldsymbol{T}_2]^T,$$
$$\boldsymbol{\delta}_j = \boldsymbol{\delta}_{11}\boldsymbol{W}_{1j1}(t+\Delta t) diag[x_{1j1}(1-x_{1j1}) x_{1j2}(1-x_{1j2}), \cdots, x_{1jm}(1-x_{1jm})]$$

and $\boldsymbol{T}_2$ is an $m \times n \times m$ tensor with the lth matrix as

$$\boldsymbol{T}_{2l} = \begin{bmatrix} \mathbf{o} \\ \vdots \\ (\boldsymbol{X}_i)^T \\ \mathbf{o} \\ \vdots \end{bmatrix} \longleftarrow \text{ at the } l\text{th row}, l = 1, 2, \cdots, m.$$

E) *Update the thresholds at the OUTPUT and the HIDDEN layer* Θ_{21}, *and* Θ_{1j}:

$$\Theta_{21}(t+\Delta t) = \Theta_{21}(t) + \Delta\Theta_{21}$$

where

$$(\Delta\Theta_{21})^T = \eta_{1\theta}\boldsymbol{\delta}_{11}$$
$$\Theta_{1j}(t+\Delta t) = \Theta_{1j}(t) + \Delta\Theta_{1j}$$

where $(\Delta\Theta_{1j})^T = \eta_\theta \boldsymbol{\delta}_j$ and $\eta_1, \eta, \eta_{1\theta}$ and $\eta_\theta > 0$ are the gain factors.

The problems of scaling input and output and updating weights have been discussed in Section IV. In what follows, the design procedures are detailed for the example and tested via simulation.

TABLE I
KINEMATIC AND DYNAMIC PARAMETERS OF THE SIMULATED REVOLUTE ROBOTS

	Length	Mass center	Mass	Moment of inertia
Link 1	1 m	0.5 m	20 kg	0.8 $kg\ m\ s^2$
Link 2	1 m	0.5 m	10 kg	0.2 $kg\ m\ s^2$

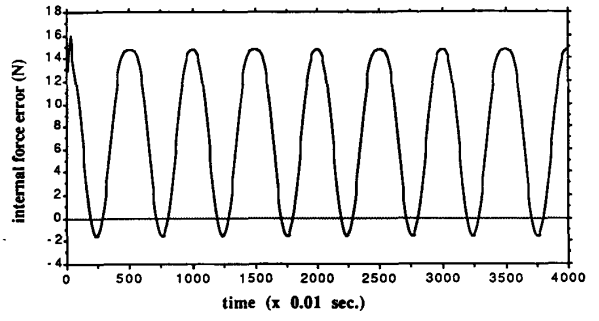


Fig. 12. The internal force error in X direction withoutthe NN–coordinator.

VII. SIMULATION RESULTS OF TWO 2-LINK ROBOTS HOLDING AN OBJECT

Referring to Fig. 10, the Cartesian frame is fixed at the base of robot 1, and the desired trajectories of the object and the robots' end-effectors are specified relative to this frame. The dynamic and kinematic parameters of the robots are presented in Table I. The task is to move the object forward and backward in X direction while keeping the height in Y direction constant. The desired trajectory that is selected by a high-level planner is to move the object in X direction from an initial position to a final position (for one meter distance) in five seconds, and then move back to the initial position. The desired velocity and acceleration of the object are zero at both initial and final positions. The sampling interval is 10 ms. The selection matrix is set to $\sigma = \text{diag}[0.5, 0.5]$ and the bias force $f_b = 0$. Each robot is position controlled with the controller in (25). Note that the NN-coordinator has no knowledge about the dynamics given by (23) and (24), except the direction matrix that is determined as follows.

Let $\boldsymbol{F} = [f_{1x}, f_{1y}, f_{2x}, f_{2y}]^T$ be the forces exerted on the object by each robot in X and Y directions, and $\tau_c = [\tau_{11c}, \tau_{12c}, \tau_{21c}, \tau_{22c}]^T$ be the torque exerted on each joint of the two robots by the NN-coordinator. Then the direction matrix is defined as:

$$\boldsymbol{D} = \text{sign}\left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\tau}_c}\right) = \text{sign}\left(\begin{bmatrix} \frac{\partial f_{1x}}{\partial \tau_{11c}} & \frac{\partial f_{1x}}{\partial \tau_{12c}} & \frac{\partial f_{1x}}{\partial \tau_{21c}} & \frac{\partial f_{1x}}{\partial \tau_{22c}} \\ \frac{\partial f_{1y}}{\partial \tau_{11c}} & \frac{\partial f_{1y}}{\partial \tau_{12c}} & \frac{\partial f_{1y}}{\partial \tau_{21c}} & \frac{\partial f_{1y}}{\partial \tau_{22c}} \\ \frac{\partial f_{2x}}{\partial \tau_{11c}} & \frac{\partial f_{2x}}{\partial \tau_{12c}} & \frac{\partial f_{2x}}{\partial \tau_{21c}} & \frac{\partial f_{2x}}{\partial \tau_{22c}} \\ \frac{\partial f_{2y}}{\partial \tau_{11c}} & \frac{\partial f_{2y}}{\partial \tau_{12c}} & \frac{\partial f_{2y}}{\partial \tau_{21c}} & \frac{\partial f_{2y}}{\partial \tau_{22c}} \end{bmatrix}\right).$$

From the configuration shown in Fig. 10, we can assume that the limitation of the joint angles are

$$0° < q_{11} < 180°, \quad -180° < q_{12} < 0°,$$
$$0° < q_{21} < 180°, \quad 0° < q_{22} < 180°.$$

Then the direction matrix can be determined as

$$\boldsymbol{D} = \text{sign}\left(\frac{\partial \boldsymbol{F}}{\partial \tau_c}\right) = \begin{bmatrix} -1 & -1 & +1 & +1 \\ +1 & +1 & +1 & +1 \\ -1 & -1 & +1 & +1 \\ +1 & +1 & +1 & +1 \end{bmatrix}.$$

This will be used in the computation of the NN-coordinator. For this example, a three-layer perceptron is used. There are four INPUT nodes with inputs

$$\boldsymbol{q} = [q_{11}, q_{12}, q_{21}, q_{22}]^T$$
$$\boldsymbol{F} = [f_{1x}, f_{1y}, f_{2x}, f_{2y}]^T$$
$$\boldsymbol{F}_d = [f_{1xd}, f_{1yd}, f_{2xd}, f_{2yd}]^T$$
$$\boldsymbol{\tau} = [\tau_{11}, \tau_{12}, \tau_{21}, \tau_{22}]^T$$

which reflect the desired and actual status of the coordinated system. The OUTPUT layer has one node with output

$$\tau_c = [\tau_{1c}^T, \tau_{2c}^T]^T = [\tau_{11c}, \tau_{12c}, \tau_{21c}, \tau_{22c}]^T.$$

The proposed NN-coordinator is evaluated via simulation and the results are summarized below.

Suppose the mass of the object is 5 kg, without the NN-coordinator, the internal force error in X direction is plotted in Fig. 12. By adding the NN-coordinator with 15 hidden nodes, the performance is greatly improved as shown in Fig. 13. The RMS (root-mean-square) error of the internal force in X direction is reduced by 94.6%. In Y direction, the RMS internal force error is reduced by 46.2%, though the internal force error is small enough due to no motion in this direction. Moreover, both the external force error and the position tracking error are kept almost the same as those without the coordinator. The detailed results are summarized in Table II.

If the mass of the object is increased to 10 kg, the NN-coordinator also works well with 20 hidden nodes. The internal force error in X direction is reduced by 89.8%, as shown in Figs. 14 and 15, and Table III.

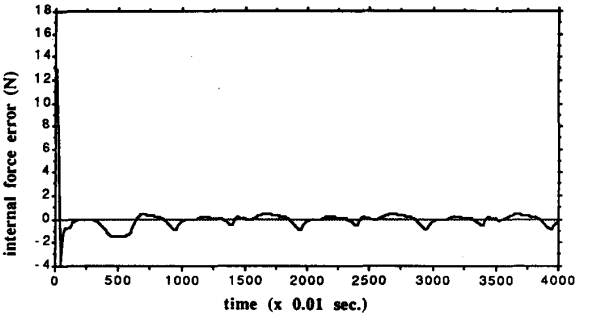


Fig. 13. The internal force error in X direction with the NN–coordinator.

TABLE II
RMS ERRORS OF INTERNAL FORCES, EXTERNAL FORCES AND OBJECT'S POSITIONS, WITH MASS = 5 kg

Sample interval		RMS errors of internal force (N)	
		without the coordinator	with the coordinator
0 — 1000	at X direction	9.58411	2.01236
	at Y direction	0.93142	0.51720
1001 — 2000	at X direction	9.57103	0.51404
	at Y direction	0.92337	0.49638
2001 — 3000	at X direction	9.57048	0.51248
	at Y direction	0.92337	0.49629
Sample interval		RMS errors of external force (N)	
		without the coordinator	with the coordinator
0 — 1000	at X direction	0.72164	0.81624
	at Y direction	2.54845	3.53886
1001 — 2000	at X direction	0.34370	0.36121
	at Y direction	0.01436	0.01104
2001 — 3000	at X direction	0.34370	0.36120
	at Y direction	0.01436	0.01105
Sample interval		RMS tracking errors of object's position (m)	
		without the coordinator	with the coordinator
0 — 1000	at X direction	0.03694	0.03772
	at Y direction	0.05733	0.00692
1001 — 2000	at X direction	0.03694	0.03773
	at Y direction	0.05759	0.00312
2001 — 3000	at X direction	0.03694	0.03773
	at Y direction	0.05759	0.00312

TABLE III
RMS ERRORS OF INTERNAL FORCES, EXTERNAL FORCES AND OBJECT'S POSITIONS, WITH MASS = 10 kg

Sample interval		RMS errors of internal force (N)	
		without the coordinator	with the coordinator
0 — 1000	at X direction	15.96137	3.75179
	at Y direction	1.12425	1.11996
1001 — 2000	at X direction	16.08066	1.89737
	at Y direction	1.10789	1.11099
2001 — 3000	at X direction	16.07986	1.84620
	at Y direction	1.10790	1.10275
Sample interval		RMS errors of external force (N)	
		without the coordinator	with the coordinator
0 — 1000	at X direction	1.40957	1.84460
	at Y direction	6.67420	9.57597
1001 — 2000	at X direction	0.67540	0.88516
	at Y direction	0.06293	0.26797
2001 — 3000	at X direction	0.67540	0.85093
	at Y direction	0.06294	0.13169
Sample interval		RMS tracking errors of object's position (m)	
		without the coordinator	with the coordinator
0 — 1000	at X direction	0.03696	0.03903
	at Y direction	0.11624	0.01390
1001 — 2000	at X direction	0.03696	0.03896
	at Y direction	0.11669	0.00657
2001 — 3000	at X direction	0.03696	0.03882
	at Y direction	0.11669	0.00652

VIII. CONCLUSION

To handle difficult control and coordination problems, we developed a direct controller and a coordinator with neural networks. Particularly, the NN—controller aims to handle industrial process control systems, in which the negative effects of a long system response delay, nonlinear elements with dead zone and/or saturation, and process noises are the main obstacles in achieving high performance. The proposed NN—controller can replace conventional controllers, and has overcome all of the problems mentioned above. The NN—coordinator is applied to the coordinated control of two robots holding an object. Such a coordinated system is organized hierarchically, where the high level is the NN—coordinator and the low level is the coordinated robots. It is assumed that each robot is a stand-alone device equipped with a commercially designed (perhaps by different vendors) servo controller. The internal structure and/or parameters of the low-level subsystems are not affected by adding the NN—coordinator. This implies that some industrial robots could be coordinated to perform more sophisticated tasks than originally intended.

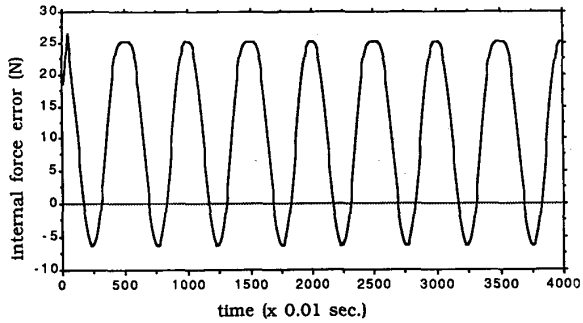


Fig. 14. The internal force error in X direction without the NN–coordinator, and mass = 10 kg.

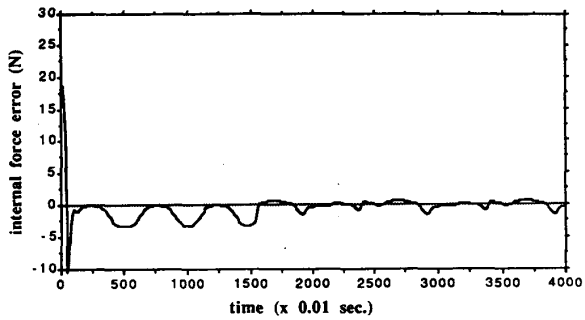


Fig. 15. The internal force error in X direction with the NN–coordinator, and mass = 10 kg.

In contrast to the scheme of indirect adaptive control [10], the proposed scheme enables the NN to be trained with system—output errors, rather than the network—output errors. The training algorithm is derived based on BP. However, in the BP algorithm, it is required to modify the weights by network—output error that is not known when a multilayer perceptron is cascaded in series to the controlled plant. Therefore, the proposed algorithm enhances the NN's ability to handle a wider range of control applications. A detailed analysis of the algorithm is presented and the associated theorems are proved. The only *a priori* knowledge about the controlled plant is the direction of its response, which is usually easy to determine for a SISO system. The direction matrix of an MIMO system can be determined, if some system constraints are imposed. Extensive simulations have been carried out and the results are shown to be quite promising. Good performance, a simple structure and algorithm, and the potential for fault tolerance make the proposed NN—controller and the NN—coordinator attractive to industrial applications.

The remaining problems include:

- Choice of the number of HIDDEN—layer nodes: there is no systematic way to choose the number of the

nodes at the HIDDEN layer(s) to approximate a given mapping. As shown in Section V, adding more HIDDEN—layer nodes may not always improve the system performance.

- Starting the NN—controller (coordinator): since the initial values of the weights are random numbers, the learning period may result in a large oscillation of the system output. This may be unacceptable for a certain controlled plant even for an open-loop stable system.

These problems will be treated in our forthcoming papers.

REFERENCES

[1] H. Asada and J.-J. E. Slotine, *Robot Analysis and Control*. New York: Wiley, 1986.

[2] I. Bar-Kana and A. Guez, "Neuromorphic adaptive control," in *Proc. 1989 IEEE Int. Conf. Decision and Contr.*, vol. 2, Dec. 1989, pp. 1739–1743.

[3] V. C. Chen and Y. H. Pao, "Learning control with neural networks," in *Proc. 1989 IEEE Conf. Robotics Automation*, Apr. 1989, pp. 1448–1453.

[4] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[5] Y. L. Gu, "On nonlinear system invertibility and learning approaches by neural networks," in *Proc. 1990 Amer. Contr. Conf.*, vol. 3, May 1990, pp. 3013–3017.

[6] A. Guez and J. Selinsky, "A neuromorphic controller with a human teacher," in *Proc. 1988 Int. Conf. Neural Networks*, vol. 2, pp. II595–II602, 1988.

[7] L. G. Kraft and D. P. Campagna, "A comparison of CMAC neural network and traditional adaptive control," in *Proc. 1989 American Control Conf.*, vol. 1, pp. 884–889, 1989.

[8] E. Levin, R. Gewirtzman, and G. F. Inbar, "Neural network architecture for adaptive system modeling and control," in *Proc. 1989 Int. Joint Conf. Neural Networks*, vol. 2, pp. 311–316, 1989.

[9] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft, "Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller," *IEEE Trans. Robotics Automat.*, vol. 6, no. 1, pp. 1–9, Feb. 1990.

[10] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Mar. 1990.

[11] M. E. Pittelkau, "Adaptive load-sharing force control for two-arm manipulators," in *Proc. IEEE Int. Conf. Robotics Automat.*, Apr. 1988, pp. 498–503.

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

[13] D. E. Rumelhart and J. L. McCelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*. Cambridge, MA: MIT Press, 1986.

[14] K. G. Shin and X. Cui, "Design of a general-purpose MIMO predictor with neural networks," *Proc. 13th IMACS World Congress on Computation and Applied Mathematics*, Dublin, Ireland, July 7, 1991.

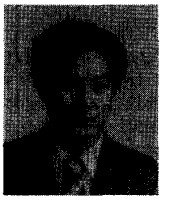
[15] E. D. Sontag, "Feedback stabilization using two-hidden-layer nets," Tech. Rep. Rutgers Center for Syst. Contr., No. SYCON–90–11, Oct. 1990.

[16] J. Wang and B. Malakooti, "On training of artificial neural networks," in *Proc. 1989 Int. Joint Conf. Neural Networks*, vol. 2, 1989, pp. 387–393.

[17] P. J. Werbos, "Back propagation: Past and future," in *Proc. 1988 Int. Conf. Neural Networks*, vol. 1, 1989, pp. I343–I353.

[18] P. J. Werbos, "Back propagation and neurocontrol: A review and prospectus," in *Proc. 1989 Int. Joint Conf. Neural Networks*, vol. 1, pp. I209–I216, 1989.

[19] P. J. Werbos, "Neural networks for control and system identification," in *Proc. 1989 IEEE Int. Conf. Decision and Contr.*, vol. 1, pp. 260–265, Dec. 1989.



Xianzhong Cui (S'89–M'92) graduated from North China Institute of Electric Power, Baoding, China, in 1976. He received the M.S. degree in power plant engineering and automation from Electric Power Research Institute (EPRI), Beijing, China, in December 1982. He received another M.S. degree and Ph.D degree, both in electrical engineering systems, in 1989 and 1992, respectively, from the University of Michigan, Ann Arbor, MI.

Currently, he is with the Renaissance System Technology, Inc., Southfield, MI, as a System Analyst working for the Powertrain Electronics Division of the Ford Motor Company. He was also an Adjunct Assistant Professor at the Department of Electrical Engineering and Computer Science, the University of Michigan in 1992. From 1976 to 1980, he served Tainjin Power Plant Construction Company as a Technician of control and instrumentation. From 1982 to 1986, he worked for EPRI as a process control engineer for power plants. He spent the 1986-1987 academic year as a Visiting Researcher in the Real-Time Computing Laboratory and Robot Systems Division, the University of Michigan. His research interests include applications of neural networks, intelligent control, industrial process control, automobile control and microprocessor-based systems, and intelligent decision support systems.

Kang G. Shin (S'75–M'78–SM'83–F'92) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

He is a Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, which he joined in 1982. He also chairs the Computer Science and Engineering Division, EECS Department. He has authored and coauthored more than 240 technical papers (more than 100 of these are in archival journals) in the areas of fault-tolerant computing, distributed real-time computing, computer architecture, and robotics and automation. From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, NY. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division, within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA.

Dr. Shin received the Outstanding IEEE TRANSACTIONS ON AUTOMATIC CONTROL Paper Award in 1987 for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from the University of Michigan, in 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are building a 19-node hexagonally mesh multicomputer a called HARTS, to validate various architectures and analytic results in the area of distributed real-time computing. He was the Program Chair of the 1986 Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS and the Guest Editor of the Special Issue on Real-Time Systems of the IEEE TRANSACTIONS ON COMPUTERS in Aug. 1987. He is a Distinguished Visitor of the Computer Society of the IEEE, an Editor for IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and an Area Editor of *International Journal of Time-Critical Computing Systems*.