Intelligent Coordination of Multiple Systems with Neural Networks

Xianzhong Cui, *Student Member, IEEE*, and Kang G. Shin, *Senior Member, IEEE*

Abstract— **Many control applications require cooperation of two or more independently designed, separately located, but mutually affecting, subsystems. In addition to the good behavior of each subsystem, effective coordination of these subsystems is very important to achieve the desired overall system performance. However, such coordination is very difficult to accomplish due mainly to the lack of precise system models and/or dynamic parameters as well as the lack of efficient tools for system analysis, design, and real-time computation of optimal solutions. A new multiple-system coordinator that combines the techniques of intelligent control and neural networks, and forms the high-level coordinator in a hierarchical structure, is proposed. The basic idea is to estimate the effects of the control commands to subsystems using a predictor and modify these commands using a knowledge-based coordinator so as to achieve the desired performance. The predictor is designed for multiple-input, multiple-output systems using neural networks. The knowledge-based coordinator is responsible for a goal-oriented search in its knowledge base and the overall system stability. Because the internal structure and parameters of the low level are not affected by using the proposed method, some commercially designed servo controllers for single systems can be coordinated to perform more sophisticated tasks for multiple systems than originally intended.**

I. Introduction

ALTHOUGH some basic principles in coordinating multiple systems were developed in early 80s [1], most related publications addressed only conceptual interpretation, and very few of them dealt with actual applications. The main difficulty in coordinating multiple systems comes from the lack of precise system models and parameters as well as the lack of efficient tools for system analysis, design, and real-time computation of optimal solutions. New methods for analysis and design are thus required for the closed-loop coordination of multiple systems.

Since intelligent control does not depend only on mathematical analyses and manipulations, it is an attractive candidate to deal with complex system control problems. An intelligent controller achieves the desired performance by searching for a goal in its knowledge base. There are three basic structures for intelligent control: *performance-adaptive*, *parameter-adaptive*, and *hierarchical structure*. The performance-adaptive structure is motivated by human expert control and/or human cognition ability, and attempts to control a system directly with an intelligent controller. Several examples of this structure are given in [2]–[5]. On the other hand, in a parameter-adaptive structure, the intelligent controller works as an on-line tuner of a conventional (usually PID) controller [6]–[8]. In a hierarchical structure, the intelligent controller [9] is a high-level controller, which attempts to modify only the reference input to the low level. The low-level subsystem could be a servo control system, and its internal structure and parameters are not affected by adding this high-level controller. One of the main tasks associated with an intelligent controller is to design a knowledge base. An inference engine will then conduct a goal-oriented search in the knowledge base according to the characteristics of system performance. The error and/or error increment of system output, and the quality of a step response are commonly used to evaluate system performance. Other additional characteristics were also suggested. For example, the estimated, dominant pole location of a closed-loop system was suggested to express system performance in [5], though no knowledge base was built on it. In [4], the output error and its derivative were arranged into a phase plane divided into 48 areas, on the basis of which rules were designed. The goal was to control the system to reach the origin of this plane. In [9], the multiple-step prediction of system output was used to characterize system performance, and the knowledge to control the system was then simply represented by a decision tree.

However, all the results reported in the literature were intended for single systems. Most of the system characteristics mentioned above may not be suitable for coordinating multiple systems, because system performance may not be easily defined and related to the measured data and control inputs. In fact, for a complex multiple-system, even human's knowledge on how to coordinate it to achieve the desired performance is limited and incomplete. So, it is difficult to design a complete knowledge base for such a system. Addition of a coordinator (not necessarily an intelligent one) leads the problem of coordinating multiple systems to form a hierarchical structure. Such an addition should not interfere with the internal structure and parameters of low-level subsystems, making the structure of performance- or parameter- adaptive intelligent controllers unsuitable for multiple-system coordination. The internal structure and/or parameters of low-level subsystems are usually not known to the coordinator. Moreover, stability analysis becomes very important, due mainly to the uncertain low-level structure and/or parameters, incomplete knowledge of the coordination and system characteristics. We should therefore answer the following questions when designing an intelligent coordinator:

Manuscript received September 12, 1990; revised March 17, 1991. This work was supported in part by the National Science Foundation under Grant No. DMC-8721492.

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 9100408.

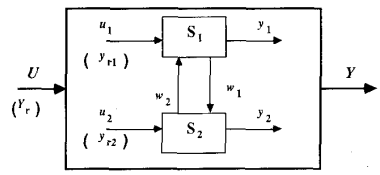


Fig. 1. Interaction of two systems.

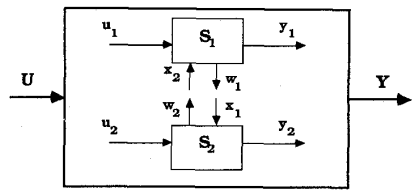


Fig. 2. Goal coordination of two systems.

1) What are the strategy and the structure to coordinate multiple systems?
2) What are the characteristics of multiple-system performance?
3) What is the knowledge necessary for coordination?
4) How should knowledge be represented?
5) How can the qualitative knowledge be extracted from sensor data?
6) How can the result of qualitative reasoning be changed into the quantitative control signals of actuators?
7) How can system stability be analyzed and guaranteed?

We propose a knowledge-based coordinator (KBC) for multiple systems by combining the techniques of intelligent control and neural networks (NN's). The KBC is a high-level coordinator within a hierarchical structure. The detailed structure and/or parameters of low-level subsystems are not required by the KBC, thus allowing individual subsystems to be designed independently. This implies that some commercially designed controllers can be coordinated to perform more sophisticated tasks than originally intended. In Section II, the problem of multiple-system coordination is stated, and some basic principles of multiple-system coordination are reviewed. The proposed scheme and the assumptions used are described in Section III. Section IV addresses the design of a KBC, including the knowledge representation, solution existence, and system stability. Section V deals with the design of an NN-based predictor with multiple-input multiple-output (MIMO). The basic structure of the NN-based predictor, a vector version of the back propagation algorithm, and the updating problem will be discussed there. As an example, the coordination of two 2-link robots holding a single object is discussed in Section VI. The paper concludes with Section VII.

II. The Problem and Principles of Multiple-System Coordination

Fig. 1 describes two interacting systems, and this description can be easily generalized to the case of more than two systems. The system dynamics are described by

$$\boldsymbol{S}_1(\boldsymbol{u}_1, \boldsymbol{y}_1, \boldsymbol{w}_2) = 0 \quad \text{and} \quad \boldsymbol{S}_2(\boldsymbol{u}_2, \boldsymbol{y}_2, \boldsymbol{w}_1) = 0$$

where $\boldsymbol{u}_i \in \boldsymbol{R}^{q_i}$, $\boldsymbol{w}_i \in \boldsymbol{R}^{m_i}$, and $\boldsymbol{y}_i \in \boldsymbol{R}^{p_i}$ for $i = 1, 2$. Let $p = p_1 + p_2$, $q = q_1 + q_2$ and $m = m_1 + m_2$, then the constraints are expressed by $\boldsymbol{S}_0 = \{(\boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{W}) : \boldsymbol{S}_1 = 0, \boldsymbol{S}_2 = 0\}$, where $\boldsymbol{U} = \left[\boldsymbol{u}_1{}^T, \boldsymbol{u}_2^T\right]^T \in \boldsymbol{R}^q$ is the augmented control input vector, $\boldsymbol{Y} = \left[\boldsymbol{y}_1{}^T, \boldsymbol{y}_2{}^T\right]^T \in \boldsymbol{R}^p$ the augmented system output vector, and $\boldsymbol{W} = \left[\boldsymbol{w}_1{}^T, \boldsymbol{w}_2{}^T\right]^T \in \boldsymbol{R}^m$ the vector representing interactions between the two systems.

Usually, the cost function of a multiple-system is the sum of the cost functions of all component systems:

$$J(\boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{W}) \equiv J_1(\boldsymbol{u}_1, \boldsymbol{y}_1, \boldsymbol{w}_2) + J_2(\boldsymbol{u}_2, \boldsymbol{y}_2, \boldsymbol{w}_1). \quad (1)$$

The problem of coordinating multiple systems can be stated as an optimization problem: minimize the cost function J subject to the constraint $\boldsymbol{S}_0$.

Though there are no general approaches to solving this problem for a complex multiple-system, some conceptual methods and basic principles have been suggested in [1]. One of these methods is called model coordination. Under this method, the problem is divided into two-level optimization problems. First, suppose the interaction $\boldsymbol{W}$ is fixed at $\boldsymbol{Z}$, then compute

$$\boldsymbol{H}(\boldsymbol{Z}) = \min_{(\boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{Z}) \in \boldsymbol{S}_0} J(\boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{Z}).$$

where $\boldsymbol{H}(\boldsymbol{Z})$ is then minimized over all allowable values of $\mathbf{Z}$. This two-level optimization problem is solved iteratively until the desired performance is achieved. Another method is called *goal coordination*, in which the system is represented as in Fig. 2. Suppose $\boldsymbol{w}_i$ is not necessarily equal to $\boldsymbol{x}_i$. The overall optimality is achieved by sequentially optimizing two subsystems, while treating $\boldsymbol{w}_i$ as an ordinary input variable of each corresponding subsystem. This requires $\boldsymbol{x}_i$ and $\boldsymbol{w}_i$ to be equal, which is called the *interaction balance principle.* Similar to the process of model coordination, the optimality is achieved iteratively. Another basic principle of coordination, called the *interaction prediction principle*, is stated as follows. Let

$$\hat{\boldsymbol{W}} = \left[\hat{\boldsymbol{w}}_1^T, \hat{\boldsymbol{w}}_2^T\right]^T$$

be the predicted interaction and $\boldsymbol{W} = \left[\boldsymbol{w}_1{}^T, \boldsymbol{w}_2{}^T\right]^T$ be the actual interaction under the control $\boldsymbol{U}$. Then the overall optimum will be achieved if the prediction gives the true value, that is: $\hat{\boldsymbol{W}} = \boldsymbol{W}$.

Obviously, solving these optimization problems largely depends on the knowledge of the structure and/or dynamic parameters of low-level subsystems and mathematical synthesis. Moreover, in a hierarchical system it is desirable that adding a high-level coordinator should not affect the internal structure and/or parameters of the low-level subsystems, and should only give appropriate coordination commands to them, so that each level can be designed independently of other levels. That is, the higher the level is, the more intelligence it has, and the less precise its knowledge about the low levels becomes. These requirements motivated us to design a knowledge-based coordinator (KBC).

To design a coordinator, we first need to define a system performance index. It should be chosen to express the desired system performance and should also be amenable to some optimization methods. For example, the performance index defined in (1) is suitable for the concepts of model coordination and goal coordination. To design a KBC, one needs an index to explicitly express system performance, and such an index will henceforth be called the *principal output.* The overall system performance index may not necessarily be the simple summation of the performance indices of all component systems. Because only the system constraints are important for coordination, one may not even be able to define subsystem performance indices. Moreover, we want to relate the principal output directly to the coordination commands. The coordination commands are defined as the reference inputs to subsystems. From the following sections, one can see that both the explicit expression for system performance and the direct relationship between the principal output and the coordination commands will simplify the design of the knowledge base and the goal-oriented search.

III. Description of the Principal Output Prediction Scheme

In a hierarchical structure, each level can be viewed as a mapping from its reference input to the output. The servo controller of each subsystem is usually designed separately from, and independently of, the others. In order not to interfere with the internal structure and/or parameters of the lower level, the only effective control variable is the reference input to the lower level. The reference inputs are a set of predesigned commands, which represent the desired overall behavior of the multiple systems. For example, when multiple robots work in a common workspace, the reference input is the desired trajectory of each robot generated without considering the presence of other robots. The purpose of a high-level coordinator is to modify the desired trajectories to avoid collision among the robots. From a high-level coordinator's points of view, the following conditions are assumed.

C1: Each subsystem is a stable, closed-loop control system.
C2: Each subsystem has a linear response to its reference input.
C3: Each subsystem will remain stable even during its interaction with other subsystems.
C4: System performance can be described explicitly by the principal output.

In Fig. 1, let $\boldsymbol{Y}$ be the principal output vector of the multiple-system, $\boldsymbol{Y}_r = [\boldsymbol{Y}_{r1}^T, \boldsymbol{Y}_{r2}^T]^T$ be the vector of reference input to the low level. Note that the components of $\boldsymbol{Y}$ may not be simply the outputs of subsystems, but could be a function of these outputs:

$$\boldsymbol{Y} = \boldsymbol{F}_0(\boldsymbol{y}_1, \boldsymbol{y}_2), \text{ where } \boldsymbol{F}_0 : \boldsymbol{R}^{p_1} \times \boldsymbol{R}^{p_2} \rightarrow \boldsymbol{R}^p.$$

Because each subsystem is a closed-loop control system, $\boldsymbol{y}_i$ can be represented as

$$\boldsymbol{y}_i = \boldsymbol{f}_i(\boldsymbol{y}_{ri}, \boldsymbol{w}_j), \text{where } i, j = 1, 2, j \neq i, \text{ and } \boldsymbol{f}_i : \boldsymbol{R}^{n_i} \times \boldsymbol{R}^{m_j} \rightarrow \boldsymbol{R}^{p_i}.$$

Then $\boldsymbol{Y}$ can be represented as

$$\boldsymbol{Y} = \boldsymbol{F}(\boldsymbol{y}_{r1}, \boldsymbol{y}_{r2}, \boldsymbol{w}_1, \boldsymbol{w}_2) \tag{2}$$

where $\boldsymbol{F} : \boldsymbol{R}^{n_1} \times \boldsymbol{R}^{n_2} \times \boldsymbol{R}^{m_1} \times \boldsymbol{R}^{m_2} \rightarrow \boldsymbol{R}^p$. The principal-output vector $\boldsymbol{Y}$ in (2) establishes an explicit relationship between the overall system performance and the reference input. Let $\hat{\boldsymbol{Y}}(k + d/k)$ and $\boldsymbol{Y}_d(k + d)$ be the $d - step$ ahead prediction and the desired value of the principal output $\boldsymbol{Y}(k)$ at time $k + d$, respectively. Then, the performance index of the overall system can be defined as

$$J(k) = \left[\boldsymbol{Y}_d(k + d) - \hat{\boldsymbol{Y}}(k + d/k)\right]^T \cdot \left[\boldsymbol{Y}_d(k + d) - \hat{\boldsymbol{Y}}(k + d/k)\right].$$

The purpose of using a coordinator is to choose a suitable reference input vector $\boldsymbol{Y}_r(k)$ so as to minimize $J(k)$ at time k subject to a set of constraints.

Suppose the prediction of the principal output corresponding to each choice of $\boldsymbol{Y}_r(k)$ is available, and the constraints can be expressed with a set of production rules. Then, in each sampling interval, the desired performance can be obtained by iteratively trying different reference inputs and adjusting them according to the principal output prediction. For example, we propose the following algorithm to coordinate two subsystems, where the superscript i denotes the iteration count.

1) Compute the principal output prediction $\hat{\boldsymbol{Y}}^0(k + d/k)$ for given reference inputs $\boldsymbol{y}_{r1}^0(k)$ and $\boldsymbol{y}_{r2}^0(k)$.
2) Using $\hat{\boldsymbol{Y}}^i(k + d/k)$, modify the reference inputs of subsystem 1, $\boldsymbol{y}_{r1}^i(k)$, $i = 0, 1, 2, \ldots$.
3) Compute $\hat{\boldsymbol{Y}}^{i+1} s(k + d/k)$ for given reference inputs $\boldsymbol{y}_{r1}^i(k)$ and $\boldsymbol{y}^0{}_r 2(k)$.
4) Set $i \leftarrow i + 1$ and repeat steps 2) and 3) until $\hat{\boldsymbol{Y}}^{i+1}(k + d/k)$ cannot be improved any further with $\boldsymbol{y}_{r1}^i(k)$ due to the constraints.
5) Set $i \leftarrow 0$.
6) Using $\hat{\boldsymbol{Y}}^i(k + d/k)$. modify the reference inputs of subsystem 2, $\boldsymbol{y}_{r2}^i(k)$, $i = 0, 1, 2, \ldots$.
7) Compute $\hat{\boldsymbol{Y}}^{i+1}(k + d/k)$ for given reference inputs $\boldsymbol{y}_{r1}^0(k)$ and $\boldsymbol{y}_{r2}^i(k)$.
8) Set $i \leftarrow i + 1$ and repeat steps (6) and (7) until $\hat{\boldsymbol{Y}}^{i+1}(k + d/k)$ cannot be improved any further with $\boldsymbol{y}_{r2}^i(k)$ due to the constraints.
9) Set $i \leftarrow 0$ and repeat steps 2)–8) until $\hat{\boldsymbol{Y}}^i(k + d/k)$ reaches its desired value.

The conceptual structure of this scheme is given in Fig. 3. Obviously, this scheme needs a multiple-step predictor to compute $\hat{\boldsymbol{Y}}^i(k + d/k)$, and a KBC for the modification process of the reference inputs. By using this principal output predictor to characterize system performance, the knowledge to coordinate multiple systems becomes clear, thus simplifying the design of a knowledge base. We now need to address the following two problems: 1) Given the principal output prediction, how can we design this KBC? This will be discussed in the next section. 2) How can we design such a principal output predictor? In Section V, an MIMO

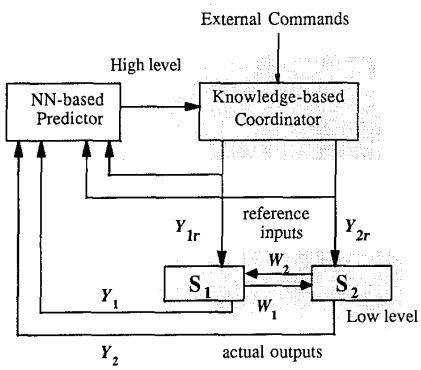


Fig. 3. Conceptual structure of the knowledge-based coordination system.

predictor is designed using NN's. With the ability of learning an input-output (I/O) mapping from experience, an NN can be used to track the variation of the mapping. However, an NN alone cannot form an intelligent coordination/control system. As a general method of representing systems with learning ability, NN's lack the ability of logical reasoning and decision making, interpretation of environmental changes, and quick response to unexpected situations. Therefore, a KBC is needed. Despite its drawbacks, the NN-based predictor establishes an explicit relationship between the principal output and the reference inputs to subsystems. Hence, the knowledge base is simplified. One can also add easily to the knowledge base such rules as the constraints of subsystems, operation monitoring, system protection, and switching of the coordination schemes. The KBC will emphasize system coordination but not data interpretation, while the ability of learning will rely mainly on the NN, that is, the NN will adapt itself to the model/parameter uncertainties, disturbances, component failures, and so on.

IV. Design of the Knowledge-Based Coordinator

A multiple-system with the KBC forms a hierarchical structure, and the low-level subsystems are viewed as a mapping from their reference input to the principal output. The goal is to modify the reference input so that the principal output reaches its desired value. For a given multiple-system we must define the principal output. Note that knowledge-based coordination is not strictly a mathematical optimization problem. The principal output must 1) have an explicit relation to the reference inputs, and 2) be measurable or computable from measured data. Because a multiple-system is designed to perform a common task(s) among the component systems, such a principal output is usually defined to express the situation of the common task(s), though it may not explicitly reflect some of generally used optimization criteria, such as energy or time.

As an example, consider the coordinated control of two robots. The two robots' operations may be tightly coupled or loosely coupled. They are tightly coupled, for example, when they hold a single object rigidly and are coordinated to move the object. On the other hand, they are loosely coupled, when they work in a common workspace and are coordinated to avoid collision. Suppose each robot is equipped with a servo controller that was originally designed for a single robot. The two robots are coordinated by modifying each robot's reference input. For the tightly coupled case, the principal output can be defined as the object's position error or the internal/external force exerted on the object. For the loosely coupled case, on the other hand, the positions and/or velocities of the robots' end-effectors can be used to represent the status of collision avoidance, and thus, they are qualified to be the principal output. For both cases, an explicit relationship between the system performance and the reference input is established by defining the appropriate principal output.

As stated in the previous sections, we want to use the principal output predictor to see where each reference input of the subsystem will lead to. In this section, it is assumed that such a predictor is well-designed and gives the true value of the principal output. (The design of such a predictor is treated in the next section.)

Given the principal output prediction, the simplified knowledge on how to coordinate a multiple-system can be stated in two steps:

1) Modify the reference input and feed the modified input to the predictor.
2) **IF** the principal output prediction yield good performance **THEN** feed the reference input to individual subsystems **ELSE** remodify it.

Since only one reference input is modified at each time, the remaining problems are then in which direction the reference input is modified (increase or decrease), how much it should be modified, and what its limits are. For a single-system, we have already developed such a KBC in [9]. For a multiple-system, the modification process of each reference input is similar to that of a single-system, so only the related results of [9] are summarized below.

A. Knowledge Representation

Using a predictor, the performance of a multiple-system is characterized by the predicted tracking error in its principal output that results from the application of the current reference input. Thus, the space of predicted tracking errors $\boldsymbol{E}$ forms the input space of the KBC's knowledge base. The goal of the KBC is then to implement the modification process discussed thus far. It is not difficult to express this process in terms of a set of production rules. The possible actions that the KBC can take include: increase the reference inputs, decrease the reference inputs, or keep them unchanged. Because our scheme is based on the modification of the reference input according to the corresponding principal output prediction, the internal structure and parameters of low-level subsystems are not affected. To simplify the design of production rules, the predicted output error is considered as the system characteristics or the input of the knowledge base. For each element of the reference input, the basic modification process can be represented by a decision tree as shown in Fig. 4. The ijth node in the tree is represented by $([a_j^i, b_j^i], c_j^i)$, where c_j^i is the

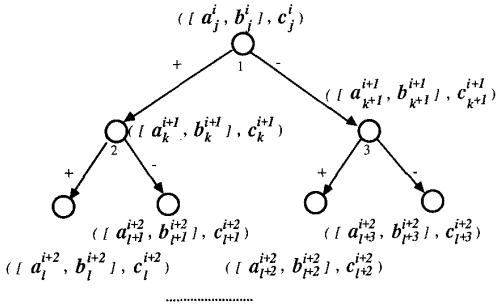


Fig. 4. Decision tree.

quantity added to the reference input,[1]

$$y_r^{i+1}(k) = y_r^o(k) + c_j^i,$$

where $y_r^o(k)$ is an element of the original reference input vector to one of the subsystems at time k, y_r^{i+1} is its modified value after the ith iteration, and $[a_j^i, b_j^i]$ is the interval to be searched, where $a_j^i < c_j^i < b_j^i$ for all i, j. By giving the reference input $y_r^i(k)$, at any node $([a_j^i, b_j^i], c_j^i)$, the interval $[a_j^i, b_j^i]$ will be split into two subintervals $[a_k^{i+1}, b_k^{i+1}] \equiv [a_j^i, c_j^i]$ and $[a_{k+1}^{i+1}, b_{k+1}^{i+1}] \equiv [c_j^i, b_j^i]$, which form two successive nodes. At the ith iteration and at the ijth node, let the predicted tracking error resulting from $y_r^i(k)$ be denoted as

$$e_j^i(k) = \hat{y}^i(k + d/k) - Y_d(k + d)$$

where y_d is an element of $\boldsymbol{Y}_d$ and $\hat{y}^i(k + d/k)$ is the corresponding element of $\hat{\boldsymbol{Y}}^i(k + d/k)$.

Then, c_j^i is computed as:

$$c_j^i = \begin{cases} b_j^i - (b_j^i - a_j^i)K, & \text{if } e_j^i(k) < 0 \\ a_j^i + (b_j^i - a_j^i)K, & \text{if } e_j^i(k) > 0 \end{cases}$$

and $0 < K < 1$ is a weighting coefficient that determines the step size of the iterative operation. a_0^0 and b_0^0 are the predesigned lower and upper bounds for the amount of reference input modification, and usually $c_0^0 = 0$, that is, at the beginning, the reference input is not modified.

The structure of this decision tree shows that the simplest inference process is similar to forward chaining, starting from the root node. However, after a period of operation, we may learn that a positive augment c_j^i is always needed. Then, the inference process may start at any node with $c_j^i > 0$ and go forward or backward according to the sign of predicted tracking error. Note that this backward search does not mean a reverse search, but rather intends to find a suitable node to start the forward search. As soon as the forward search begins, the process is not reversible.

[1] Because only the reference input to one subsystem is modified at a time, to simplify the notation, subsystem 1 and 2 will not be distinguished within this section, that is, $y_r(k)$ will represent one element of either $\boldsymbol{y}_{r1}(k)$ or $\boldsymbol{y}_{r2}(k)$.

B. Solution Existence and Stability Analysis

The basic forms of production rules are as follows.

IF $((e_j^i(k) < 0)$ **AND** $(|e_j^i(k)| > \epsilon)$
THEN increase c_j^i **AND** compute $y_r^{i+1}(k) = y_r^0(k) + c_j^i$;
IF $(e_j^i(k) > 0)$ **AND** $(|e_j^i(k)| > \epsilon)$
THEN decrease c_j^i **AND** compute $y_r^{i+1}(k) = y_r^0(k) + c_j^i$;
IF $|e_j^i(k)| \leqslant \epsilon$
THEN set $y_r^{i+1}(k) = y_r^i(k)$ **AND** stop the iterative operation.

$\epsilon > 0$ is a prespecified error tolerance. Because the amount of modification to the reference input is bounded, or $a_0^0 < c_j^i < b_0^0$ for all i, j, there may be a case where $|e_j^i(k)| > \epsilon$ for all c_j^i. To avoid such a case, the desired trajectory needs to be designed carefully. For example, when the desired trajectory is a step function, for a second order system, at $k = 0$ the system response cannot have a jump no matter how large the reference input is. A reasonable choice of ϵ is another way to prevent this problem. This existence problem can be monitored by adding, for example, the following rule into the knowledge base:

IF $((|c_j^i - b_0^0| < \delta)$ **OR** $(|c_j^i - a_0^0| < \delta))$ **AND** $(|e_j^i(k)| > \epsilon$
THEN (change a_0^0 or b_0^0 automatically and continue the search)
OR (ask the operator for an adjustment)
OR (stop the iterative operation and choose c_j^i with the smallest $e_j^i(k)$ as the best output).

$\delta > 0$ is a prespecified tolerance.

Suppose the weighting factor K is set too small or too large, then the search for c_j^i may take a long time. This would not be acceptable if the required computation cannot be completed within one sampling interval. The case of the computation/search time exceeding one sampling interval is equivalent to having no solution. This case is monitored by

IF (the search time $> T_{\max}$) **AND**
$(|e_j^i(k)| > \epsilon)$ **THEN** (stop the iterative operation)
AND (choose c_j^i with the smallest $e_j^i(k)$ as the best output) **AND** (modify the weighting coefficient K).

$T_{\max}$ is the prespecified maximum search time.

Suppose the prediction gives the true principal output, and let us consider the KBC and the closed-loop subsystem. The KBC can then be viewed as a map $\boldsymbol{M}_0 : \boldsymbol{E} \to \boldsymbol{Y}_R$, specified by all the production rules, where $\boldsymbol{E}$ is the space of predicted principal output tracking error and $\boldsymbol{Y}_R$ the reference input space. The low-level closed-loop subsystem is also a map, $\boldsymbol{L}$:$\boldsymbol{Y}_R \to \boldsymbol{E}$, which is specified by the desired dynamic properties of the servo controller. Because $\boldsymbol{L}$ represents a well-designed controller and there exists a reference input at time k, $\boldsymbol{Y}_r^i(k) \in \boldsymbol{Y}_R$ such that $e_j^i(k) = 0$. Thus, it is reasonable to assume that $\boldsymbol{L}$ is a linear map. The properties of the map $\boldsymbol{M} \equiv \boldsymbol{L}\boldsymbol{M}_0 : \boldsymbol{E} \to \boldsymbol{E}$ depends mainly on the properties of the map $\boldsymbol{M}_0$. In fact, all the antecedents of production rules are established based on the prediction of principal output. If the predictor gives the true principal output, then the properties of the invariant map $\boldsymbol{M}$:$\boldsymbol{E} \to \boldsymbol{E}$ is determined solely by the knowledge base. For system stability, all production rules in the knowledge base must form a contraction map. More formally, we give the following theorem without proof. (See [9] for its proof.)

Theorem: Suppose 1) the principal output prediction of a multiple-system is computable and the predictor gives the true principal output, and 2) $\boldsymbol{L}$:$\boldsymbol{Y}_R \rightarrow \boldsymbol{E}$ of the low-level closed-loop subsystem is a linear map. If the map $\boldsymbol{M}_0$:$\boldsymbol{E} \rightarrow \boldsymbol{Y}_R$ is given by a decision tree, then the composite map $\boldsymbol{M} \equiv \boldsymbol{L}\boldsymbol{M}_0$:$\boldsymbol{E} \rightarrow \boldsymbol{E}$ is a contraction map.

At each node of the decision tree, the iterative learning process is performed and the rules always keep the search direction pointed to the node where the tracking error decreases. This implies that the iterative learning process decreases the tracking error. As mentioned in Section III, the inference process is not reversible, and thus, it is impossible to have an unstable system response.

V. Design of an NN-Based Predictor

Though it is assumed that the principal output $\boldsymbol{Y}$ is measurable or computable from the measured data, it may be very difficult to derive a closed-form expression for (2). Therefore, it is almost impractical to design such a principal output predictor with mathematical synthesis alone, even if such a closed form exists. The development of NN's suggested that an I/O mapping can be approximated by a multilayer perceptron [10], [11]. With the ability of learning from examples, an NN can be trained to retain the dynamical property of an I/O mapping. Typically, a set of I/O pairs is arranged as $(u_1, y_1), (u_2, y_2), \ldots$, where $y_i = f(u_i)$ is a mapping. Using these training data, the connection weights within the NN are reorganized so as to represent the mapping relation. One of the most popular NN structures is the multilayer perceptron with the back propagation (BP) algorithm [12], [13]. The computation of BP includes two steps: 1) compute the NN's output forward from its INPUT to OUTPUT layers, and 2) modify the connection weights backward from its OUTPUT to INPUT layers. In what follows, an MIMO predictor is designed using an NN. The BP algorithm is extended to a vector form, and the problem of tracking a time-varying system is also addressed.

Referring to (2), the *d-step* ahead prediction of $\boldsymbol{Y}$ can be represented by

$$\hat{\boldsymbol{Y}}(k+d/k) = \boldsymbol{F}_p(\bar{\boldsymbol{Y}}_{r1}, \bar{\boldsymbol{Y}}_{r2}, \bar{\boldsymbol{Y}}) \tag{3}$$

where

$$\bar{\boldsymbol{Y}}_{r1} = \Big(\boldsymbol{Y}_{r1}(k+i_1), \ldots, \boldsymbol{Y}_{r1}(k), \boldsymbol{Y}_{r1}(k-1), \ldots, \boldsymbol{Y}_{r1}(k-i_2)\Big)$$

$$\bar{\boldsymbol{Y}}_{r2} = \Big(\boldsymbol{Y}_{r2}(k+j_1), \ldots, \boldsymbol{Y}_{r2}(k), \boldsymbol{Y}_{r2}(k-1), \ldots, \boldsymbol{Y}_{r2}(k-j_2)\Big)$$

$$\bar{\boldsymbol{Y}} = \Big(\boldsymbol{Y}(k), \boldsymbol{Y}(k-1), \ldots, \boldsymbol{Y}(k-i)\Big)$$

$$\boldsymbol{F}_p : \boldsymbol{R}^{n_1} \times \boldsymbol{R}^{n_2} \times \boldsymbol{R}^{p} \rightarrow \boldsymbol{R}^{p}$$

where i, i_1, i_2, j_1 and j_2 are constant integers. The interaction effects among subsystems are implicitly included in the historical data of $\bar{\boldsymbol{Y}}$. In (3), the principal output prediction is directly represented as a mapping of the reference inputs and the historical data of $\bar{\boldsymbol{Y}}$. A three-layer (with one hidden layer) perceptron is designed to learn the relationship of (3), and forms the backbone of the NN-based predictor. Fig. 5 shows the structure of the predictor, where the reference inputs $\bar{\boldsymbol{Y}}_{r1}$ and $\bar{\boldsymbol{Y}}_{r2}$, and the historical data $\bar{\boldsymbol{Y}}$ are fed to the nodes at the INPUT layer. When the NN becomes well-trained, the predictions $\hat{\boldsymbol{Y}}(k+d/k)$ for $d = 1, 2, \ldots$ are then produced from the OUTPUT nodes. Two problems should be solved before implementing this NN-based predictor:

1) How to efficiently represent and compute an MIMO mapping with the NN ?
2) How to track a time-varying mapping ?

These problems have been treated in [14], and thus, only the key points are summarized below for completeness.

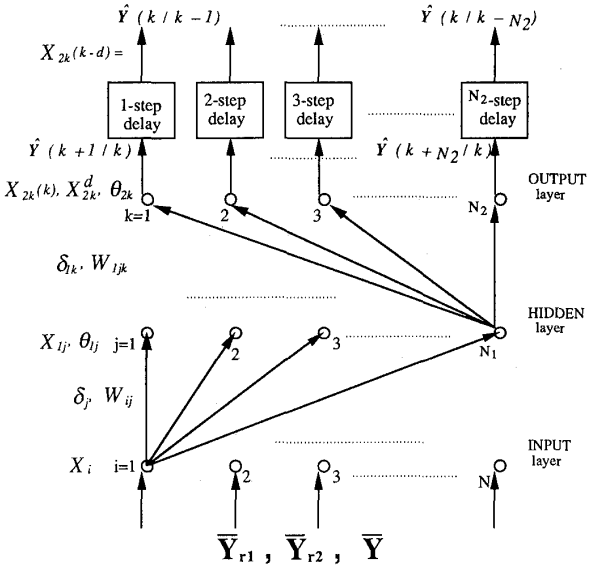


Fig. 5. Structure of the NN-based predictor.

A. Multidimensional Back Propagation Algorithm

Traditionally, each node of a multilayer perceptron is designed only to perform scalar operations, and the number of nodes or layers is increased to represent a more complex I/O mapping. However, if each node handles only one element of a vector, then a multilayer mesh is required to learn a complex MIMO mapping, such as (3). Training such a mesh is impractical due to the lack of a systematic algorithm. Therefore, we design a multilayer perceptron with the ability of vector operations in order to easily specify some known coupling relations within the predicted system, and to get an easier (thus more intuitive) form of the training algorithm. Referring to Fig. 5, the corresponding multi-dimensional BP algorithm is summarized below.

All inputs and outputs of this NN are vectors, $\boldsymbol{X}_i \in \boldsymbol{R}^{n\times 1}, \boldsymbol{X}_{1j} \in \boldsymbol{R}^{m\times 1}$, and $\boldsymbol{X}_{2k} \in \boldsymbol{R}^{p\times 1}$ are the output

of INPUT, HIDDEN and OUTPUT layers, respectively, for $1 \le i \le N, 1 \le j \le N_1$, and $1 \le k \le N_2$.

1) Compute the Output of the HIDDEN Layer $\boldsymbol{X}_{1j}$:

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

where $\boldsymbol{O}_{1j} = \sum_{i=1}^{N} \boldsymbol{W}_{ij}\boldsymbol{X}_i$, $j = 1, 2, \ldots, N_1$, and $\boldsymbol{W}_{ij} \in \boldsymbol{R}^{m\times n}$ is the weights from the INPUT to the HIDDEN layer, $\boldsymbol{\theta}_{1j} = [\theta_{1j1}, \ldots, \theta_{1jm}]^T$ the threshold at the HIDDEN layer.

2) Compute the Output of the OUTPUT Layer

$$\begin{aligned}\boldsymbol{X}_{2k} &\equiv [x_{2k1}, \ldots, x_{2kp}]^T = \boldsymbol{f}_k(\boldsymbol{O}_{2k}) \\ &= \left[\frac{1}{1+\exp\left(-(o_{2k1}+\theta_{2k1})\right)}, \right. \\ &\qquad \left. \ldots, \frac{1}{1+\exp\left(-(o_{2kp}+\theta_{2kp})\right)}\right]^T\end{aligned}$$

where $\boldsymbol{O}_{2k} = \sum_{j=1}^{N_1} \boldsymbol{W}_{1jk}\boldsymbol{X}_{1j}, k = 1, 2, \ldots, N_2$, and $\boldsymbol{W}_{1jk} \in \boldsymbol{R}^{p\times m}$ is the weights from the HIDDEN to the OUTPUT layer, $\boldsymbol{\theta}_{2k} = [\theta_{2k1}, \ldots, \theta_{2kp}]^T$ the threshold at the OUTPUT layer.

3) Update the Weights from the HIDDEN to the OUTPUT Layer $\boldsymbol{W}_{1jk}$:

$$\boldsymbol{W}_{1jk}(k+1) = \boldsymbol{W}_{1jk}(k) + \Delta\boldsymbol{W}_{1jk}$$

where

$$\begin{aligned}\Delta\boldsymbol{W}_{1jk} &= \eta_1[\delta_{1k}\boldsymbol{T}]^T \\ \delta_{1k} &= \left[\boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k}\right]^T diag\,[x_{2k1}(1-x_{2k1}), \\ &\qquad x_{2k2}(1-x_{2k2}), \cdots x_{2kp}(1-x_{2kp})]\end{aligned}$$

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

$$\boldsymbol{T}_{1l} = \begin{bmatrix} 0 \\ \cdots \\ (\boldsymbol{X}_{1j})^T \\ 0 \\ \cdots \end{bmatrix}^T \quad \text{—at the } l\text{th row, } l=1,2,\cdots,p.$$

4) Update the Weights from the INPUT to the Hidden Layer $\boldsymbol{W}_{ij}$:

$$\boldsymbol{W}_{ij}(k+1) = \boldsymbol{W}_{ij}(k) + \Delta\boldsymbol{W}_{ij}$$

where

$$\begin{aligned}\boldsymbol{W}_{ij} &= \eta[\delta_j\boldsymbol{T}_2]^T \\ \delta_j &= \left(\sum_{k=1}^{N_2}\delta_{1k}\boldsymbol{W}_{1jk}\right) diag\,[x_{1j1}(1-x_{1j1}), \\ &\qquad x_{1j2}(1-x_{1j2}), \cdots, x_{1jm}(1-x_{1jm})]\end{aligned}$$

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

$$\boldsymbol{T}_{2l} = \begin{bmatrix} 0 \\ \cdots \\ (\boldsymbol{X}_i)^T \\ 0 \\ \cdots \end{bmatrix}^T \quad \text{—at the } l\text{th row , } l=1,2,\cdots,m.$$

5) Update the Thresholds at the OUTPUT and the HIDDEN Layers $\boldsymbol{\theta}_{2k}$, $\boldsymbol{\theta}_{1j}$:

$$\begin{aligned}\boldsymbol{\theta}_{2k}(k+1) &= \boldsymbol{\theta}_{2k}(k)\eta_{1\theta}[\delta_{1k}]^T \\ \boldsymbol{\theta}_{1j}(k+1) &= \boldsymbol{\theta}_{1j}(k) + \eta_\theta[\delta_{1k}]^T\end{aligned}$$

where $\eta_1, \eta, \eta_{1\theta}$, and $\eta_\theta > 0$ are the gain factors.

B. Tracking a Time-Varying System

In the above algorithm, the NN is trained by using the prediction error

$$\boldsymbol{E}_k(k) = \boldsymbol{X}_{2k}^d(k) - \boldsymbol{X}_{2k}(k) = \boldsymbol{Y}(k+d) - \hat{\boldsymbol{Y}}(k+d/k), \quad (4)$$

where $\boldsymbol{X}_{2k}^d(k)$ is the desired value of $\boldsymbol{X}_{2k}^d(k)$ at time k. This implies that the NN should be trained by using the system's future output $\boldsymbol{Y}(k+d)$, which are unknown. A set of training data can be acquired beforehand, and used to train the NN. After the NN is "well trained," its weights will no longer be modified. Then, the NN produces the correct outputs, whenever the inputs are present at the INPUT layer. However, for time-varying mappings, it is meaningless to say that an NN is "well trained". Moreover, in a real-time control system, it is desirable to always operate the system in closed-loop. This means that the NN-based predictor should be "updated" (rather than trained) so as to track a time-varying system. To update the NN-based predictor on-line, the basic idea is to modify the weights of the NN using the *a posteriori* prediction error:

$$\boldsymbol{E}_k(k-d) = \boldsymbol{X}_{2k}^d(k-d) - \boldsymbol{X}_{2k}(k-d) = \boldsymbol{Y}(k) - \hat{\boldsymbol{Y}}(k/t-d). \quad (5)$$

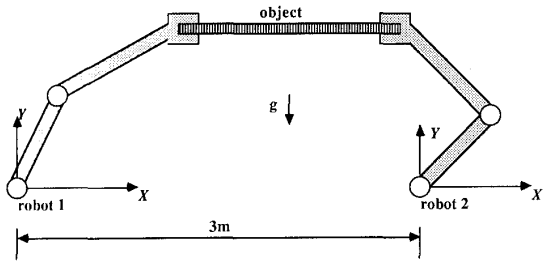
It is shown in [14] that one can use (5) instead of (4) in the algorithm, and keep all other formulas unchanged. The scaling problem and error analysis have also been addressed in [14], and concluded that the accuracy of the NN-based predictor depends only on the accuracy of the NN's approximation of the actual mapping.

VI. Coordinated Control of Two 2-Link Robots

To demonstrate how to apply the proposed scheme for solving real life problems, we consider the problem of coordinating two 2-link robots holding a rigid object. The low-level subsystems include two robots each with a separately designed servo controllers. The basic configuration of this example is given in Fig. 6. The Cartesian frame is fixed at the base of robot 1, and the trajectories of the object and the robots' end-effectors are specified relative to this frame. The task is to move the object forward and then backward in X direction while keeping the height in Y direction constant. The desired trajectory of the object is selected by a high-level planner as the reference input to the low level. If the two robots hold the object firmly, then the dynamics of the system are modeled as follows.

Dynamics of the Object: Let $\boldsymbol{f}_i = (f_{ix}, f_{iy})^T$ be the force exerted by the end-effector of robot i on the object in Cartesian space. Then the motion of the object is described by

$$m\ddot{\boldsymbol{P}} + m\boldsymbol{g} = \boldsymbol{f}, \quad \boldsymbol{f} = \boldsymbol{W}\boldsymbol{F} \equiv [\boldsymbol{I}_2, \boldsymbol{I}_2]\begin{bmatrix}\boldsymbol{f}_1 \\ \boldsymbol{f}_2\end{bmatrix} \quad (6)$$



Robots	Link 1	Link 2
Length	1 m	1 m
Mass Center	0.5 m	0.5 m
Mass	20 kg	10 kg
Moment of inertia	0.8 kg ms^2	0.2 kg ms^2
Object: mass=5 kg; length=1 m.		

Fig. 6. Two 2–link robots holding an object.

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 (6), one can see that, to achieve the object's specified acceleration, the combination of forces shared by the two robots is not unique.

Dynamics of Each Robot with Servo Controller: Suppose two robots have an identical mechanical configuration, then the force-constrained dynamic equation of robot i in joint space is given by [15]:

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

where $\boldsymbol{q}_i$ is the vector of the robot's joint positions, $\boldsymbol{H}$ is the inertia matrix, $\boldsymbol{h}$ is the centrifugal, Coriolis, and gravitational forces, $\boldsymbol{J}_i$ is the Jacobian matrix, and $\boldsymbol{\tau}_i$ is the vector of joint torques. Suppose both robots are position-controlled with the computed torque algorithm. That is, the control input to robot i is

$$\boldsymbol{\tau}_i = \hat{\boldsymbol{H}}(\ddot{\boldsymbol{q}}_{id} - \boldsymbol{K}_{Di}(\dot{\boldsymbol{q}}_i - \dot{\boldsymbol{q}}_{id}) - \boldsymbol{K}_{pi}(\boldsymbol{q}_i - \boldsymbol{q}_{id})) + \hat{\boldsymbol{h}} \quad (7)$$

where $\hat{\boldsymbol{H}}$ and $\hat{\boldsymbol{h}}$ are the estimated values of $\boldsymbol{H}$ and $\boldsymbol{h}$, $\boldsymbol{q}_{id}$ is the desired value of $\boldsymbol{q}_i$, $\boldsymbol{K}_{Di}$ and $\boldsymbol{K}_{pi}$ are the controllers' gains. The reference input to the system is the desired trajectory of the object specified by $\boldsymbol{P}_d, \dot{\boldsymbol{P}}_d$ and $\ddot{\boldsymbol{P}}_d$, which will be transformed into the desired trajectories of the end-effector and the joints of each robot.

Problem Statement: Suppose the object is a rigid body and there is no relative motion between the end-effectors and the object. For (6), let $\boldsymbol{f}_d$ and $\boldsymbol{F}_d$ be the desired values of $\boldsymbol{f}$ and $\boldsymbol{F}$, respectively. Then, we have

$$\boldsymbol{F}_d = \boldsymbol{F}_{Md} + \boldsymbol{F}_{Id} \equiv \boldsymbol{W}^* \boldsymbol{f}_d + (\boldsymbol{I}_4 - \boldsymbol{W}^* \boldsymbol{W})\boldsymbol{y}_0 \quad (8)$$

where $\boldsymbol{W}^* \in \boldsymbol{R}^{4\times 2}$ is the pseudo-inverse of $\boldsymbol{W}$, $\boldsymbol{I}_4$ is a 4×4 unit matrix, and $\boldsymbol{y}_0 \in \boldsymbol{R}^{4\times 1}$ an arbitrary vector in the null space of $\boldsymbol{W}$. Therefore, the forces exerted by the end-effectors consist of two parts:

$$\boldsymbol{F}_{Md} \equiv \begin{bmatrix} \boldsymbol{F}_{M1d} \\ \boldsymbol{F}_{M2d} \end{bmatrix} \in \boldsymbol{R}^{4\times 1}$$

is the force to move the object and

$$\boldsymbol{F}_{Id} \equiv \begin{bmatrix} \boldsymbol{F}_{I1d} \\ \boldsymbol{F}_{I2d} \end{bmatrix} \in \boldsymbol{R}^{4\times 1}$$

is the internal force. The following two problems arise: 1) sharing the moving force $\boldsymbol{F}_{Md}$ by two robots, 2) changing the internal force so as to satisfy a set of constraints, such as joint torque limits or energy capacity.

In (8), $\boldsymbol{F}_{Md}$ can be specified by the desired trajectory. $\boldsymbol{F}_{Id}$ is given as the desired internal force, for example, $\boldsymbol{F}_{Id} = 0$ for the least energy consumption. Because $\boldsymbol{W}^*$ is a constant matrix and both $\boldsymbol{f}_d$ and $\boldsymbol{F}_{Id}$ are specified, the desired force $\boldsymbol{F}_d$ is determined uniquely. However, this ideal situation of load sharing may not be achieved due to the 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 reassign the load to, each robot dynamically. Our goal is to design a KBC to coordinate the two robots moving the object while minimizing the internal force.

Principal Output and Its NN-Based Predictor: The reference inputs to the low-level subsystems are the desired acceleration $\ddot{\boldsymbol{P}}_{id}$, velocity $\dot{\boldsymbol{P}}_{id}$ and position $\boldsymbol{P}_{id}$ of robot i's end-effector, $i = 1, 2$. The internal force can be used to evaluate system performance, and has an explicit relation to the reference inputs. So, the internal force is defined as the principal output. Because the force exerted by each robot to achieve a specified acceleration of the object is not unique, it is possible to adjust the internal force by modifying the reference inputs. Since the position tracking error needs to be kept small and the desired acceleration has an explicit relationship to the force exerted on the object, only the desired acceleration is modified so as to reduce the internal force. Then, the desired acceleration issued to each robot is $\ddot{\boldsymbol{P}}_{idm}$ – the modified value of $\ddot{\boldsymbol{P}}_{id}$, $i = 1, 2$. An NN-based predictor is designed to predict the force exerted on the object, which corresponds to each reference input. The predicted internal force (that is, the principal output) is then computed. The NN-based predictor has eight nodes at the INPUT layer, and the inputs are $\boldsymbol{P}_{1d}(k), \boldsymbol{P}_{1d}(k-1), \boldsymbol{P}_{2d}(k)$, $\boldsymbol{P}_{2d}(k-1)$, $\ddot{\boldsymbol{P}}_{1dm}(k)$, $\ddot{\boldsymbol{P}}_{1dm}(k-1), \ddot{\boldsymbol{P}}_{2dm}(k)$, and $\ddot{\boldsymbol{P}}_{2dm}(k-1)$. There are five HIDDEN nodes and six OUTPUT nodes with outputs $\hat{\boldsymbol{f}}_i(k + d/k)$, for $\subset = 1, 2, d = 1, 2, 3$.

Simulations Results: The two robots move the object in X direction from the initial position to the final position over one-meter distance in five seconds, and then move back to the initial position. The sampling interval is $10ms$. Force predictions are used for the modification process, and position tracking is achieved by the position controllers. The 1-step ahead predictions $\hat{\boldsymbol{f}}_i(k + 1/k), i = 1, 2$ are used in the KBC. The desired internal force is set to zero. Without the KBC, the internal force error in X direction is plotted in Fig. 7. After adding the KBC, the root-mean-square error of the internal force in X direction is reduced by 63% as shown in Fig. 8. Moreover, both the external force error and the position tracking error are kept almost the same as those without the KBC. The detailed results are summarized in Table I. Since there is no motion in Y direction, the internal force error in that direction is small enough not to require the KBC.

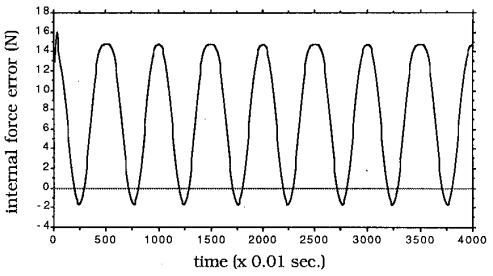


Fig. 7. Internal force error in X direction without the KBC.

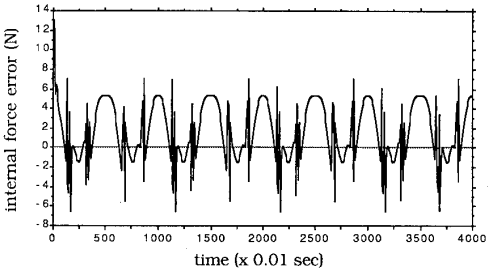


Fig. 8. Internal force error in X direction with the KBC.

TABLE I
RMS ERRORS OF INTERNAL FORCES, EXTERNAL FORCES AND OBJECT'S POSITION TRACKING

Sample Interval for Statistics		RMS Errors of Internal Forces (N)	
		without KBC	with KBC
0 – 1000	in X direction	9.58447	3.85020
	in Y direction	0.93141	0.53177
1001 – 2000	in X direction	9.57130	3.53340
	in Y direction	0.92339	0.49949
2001 – 3000	in X direction	9.57097	3.53688
	in Y direction	0.92339	0.49956

Sample Interval for Statistics		RMS Errors of External Forces (N)	
		without KBC	with KBC
0 – 1000	in X direction	0.72359	0.95822
	in Y direction	2.54853	2.54345
1001 – 2000	in X direction	0.34883	0.70199
	in Y direction	0.01436	0.03345
2001 – 3000	in X direction	0.34883	0.70346
	in Y direction	0.01436	0.03355

Sample Interval for Statistics		RMS Tracking Errors of Object's Positions (m)	
		without KBC	with KBC
0 – 1000	in X direction	0.03509	0.03529
	in Y direction	0.05733	0.05784
1001 – 2000	in X direction	0.03509	0.03530
	in Y direction	0.05759	0.05813
2001 – 3000	in X direction	0.03509	0.03530
	in Y direction	0.05759	0.05813

VII. CONCLUSION

Focusing on the problem of coordinating multiple systems, a knowledge-based coordinator is designed using the techniques of both intelligent control and neural networks. As the high-level coordinator in a hierarchical structure, its basic principle is to modify the reference inputs of low-level subsystems according to the principal output prediction so as to achieve the desired performance. By adding the proposed KBC, the internal structure and parameters of the low-level subsystems are not affected. Hence, each servo controller of the low-level subsystems can be designed separately from, and independently of, the others; no constraints need to be imposed on the design of low-level controllers. This implies that some commercially designed servo controllers for a single system can be coordinated to work for a multiple-system.

Using the principal output and its prediction, and a structure of decision tree for knowledge representation, the knowledge base necessary to coordinate multiple systems is greatly simplified while guaranteeing system stability. By using a predictor, the negative effects of system time delay is eliminated and each reference input is analyzed before putting it in operation. The unknown parameters and/or time-varying properties of a multiple-system are handled by the NN-based predictor, while leaving the logical reasoning and decision making on the coordination to the KBC.

To test this new scheme, the coordination problem for two 2-link robots holding a rigid object is simulated. By modifying the reference input of each robot, the internal force exerted on the object is reduced by 63%, indicating the scheme's potential for the effective coordination of multiple robots.

REFERENCES

[1] R. E. Larson, P. L. McEntire, and J. G. O'Reilly, Eds. "Tutorial: Distributed control (2nd ed.)," *IEEE Computer Society*, 1982.
[2] S. Lee and M. H. Kim, "Cognitive control of dynamic systems," in *Proc. IEEE 2nd Int. Symp. Intelligent Contr.*, 1987, pp. 455–460.
[3] Z. Geng and M. Jamshidi, "Expert self-learning controller for Robot manipulator," in *Proc. 1988 IEEE Int. Conf. Decision and Contr.*, pp. 1090–1095.
[4] A. J. Krijgsman, H. M. T. Broeders, H. B. Verbruggen, and P. M. Bruijn, "Knowledge-based control," in *Proc. 1988 IEEE Int. Conf. Decision and Contr.*, pp. 570–574.
[5] J. Jiang and R. Doraiswami, "Information acquisition in expert control system design using adaptive filters," in *Proc. IEEE The 2nd Int. Symp. Intelligent Contr.*, 1987, pp. 165–170.
[6] K. L. Anderson, G. L. Blankenship, and L. G. Lebow, "A rule-based adaptive PID controller," in *Proc. 1988 IEEE Int. Conf. Decision and Contr.*, pp. 564–569.
[7] B. Porter, A. H. Jones, and C. B. McKeown, "Real-time expert controller for plants with actuator non-linearities," in *Proc. IEEE The 2nd Int. Symp. Intelligent Contr.*, 1987, pp. 171–177.
[8] G. K. H. Pang, "A blackboard control architecture for real-time control," in *Proc. 1988 Amer. Contr. Conf.*, pp. 221–226.
[9] K. G. Shin and X. Cui, "Design of a knowledge-based controller for intelligent control systems," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 368–375, Mar./Apr. 1991.
[10] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.
[11] V. Vemuri (reprint edited by) "Artificial neural networks: Theoretical concepts," *Press of IEEE Computer Society*, 1988.
[12] P. J. Werbos, "Back propagation: Past and future," *Proc. 1988 Int. Conf. Neural Networks*, vol. 1, pp. I343-I353.
[13] D. E. Rumelhart and J. L. McCelland, "Learning internal representations by error propagation," *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," in *Proc. 13th IMACS World Congress on Computation and Applied Mathematics*, Dublin, Ireland, July 1991.
[15] H. Asada and J.-J. E. Slotine, *Robot Analysis and Control.* New York: Wiley-Interscience, 1986.

Xianzhong Cui graduated form North China Institute of Electric Power, Baoding, China, in 1976. He received the first M.S. degree in power plant engineering and automation from Electric Power Research Institute (EPRI), Beijing, China, in 1982, and the second M.S. degree in electrical engineering-systems from the University of Michigan, Ann Arbor, in 1989.

Since 1987, he has been working toward the Ph.D degree as Research Assistant in the Department of Electrical Engineering and Computer Science of the University of Michigan. From 1976 to 1980, he served as a Technician of instrumentation and control systems at the Tianjin Power Plant Construction Company, Tianjin, China. From 1982 to 1986, he worked for EPRI as an Electrical Engineer for power plant automation. He spent 1986–1987 as a Visiting Researcher at the University of Michigan. His research interests include neural networks and intelligent control systems, process control, real time control and computer systems, and robotics.

Kang G. Shin (S'75–M'78–SM'83) He 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 180 technical papers (more than 80 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 August 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.*