A Reinforcement Neuro-Fuzzy Combiner for Multiobjective Control

Chin-Teng Lin and I-Fang Chung

***Abstract*—This paper proposes a *neuro-fuzzy combiner* (NFC) with reinforcement learning capability for solving multiobjective control problems. The proposed NFC can combine n existing low-level controllers in a hierarchical way to form a multiobjective fuzzy controller. It is assumed that each low-level (fuzzy or nonfuzzy) controller has been well designed to serve a particular objective. The role of the NFC is to fuse the n actions decided by the n low-level controllers and determine a proper action acting on the environment (plant) at each time step. Hence, the NFC can combine low-level controllers and achieve multiple objectives (goals) at once. The NFC acts like a switch that chooses a proper action from the actions of low-level controllers according to the feedback information from the environment. In fact, the NFC is a *soft switch*; it allows more than one low-level actions to be active with different degrees through fuzzy combination at each time step. An NFC can be designed by the trial-and-error approach if enough *a priori* knowledge is available, or it can be obtained by supervised learning if precise input/output training data are available. In the more practical cases when there is no instructive teaching information available, the NFC can learn by itself using the proposed reinforcement learning scheme. Adopted with reinforcement learning capability, the NFC can learn to achieve desired multiobjectives simultaneously through the rough reinforcement feedback from the environment, which contains only critic information such as "success (good)" or "failure (bad)" for each desired objective. Computer simulations have been conducted to illustrate the performance and applicability of the proposed architecture and learning scheme.**

***Index Terms*— Critic information, mutual credit assignment, priori knowledge, reinforcement learning, soft switch.**

I. Introduction

In fuzzy logic control, the hierarchical fuzzy controllers are usually considered for solving the control problems with complex requirements, multiple goals (objectives), or multiple tasks [1]. The application examples ranging from the cart-pole balancing problem [2] to recuperative turboshaft engine control [3] and aircraft engine control [4]. A hierarchical fuzzy control scheme permits the decomposition of a complex problem into a set of simpler ones. Once these simpler problems have been solved using low-level controllers, they can be combined through a high-level fuzzy controller to solve the larger problem. The high-level fuzzy controller contains the explicit tradeoff knowledge and defines the degree of compromise among usually competing control goals.

At present, the design of a hierarchical fuzzy controller for multiobjective control relies heavily on trial-and-error and expert knowledge [1], [2]. The existing supervised-learning-based neuro-fuzzy controllers [5]–[9] are usually impractical to use for multiobjective control, since they require precise training data covering all control objectives. For some real-world applications, precise training data are usually difficult and expensive, if not impossible, to obtain. This is even true for the multiobjective control problems, where different objectives have different priorities and are correlated with each other. The reinforcement learning techniques, which only require scalar evaluation (called *reinforcement signal*) of the controller output from the environment, could solve this problem. However, most applications of control architectures based on reinforcement learning have focused on single objective [11], [12]. In general reinforcement learning environments, we need to solve both the *structural* and *temporal* credit assignment problems. In the structural credit assignment problem, we need to attribute network error to different nodes or weights. Most supervised neural learning algorithms (e.g., the backpropagation rule) [13] can be used to solve this problem. In the temporal credit assignment problem, we need to assign credit or blame to each step (control output) individually in long sequences leading up to eventual successes or failures. This problem results from the fact that a reinforcement signal may only be available at a time long after a sequence of actions has occurred; this signal may be caused by a network output (action) several time steps before or by the whole sequence of actions with varying degrees of contribution. Several reinforcement learning methods such as temporal difference [14], [15] and Q-learning methods [16] can be used to solve the temporal credit assignment problem. When consider the reinforcement learning for multiobjective control, we encounter another problem—the *mutual* credit assignment problem. In this problem, we need to assign credit or blame to each action individually among all the actions decided by the low-level controllers that try to achieve the multiple control objectives. This problem becomes severe when the multiple control objectives are complexly related and have different priorities. This is the main problem we want to attack in this paper.

For the control with multiple tasks, Jacobs [17] developed a modular gating architecture that dose task decomposition. The gating architecture consists of several expert modules and a special gating module that has an output for each expert

Manuscript received April 18, 1997; revised July 10, 1998 and April 2, 1999. This work was supported by the R.O.C. National Science Council under Grant NSC87-2213-E-009-146. This paper was recommended by Associate Editor P. Borne.

The authors are with the Department of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: ctlin@fnn.cn.nctu.edu.tw).

Publisher Item Identifier S 1083-4419(99)07945-5.

module. When presented with supervised training patterns from multiple tasks, the expert modules compete with each other to learn the training patterns, and this competition is mediated by the gating module. The gating architecture has been used to learn multiple tasks within the supervised learning paradigm. Singh [18] extended the gating architecture to a CQ (*compositional* Q)-learning architecture that fits in reinforcement learning environments. Similar architectures are also studied in [19], [20]. In these architectures, only one action is selected at each time step. Moreover, they focus on the control problems with multiple discrete and sequential tasks. Hence, they do not suit directly to the multiobjective control problems considered in this paper, where we aim at achieving multiple control objectives simultaneously. In [1], Bonissone proposed the idea of using fuzzy gating module in the gating architecture to design a hierarchical fuzzy controller. The soft gating module allows more than one actions to be active at any one time. However, no detailed design procedure or learning scheme are proposed in [1].

In this paper, we adopt the concept of gating architecture shown in Fig. 1 to solve the multiobjective control problems. We propose a *neuro-fuzzy combiner* (NFC) with reinforcement learning capability to work as the gating module. The NFC performs soft switching among the outputs (actions) of different low-level controllers. The soft switching allows more than one actions to be active with different degrees at each time step according to the environment states and feedback, which can be supervised- or reinforced-type. The proposed NFC can combine n existing low-level controllers in a hierarchical way to form a multiobjective fuzzy controller. It is assumed that each low-level (fuzzy or nonfuzzy) controller has been well designed to serve a particular objective. The role of the NFC is to fuse the n actions decided by the n low-level controllers and determine a proper action acting on the environment (plant) at each time step. Hence, the NFC can combine low-level controllers and achieve multiobjectives at once. The NFC results in smoother transitions between control actions. Moreover, it allows expert knowledge to be moved up into the gating module, so that the low-level controllers handle the dynamics, while the high-level NFC deals with quasisteady state conditions, thus making the design of low-level controllers easier. An NFC can be designed by the trial-and-error approach if enough *a priori* knowledge is available, or it can be obtained by supervised learning if precise input/output training data are available. In the more practical cases when no instructive teaching information is available, the NFC can learn by itself using the proposed reinforcement learning scheme. This learning scheme is developed based on the proposed supervised learning scheme. Adopted with reinforcement learning capability, the NFC can learn to achieve desired multiobjectives simultaneously through the rough reinforcement feedback from the environment. The reinforcement feedback contains only critic information such as "success (good)" or "failure (bad)" for each desired objective.

The outline of this paper is as follows. In Section II, we introduce the structure of a multiobjective fuzzy control system along with the proposed neural fuzzy combiner (NFC). A hierarchical design method for the NFC is presented in Section III.

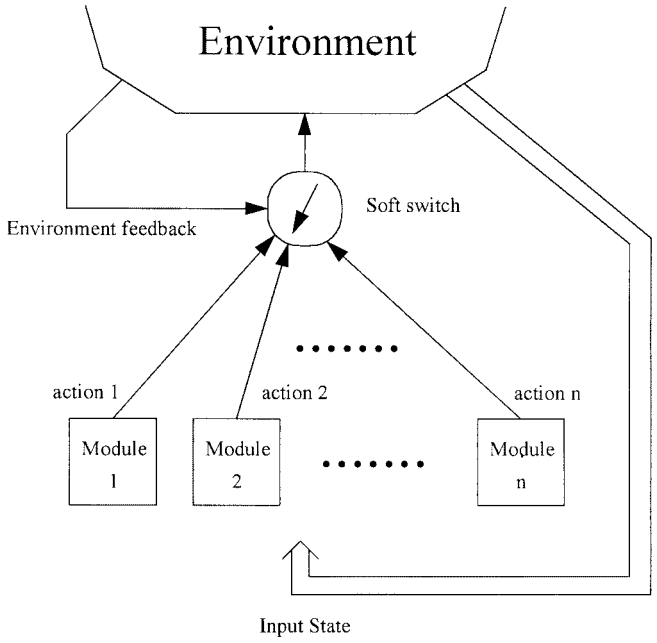


Fig. 1. Gating architecture.

A supervised learning scheme and a reinforcement learning scheme for the NFC are then developed in Sections IV and V, respectively. Convergence analysis of reinforcement learning is proposed in Section VI. In Section VII, we apply the proposed architecture to the cart-pole balancing problem and the crane control problem. Section VIII summarizes the conclusions.

II. Multiobjective Fuzzy Control System

This section introduces the structures and functions of the proposed multiobjective fuzzy control system (MFCS) and its key component, neural fuzzy combiner (NFC). As shown in Fig. 2, the structure of the MFCS is basically a gating architecture shown in Fig. 1, in which the soft switch is realized by the NFC in the MFCS. In the proposed MFCS, we use an NFC to combine n low-level controllers, which are denoted as modules 1 to n in Fig. 2. Each module receives an input vector I_i from the environment, and produce a control signal f_i fed into the NFC for further processing at each time step, where I_i is a sub-vector of the environment state vector I. It is assumed that each low-level (fuzzy or nonfuzzy) controller has been well designed to serve a particular objective of the required multiple objectives. Since many techniques have been developed to design a controller for a single objective control problem, these techniques can be applied to design each low-level controller in the MFCS. These techniques include traditional control theorems such as PID control, fuzzy control [21], neuro-fuzzy control [5]–[10], etc. Notice that while achieving decomposition of a multiobjective task into a set of single-objective tasks may be difficult, or even impossible, there are useful classes of tasks where such decomposition is achievable. The technique proposed in this paper is applicable to the multiobjective control problems, where the decomposition is achievable and the controller

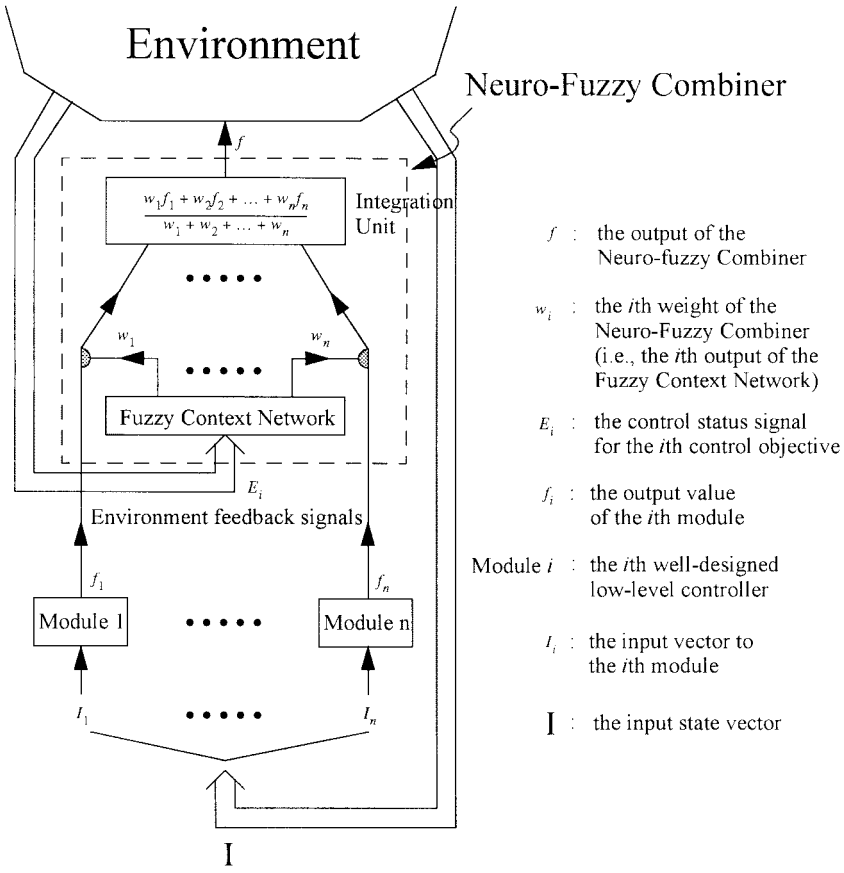


Fig. 2. Basic structure of the MFCS.

for each single objective exists. The role of the NFC is to fuse the n actions, $f_1, f_2, \cdots, f_n$, decided by the n low-level controllers and determine a proper action f acting on the environment (plant) according to control status signals at each time step. The remaining of this paper will focus on the design of the NFC.

As shown in Fig. 2, the proposed NFC is composed of two elements, a fuzzy context network (FCN) and an integration unit. The FCN decides a set of weights, $w_1, w_2, \cdots, w_n$, for the n low-level control actions according to n control status signals $E_1, E_2, \cdots, E_n$ (the "context") at each time step. The control status signals indicate the status of the control objectives and are defined by

$$E_i = \text{degree}(\text{distance of goal}_i), \qquad i = 1, \cdots, n \quad (1)$$

where any proper distance measure can be used. The control status signal E_i defined in the above indicates the degree that the ith control objective is achieved at the current time step. Such information can be obtained by simply checking the status of each control objective independently. The weight w_i produced by the FCN determines the degree (weighting) of the low-level control action f_i to act on the environment. With these weighting values, the integration unit will do the following linearly weighted summation to combine the n low-level actions into a final action f as output to the environment:

$$f = \frac{w_1 f_1 + w_2 f_2 + \cdots + w_n f_n}{w_1 + w_2 + \cdots + w_n}. \quad (2)$$

The function of the FCN is basically to find the (dynamic) mapping between the control status signals E_i and low-level control actions' weighting values w_i. Due to the powerful learning ability of neural fuzzy networks, we consider the use of neuro-fuzzy techniques to realize the FCN in this paper. The proposed FCN (see Fig. 3) has five layers with node and link numbering defined by the brackets on the right-hand side of the figure. Layer-1 nodes are input nodes (*input linguistic nodes*) representing input linguistic variables. Layer-5 nodes are output nodes (*output linguistic nodes*) representing output linguistic variables. Layer-2 and layer-4 nodes are *term nodes* that act as membership functions representing the terms of the respective input and output linguistic variables. Each layer-3 node is a rule node representing one fuzzy logic rule. Layer-3 links define the preconditions of the rule nodes, and layer-4 links define the consequents of the rule nodes. The links in layers 2 and 5 are fully connected between linguistic nodes and their corresponding term nodes.

The FCN uses the technique of *complement coding* from Fuzzy ART [22] to normalize the input/output training vectors. Complement coding is a normalization process that rescales an n-dimensional vector in $\Re^n, \boldsymbol{x} = (x_1, x_2, \cdots, x_n)$, to its $2n$-dimensional complement coding form in $[0,1]^{2n}, \boldsymbol{x}'$, such that

$$\begin{aligned} \boldsymbol{x}' &\equiv (\overline{x}_1, \overline{x}_1^c, \overline{x}_2, \overline{x}_2^c, \cdots, \overline{x}_n, \overline{x}_n^c) \\ &= (\overline{x}_1, 1-\overline{x}_1, \overline{x}_2, 1-\overline{x}_2, \cdots, \overline{x}_n, 1-\overline{x}_n) \end{aligned} \quad (3)$$

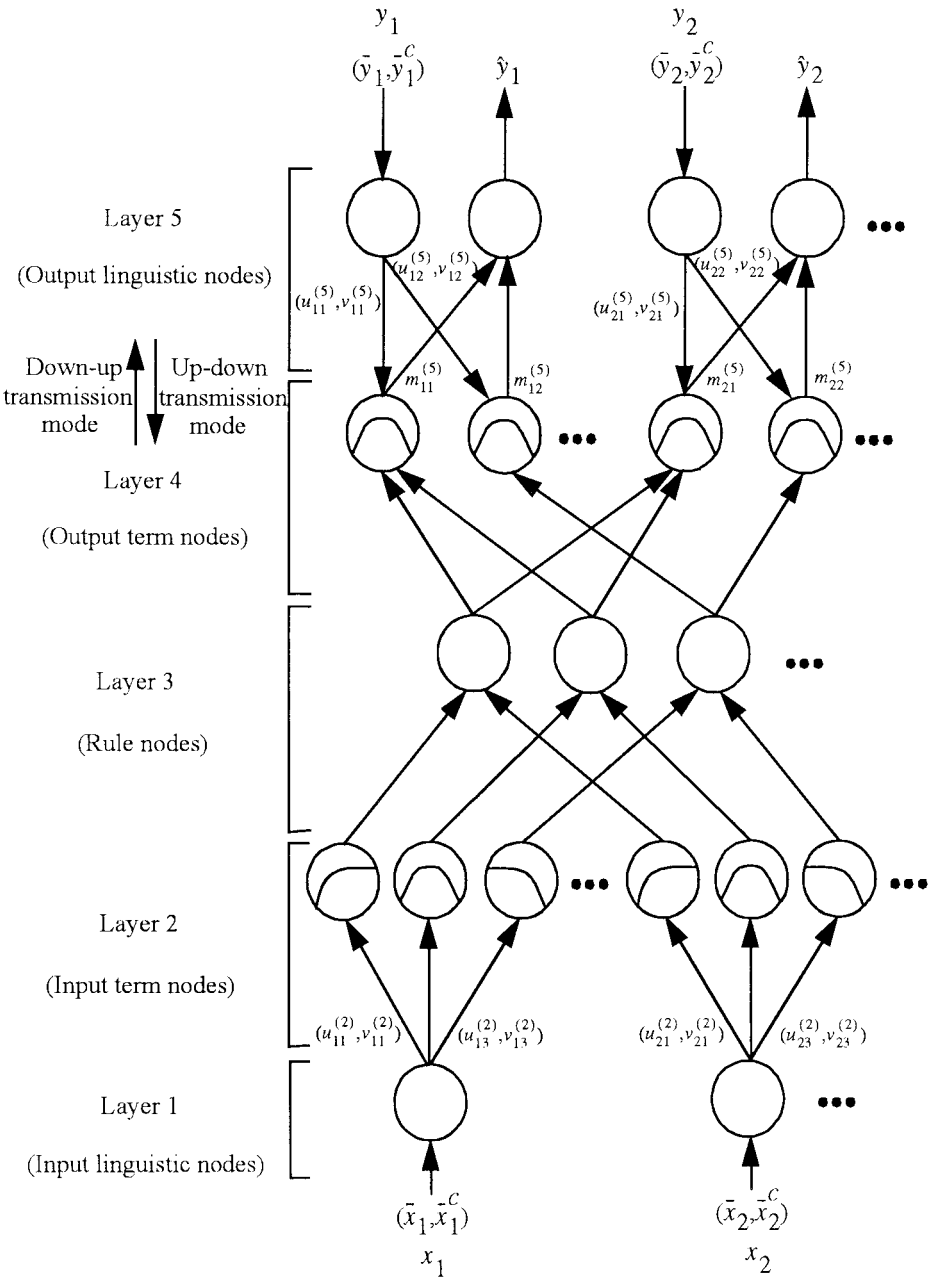


Fig. 3. Fuzzy context network.

where $(\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_n) = \overline{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|$ and $\overline{x}_1^c$ is the complement of $\overline{x}_1$, i.e., $\overline{x}_1^c = 1 - \overline{x}_1$. As mentioned in [22], complement coding helps avoid the problem of category proliferation when using fuzzy ART for fuzzy clustering. It also preserves training vector amplitude information.

We shall next describe the functions of the nodes, $a(f)$, in each of the five layers of the FCN, where $f(\cdot)$ is an integration function, and $a(\cdot)$ is an activation function. Assume that the dimension of the input space is n, and that of the output space is m.

Layer 1: Each node in this layer is called an input linguistic node and corresponds to one input linguistic variable. Layer-1 nodes just transmit input signals to the next layer directly. That is

$$f(\overline{x}_i, \overline{x}_i^c) = (\overline{x}_i, \overline{x}_i^c) = (\overline{x}_i, 1 - \overline{x}_i) \quad \text{and} \quad a(f) = f. \tag{4}$$

Note that due to the complement coding process, for each input node i, there are two output values, $\overline{x}_i$ and $\overline{x}_i^c = 1 - \overline{x}_i$.

Layer 2: Nodes in this layer are called input term nodes and each represents a term of an input linguistic variable, and acts as a one-dimensional membership function. The following trapezoidal membership function [24] is used:

$$f(z_{ij}^{(2)}) = \frac{1}{n}\left[1 - g(z_{ij}^{(2)} - v_{ij}^{(2)}, \gamma) - g(u_{ij}^{(2)} - z_{ij}^{(2)}, \gamma)\right] \quad \text{and} \quad a(f) = f \tag{5}$$

where $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ are, respectively, the left-flat and right-flat points of the trapezoidal membership function of the jth input term node of the ith input linguistic node; $z_{ij}^{(2)}$ is the input to the jth input term node from the ith input linguistic node (i.e., $z_{ij}^{(2)} = \overline{x}_i$); and

$$g(s,\gamma) = \begin{cases} 1, & \text{if } s\gamma > 1 \\ s\gamma, & \text{if } 0 \le s\gamma \le 1. \\ 0, & \text{if } s\gamma < 0. \end{cases} \tag{6}$$

The parameter γ is the sensitivity parameter that regulates the fuzziness of the trapezoidal membership function.

Layer 3: Nodes in this layer are called rule nodes and each represents one fuzzy logic rule. Each layer-3 node has n input term nodes fed into it, one for each input linguistic node. The links in layer 3 are used to perform precondition matching of fuzzy logic rules. Hence the rule nodes perform the following operation:

$$f(z_1^{(3)}, z_2^{(3)}, \cdots, z_n^{(3)}) = \sum_{i=1}^{n} z_i^{(3)} \quad \text{and} \quad a(f) = f \tag{7}$$

where $z_i^{(3)}$ is the ith input to a node in layer 3 and the summation is over the inputs of this node.

Layer 4: The nodes in this layer are called output term nodes; each has two operating modes: *down–up* transmission and *up–down* transmission modes (see Fig. 3). In down-up transmission mode, the links in layer 4 perform the fuzzy OR operation on fired (activated) rule nodes that have the same consequent

$$f(z_1^{(4)}, z_2^{(4)}, \cdots, z_p^{(4)}) = \max(z_1^{(4)}, z_2^{(4)}, \cdots, z_p^{(4)}) \quad \text{and} \quad a(f) = f \tag{8}$$

where $z_i^{(4)}$ is the ith input to a node in layer 4 and p is the number of inputs to this node from the rule nodes in layer 3. In up-down transmission mode, the nodes in this layer and the up-down transmission links in layer 5 function exactly the same as those in layer 2: each layer-4 node represents a term of an output linguistic variable and acts as a one-dimensional membership function. A set of m output term nodes, one for each output linguistic node, defines an m-dimensional hyperbox (membership function) in the output space.

Layer 5: Each node in this layer is called a output linguistic node and corresponds to one output linguistic variable. There are two kinds of nodes in layer 5. The first kind of node performs up-down transmission for training data (desired outputs) to feed into the network, acting exactly like the input linguistic nodes. For this kind of node, we have

$$f(\overline{y}_i, \overline{y}_i^c) = (\overline{y}_i, \overline{y}_i^c) = (\overline{y}_i, 1 - \overline{y}_i) \quad \text{and} \quad a(f) = f \tag{9}$$

where $\overline{y}_i$ is the ith element of the normalized desired output vector. Note that complement coding is also performed on the desired output vectors. The second kind of node performs down–up transmission for decision signal output. These nodes and the layer-5 down-up transmission links attached to them act as a defuzzifier. If $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ are the corners of the hyperbox of the jth term of the ith output linguistic variable y_i, then the following functions can be used to simulate the *center of area* defuzzification method

$$f(z_j^{(5)}) = \sum_j w_{ij}^{(5)} z_j^{(5)} = \sum_j m_{ij}^{(5)} z_j^{(5)} \quad \text{and} \quad a(f) = \frac{f}{\sum_j z_j^{(5)}} \tag{10}$$

where $z_j^{(5)}$ is the input to the ith output linguistic node from its jth term node, and $m_{ij}^{(5)} = (u_{ij}^{(5)} + v_{ij}^{(5)})/2$ denotes the center value of the output membership function of the jth term of the ith output linguistic variable.

III. Hierarchical Design Method for the Neural Fuzzy Combiner

The proposed FCN maintains the spirit of human-like thinking and reasoning as in fuzzy logic systems. Hence, if necessary and available, experts' knowledge can be easily incorporated into the structure of the FCN. In this section, we shall take use of this property and propose a hierarchical design method for the FCN.

The proposed hierarchical FCN design method is based on the hierarchical fuzzy controller design approach proposed in [2] for the controller design problem with multiple goals. In this approach, it is assumed that priorities can be assigned among the goals. Let $G = \{g_1, g_2, \cdots, g_n\}$ be a set of n goals that the system should achieve and maintain. Assume that the priorities of the goals can be ordered as $g_1, g_2, \cdots, g_n$. As an example, in the simple problem of balancing a pole on the palm of a hand and also moving the pole to a predetermined location, we can set the goal of keeping the pole balanced as the first priority, and the goal of moving to the desired location as the second priority. Furthermore, assuming that x_i is the input linguistic variable related to achieving goal g_i, we can derive the control rules recursively by the following steps.

Step 1: Acquire the rule set R_1 of appropriate control rules directly related to the highest priority goal. These rules are in the general form of

$$\text{IF } x_1 \text{ is } A_1, \text{THEN } z \text{ is } C_1.$$

Step 2: Subsequently form the rule set R_i, for $i = 2$ to n. The format of the rules in these rule sets is similar to the ones in the previous step except that they include aspects of approximately achieving the previous goal, i.e., "IF g_{i-1} is approximately achieved AND x_i is A_i, THEN z is C_i."

The above design concept can be applied to design the FCN in our MFCS. The fundamental concept of the hierarchical FCN design method is to let the FCN switch *softly* to the low-level controller that tries to achieve a high-priority goal. When the high-priority goal is achieved *approximately*, the FCN then switch softly to the low-level controller that can achieve the goal with next high priority. Take the aforementioned pole-balancing problem as an example. Assume that E_1 and E_2 are the control status signals for the first control goal $g_1 =$ "balancing the pole," and the second goal $g_2 =$ "moving the

TABLE I
EXAMPLE RULE TABLES FOR THE FCN. (a) RULE TABLE FOR w_1 AND (b) RULE TABLE FOR w_2

(a)

E_2 \ E_1	N	Z	P
N	L	S	L
Z	L	M	L
P	L	S	L

(b)

E_2 \ E_1	N	Z	P
N	S	L	S
Z	S	M	S
P	S	L	S

pole to a predetermined location," respectively. Suppose that module 1 is a well-designed controller for achieving goal g_1, and module 2 is a well-designed controller for achieving goal g_2. Then the fuzzy rule table for the FCN could be designed as in Table I. In the FCN, the inputs are the control status signals E_1, E_2, and the output are the weighting values of the low-level controllers (modules 1 and 2), w_1, w_2. In this design, the term set for E_1 and E_2 is $\{N$ (negative), Z (zero), P (positive)$\}$, and the term set for w_1 and w_2 is $\{S$ (small), M (medium), L (large)$\}$. In this example, we assume that the goal g_i is *approximately* achieved if E_i approaches zero, i.e., $E_i = Z$.

From the rule tables, we observe that module 1 (the first low-level controller) will be activated (i.e., $w_1 = L$) when the first goal g_1 is not achieved (i.e., $E_1 \neq Z$). At the same time, module 2 (the second low-level controller) is suppressed (i.e., $w_2 = S$). We also observe that module 2 is activated (i.e., $w_2 = L$) only when the first goal is achieved (i.e., $E_1 = Z$). At the same time, module 1 is suppressed (i.e., $w_1 = S$). One special case occurs when both goals are achieved, i.e., $E_1 = E_2 = Z$. In this case, the FCN will give the two low-level controllers equal weights (i.e., $w_1 = w_2 = M$).

IV. SUPERVISED LEARNING OF THE NFC

The hierarchical design method for the NFC proposed in the last section is straightforward but it relies heavily on expert knowledge and the precise analysis of controlled problem. In this section, we present an on-line two-step learning scheme for the proposed NFC (more precisely, the FCN). For an on-line incoming training pattern, the following two steps are performed in this learning scheme. First, a structure learning scheme is used to decide proper fuzzy partitions and to find the presence of rules. Second, a supervised learning scheme is used to optimally adjust the membership functions for the desired outputs. In this learning method, only the training data need to be provided from the outside world. The users don't need to provide the initial fuzzy partitions, membership functions and fuzzy logic rules.

A. Structure-Learning Step

The structure-learning task can be stated as: Given input training data at time t, $x_i(t), i = 1, \cdots, n$ and desired output value $y_i(t), i = 1, \cdots, m$, we need proper fuzzy partitions, membership functions, and fuzzy logic rules. At this stage, the network works in a two-sided manner; that is, the nodes and links in layer 4 are in the up-down transmission mode so training input and output data are fed in the network from both sides.

The structure-learning step consists of three learning processes: input fuzzy clustering process, output fuzzy clustering process, and mapping process. The first two processes are performed simultaneously on both sides of the network, and are described below.

Input Fuzzy Clustering Process: We use the fuzzy ART fast learning algorithm [22], [23] to find the input membership function parameters, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$. This is equivalent to finding proper input space fuzzy clustering or, more precisely, to forming proper fuzzy hyperboxes in the input space. Initially, for each complement coded input vector $\boldsymbol{x}'$ [see (3)], the values of choice functions, T_j, are computed by

$$T_j(\boldsymbol{x}') = \frac{|\boldsymbol{x}' \wedge \boldsymbol{w}_j|}{\alpha + |\boldsymbol{w}_j|}, \qquad j = 1, 2, \cdots, N \tag{11}$$

where "$\wedge$" is the minimum operator performed for the pairwise elements of two vectors, $\alpha \geq 0$ is a constant, N is the current number of rule nodes, and $\boldsymbol{w}_j$ is the *complement weight vector*, which is defined by $\boldsymbol{w}_j \equiv [(u_{1j}^{(2)}, 1 - v_{1j}^{(2)}), \cdots, (u_{ij}^{(2)}, 1 - v_{ij}^{(2)}), \cdots, (u_{nj}^{(2)}, 1 - v_{nj}^{(2)})]$, where $[(u_{1j}^{(2)}, v_{1j}^{(2)}), \cdots, (u_{ij}^{(2)}, v_{ij}^{(2)}), \cdots, (u_{nj}^{(2)}, v_{nj}^{(2)})]$ is the weight vector of layer-2 links associated with rule node j. The choice function value indicates the similarity between the input vector $\boldsymbol{x}'$ and the complement weight vector $\boldsymbol{w}_j$. We then need to find the complement weight vector closest to $\boldsymbol{x}'$. This is equivalent to finding a hyperbox (category) that $\boldsymbol{x}'$ could belong to. The chosen category is indexed by J, where

$$T_J = \max\{T_j \colon j = 1, \cdots, N\}. \tag{12}$$

Resonance occurs when the match value of the chosen category meets the vigilance criterion

$$\frac{|\boldsymbol{x}' \wedge \boldsymbol{w}_J|}{|\boldsymbol{x}'|} \geq \rho \tag{13}$$

where $\rho \in [0, 1]$ is a vigilance parameter. If the vigilance criterion is not met, we say *mismatch reset* occurs. In this case, the choice function value T_J is set to zero for the duration of the input presentation to prevent persistent selection of the same category during search (we call this action "disabling J"). A new index J is then chosen using (12). The search process continues until the chosen J satisfies (13). If no such J is found, then a new input hyperbox is created by adding a set of n new input term nodes, one for each input linguistic variable, and setting up links between the newly added input term nodes and the input linguistic nodes. The complement weight vectors on these new layer-2 links are simply given as the current input vector, $\boldsymbol{x}'$. These newly added input term nodes and links define a new hyperbox, and thus a new category, in the input space. We denote this newly added hyperbox as J.

Output Fuzzy Clustering Process: The output fuzzy clustering process is exactly the same as the input fuzzy clustering process except that it is performed between layers 4 and 5 which are working in the up-down transmission mode. Of course, the training pattern used now is the desired output vector after complement coding $\boldsymbol{y}' = (\overline{y}, \overline{y}^c)$. We denote the chosen or newly added output hyperbox by K. This hyperbox

is defined by the complement weight vector in layer 5: $\boldsymbol{w}_K = [(u_{1j}^{(5)}, 1-v_{1j}^{(5)}), \cdots, (u_{ij}^{(5)}, 1-v_{ij}^{(5)}), \cdots, (u_{mj}^{(5)}, 1-v_{mj}^{(5)})]$.

The above two fuzzy clustering processes produce a chosen input hyperbox indexed as J and a chosen output hyperbox indexed as K. If the chosen input hyperbox J is not newly added, then there is a rule node, J, that corresponds to it. If the input hyperbox J is a newly added one, then a new rule node (indexed as J) in layer 3 is added, and connected to the input term nodes that constitute it.

Mapping Process: After the two hyperboxes in the input and output spaces are chosen in the input and output fuzzy clustering processes, the next step is to perform the mapping process which decides the connections between layer-3 and layer-4 nodes. This is equivalent to deciding the consequents of fuzzy logic rules. This mapping process is described by the following algorithm, wherein connecting rule node J to output hyperbox K we means connecting the rule node J to the output term nodes that constitutes the hyperbox K in the output space.

Step 1: IF rule node J is a newly added node
THEN connect rule node J to output hyperbox K.
Step 2: ELSE IF rule node J is not connected to output hyperbox K originally
THEN disable J and perform Input Fuzzy Clustering Process to find the next qualified J (i.e., the next rule node that satisfies (12) and (13)].
Go to Step 1.
Step 3: ELSE no structure change is necessary.

In the mapping process, hyperboxes J and K are resized according to the *fast learning rule* [22], [23] by updating weights, $\boldsymbol{w}_J$ and $\boldsymbol{w}_K$, as follows:

$$\boldsymbol{w}_J^{(\text{new})} = \boldsymbol{x}' \wedge \boldsymbol{w}_J^{(\text{old})}, \qquad \boldsymbol{w}_K^{(\text{new})} = \boldsymbol{y}' \wedge \boldsymbol{w}_K^{(\text{old})}. \tag{14}$$

B. Parameter Learning Step

After the network structure has been adjusted according to the current training pattern, the network then enters the second learning step to adjust the parameters of the membership functions optimally with the same training pattern. In the parameter learning, the network works in the feedforward manner; that is, the node and links at layer four are in the down-up transmission mode. Basically, the idea of backpropagation algorithm is used for this parameter learning to find the errors of node outputs in each layer. Then, these errors are analyzed to perform parameters adjustment. The goal is to minimize the error function

$$E = \sum_i \tfrac{1}{2}[y_i(t) - \hat{y}_i(t)]^2, \qquad i = 1, \cdots, n \tag{15}$$

where $y_i(t)$ is the ith desired output and $\hat{y}_i(t)$ is the ith current output. It is noted that in the parameter learning we use only normalized training vectors, $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{y}}$, rather than the complement coded ones $\boldsymbol{x}'$ and $\boldsymbol{y}'$. Assuming that w is the adjustable parameter in a node, the general learning rule used is

$$w(t+1) = w(t) + \eta\left(-\frac{\partial E}{\partial w}\right) \tag{16}$$

$$\frac{\partial f}{\partial w} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial f}\frac{\partial f}{\partial w} \tag{17}$$

where η is the learning rate. To show the learning rules, we derive the rules layer by layer using the hyperbox membership functions with corners u_{ij}'s and v_{ij}'s as the adjustable parameters for these computations. For clarity, we consider the single-output case.

Layer 5: Using (10), (16), and (17), the updating rules of the corners of hyperbox membership function, v_i and u_i, are derived as

$$v_i(t+1) = v_i(t) + \eta[y(t) - \hat{y}(t)]\frac{z_i}{2\sum z_i} \tag{18}$$

$$u_i(t+1) = u_i(t) + \eta[y(t) - \hat{y}(t)]\frac{z_i}{2\sum z_i}. \tag{19}$$

The error to be propagated to the preceding layer is

$$\delta^5 = -\frac{\partial E}{\partial a^5} = y(t) - \hat{y}(t). \tag{20}$$

Layer 4: In the down-up transmission mode, there is no parameter to be adjusted in this layer. Only the error signal (δ_i^4) needs to be computed and propagated. According to (10), the error signal δ_i^4 is derived as

$$\delta_i^4 = \delta^5 \frac{m_i \sum z_i - \sum m_i z_i}{\left(\sum z_i\right)^2}. \tag{21}$$

As for the case of multiple-outputs, the computations in layers five and four are exactly the same as the above and proceed independently for each output linguistic variable.

Layer 3: As in layer four, only the error signals need to be computed. According to (8), this error signal can be derived as

$$\delta_i^3 = \frac{z_i^4}{z_{\max}}\delta_i^4 \tag{22}$$

where $z_{\max} = \max$ (*inputs of output terms node* j). As for the case of multiple-outputs, the error signal becomes $\delta_i^3 = \Sigma_k\ (z_k^4/z_{\max})\delta_k^4$, where the summation is performed over the consequents of a rule node; that is, the error of a rule node is the summation of the errors of its consequents.

Layer 2: Using (5), (16), and (17), the updating rules of v_{ij} and u_{ij} are derived as

$$v_{ij}(t+1) = v_{ij}(t) + \eta\frac{\partial a^2}{\partial v_{ij}}\delta_i^3 \quad \text{and}$$

$$u_{ij}(t+1) = u_{ij}(t) + \eta\frac{\partial a^2}{\partial u_{ij}}\delta_i^3 \tag{23}$$

where

$$\frac{\partial a^2}{\partial v_{ij}} = \begin{cases} \dfrac{\gamma}{n}, & \text{if } 0 \le (x_i - v_{ij})\gamma \le 1 \\ 0 & \text{otherwise,} \end{cases}$$

$$\frac{\partial a^2}{\partial u_{ij}} = \begin{cases} -\dfrac{\gamma}{n}, & \text{if } 0 \le (u_{ij} - x_i)\gamma \le 1 \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

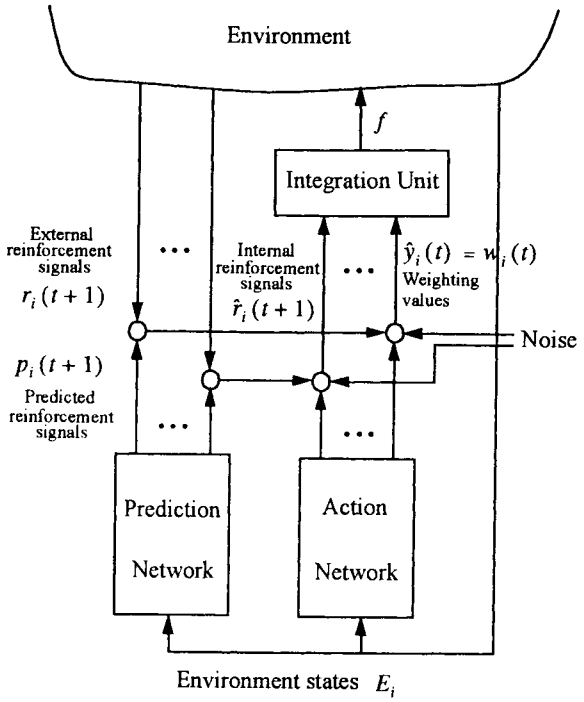


Fig. 4. Reinforcement fuzzy context network.

The above supervised learning algorithm performs well only when a set of supervised training data are available. Unfortunately, such detailed and precise training data may be very expensive or even impossible to obtain in some real-world applications. This problem becomes even severe for multiobjective control. In many cases, the controlled system may only be able to provide the learning algorithm with reinforcement signals such as binary decisions of right/wrong of the current controller. In order to train the FCN with this kind of evaluative feedback, we shall present a reinforcement learning scheme in the next section.

V. Reinforcement Learning of the NFC

Fig. 4 shows how a controller and its training environment interact in a reinforcement learning problem. The environment supplies a time-varying input vector to the network, receives its time-varying output vectors, and then provides time-varying scalar reinforcement signals. For each control objective i, there is a corresponding reinforcement signal $r_i(t)$ indicating the degree (status) of fulfillment of this objective. In this paper, a reinforcement signal $r_i(t)$ can be both forms

1) two-valued number, $r_i(t) \in \{-1, 0\}$, such that $r_i(t) = 0$ means "a success" and $r_i(t) = -1$ means "a failure";
2) a real number, $r_i(t) \in [-1, 0]$, which represents a more detailed and continuous degree of failure or success.

We also assume that each $r_i(t)$ is the reinforcement signal available at time step t and is caused by the inputs and outputs chosen at time step $t-1$ or even affected by earlier inputs and outputs.

To resolve the reinforcement learning problems, an integrated network, called the reinforcement fuzzy context network (RFCN) is proposed. As shown in Fig. 4, the proposed RFCN consists of two FCNs; one FCN is used as the action network (fuzzy controller) and the other FCN as the prediction network (fuzzy predictor). Each network has exactly the same structure as that shown in Fig. 3. The action network plays the role of the FCN in Fig. 2; it decides a set of weights, $w_1, w_2, \cdots, w_n$, for the n low-level control actions according to n control status signals $E_1, E_2, \cdots, E_n$ (the "context") at each time step. The prediction network models the environment such that it can predict the reinforcement signals that will eventually be obtained from the environment for the current action (more precisely, the current weighting values) determined by the action network. The predicted reinforcement signals can provide the action network beforehand and more detailed reward/penalty information ("internal reinforcement signals") about the candidate outputs (weighting values) for the action network to learn and decrease the uncertainty it faces to speed up the learning.

Associated with the RFCN is a reinforcement learning scheme. We shall next introduce the functions and learning algorithms of the action network and the prediction network, respectively, in the following two subsections.

A. Action Network

The goal of the reinforcement learning of the action network is to adjust the parameters w (composed of u_{ij} and v_{ij}) of the action network, to change the connection types, or even to add new nodes, if necessary, such that each reinforcement signal is maximum; that is

$$\Delta w \propto \sum_i \frac{\partial r_i}{\partial w}. \tag{25}$$

To determine $\partial r_i/\partial w$, we need to know $\partial r_i/\partial y_i$, where y_i is the ith-output of the action network with r_i as its reinforcement signal. Since the reinforcement signal does not provide any hint as to what the right answer should be in terms of a cost function, there is no gradient information, and the gradient, $\partial r_i/\partial y_i$, can only be estimated. If we can estimate $\partial r_i/\partial y_i$, then the on-line supervised learning algorithm developed in the last section can be directly applied to the action network to solve the reinforcement learning problem.

In estimating the gradient information using the stochastic exploration method [25], [26], the output y_i of the action network does not directly act on the environment. Instead, it is treated as a mean (expected) value. The actual output value, $\hat{y}_i$, is chosen by exploring a range around this mean point. This range of exploration corresponds to the variance of a probability function which is the normal distribution in our design. The amount of exploration, $\sigma_i(t)$, is chosen as

$$\sigma_i(t) = \frac{1}{(1 + \exp(2p_i(t)))} \tag{26}$$

where $p_i(t)$ is the predicted (expected) reinforcement signal coming from the prediction network to predict $r_i(t)$. Once the variance has been decided, the actual output of the stochastic node can be set as

$$\hat{y}_i(t) = N(y_i(t), \sigma_i(t)). \tag{27}$$

That is, $\hat{y}_i(t)$ is a normal or Gaussian random variable with the density function

$$\text{Prob}(\hat{y}_i) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left\{\frac{-(\hat{y}_i - y_i)^2}{2\sigma_i^2}\right\}. \tag{28}$$

Using the above stochastic exploration method with multiparameter distribution, the gradient information is estimated as

$$\begin{aligned}\frac{\partial r_i}{\partial y_i} &\approx \hat{r}_i(t)\hat{y}_i(t-1)\left[\frac{\hat{y}_i(t-1) - y_i(t-1)}{\sigma_i(t-1)}\right] \\ &\equiv \hat{r}_i(t)\left[\hat{y}_i\frac{\hat{y}_i - y_i}{\sigma_i}\right]_{t-1}\end{aligned} \tag{29}$$

where the subscript, $t-1$, represents time displacement. The signal $\hat{r}_i(t)$ is called the internal reinforcement signal coming from the prediction network. It indicates the prediction error of the prediction network (e.g., the difference between $p_i(t)$ and r_i). The exact mathematical formula of the internal reinforcement signals, $\hat{r}_i(t)$, will be studied in the next subsection. Equation (29) is in fact a *competitive* learning rule due to the product term $\hat{y}_i(t-1)$ and thus we call this new scheme as *competitive stochastic-exploration* rule. This term functions as a "gating" factor for scaling the effects caused by different low-level controllers. This competitive stochastic-exploration learning rule is a novel scheme to solve the reinforcement learning problem in multiobjective control.

Basically, the training of an action network is not a supervised learning problem. There are no correct "target" output values for each input pattern. However, in the structure-learning step of the action network, we need desired output values to determine proper output fuzzy partitions as well as membership functions and to find fuzzy logic rules. The desired output values can be estimated as

$$y_i^d(t) \approx y_i(t) + \kappa\frac{\partial r_i}{\partial y_i} \tag{30}$$

where κ is a real number in the range $[0, 1]$ and $\partial r_i/\partial y_i$ can be replaced by $\hat{r}_i(t)[\hat{y}_i(\hat{y}_i - y_i)/\sigma_i]_{t-1}$ in (29). According to the input state values and the estimated desired output values, the structure-learning step in Section IV can be performed on the action network directly.

As for the parameter learning of the action network, the goal is to maximize the external reinforcement signal, $r_i(t)$. Thus, we need to estimate the gradient information, $\partial r_i/\partial y_i$, as we have obtained in (29). With this gradient information, the parameter-learning step can be performed in the way similar to that in the supervised learning scheme in Section IV. More clearly, the exact parameter learning rules of the action network are the same as (18)–(24) except that the term $-[y(t) - \hat{y}(t)]$ in these equations is replaced by the new error term $\hat{r}_i(t)[\hat{y}_i(\hat{y}_i - y_i)/\sigma_i]_{t-1}$.

B. Prediction Network

An FCN is used as the prediction network in the RFCN shown in Fig. 4. The prediction network receives external reinforcement signals from the environment and produces more informative internal reinforcement signals to the action network. It is noted that a prediction network in the RFCN has n outputs, one for predicting each external reinforcement signal. To solve the temporal credit assignment problem mentioned in Section I, the technique based on the temporal-difference methods, which are closely related to the dynamic programming technique [27], is used [14], [15]. Unlike the supervised learning method which assigns credit according to the difference between the predicted and actual output, the temporal-difference methods assign credit according to the difference between temporally successive predictions.

A general learning rule for the temporal-difference procedures [15] is

$$\Delta w(t) = \eta(r(t) + \gamma p(t) - p(t-1))\nabla_w p(t-1) \tag{31}$$

where

w	predictor's adjustable parameter;
p	predictor's output;
r	external reinforcement signal (i.e., desired output);
$\gamma\ (0 \le \gamma < 1)$	discount-rate parameter;
η	learning rate.

We shall derive the learning rule of the prediction network according to (31). According to this equation, the error signal $\hat{r}_i(t)$ of the output node of the prediction network is

$$\hat{r}_i(t) = r_i(t) + \gamma p_i(t) - p_i(t-1), \qquad 0 \le \gamma < 1. \tag{32}$$

The goal of training the prediction network is to minimize the squared error

$$\begin{aligned}E &= \sum_i \frac{1}{2}(\hat{r}_i(t))^2 \\ &= \sum_i \frac{1}{2}(r_i(t) + \gamma p_i(t) - p_i(t-1))^2, \\ &\qquad i = 1, \cdots, n.\end{aligned} \tag{33}$$

With the above error function, we can derive the structure/parameter learning algorithm for the prediction network using the gradient descent method. The resulting structure learning step is exactly the same as that of the supervised learning algorithm in Section IV. The parameter learning algorithm of the prediction network can be obtained by replacing the term $y_i(t) - \hat{y}_i(t)$ by the term $r_i(t) + \gamma p_i(t) - p_i(t-1)$ in (18) to (24).

Since both the action and prediction networks are FCN's, expert knowledge, if available, can be put into the network directly as *a priori* knowledge to speed up the learning. Obviously, the hierarchical design method introduced in Section III could be used to design the action network directly. When proper fuzzy rules for the action network have been determined, we can design the prediction network based on these rules using the concept of hierarchical design approach.

Consider again the aforementioned pole-balancing problem in Section III as an example. Assume the fuzzy rule table for the action network is shown in Table I. Then the fuzzy rule table for the prediction network could be designed as Table II. For the prediction network, the inputs are the control status signals E_1, E_2, and the outputs are the predicted reinforcement

TABLE II
EXAMPLE RULE TABLES FOR THE PREDICTION NETWORK FOR THE RFCN. (a) RULE TABLE FOR p_1 AND (b) RULE TABLE FOR p_2

(a)

E_2 \ E_1	N	Z	P
N	M	B	M
Z	M	G	M
P	M	B	M

(b)

E_2 \ E_1	N	Z	P
N	B	M	M
Z	B	G	B
P	M	M	B

signals, p_1, p_2. In this design, the term set for E_1 and E_2 is $\{N$ (negative), Z (zero), P (positive)$\}$, and the term set for p_1 and p_2 is $\{B$ (bad), M (median), G (good)$\}$. It is observed that the rules in Table II predict the reinforcement signals for the actions chosen by the rules in Table I under certain control status. For example, from Table I, when $E_1 = N$, the action network chooses $w_1 = L$ and $w_2 = S$, which suppresses the second low-level controller and allows the first low-level controller to try to achieve the first control objective (i.e., balancing the pole). Hence, we expect to obtain a larger reinforcement signal for the first control objective $(p_1 = M)$. To keep the pole balanced when $E_1 = N$, we need a force to the left, which will push the pole position further away from the desired position when $E_2 = N$, but push the pole position closer to the desired position when $E_2 = P$. Hence we can expect a bad reinforcement feedback for the second control objective $(p_2 = B)$ when $E_2 = N$, and a better reinforcement feedback for the second control objective $(p_2 = M)$ when $E_2 = P$. The other rules in Table II can be obtained by similar consideration.

VI. CONVERGENCE ANALYSIS OF REINFORCEMENT LEARNING

In this section, we shall study the convergence property of the proposed reinforcement learning scheme. As shown in Fig. 4, the proposed RFCN is composed of two FCN's acting as the prediction network and action network. In Section V, we have proposed a reinforcement learning algorithm for RFCN, where the weight updating rules for the action network and the prediction network were derived, respectively. The weight updating rule for the action network derived in Section V-A is basically a gradient descent procedure in which the network parameters are modified along the negative direction of the error function with respect to these parameters. Also, we can show that the FCN is a universal approximator via the Stone–Weierstrass Theorem. So, it can be expected that the parameters will eventually converge to the values which minimize the error function to within some small fluctuations. However, this expectation is based on the condition that the *estimated gradient information is correct*. As we know, the gradient information is estimated based on the internal reinforcement signal, $\hat{r}_i(t)$, in (32) which is provided by the prediction network. Hence, to study the convergence property of the reinforcement learning of RFCN, we need to study the convergence property of the weight updating rule of the prediction network.

As described in Section V-B, the reinforcement learning of the prediction network is mainly based on the *temporal difference* (TD) learning algorithm. The convergence properties of TD learning algorithm have been studied by Sutton [15], Dayan [28], [29], etc. They transformed the TD learning algorithm into absorbing Markov chain and exploited the concept of the convergence analysis in *learning automata*, such as *ϵ-optimality, absolutely expedient*, $\cdots$, etc., to study the convergence property. However, these results are only suitable for the predictor (e.g., a neural network) with a single output. Since we concern with the predictor (FCN) with multiple outputs in this paper and thus face the mutual credit assignment problem, we need to come out our own convergence proof for the proposed reinforcement learning scheme.

Before we perform the convergence analysis, some symbols used in TD learning algorithm need to be explained or redefined in advance. As shown in (31), the rule of weight vector updating for TD learning algorithm is given by

$$\boldsymbol{w}(t) = \boldsymbol{w}(t-1) + \eta(\boldsymbol{r}(t) + \gamma \boldsymbol{p}(t) - \boldsymbol{p}(t-1))\nabla_w \boldsymbol{p}(t-1)$$

where $\boldsymbol{p}(t) = \boldsymbol{p}(\boldsymbol{x}(t), \boldsymbol{w}(t-1))$, and $\boldsymbol{x}(t)$ is the input vector at time t. However, the above weight vector updating would be inconvenient because prediction differences result from changes in $\boldsymbol{x}$; changes in $\boldsymbol{x}$ should be reflected in weight vector $\boldsymbol{w}$ updating. Hence, Sutton proposes that the above equation can be transformed to

$$\begin{aligned}\boldsymbol{w}(t) =& \boldsymbol{w}(t-1) + \eta(\boldsymbol{r}(t) + \gamma \boldsymbol{p}(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) \\ & - \boldsymbol{p}(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\nabla_w \boldsymbol{p}(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\end{aligned} \tag{34}$$

we shall now study the convergence characteristic of the TD learning algorithm in (34) on the prediction network with two output nodes by using the technique in [30].

Step 1: First, we normalize the gradient vector $\nabla_w p_i(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))$ in (34) and obtain

$$\begin{aligned}\Delta \boldsymbol{w}_i(t) =& \eta(r_i(t) + \gamma p_i(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) \\ & - p_i(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \\ & \times \frac{\nabla_w p_i(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))}{||\nabla_w p_i(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))||} \qquad i = 1, 2.\end{aligned} \tag{35}$$

Notice that (35) is equivalent to the original TD learning algorithm [(31)] used in our prediction network, since the input/output data are normalized to the range of [0, 1] in the process of complement coding in the proposed learning scheme.

Step 2: In (33), we formulate the error function of the prediction network as the summation of every single error value of the prediction outputs. Hence, we can express the updating rule of the weight vector in the prediction network by

$$\begin{aligned}\Delta \boldsymbol{w}(t) &= \Delta \boldsymbol{w}_1(t) + \Delta \boldsymbol{w}_2(t), \\ \boldsymbol{w}(t) &= \boldsymbol{w}(t-1) + \Delta \boldsymbol{w}(t).\end{aligned} \tag{36}$$

Step 3: With the preparation in Steps 1 and 2, we can show that if the learning constant η is small enough, then the updated weight vector has greater effect of bringing $p_i(\boldsymbol{x}(t-1), \boldsymbol{w}(t))$ closer to $p_i(\boldsymbol{x}(t), \boldsymbol{w}(t-1))$ than the effect of bringing $p_j(\boldsymbol{x}(t-1), \boldsymbol{w}(t))$ closer to $p_j(\boldsymbol{x}(t), \boldsymbol{w}(t-1))$ assuming that the difference in output $r_i(t) + \gamma p_i(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) - p_i(\boldsymbol{x}(t-$

$1), \boldsymbol{w}(t-1))$ is greater than $r_j(t) + \gamma p_j(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) - p_j(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))$, where $i, j \in \{1, 2\}, i \neq j$, are the indexes of the two output nodes of the prediction network. In other words, larger correction will be made to the output node which produces bigger prediction error. Namely, we can prove that the reinforcement learning process of the prediction network will go toward the convergence direction. This property is shown in the rest of this section.

For simplification, we assume the absolute variation of the prediction value from the first output is greater than that of the absolute variation from the second output, i.e.,

$$|A| > |B|, \tag{37}$$

where A and B are set as follows:

$$\begin{aligned} A &\equiv r_1(t) + \gamma p_1(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) - p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)), \\ B &\equiv r_2(t) + \gamma p_2(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) - p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)). \end{aligned} \tag{38}$$

According to (36), we can write $p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t))$ as follows:

$$\begin{aligned} &p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t)) \\ &\quad = p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1) + \Delta \boldsymbol{w}(t)) \\ &\quad = p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1) + \Delta \boldsymbol{w}_1(t) + \Delta \boldsymbol{w}_2(t)). \end{aligned} \tag{39}$$

If $\Delta \boldsymbol{w}(t)$ in (39) is a microvector, the above equation becomes

$$\begin{aligned} &p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t)) \\ &\quad \approx p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) + (\Delta \boldsymbol{w}_1(t) \\ &\qquad + \Delta \boldsymbol{w}_2(t))^T \nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)). \end{aligned} \tag{40}$$

By (35), the above equation can be rewritten as

$$\begin{aligned} &p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t)) \\ &\quad \approx p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \\ &\qquad + \eta(r_1(t) + \gamma p_1(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) \\ &\qquad - p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))) \\ &\qquad \times \left\langle \frac{\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))}{\|\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\|}, \right. \\ &\qquad \left. \nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \right\rangle \\ &\qquad + \eta(r_2(t) + \gamma p_2(\boldsymbol{x}(t), \boldsymbol{w}(t-1)) \\ &\qquad - p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))) \\ &\qquad \times \left\langle \frac{\nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))}{\|\nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\|}, \right. \\ &\qquad \left. \nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \right\rangle. \end{aligned} \tag{41}$$

According to (38) and through a simple vector operation, (41) becomes (42), as shown at the bottom of the page.

In (42)

$$\left| \frac{\langle \nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)), \nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \rangle}{\|\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\| \, \|\nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\|} \right| \leq 1$$

holds. Hence, it is possible to set the value of the learning rate η as shown in (43) at the bottom of the page.

Then, (42) becomes

$$\begin{aligned} &p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t)) \\ &\quad \approx p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \\ &\qquad + \eta \|\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\| (A + B \cos\eta). \end{aligned} \tag{44}$$

Previously, we assumed $|A| > |B|$ [(37)]. Consider the case of $A > B \geq 0$ now. We know that $|\cos\eta| \leq 1$, so we have

$$\begin{aligned} &p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t)) \\ &\quad \geq p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \\ &\qquad + \eta \|\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\| (A - B) \\ &\quad > p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)). \end{aligned} \tag{45}$$

The above evaluation shows that in the case of $A > B \geq 0$, $p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t))$ approaches $p_1(\boldsymbol{x}(t), \boldsymbol{w}(t-1))$ rather than $p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))$, provided that η is small enough. By using the same technique, we can obtain the same conclusion for the cases of $A > 0 \geq B$, $0 \geq B > A$, and $B \geq 0 > A$. This completes the proof of the convergence property stated in Step 3 in this section.

VII. Illustrative Examples

In this section, the proposed MFCS is applied to two simulated examples to show its performance and applicability. These two examples are the cart-pole balancing control and the crane control problems.

$$\begin{aligned} p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t)) \approx p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) + \eta \|\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\| \\ \times \left(A + B \frac{\langle \nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)), \nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \rangle}{\|\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\| \, \|\nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\|} \right). \end{aligned} \tag{42}$$

$$\cos\eta = \frac{\langle \nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)), \nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1)) \rangle}{\|\nabla_w p_1(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\| \, \|\nabla_w p_2(\boldsymbol{x}(t-1), \boldsymbol{w}(t-1))\|}. \tag{43}$$

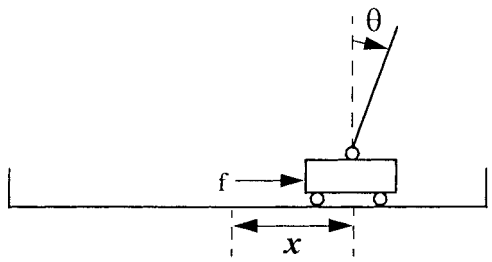


Fig. 5. Cart–pole balancing system.

A. Control of the Cart–Pole Balancing System

The cart–pole balancing problem involves of learning how to balance an upright pole as shown in Fig. 5. The bottom of the pole is hinged to a cart that travels along a finite-length track to its right or its left. Both the cart and pole can move only in the vertical plane; that is, each has only one degree of freedom. There are two control objectives in our problem; move the cart to a desired position and keep the pole at the upright position at the same time. Hence this is a multiobjective control problem and the proposed MFCS will be applied here to achieve the two objectives simultaneously. There are four input state variables in this system:

- θ angle of the pole from an upright position (in degrees);
- $\dot{\theta}$ angular velocity of the pole (in degrees per second);
- x horizontal position of the cart's center (in meters);
- $\dot{x}$ velocity of the cart (in meters per second).

The only control action is f, which is the amount of force (in *Newtons*) applied to the cart to move it left or right. The goal of this control problem is to design or train the MFCS such that it can determine a sequence of forces with proper magnitudes to apply to the cart to balance the pole as well as to push the cart to a desired position.

The model and the corresponding parameters of the cart-pole balancing system for our computer simulation are adopted from [14] with the consideration of friction effects. The equations of motion that we used are

$$\begin{aligned}
\theta(t+1) &= \theta(t) + \Delta\dot{\theta}(t),\\
\dot{\theta}(t+1) &= \dot{\theta}(t) + \Delta\Big\{(m+m_p)g\ \sin\theta(t)\\
&\quad - \cos\theta(t)[f(t) + m_p l\dot{\theta}(t)^2\ \sin\theta(t)\\
&\quad - \mu_c\,\mathrm{sgn}(\dot{x}(t))] - \frac{\mu_p(m+m_p)\dot{\theta}(t)}{m_p l}\Big\}\Big/\\
&\quad [(4/3)(m+m_p)l - m_p l\ \cos^2\theta(t)],\\
x(t+1) &= x(t) + \Delta\dot{x}(t),\\
\dot{x}(t+1) &= \dot{x}(t) + \Delta\{f(t) + m_p l[\dot{\theta}(t)^2\ \sin\theta(t)\\
&\quad - \ddot{\theta}(t)\ \cos\theta(t)] - \mu_c\ sgn(\dot{x}(t))\}/(m+m_p)
\end{aligned} \tag{46}$$

where

- $g = 9.8$ m/s^2 acceleration due to the gravity;
- $m = 1$ kg mass of the cart;
- $m_p = 0.1$ kg mass of the pole;
- $l = 0.5$ m half-pole length;
- $\mu_c = 0.0005$ coefficient of friction of the cart on the track;
- $\mu_p = 0.000\,000\,2$ coefficient of friction of the pole on the cart;
- $\Delta = 0.02$ s sampling interval.

The constraints on the variables are $-12^\circ < \theta < 12^\circ$, $-2.4m \le x \le 2.4m$, and $-10N \le f \le 10N$.

To apply the proposed MFCS to solve the above two-objective problem, we need to define two control status signals, E_1 and E_2. In this problem, they are defined as follows according to the two control objectives

$$E_1 \stackrel{\text{def}}{=} \theta^\circ/12^\circ, \qquad E_2 \stackrel{\text{def}}{=} (x - x_{\text{desired}})/4.8. \tag{47}$$

The control status signal E_1 indicates the status of the first control objective, i.e., to keep the pole balanced. The control status signal E_2 indicates the status of the second control objective; i.e., to move the cart to a desired position, x_{desired}. They are normalized to be in $[-1, 1]$. Hence an E_i value close to zero means the respective control objective is almost achieved.

Before training an NFC for the MFCS to control the two-objective cart-pole system, we first need to design two low-level controllers, one for each control objective. The first low-level controller (called "angle controller") is to determine a sequence of forces to keep the pole balanced regardless of the cart position. The second low-level controller (called "position controller") is to determine a sequence of forces to move the cart to a desired position as soon as possible regardless of the pole angle. Since both the angle controller and position controller are normal single-objective controllers, they can be designed easily by existing control schemes. In this work, we designed two fuzzy controllers to act as the angle and position controllers, respectively, based on the trial-and-error design approach [21]. We shall use the NFC with reinforcement learning to combine these two low-level controllers to achieve both control objectives simultaneously.

We formulate the cart–pole balancing problem as a reinforcement learning problem by defining two reinforcement signals, one for each control objective, as follows:

$$r_1(t) = \begin{cases} -1, & \text{if } |\theta^\circ(t)| > 12^\circ \\ 0, & \text{otherwise,} \end{cases} \tag{48}$$

$$r_2(t) = \begin{cases} -1, & \text{if } |x - x_{\text{desired}}| > 2.4m \\ & \text{or } |x - x_{\text{desired}}| > 0.3m \text{ over} \\ & \quad 300 \text{ time steps} \\ 0, & \text{otherwise.} \end{cases} \tag{49}$$

These reinforcement signals indicate that a failure occurs if the pole angle deviation exceeds $\pm 12^\circ$, the cart bumps into the walls on both sides (i.e., $|x - x_{\text{desired}}| > 2.4m$), or the cart dose not reach the desired position in 300 time steps. The last condition is to ensure that the cart can arrive in the desired position in short time instead of wandering over the track, where the number "300 time steps" is chosen objectively in the simulations.

In the simulations, the learning system was tested for four runs. Each run consists of a sequence of trials; each trial begins with the same initial condition and ends with a failure signal

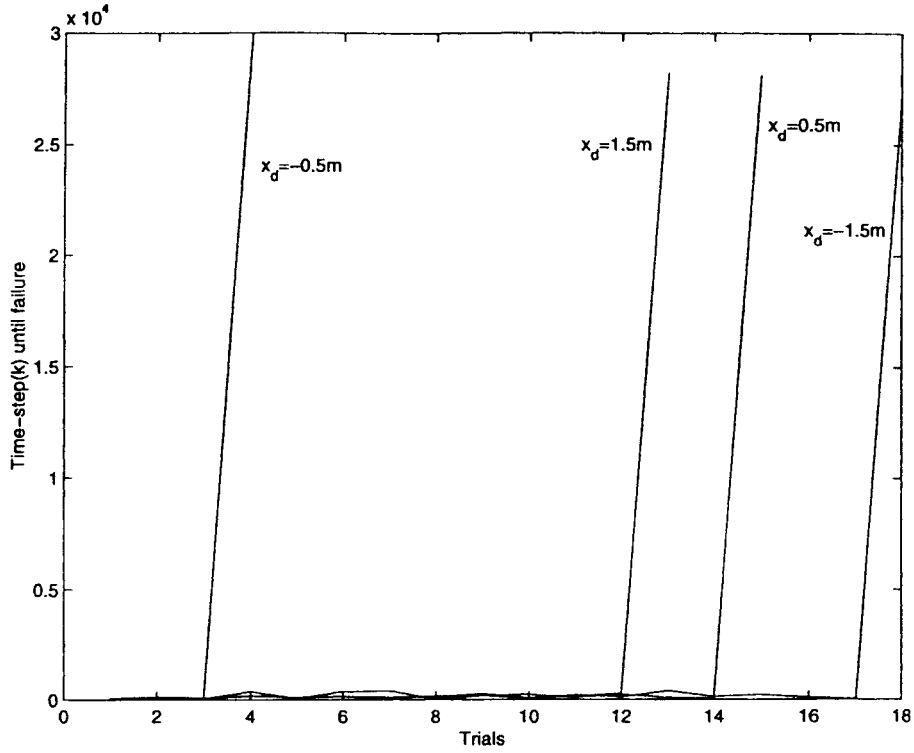


Fig. 6. Performance of the RFCN on the cart–pole balancing problem.

appearing, i.e., $r_i(t) = -1$. A run consists of at most 50 trials, unless the duration of each run exceeds 30 000 time steps. In the latter case, we consider the run "successful." The following learning parameters were used for each trial. The learning rate $\eta = 0.001$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.7, \rho_{\text{output}} = 0.7$ were used for the action network. The learning rate $\eta = 0.005$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.7, \rho_{\text{output}} = 0.7$ were used for the critic network, where ρ_{input} was the vigilance parameter used in the input fuzzy clustering process and ρ_{output} was the vigilance parameter used in the output fuzzy clustering process.

Fig. 6 shows the number of trials until success for each run, where the four runs start at different initial states. The results indicate that the RFCN can achieve both control objectives simultaneously after several trials (11 trials on average) for four different initial conditions and desired cart positions. To observe the learning process more clearly, Fig. 7 shows the state status, θ and x, and the corresponding two-valued external reinforcement signals of a run. From the figure, we can tell the cause of each failure signal (i.e., $r_i = -1$); a failure signal results either from the unacceptable pole angle or from the unacceptable cart position as defined in (48), (49). It is noted from (49) that there are two causes for $r_2 = -1$, i.e., $|x - x_{\text{desired}}| > 2.4m$ or $|x - x_{\text{desired}}| > 0.3m$ over 300 time steps. Therefore, in Fig. 7, we use $r_{21}(t)$ to represent the reinforcement signal corresponding to the situation, $|x - x_{\text{desired}}| > 2.4m$, and $r_{22}(t)$ to represent the reinforcement signal corresponding to the situation, $|x - x_{\text{desired}}| > 0.3m$ over 300 time steps.

After reinforcement learning, the action network in the RFCN is well trained and can be used alone as the fuzzy context network (FCN) in Fig. 2 to form the whole MFCS for controlling the multiobjective cart–pole system. The learned FCN should function as the one designed by the hierarchical design method in Section III. When we applied this learned FCN to the above cart-pole system for the case of $x_{\text{desired}} = -1.5$, the outputs (weighting values) of the FCN (w_1 and w_2), the weighted-sum force (f), and the resulted state status (θ and x) are shown in Fig. 8. It is observed that before the two objectives are achieved simultaneously (i.e., $\theta = 0$ and $x_{\text{desired}} = -1.5$ constantly), the two weighting factors, w_1 and w_2, appear to be complementary to keep the priority relation of the two low-level controllers.

As mentioned in Section III and Section V-B, expert knowledge, if available, can be put into the action and prediction networks directly as *a priori* knowledge to speed up the learning. To illustrate this point, we performed another simulation in which the rule tables in Tables I and II were put into the action and prediction networks of the RFCN, respectively, as *a priori* knowledge before starting the reinforcement learning. The membership functions of the fuzzy terms in the two rule tables are given in Fig. 9. In the reinforcement learning process of the rule-embedded RFCN, the number of trials until success for each run is shown in Fig. 10, where the four runs start at different initial states as we did in the above. As compared to Fig. 6, Fig. 10 indicates that the embedded expert knowledge has reduced the average number of trials of RFCN from 11 to 5.

Comparing RFCN to the manually designed FCN, we see that although the latter could outperform the trained RFCN if manually trial-and-error design is performed exhaustively, the former has some advantages over the latter. First, the hierarchical (manual) design method for the FCN is straightforward

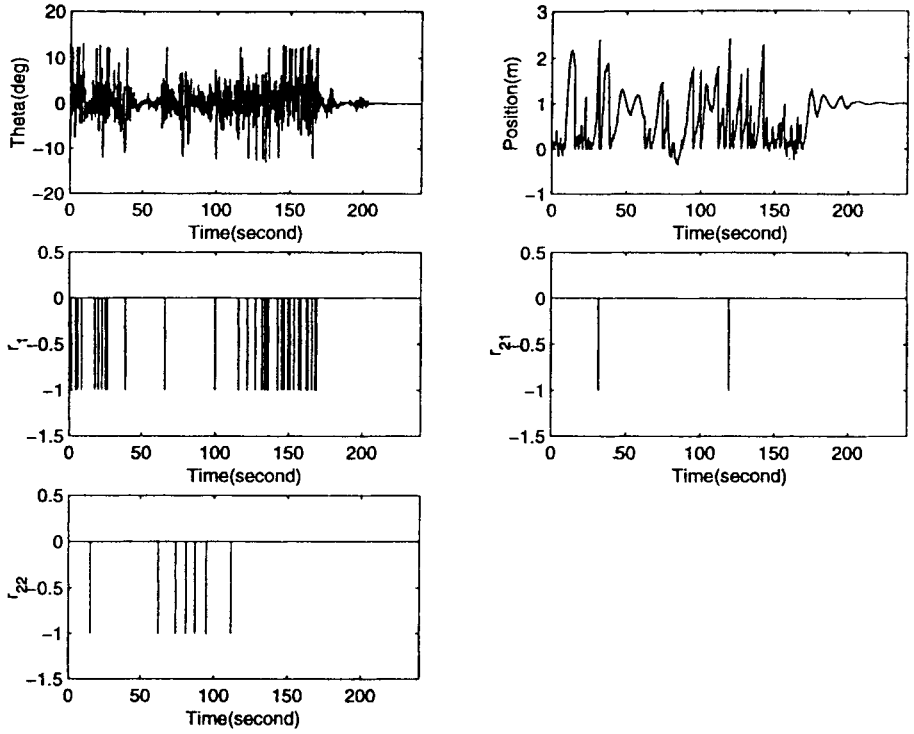


Fig. 7. Variations of pole angle, cart position and reinforcement signals during the reinforcement learning process of the RFCN on the cart-pole problem.

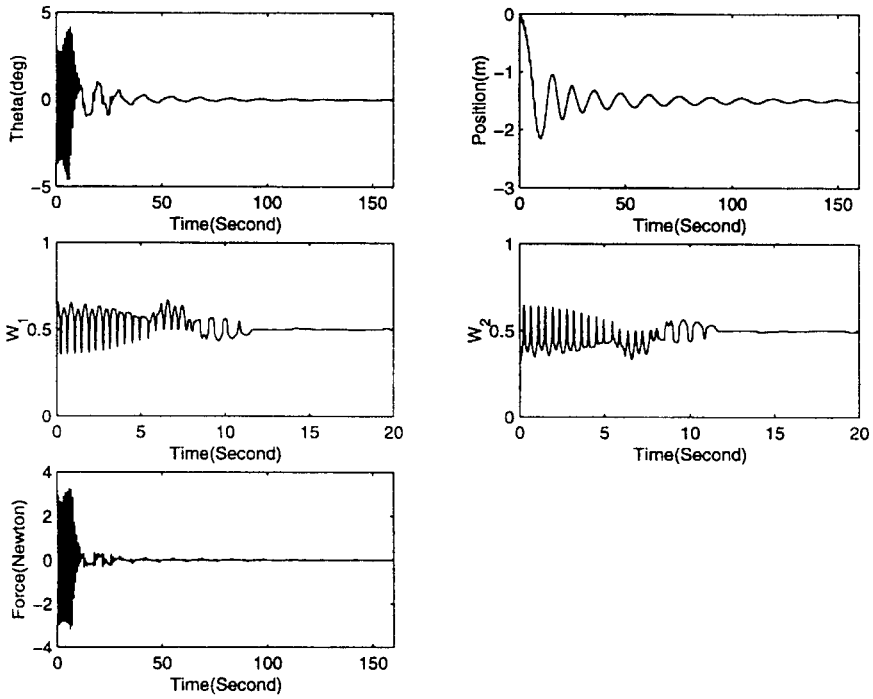
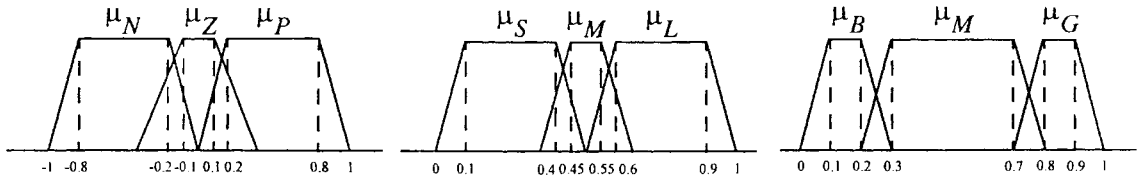


Fig. 8. Weighting values (w_1 and w_2—outputs of learned FCN), weighted-sum (integrated) force (f—output of neural fuzzy combiner), and the resulted state status (θ and x) of the cart-pole system controlled by the trained RFCN in the case of $x_{\text{desired}} = -1.5$.

but it relies heavily on expert knowledge and precise analysis of controlled problem. On the other hand, the RFCN can learn by itself automatically through the rough reinforcement feedback from the environment and avoid the needs of expert knowledge and trial-and-error design. Second, the manually designed FCN is static; its rules and membership functions are fixed once they are designed. Hence it cannot adapt to new environment, and might not keep the system stable in face of big disturbances which are out of the control range of designed rules. On the contrary, RFCN is a dynamic system with on-line learning ability and can adapt to new environment through continuous learning. Fig. 11 illustrates this point, where the state status, θ and x, of the cart–pole system controlled by the RFCN and by the manually designed FCN, respectively,



(a) (b) (c)

Fig. 9. Membership functions of the fuzzy terms in Tables I and II used as *a priori* knowledge of the action and prediction networks of the RFCN. (a) Membership functions for environment signals E_i, (b) membership functions for weighting factors w_i, and (c) membership functions for predicted reinforcement signal p_i.

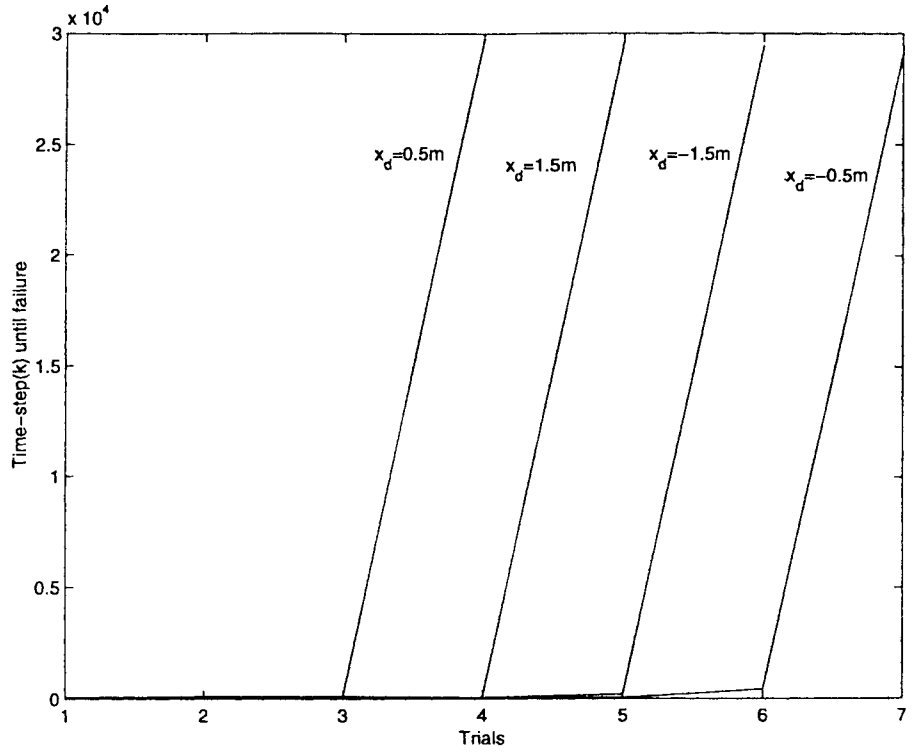


Fig. 10. Performance of the rule-embedded RFCN on the cart–pole balancing problem.

are compared. A big disturbance was added to the cart at the 200th s. This disturbance caused the cart-pole system controlled by the manually designed FCN corrupt (see the low two figures in Fig. 11). However, the RFCN regained the two control objectives in short time after the disturbance through on-line learning as shown in the upper two figures of Fig. 11.

Since the cart-pole control has become a benchmark problem in the reinforcement learning research realm, we can compare the performance of our RFCN to that of others. In Fig. 12, the performance of the rule-embedded RFCN and blank RFCN is compared to that of seven other reinforcement learning schemes. These schemes can be found in the references indicated in the parentheses of Fig. 12. The results show that the proposed RFCN outperforms the other schemes in terms of average number of trials, average number of generated rules, and average angular deviation of the pole under the control of learned controller. It is noted that all the compared schemes except [10] solved the single objective cart-pole control problem, i.e., they only tried to keep the pole at upright position regardless of the cart position. Hence, the comparisons in Fig. 12 are not exactly fair to the RFCN which faces a more difficult problem. Even though, the RFCN shows the best performance in the comparisons.

B. Control of the Crane System

The overhead-traveling crane as shown in Fig. 13 is usually used for loading work at ironworks and various factories. In the operation of crane motion, the load is required to arrive at the target point accurately without any swing. In this section, we shall apply our MFCS to solve this multiobjective crane control problem. In this system, the only control action is f, which is the amount of force (in *Newtons*) applied to the trolley to move it. The model and the corresponding parameters of the crane system for our computer simulation are adopted from [35] with the consideration of friction effects. The dynamic equations of motion that we used are

$$\begin{aligned} R\ddot{\theta} + \ddot{D}\cos\theta + g\ \sin\theta &= 0, \\ (M+m)\ddot{D} + c_d\dot{D} - mR(\dot{\theta})^2 + mR\ddot{\theta}\ \cos\theta &= f \end{aligned} \quad (50)$$

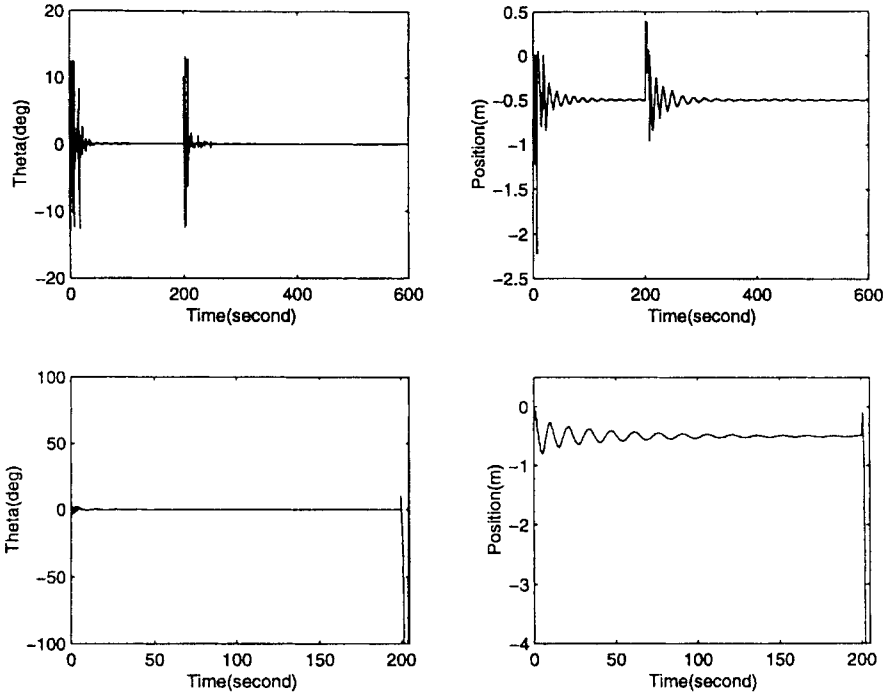


Fig. 11. Tests of adaptation ability of the RFCN (upper two figures) and the manually designed FCN (low two figures), where a big disturbance acts on the cart at the 200th second.

	Ours*	Ours	Lin and Lin [31]	Barto's original system [14]	Barto's system with continuous output [31]	Neural Networks	Fuzzy Neural Networks		
							Fuzzy rules must be given in advance		Lin and Lee [34]
						Anderson [32]	Lee and Berenji [33]	Berenji and Khedkar [10]	
Trials	5	11	15	35	40	8000	10	300	10
Rule number	10	10	10	162	162	—	189	13	35
Angular deviation	0.5°	0.5°	0.5°	4°	2.5°	—	1°	1°	1°

Ours*: Putting expert knowledge (fuzzy rules) into the RFCN in advance.

Fig. 12. Performance comparisons of the blank RFCN (Ours) and rule-embedded RFCN (Ours*) to other existing reinforcement learning schemes.

where

θ (in degrees)	vertical deviation;
D (in meters)	trolley position;
$g = 9.8$ m/s^2	acceleration due to the gravity;
$m = 5000$ kg	trolley-load mass;
$M = 5000$ kg	trolley mass;
$R = 5$ m	rope length;
$c_d = 500$ kg/s	coefficient of friction.

The constraints on the variables are $-30° < \theta < 30°$ and $-10^6 N \leq f \leq 10^6 N$.

For computer simulation, we reformulate the dynamic equations in (50) into state equations as follows:

$$x_1(t+1) = x_1(t) + \Delta\left\{\frac{x_2(t)}{R} - \frac{x_4(t)}{R}\cos x_1(t)\right\},$$

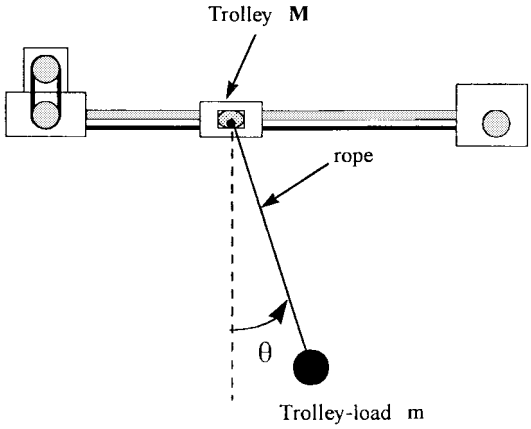


Fig. 13. Crane system.

$$
\begin{aligned}
x_2(t+1) &= x_2(t) + \Delta\Big\{-g\ \sin x_1(t) - \frac{1}{R}x_2(t)x_4(t) \\
&\quad \cdot \sin x_1(t) - \frac{1}{R}\ x_4^2(t)\ \sin x_1(t)\ \cos x_1(t)\Big\}, \\
x_3(t+1) &= x_3(t) + \Delta x_4(t), \\
x_4(t+1) &= x_4(t) + \frac{\Delta}{M+m-\dfrac{m\ \cos^2 x_1(t)}{(M+m)}} \\
&\quad \cdot \Big\{f - c_d x_4(t) + mR\Big[\frac{x_2(t)}{R} \\
&\quad - \frac{x_4(t)}{R}\ \cos x_1(t)\Big]^2 \\
&\quad \cdot \sin x_1(t) + mg\ \sin x_1(t)\ \cos x_1(t)\Big\}, \\
x_5(t+1) &= x_5(t) + \Delta[-(x_4(t+1) - x_4(t))\ \cos x_1(t) \\
&\quad - g\ \sin x_1(t)]/R
\end{aligned} \tag{51}
$$

where

x_1	θ;
x_2	$R\dot{\theta} + \dot{D}\ \cos\theta$;
x_3	D;
x_4	$\dot{D}$;
x_5	$\dot{\theta}$;
sampling interval Δ	0.02 s.

To apply the proposed MFCS to solve the above two-objective control problem, we define two control status signals, E_1 and E_2, according to the two control objectives as follows:

$$
\begin{aligned}
E_1 &\stackrel{\text{def}}{=} x_1(t)/30^\circ \\
E_2 &\stackrel{\text{def}}{=} (x_3(t) - x_{\text{desired}})/x_{\text{desired}}.
\end{aligned} \tag{52}
$$

The control status signal E_1 indicates the status of the first control objective, i.e., to keep the swing of the trolley-load as little as possible. The control status signal E_2 indicates the status of the second control objective, i.e., to push the trolley to a desired position, x_{desired}. The signal E_1 is normalized to be in $[-1,\ 1]$ and the signal E_2 is normalized to be in $[-1,\ 0]$. The E_i's value close to zero means the respective control objective is almost achieved. Notice that we normalize the signal E_2 to be in $[-1,\ 0]$ instead of $[-1,\ 1]$, since we do not want the trolley to go beyond and oscillate around the desired position at any time.

Before we train an NFC for the MFCS to control the two-objective crane system, we need to design two low-level controllers, one for each control objective. As we did in the last subsection, we used the trial-and-error design approach of fuzzy control [21] to design these two low-level controllers. The first low-level controller (angle controller) is to control the swinging angles of the trolley-load. The second low-level controller (position controller) is to determine a sequence of forces to move the trolley to a desired position as soon as possible regardless of the trolley-load's swing angle. We shall use an NFC with reinforcement learning to combine these two low-level controllers to achieve both control objectives of the crane system simultaneously.

We formulate the crane control problem as a reinforcement learning problem by defining two reinforcement signals, one for each control objective, as follows:

$$
r_1(t) = \begin{cases} -1, & \text{if } |x_1(t)| > 30^\circ \\ & \text{or } |x_1(t)| > 3^\circ \text{ over 1000} \\ & \quad \text{time steps} \\ 0, & \text{otherwise,} \end{cases} \tag{53}
$$

$$
r_2(t) = \begin{cases} -1, & \text{if } |x_3(t) - x_{\text{desired}}| > 0.3m \\ & \quad \text{over 1000 time steps} \\ 0, & \text{otherwise.} \end{cases} \tag{54}
$$

These reinforcement signals indicate that a failure occurs if the trolley-load swing angle exceeds $\pm 30^\circ$, the trolley-load swing angle is not stabilized to be within $\pm 3^\circ$ in 1000 time steps, or the trolley dose not reach the desired position in 1000 time steps. The last two conditions are to ensure that the trolley-load can arrive in the desired position stably (without swinging) in short time, where the number "1000 time steps" is chosen objectively in the simulations.

In the simulations, the learning system was tested for four runs. Each run consists of a sequence of trials; each trial begins with the same initial condition and ends with a failure signal appearing, i.e., $r_i(t) = -1$. A run consists of at most 30 trials, unless the duration of each run exceeds 30 000 time steps. In the latter case, we consider the run "successful." The following learning parameters were used for each trial. The learning rate $\eta = 0.005$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.9, \rho_{\text{output}} = 0.9$ were used for the action network. The learning rate $\eta = 0.01$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.8, \rho_{\text{output}} = 0.8$ were used for the critic network, where ρ_{input} was the vigilance parameter used in the input fuzzy clustering process and ρ_{output} was the vigilance parameter used in the output fuzzy clustering process.

The simulation results of the crane system, corresponding to Figs. 6 and 7 for the cart-pole system, are shown in Figs. 14 and 15. Fig. 14 indicates that the RFCN can achieve both control objectives simultaneously after several trials (six trials on average) for four different desired trolley positions. Fig. 15 shows the state status, x_1 and x_3, and the corresponding two-valued external reinforcement signals of a run. It is noted from (53) that there are two causes for $r_1 = -1$, i.e., $|x_1(t)| > 30^\circ$ or $|x_1(t)| > 3^\circ$ over 1000 time steps. Therefore, in Fig. 15, we use $r_{11}(t)$ to represent the reinforcement signal corresponding to the situation, $|x_1(t)| > 30^\circ$, and $r_{12}(t)$ to represent the reinforcement signal corresponding to the situation, $|x_1(t)| > 3^\circ$ over 1000 time steps.

VIII. Conclusions

This paper proposes a neural fuzzy combiner (NFC) to combine a set of well-designed low-level controllers into a multiobjective fuzzy control system (MFCS) for solving the control problems with multiple goals. Three approaches have been developed to obtain a proper NFC, a hierarchical design method, a supervised learning scheme, and a reinforcement learning scheme. The first one is based on expert knowledge, and the other two are based on neural learning techniques suitable for different learning environments. This paper extends the application domain of reinforcement learning from single-objective to multiobjective control by proposing the concept of

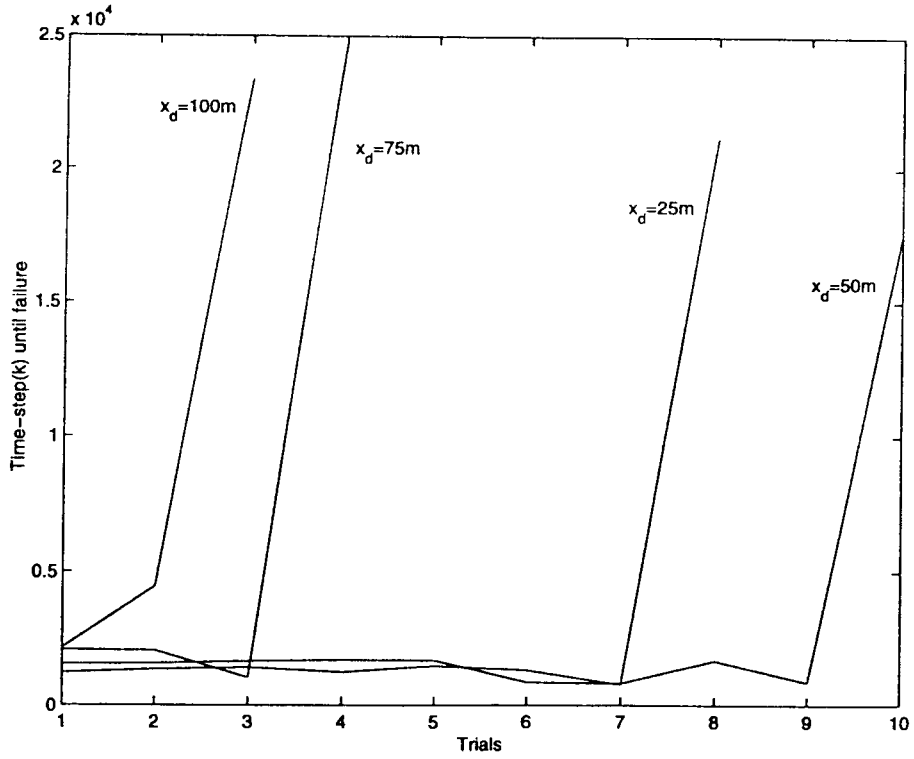


Fig. 14. Performance of the RFCN on the crane system.

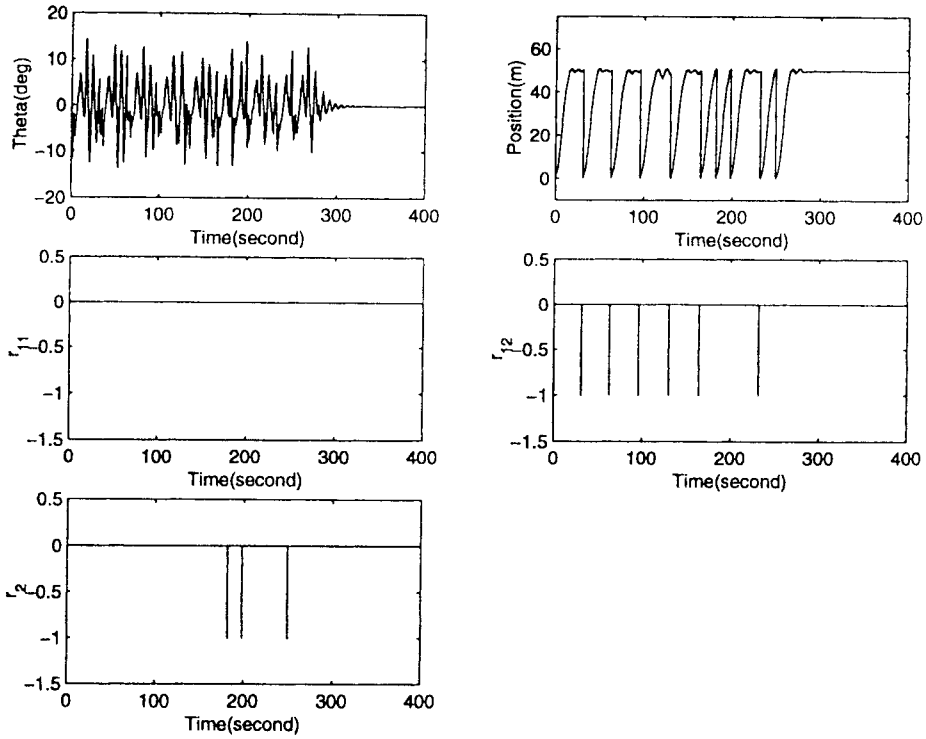


Fig. 15. Variations of trolley-load swing angle, trolley position, and reinforcement signals during the reinforcement learning process of the RFCN on the crane system.

mutual credit assignment and developing a novel competitive stochastic-exploration scheme. Computer simulations on the cart-pole balancing problem and the crane control problem have shown the applicabilities of the proposed structure and learning schemes. The basic idea of the proposed multiobjective control techniques is to decompose a multiobjective task into a set of single-objective tasks. While achieving such decomposition may be difficult, or even impossible,

there are useful classes of tasks where such decomposition is achievable. Future research will focus on identifying the classes of multiobjective tasks where the decomposition is achievable.

REFERENCES

[1] P. Bonissone, "Fuzzy logic controllers: An industrial reality," in *Proc. IEEE Computational Intelligence: Imitating Life*, 1994, pp. 316–327.

[2] H. R. Berenji, Y. Y. Chen, C. C. Lee, J. S. Jang, and S. Murugesan, "A hierarchical approach to designing approximate reasoning-based controllers for dynamic physical systems," in *Proc. 6th Conf. Uncertainty Artificial Intelligence*, 1990, vol. 8, no. 5, pp. 362–369.

[3] P. Bonissone and K. Chiang, "Fuzzy logic hierarchical controller for a recuperative turboshaft engine: From mode selection to mode melding," in *Industrial Applications of Fuzzy Control and Intelligent Systems*, J. Yen, R. Langari and L. Zadeh, Eds. New York: Van Nostrand Reinhold, 1994.

[4] M. Sugeno, T. Murofushi, J. Nishino and H. Miwa, "Helicopter flight control based on fuzzy logic," in *Proc. Int. Fuzzy Eng. Symp. '91(IFES'91)*, Nov. 1991, pp. 1120–1121.

[5] P. J. Werbos, "Neural control and fuzzy logic: Connections and designs," *Int. J. Approx. Reas.*, vol. 6, pp. 185–219, 1992.

[6] J. S. Jang, "Self-learning fuzzy controllers based on temporal back propagation," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 723–741, 1992.

[7] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[8] C. T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[9] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1414–1427, 1992.

[10] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 724–740, 1992.

[11] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Read-time learning and control using asynchronous dynamic programming," Tech. Rep. 91-57, Univ. Mass., Amherst, 1991.

[12] R. S. Sutton, "Integrating architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proc. 7th Int. Workshop Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1990, pp. 216–224.

[13] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: Macmillan, 1994.

[14] A. G. Barto, R. S. Sutton and C. W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 835–846, 1983.

[15] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, pp. 9–44, 1988.

[16] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins, "Sequential decision problems and neural networks," in *Advances in Neural Information Processing System 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufman, 1990.

[17] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computat.*, vol. 3, 1991.

[18] S. P. Singh, "Transfer of learning by composing solutions of elemental sequential tasks," *Mach. Learn.*, vol. 8, pp. 323–339, 1992.

[19] P. Maes and R. Brooks, "Learning to coordinate behaviors," in *Proc. 8th AAAI*. San Mateo, CA: Morgan Kaufman, 1990, pp. 796–802.

[20] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," Tech. Rep., IBM Res. Div., T. J. Watson Res, Center, Yorktown Heights, NY, 1990.

[21] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Parts I and II," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 404–435, 1990.

[22] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.

[23] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 698–712, 1992.

[24] P. K. Simpson, "Fuzzy min-max neural networks—Part 2: Clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 1, pp. 32–45, 1993.

[25] J. A. Franklin, "Input space representation for reinforcement learning control," in *Proc. IEEE Int. Symp. Intelligent Control*, 1989, pp. 115–122.

[26] R. J. Williams, "A class of gradient-estimating algorithms for reinforcement learning in neural networks," in *Proc. Int. Joint Conf. Neural Networks*, San Diego, CA, 1987, vol. 2, pp. 601–608.

[27] P. J. Werbos, "A menu of design for reinforcement learning over time," in *Neural Networks for Control*, W. T. Miller III, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1990.

[28] P. Dayan, "The convergence of TD(λ) for general λ," *Mach. Learn.*, vol. 8, pp. 341–362, 1992.

[29] P. Dayan and T. J. Sejnowski, "TD(λ) converges with probability 1," *Mach. Learn.*, vol. 14, pp. 295–301, 1994.

[30] N. Baba, "TD learning with neural networks," *J. Robot. Mechatron.*, vol. 10, no. 4, pp. 289–294, 1998.

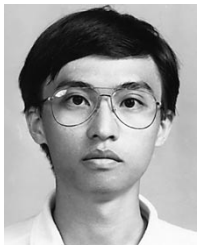
[31] C. J. Lin and C. T. Lin, "Reinforcement learning for an ART-based fuzzy adaptive learning control network," *IEEE Trans. Neural Networks*, vol. 7, pp. 709–731, May 1996.

[32] C. W. Anderson, "Strategy learning with multilayer connectionist representations," in *Proc. 4th Int. Workshop Machine Learning*, Irvine, CA, June 1987, pp. 103–114.

[33] C. C. Lee and H. R. Berenji, "An intelligent control based on approximate reasoning and reinforcement learning," in *Proc. IEEE Int. Symp. Intell. Contr.*, 1989, pp. 200–205.

[34] C. T. Lin and C. S. G. Lee, "Reinforcement structure/parameter learning for neural network-based fuzzy logic control systems," *IEEE Trans. Fuzzy Syst.*, vol. 2, pp. 46–63, Feb. 1994.

[35] M. Fliess, J. Lévine and P. Rouchon, "A simplified approach of crane control via a generalized state-space model," in *IEEE Proc. 30th Conf. Decision Control*, Brighton, U.K., Dec. 1991, pp. 736–741.



Chin-Teng Lin received the B.S. degree in control engineering from the National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1986, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1989 and 1992, respectively.

Since August 1992, he has been with the College of Electrical Engineering and Computer Science, National Chiao-Tung University, Hsinchu, where he is currently a Professor of Electrical and Control Engineering. He also serves as the Deputy Dean of the Research and Development Office of the National Chiao-Tung University since 1998. His current research interests are fuzzy systems, neural networks, intelligent control, human-machine interface, and video and audio processing. He is the co-author of *Neural Fuzzy Systems—A Neuro-Fuzzy Synergism to Intelligent Systems* (Englewood Cliffs, NJ: Prentice-Hall), and the author of *Neural Fuzzy Control Systems with Structure and Parameter Learning* (Singapore: World Scientific). He has published around 40 journal papers in the areas of neural networks and fuzzy systems.

Dr. Lin is a member of Tau Beta Pi and Eta Kappa Nu. He is also a member of the IEEE Computer Society, the IEEE Robotics and Automation Society, and the IEEE Systems, Man, Cybernetics Society. He has been the Executive Council Member of Chinese Fuzzy System Association (CFSA) since 1995, and the Supervisor of Chinese Automation Association since 1998. He was the Vice Chairman of the IEEE Robotics and Automation Society, Taipei Chapter, in 1996 and 1997. He won the Outstanding Research Award from National Science Council (NSC), Taiwan, in 1997 and 1999, and the Outstanding Electrical Engineering Professor Award granted by the Chinese Institute of Electrical Engineering (CIEE) in 1997.

I-Fang Chung received the B.S. and M.S. degrees in control engineering from the National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1993 and 1995, respectively. He is currently pursuing the Ph.D. degree in the department of electrical and control engineering at the same university.

His current research interests are neural networks, learning systems, fuzzy control, virtual reality, and dynamical simulation.