An Approach to Enlarge Learning Space Coverage for Robot Learning Control

Kuu-Young Young, *Member, IEEE*, and Shaw-Ji Shiah

***Abstract*— In robot learning control, the learning space for executing general motions of multijoint robot manipulators is quite large. Consequently, for most learning schemes, the learning controllers are used as subordinates to conventional controllers or the learning process needs to be repeated each time a new trajectory is encountered, although learning controllers are considered to be capable of generalization. In this paper, we propose an approach for larger learning space coverage in robot learning control. In this approach, a new structure for learning control is proposed to organize information storage via effective memory management. The proposed structure is motivated by the concept of human motor program and consists mainly of a fuzzy system and a cerebellar model articulation controller (CMAC)-type neural network. The fuzzy system is used for governing a number of sampled motions in a class of motions. The CMAC-type neural network is used to generalize the parameters of the fuzzy system, which are appropriate for the governing of the sampled motions, to deal with the whole class of motions. Under this design, in some sense the qualitative fuzzy rules in the fuzzy system are generalized by the CMAC-type neural network and then a larger learning space can be covered. Therefore, the learning effort is dramatically reduced in dealing with a wide range of robot motions, while the learning process is performed only once. Simulations emulating ball carrying under various conditions are presented to demonstrate the effectiveness of the proposed approach.**

***Index Terms*—CMAC-type neural network, fuzzy system, human motor program, learning space coverage, robot learning control.**

I. Introduction

THE dynamics of robot manipulators are, in general, nonlinear and complex. Therefore, conventional fixed-gain, linear feedback controllers are not capable of effectively controlling movements of multijoint robot manipulators under different distance, velocity, and load requirements. The nonlinear dynamic interaction in multijoint movements can be compensated for through the use of feedback. To achieve better compensation for dynamic interaction in approaches using feedback for conventional robot control (e.g., the computed torque method), a complete, nonlinear dynamic model describing the robot manipulator is needed [6]. The use of a complicated nonlinear dynamic model makes real-time implementation difficult [16]. Moreover, it is by no means an easy task to identify the model parameters accurately. On the other hand, if a learning controller is used, the consequences of using incomplete models and inaccurate model parameters may not be very significant. Because they are capable of tackling highly complex dynamics without explicit model dependence and identification, learning controllers are an attractive alternative in robot motion control [10], [24].

Two well-known types of learning controllers for robot motion control are artificial neural networks and fuzzy systems. Both are biologically inspired and intended to model human experience [25], [28]. The structure of artificial neural networks is modeled after the organization of the brain, although the similarity between the two is actually slight [25]. On the other hand, fuzzy systems are meant to encode pieces of knowledge presented by experts [13], [17], [24]. In some previous research involving the application of these two types of learning controllers, they are used to assist in the control of a robot manipulator, while a conventional control algorithm, e.g., PD or PID control, is responsible for the major portion of the control [14], [16], [18]. In this approach, the conventional control algorithm brings the system close to the desired state and the learning mechanism then compensates for the remaining error. On the other hand, some systems use learning algorithms alone to execute the control. However, although these learning controllers are considered to be capable of generalization, most of them need to repeat the learning process each time a new trajectory is encountered [11]. Otherwise, a neural network will consist of a huge number of neurons or a fuzzy system will require too many rules because the learning space needed to handle arbitrary trajectories is too large.

To sum up, one major problem in using learning controllers is that the learning space for executing general motions of multijoint robot manipulators is too large [20], [22]. In this paper, we propose an approach for larger learning space coverage in robot learning control. In this approach, a new structure for learning control is proposed to organize information storage via effective memory management. The proposed structure is motivated by the concept of human motor program and consists mainly of a fuzzy system and a cerebellor model articulation controller (CMAC)-type neural network. The fuzzy system is used for governing a number of sampled motions in a class of motions. To allow for automatic adjustment of the system's parameters, the fuzzy system will be implemented with the structure of a fuzzy-neural network [4], [15]. The CMAC-type neural network is used to generalize the parameters of the fuzzy system, which are appropriate for the governing of the sampled motions,

Manuscript received May 21, 1996; revised January 29, 1997. This work was supported in part by the National Science Council, Taiwan, under Grant NSC 83-0422-E-009-065.

The authors are with the Department of Control Engineering, National Chiao-Tung University, Hsinchu, 300 Taiwan.

Publisher Item Identifier S 1063-6706(97)04835-2.

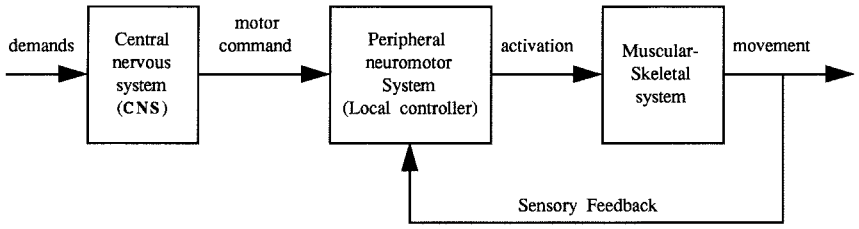


Fig. 1. A simplified human motor control block diagram.

to deal with the whole class of motions. One reason for adopting the CMAC-type neural network rather than another type of neural network is that its learning for certain motions will not affect that for other motions too much [1]. Thus, training patterns can be added or deleted easily according to the performance. In addition, this type of network has a strong generalization capability and simple structure. Under this design, in some sense the qualitative fuzzy rules in the fuzzy system are generalized by the CMAC-type neural network, and then a larger learning space can be covered. Therefore, the learning effort is dramatically reduced in dealing with a wide range of robot motions, while the learning process is performed only once. Biological backgrounds of human motor program are discussed in Section II. The proposed approach and a robot learning control scheme based on it are described in Section III. Simulations emulating ball carrying with various distances, velocities and loads that demonstrate the effectiveness of the proposed approach are described in Section IV. Finally, discussions and conclusions are stated in Section V.

II. Human Motor Program

The proposed approach is motivated by the concept of human motor program, which is derived from the study of human movement and has stimulated research in human control strategies [3], [5], [7], [12], [19], [21], [26]. Fig. 1 shows a simplified human motor control block diagram, which governs human limb movements. In Fig. 1, we see that human movement is governed by a hierarchical structure [11], [21]. According to different demands, the central nervous system (CNS) makes movement plans. Appropriate motor commands are then generated and sent to the peripheral neuromotor system, which modifies the motor commands via sensory feedback. The peripheral neuromotor system behaves as a local controller that adapts to different movements, loads, and environments in addition to accepting commands from the CNS [8]. Finally, the modified commands are sent to the muscular-skeletal system for movement execution. With this hierarchical structure, the difficulty of performing complex movements can be shared by the CNS at the higher level and the local controller at the lower level.

Because feedback processing in the human motor control system is slow, long delays are experienced in the transfer of sensory information to the higher level of the hierarchy. For slow movement, long delays may cause no serious problems when feedback control is employed by the higher level of the hierarchy; however, the effect of delays cannot be ignored in dealing with fast movement. Therefore, open-loop control might be more appropriate for governing fast human movement and a concept of motor program was proposed [19], [21]. The basic idea in human motor program is that movement demands are specified by the CNS in advance and then executed in an essentially open-loop manner. When activated, the motor program generates motor commands and sends them to the peripheral neuromotor system for execution. Where a class of movements is concerned, the storage of complex motor programs for each movement may not be appropriate. Instead, the motor program should be generalized, simple to operate, and efficient in storage. Thus, a single motor program will be enough to accommodate a wide variety of movements. Proper parameters corresponding to a particular movement can then be supplied to the motor program in execution. In other words, the motor program should be *abstract* and the corresponding control parameters should be efficiently stored and manipulated [21]. Although there is still doubt concerning whether the control strategy employed by the CNS is in fact open-loop and it is not yet clear which parameters of movement motor programs control [19], the concept of human motor programs is attractive.

III. Proposed Approach

In the proposed approach, we take advantage of the merits of a fuzzy system and a CMAC-type neural network to emulate a motor program with the appealing characteristics described above. A fuzzy system is used to represent the abstract motor program and a CMAC-type neural network is used to manage the parameters. Through fuzzy logic and representation, the fuzzy system manipulates information in a linguistic way. In a sense, the fuzzy system encodes knowledge via qualitative rules rather than via precise quantitative description. This feature makes the fuzzy system an appealing choice for emulating a motor program. The parameters specifying the fuzzy rules for governing sampled motions are stored and manipulated by the CMAC-type neural network to deal with a wide range of motions. The combination of a fuzzy system and a neural network in this way is novel and exploits the merits of the motor program.

In order to simplify the complexity in learning, we also classify robot motions according to their features in advance. As an example, a group of arbitrary robot motions can be categorized into a class of motions of various movement distances with the same movement velocity and load or a class of motions of various movement velocities with the same movement distance and load, etc. Under this arrangement, a class of motions with

the same feature are expected to correspond to similar fuzzy parameters. Consequently, the data with which the CMAC-type neural network, which executes the generalization, will have to deal will exhibit less nonlinearity. Further discussions on motion feature selection will be given in Section V. Parameters of the fuzzy system for various classes of motions can also be incorporated into the same neural network at the expense of greater memory requirements. For instance, the neural network can store parameters corresponding to a group of motions which reach different destinations with different velocities and loads. Naturally, a fuzzy system is not the only choice for representing a motor program. For instance, the gains of a conventional PD or PID controller or the torques to move the robot links can also be generalized by a neural network to govern various robot motions. However, we believe that generalization of qualitative fuzzy rules is more effective than that of quantitative numerical data, because the former involves the generalization of abstract representations and tends to cover a larger learning space. In Section III-E, discussions are given for comparing the generalization capability based on using qualitative fuzzy rules and quantitative numerical data.

A. Robot Learning Control Scheme

At the current stage of the study, we do not intend to develop a learning control scheme that can govern general motions of general multijoint robot manipulators. Instead, we will concentrate on demonstrating the effect of the combination of the fuzzy system and the CMAC-type neural network in our proposed approach for learning space coverage. For this purpose, a robot learning control scheme based on the proposed approach is proposed and its conceptual organization shown schematically in Fig. 2. In Fig. 2, the inputs to the system are motions from classes of motions with the same features. For each motion from a class of motions, its trajectory descriptor, e.g., movement distance, velocity, load, or other features, is used as an index and inputted into the CMAC-type neural network (CMAC). A set of parameters will then be solicited from the CMAC and sent forward to the fuzzy system. With the set of parameters representing the fuzzy rules, the fuzzy system can govern the motion, along with the desired position and velocity trajectories of the motion, which are also inputted to the fuzzy system. The fuzzy system is implemented in the form of a fuzzy neural network (FNN) and generates motion commands sent forward to a local controller. In turn, the local controller, which emulates the peripheral neuromotor system shown in Fig. 1, modulates the motion command via sensory feedback and uses the resultant signal to move the robot manipulator [27]. According to some biological evidence, human motor program may provide only the desired position for movement control [19]. Therefore, to simplify the design of the learning scheme, only the desired position and no desired velocity is specified in the motion command. A simple position control law with linear damping is then adopted for the local controller [23]

$$\tau = \mathbf{K_p}(\mathbf{C_m} - \mathbf{q}) - \mathbf{K_d}\dot{\mathbf{q}} \tag{1}$$

where $\mathbf{C_m}$ stands for the motion command vector, $\mathbf{q}$ and $\dot{\mathbf{q}}$ are

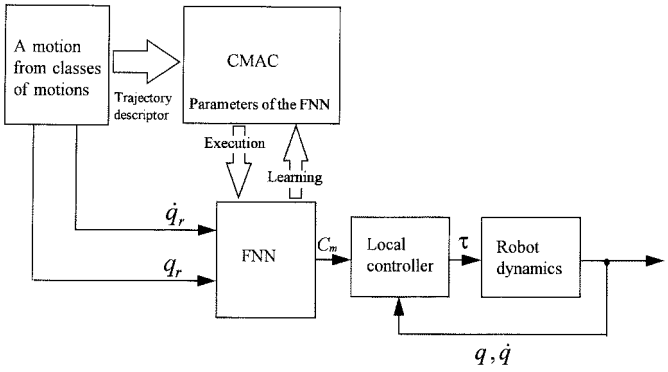


Fig. 2. Conceptual organization of a robot learning control scheme.

the actual position and velocity vectors obtained from sensory feedback, and $\mathbf{K_p}$ and $\mathbf{K_d}$ are symmetric positive definite gain matrices for stability considerations [6].

In the proposed scheme, in some sense, the intelligence for motion governing is mainly attributed to the FNN and CMAC, and the local controller deals with the sensory feedback only. Thus, the success of the proposed scheme depends on designing the FNN and CMAC to generate proper motion commands for various motions in the classes. Proper learning processes are also imperative for determining the weights in the networks. The structures and learning processes for the FNN and CMAC will be discussed in Sections III-B and C, respectively. Two stages of learning, the first involving the FNN and the second the CMAC, are needed for the system to learn to govern classes of motions. In the first stage, proper parameters of the FNN will be learned for sampled motions. These parameters will then be used as training patterns to set up the CMAC in the second stage of learning. Later, after learning the CMAC will supply proper parameters to the FNN for the governing of input desired motions. The interaction between the FNN and the CMAC in learning classes of motions is described in Section III-D.

B. Implementation of the FNN

The fuzzy system is implemented in the form of a fuzzy-neural network, as shown in Fig. 3. An FNN is designed to realize the process of fuzzy reasoning using the structure of a neural network. The parameters of fuzzy reasoning are expressed by the connection weights or node functions of the neural network [4], [15]. The representation of a fuzzy system using a fuzzy-neural network enables us to take advantage of the learning capability of the neural network for automatic tuning of the parameters in the fuzzy system. In Fig. 3, the inputs to the FNN are the position and velocity trajectories of a sampled motion and the output is the motion command. We chose an FNN with a structure similar to that in [4]. The reason for this choice is that the numbers of fuzzy rules and membership functions for input and output are prespecified in this type of FNN. Thus, for various motions there are the same number of fuzzy parameters with the same attributes. Consequently, these fuzzy parameters are appropriate to be generalized by the neural network to deal with a class of motions. On the other hand, an FNN with a variable structure

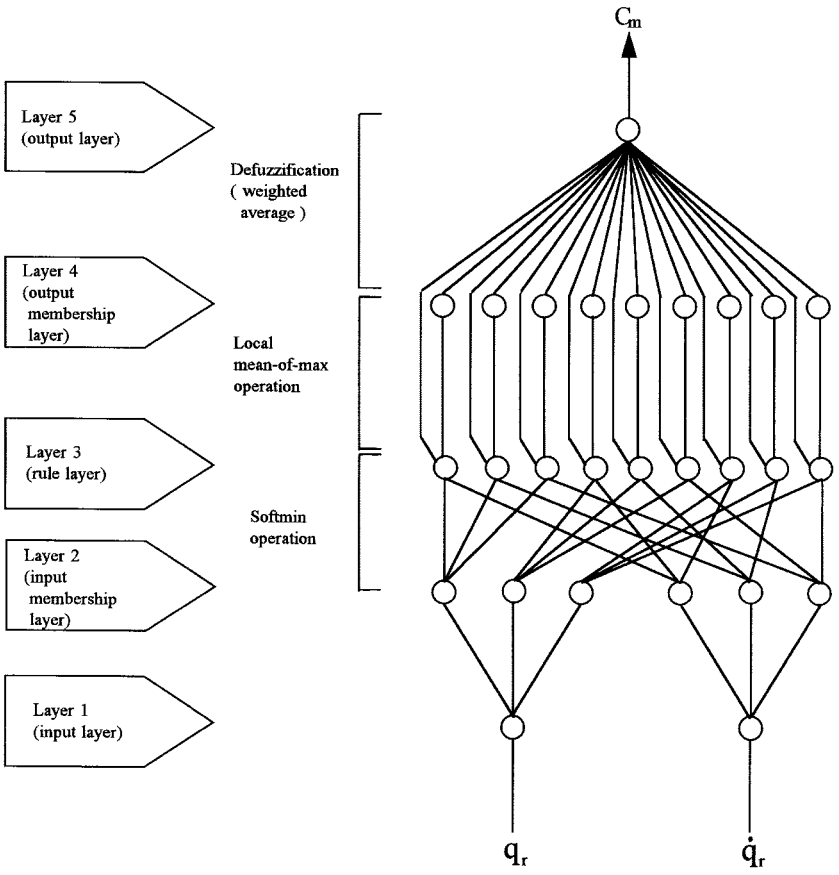


Fig. 3. The structure of the FNN.

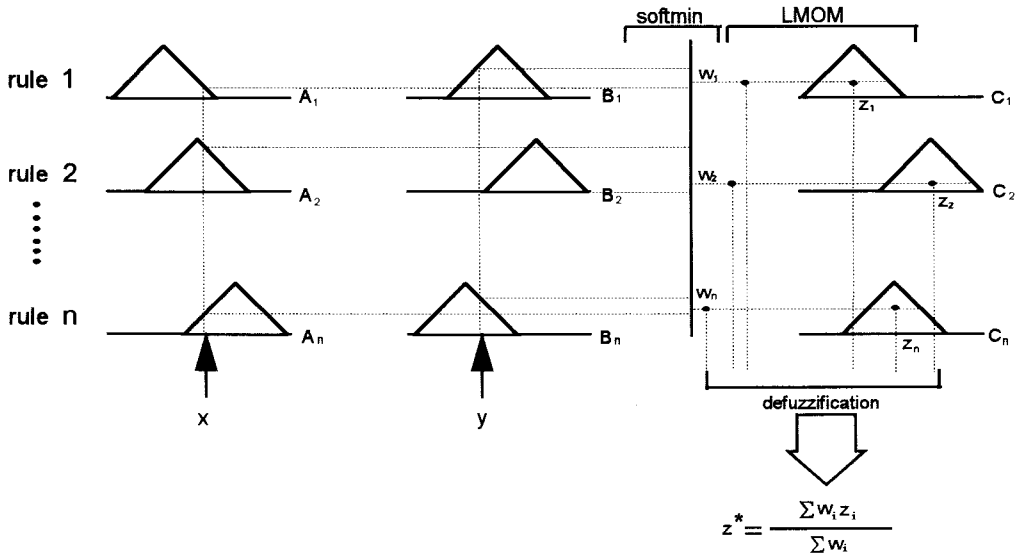


Fig. 4. The fuzzy inference process in the FNN.

(such as that in [15]), is flexible and effective in learning. However, in this type of FNN, the number of fuzzy parameters may vary with different robot motions. This feature makes this type of FNN unsuitable for the present application in which the fuzzy parameters must be generalized.

The structure of the FNN adopted here consists of five layers of nodes, which are of the same type within the same layer. The fuzzy inference process is as shown in Fig. 4. Each of the five layers performs one stage of the fuzzy inference process, as described below.

Layer 1: This layer is the input layer, and inputs are transmitted to the next layer directly without any computation. In Fig. 3, there are two nodes for two inputs q_r and $\dot{q}_r$ of motions of a single degree-of-freedom.

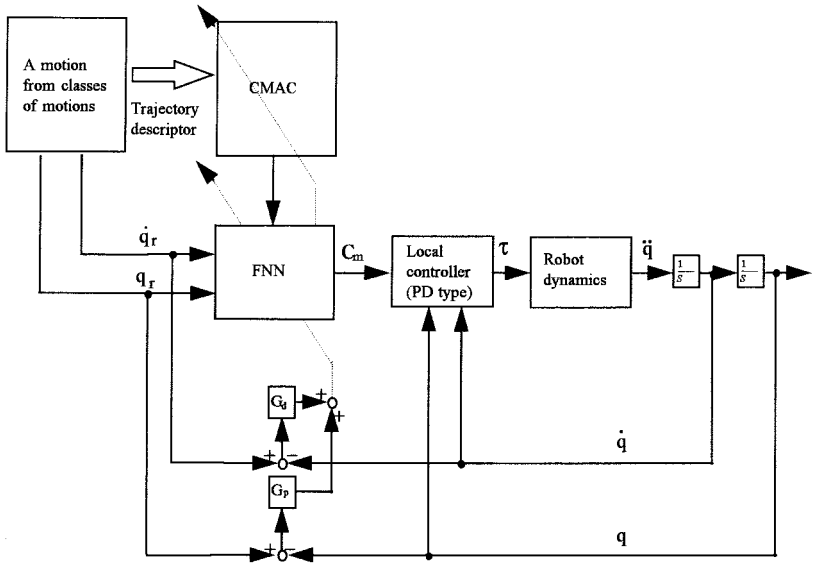


Fig. 5. Learning processes for the proposed approach.

Layer 2: This layer is intended for the input membership functions. Each node i in this layer has a node function

$$O_i^2 = \mu(x) \tag{2}$$

where μ: $X \rightarrow [0, 1]$ is a membership function and x is the input to node i. The triangular membership function is adopted, as described below.

$$\mu(x) = \begin{cases} 1 - \dfrac{(x-b)}{c}, & x \in [b, b+c] \\ 1 + \dfrac{(x-b)}{a}, & x \in [b-a, b] \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Different membership grades at the same crisp point can be obtained by adjusting the parameter set (a, b, c).

Layer 3: This layer is intended for the implementation of the fuzzy rules. Each node in this layer corresponds to a rule, which is defined as a fuzzy conditional statement of the form

$$\text{Rule: IF } X \text{ is } A \text{ and } Y \text{ is } B \text{ THEN } Z \text{ is } C \tag{4}$$

where X and Y are fuzzy sets representing the inputs, Z represents the output, and A, B, and C represent linguistic variables, such as small, medium, and large. The number of rules involved in the input–output relation in our design is prespecified. In this layer, each node also outputs the firing strength of the rule O_i^3 by performing a differentiable softmin operation [4]

$$O_i^3 = \frac{\sum_j O_j^2 \exp(-kO_j^2)}{\sum_j \exp(-kO_j^2)} \tag{5}$$

where O_j^2 is the output of the jth node in Layer 2, connected to the ith node in Layer 3, and k is a constant. When k approaches infinity, the softmin operator becomes a min operator; for finite k, O_i^3 is differentiable, which is required in the learning process.

Layer 4: This layer is intended for the output membership functions. Each node i in this layer performs an inverse of μ_i to locate the X-coordinate of the centroid of the membership function O_i^4 by using the local mean-of-maximum method (LMOM) [4]

$$O_i^4 = \mu_i^{-1}(O_i^3). \tag{6}$$

Both O_i^3 and O_i^4 will be needed in the defuzzification process performed by Layer 5. For the triangular output membership function adopted, the LMOM gives

$$\mu_i^{-1}(O_i^3) = b + \tfrac{1}{2}(c-a)(1-O_i^3) \tag{7}$$

which yields the median of the triangular membership function on the X axis.

Layer 5: This layer is the output layer, which has as many nodes as there are output action variables. In Fig. 3, only one node is needed for a single motion command C_m. The defuzzification approach adopted is the weighted averaging method [13]

$$O^5 = \frac{\sum_i O_i^3 O_i^4}{\sum_i O_i^3}. \tag{8}$$

Because the number of rules in Layer 3 is prespecified and weights for the input and output layers (Layers 1 and 5) are fixed, the parameters to learn in this FNN are the modifiable weights present on the input links to Layers 2 and 4, which correspond to the input and output membership functions. When the FNN learns the parameters of the input and output membership functions for generating the motion command C_m corresponding to a sampled motion, an error rate, related to the motion command C_m and the resultant motion, is first specified in the last layer (Layer 5). This error rate is then backpropagated to adjust the parameters from layer to layer sequentially. Because a concise form of the inverse dynamic model of the robot manipulator is not available, the error rate

cannot be obtained directly by differentiating the error between the desired motion and the actual motion relative to the motion command. Instead, as indicated in the learning process shown in Fig. 5, we use the combined feedback error of position (e) and velocity ($\dot{e}$) between the desired and actual motions, denoted as $E = G_p e + G_d \dot{e}$ to derive the error rate $\partial E/\partial C_m$ [2], [11]

$$\frac{\partial E}{\partial C_m} = \frac{\partial E}{\partial O^5} \simeq \eta(G_p e + G_d \dot{e}) \tag{9}$$

where η is a learning rate and G_p and G_d are gains that result in a stable system. The error rate $\partial E/\partial C_m$ in (9) is estimated, but not exact, for describing the differential relationship between the motion command C_m and the resultant motion. Nevertheless, the results in [2], [11] (and also ours) show that the use of this error rate is appropriate for the learning. With the error rate in (9) and some straightforward manipulation, we can derive updates of the parameters in Layers 2 and 4. In the training stage, the parameters are obtained and sent to the CMAC. After the CMAC is set up, the fuzzy parameters will be supplied from the CMAC to the FNN. Details of the learning processes will be discussed in Section III-D.

C. Implementation of the CMAC

The CMAC-type neural network has been successfully used in learning control functions [1]. It has also been applied to learn robot dynamics for controlling robot tasks [16]. The CMAC is a trainable discrete linear network pattern classifier that emulates the behavior of human beings in dealing with stimuli and responses. The CMAC computes control functions by referring to a table rather than by solving analytic equations. Function values are stored in a distributed fashion such that the value of a function at any point in input space is derived by summing the contents over a number of memory locations. A unique feature of the CMAC is a mapping algorithm which converts the distance between input vectors into the degree of overlap between sets of data where the function values are stored. Thus, the CMAC serves our purposes well, because mappings can be provided to generate proper parameters for a whole class of motions based on the finite sets of parameters for the sampled motions used as training patterns.

The basic concept behind the CMAC can be represented by a pair of mappings (as shown in Fig. 6) [1]

$$\text{f:}\quad \mathbf{S} \rightarrow \mathbf{A} \tag{10}$$

$$\text{g:}\quad \mathbf{A} \rightarrow \mathbf{P} \tag{11}$$

where $\mathbf{S}$ represents the set of input vectors, $\mathbf{A}$ represents the set of association cell vectors, and $\mathbf{P}$ represents the set of response output vectors. The first mapping maps the input data onto a finite set of intermediate states called association cells. The mapping is generally a fixed relation since it is a process of indexing the input data. The number of units of $\mathbf{A}$ that become excited in response to both of two different inputs S_i and S_j decreases monotonically as the similarity between S_i and S_j decreases. This arrangement produces generalization between nearby input vectors and no generalization between distant input vectors. In order to reduce the memory size of association cells for a practical mapping, Albus [1] proposed a procedure for hash-coding, which is basically a uniform random mapping from a larger memory to a smaller one. The second mapping depends on the values of weights, assigned to every association cell, which will be modified during the training stage. These weights can be adjusted by the difference between the desired output and the produced output. This mapping then sums up the weights attached to the active association cells to produce the output $\mathbf{P}$.

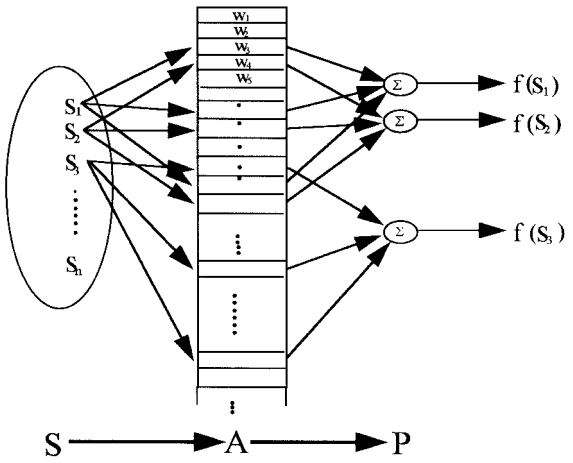


Fig. 6. Basic CMAC mappings.

To demonstrate how to store the sets of parameters corresponding to a class of motions into the CMAC, we use a group of motions of different movement distances with different velocities and loads as an example. The input vector $\underline{s}$ will contain three elements (d, v, m) for indexing these motions:

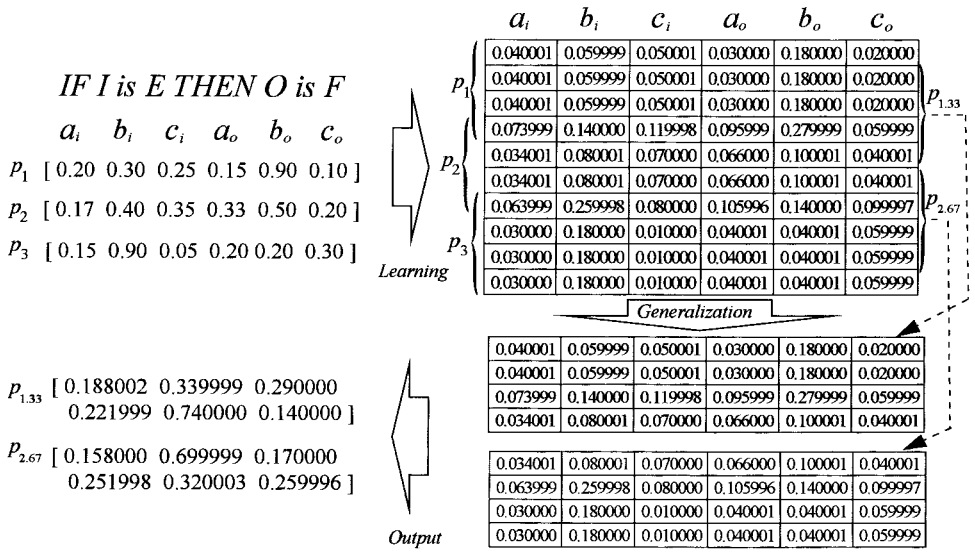
$$\underline{s} = (d, v, m) \tag{12}$$

where d stands for the distance, v for the velocity, and m for the load. Via the mappings of the network, the input vector should correspond to an output response consisting of a desired set of parameters. Thus, we need to find appropriate weights to attach to the active association cells in $\mathbf{A}$ by utilizing the sets of parameters for the sampled motions as training patterns. However, when the training patterns are stored into the network, the set of parameters generated by the CMAC may be different from the set of desired parameters because of memory overlapping in the association cells in the first mapping from the input vector to the association cells in (10). Thus, a learning process is needed to modify the weights through an updating function using the difference between the set of desired parameters and that generated by the CMAC for each training pattern. The updating function is as follows:

$$\underline{w}_{k+1} = \underline{w}_k + \beta\left(\frac{\underline{p} - \underline{p}'}{c}\right) \tag{13}$$

where k denotes the stage of the training process, β is the learning rate, c is the number of weights contributing to the output, $\underline{w}_k$ and $\underline{w}_{k+1}$ stand for vectors of weights before and after the kth stage of learning, respectively, and $\underline{p}$ and $\underline{p}'$

TABLE I
AN EXAMPLE OF HOW THE FUZZY PARAMETERS IN THE FNN ARE GENERALIZED BY THE CMAC



stand for desired and actual output vectors, respectively. The learning process will terminate when the difference between the sets of parameters generated by the network and the desired sets of parameters for the sampled motions are within a certain tolerance ϵ_c.

With the FNN and CMAC implemented, Table I shows an example of how the fuzzy parameters in the FNN are generalized by the CMAC. In Table I, there is one fuzzy rule in the form: *IF I is E THEN O is F*, where I and O are fuzzy sets representing the input and output, respectively, and E and F represent linguistic variables. Assume that I and O correspond to only one membership function with (a_i, b_i, c_i) and (a_o, b_o, c_o), described in (3), standing for the parameters for the input and output triangular membership functions, respectively. In Table I, three sets of parameters (P_1, P_2, P_3) for the fuzzy rule are obtained from three sampled motions, and sent for generalization by the CMAC. In the CMAC, four memory locations are used to store one fuzzy parameter with one memory location overlapping between adjacent fuzzy parameters. After successful learning, the CMAC can generate sets of parameters within the range expanded by (P_1, P_2, P_3), such as the outputs $P_{1.33}$ and $P_{2.67}$ shown in Table I.

D. Learning to Govern Classes of Motions

The process of learning to govern classes of motions, shown in Fig. 5, can be divided into two stages, one involving the FNN and the other the CMAC. In the first stage of learning, sampled motions will be selected from a class of motions and sent to the FNN. Here, we assume that stability and convergence of the FNN in learning to govern each sampled motion are guaranteed, and these issues are well dealt with in previous research [4], [15]. Proper parameters of the FNN will then be obtained for each of the sampled motions via the learning process described in Section III-B. In the second stage of learning, these parameters will be used as training patterns to set up the CMAC following the procedure described in Section III-C. After these two stages of learning, fuzzy parameters will be manipulated and supplied by the CMAC to the FNN for the governing of input desired motions. The learning in the FNN and the CMAC is, in fact, interactive. When the fuzzy parameters generated by the CMAC are not appropriate for governing a particular motion, the FNN can provide more training patterns by learning to govern more sampled motions around that motion to increase the resolution of the CMAC. In other words, the FNN and the CMAC will cooperate to yield homogeneous reliable performance for a whole class of motions.

In the application of the proposed scheme to the control of multijoint robot manipulators, input motions may be specified in Cartesian or joint space. If the input motion is specified in Cartesian space, the input positions $\mathbf{q_r}$ and velocities $\dot{\mathbf{q}}_\mathbf{r}$ for an n-joint robot manipulator will be mapped to corresponding joint positions $q_{j1}, \cdots, q_{jn}$ and joint velocities $\dot{q}_{j1}, \cdots, \dot{q}_{jn}$ via an inverse kinematic transformation; if the input motion is specified in joint space, this transformation will be unnecessary. The learning process will be performed only once to deal with classes of robot motions. The algorithm for learning to govern classes of motions is as follows.

Algorithm for Learning to Govern Classes of Motions: Learn to govern classes of motions for multijoint robot manipulators using both the FNN and CMAC.

Step 1) Initialize both the FNN and CMAC by choosing initial values of parameters (a, b, c) for the input and output membership functions in the FNN and by setting all the weights in the CMAC to zero.

Step 2) Select sampled motions from a class of motions and use the FNN to learn to govern the sampled motions. Proceed with the learning process until errors are within the tolerance ϵ_f following the procedure described in Section III-B. Send the parameters derived for the input and output membership functions to the CMAC.

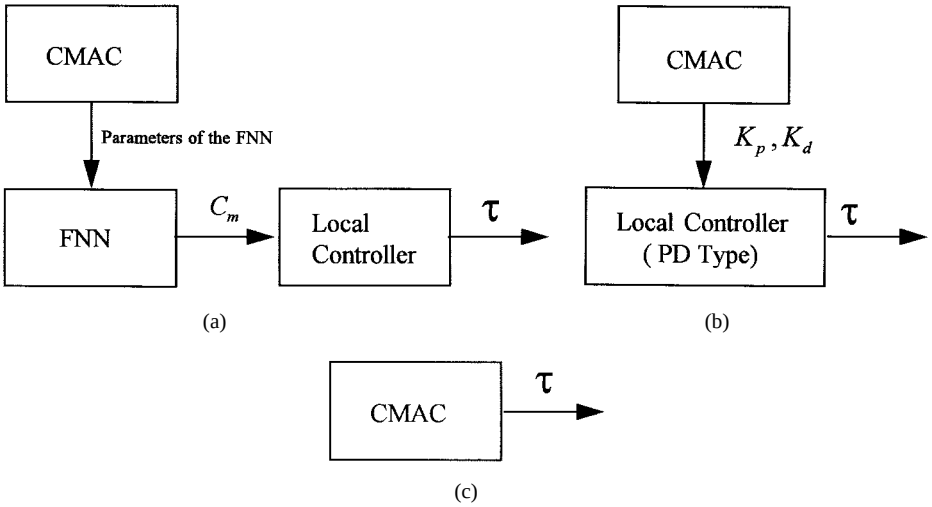


Fig. 7. The block diagram for generalization capability comparison. (a) Fuzzy rule. (b) Controller gain. (c) Torque.

Step 3) Use the parameters derived in Step 2) as training patterns for the learning in the CMAC. Proceed with the learning process until errors are within the tolerance ϵ_c following the procedure described in Section III-C.

Step 4) If the parameters generated by the CMAC fail to provide acceptable performance for an arbitrary motion in the class of motions, go back to Step 2 for learning to govern more sampled motions around that motion.

E. Generalization Capability Comparison

In this section, the generalization capability based on using qualitative fuzzy rules and quantitative numerical data will be investigated. In Fig. 7, we show the block diagram for three kinds of generalization: fuzzy rule, controller gain, and torque. Intuitively, generalization in the higher level of a control system is less sensitive to system variations. In [9], Jung and Hsia demonstrate via simulations that in the application of the neural network for nonlinearity compensation in robot control, significant performance improvements can be obtained when the compensation is performed at the command level instead of at the torque level. Take the block diagram for torque generalization in Fig. 7(c) for example. In Fig. 7(c), the CMAC generates torques to govern the robot manipulator directly without the modulation of the local controller. Thus, the CMAC may demand very high network resolution to provide precise torque commands to deal with variations in robot dynamics; when a small deviation in torque occurs, the system performance will be seriously affected. This phenomenon, however, is less evident in the higher level generalization, because the system, in some sense, is more robust.

Simulations were performed for generalization capability comparison based on using a two-joint robot manipulator, shown in Fig. 8. The performances of the fuzzy rule and torque generalization in Fig. 7(a) and (c) were evaluated. We consider the controller gain generalization in Fig. 7(b) is not suitable for the generalization purpose because it is not that intuitive to choose proper sets of K_p and K_d for generalization among a number of various combinations of K_p and K_d sets that perform nicely in governing a group of motions. On the other hand, the qualitative feature of the fuzzy rule makes it easier to distinguish a linguistic variable *small* from *large*. In the simulations, first a set of sampled motions were chosen spreading evenly in the robot workspace, and the system in Fig. 7(a) without the CMAC was then used to govern these sampled motions. After successful governing of the sampled motions was achieved via proper learning, the fuzzy parameters of the FNN and the generated torques from the local controller were sent for fuzzy rule and torque generalization in Fig. 7(a) and (c), respectively. Thus, the same accuracy was reached in governing the sampled motions by applying both schemes in Fig. 7(a) and (c). The same CMAC neural network is employed to generalize the fuzzy parameters and torques for these two kinds of generalization and some test motions, different from the sampled motions, were used to evaluate their generalization capabilities. The results show that fuzzy rule generalization performed much better than torque generalization in governing the test motions and torque generalization even resulted in quite large tracking errors for some test motions. It indicates that the generalization of the qualitative fuzzy rules is more effective than that of the quantitative numerical data.

IV. Simulation

To demonstrate the effectiveness of the proposed approach, simulations emulating ball carrying were performed. Fig. 8(a) shows the setup for this ball carrying simulation and a two-joint robot manipulator is used to carry balls of various weights to baskets of various heights with various velocities. The dynamic equations for this two-joint planar manipulator, as shown in Fig. 8(b), are expressed as follows:

$$\tau_1 = H_{11}\ddot{\theta}_1 + H_{12}\ddot{\theta}_2 - H\dot{\theta}_2^2 - 2H\dot{\theta}_1\dot{\theta}_2 + G_1 \quad (14)$$

$$\tau_2 = H_{21}\ddot{\theta}_1 + H_{22}\ddot{\theta}_2 + H\dot{\theta}_1^2 + G_2 \quad (15)$$

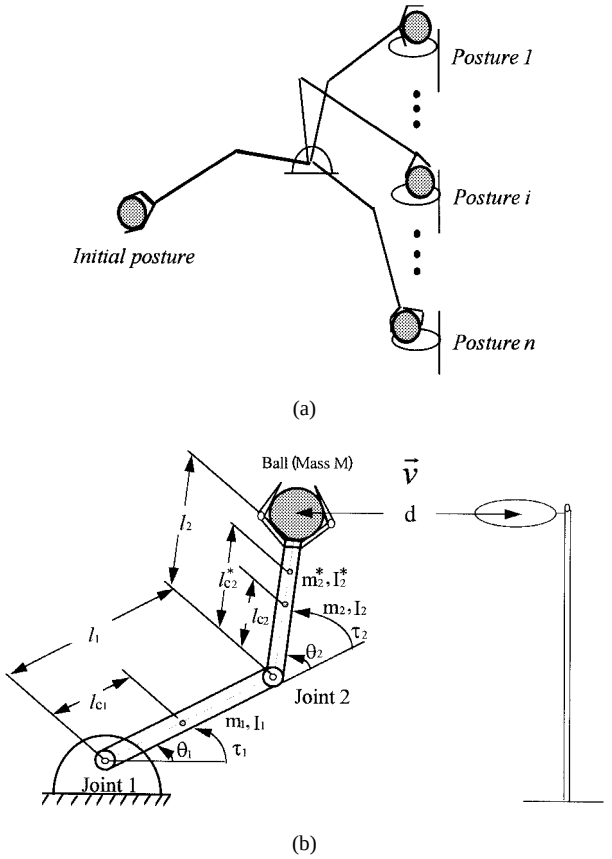


Fig. 8. (a) The setup for ball carrying simulation. (b) The two-joint robot manipulator.

where

$$H_{11} = m_1 l_{c1}^2 + I_1 + m_2^*[l_1^2 + l_{c2}^{*2} + 2l_1 l_{c2}^* \cos(\theta_2)] + I_2^* \quad (16)$$

$$H_{22} = m_2^* l_{c2}^{*2} + I_2^* \quad (17)$$

$$H_{12} = m_2^* l_1 l_{c2}^* \cos(\theta_2) + m_2^* l_{c2}^{*2} + I_2^* \quad (18)$$

$$H_{21} = H_{12} \quad (19)$$

$$H = m_2^* l_1 l_{c2}^* \sin(\theta_2) \quad (20)$$

$$G_1 = m_1 l_{c1} g \cos(\theta_1) + m_2^* g[l_{c2}^* \cos(\theta_1 + \theta_2) + l_1 \cos(\theta_1)] \quad (21)$$

$$G_2 = m_2^* l_{c2}^* g \cos(\theta_1 + \theta_2) \quad (22)$$

and

$$m_2^* = m_2 + M \quad (23)$$

$$l_{c2}^* = \frac{m_2 l_{c2} + M l_2}{m_2^*} \quad (24)$$

$$I_2^* = I_2 + m_2(l_{c2}^* - l_{c2})^2 + M(l_2 - l_{c2}^*)^2 \quad (25)$$

with $m_1 = 2.815$ kg, $m_2 = 1.640$ kg, $l_1 = 0.30$ m, $l_2 = 0.32$ m, $l_{c1} = 0.15$ m, $l_{c2} = 0.16$ m, and $I_1 = I_2 = 0.0234$ kgm^2. The effect of gravity was ignored for simplicity in the simulation. The inclusion of the gravity term will not affect the simulation results much and its effect is usually compensated for via feedforward or other means in robot control.

The motions needed to perform these different ball carrying tasks are grouped into classes of motions according to movement distance, velocity, and load. The ranges for these indexes specifying the motions are distance $\in [0.2, 1.2]$ (m), velocity $\in [0.5, 3]$ (m/s), and load $\in [0.5, 5]$ (kg). Because the motions are specified in Cartesian space, they are mapped into joint space via an inverse kinematic transformation. There are four inputs [one position (q_r) and one velocity ($\dot{q}_r$) for each joint] and two outputs for two motion commands (C_m) in the FNN. For each joint, we assign three linguistic grades for each of the two input variables, q_r and $\dot{q}_r$, and the corresponding fuzzy rules are formulated in the form of fuzzy conditional statements

$$\text{Rule: IF } q_r \text{ is } A_i \text{ and } \dot{q}_r \text{ is } B_j \text{ THEN } C_m \text{ is } C_k \quad (26)$$

where $i, j = 1 \cdots 3$ and $k = 1 \cdots 9$, and A_i, B_j, and C_k represent linguistic variables. Therefore, for each joint in the FNN, there are two nodes in Layer 1, six nodes in Layer 2, nine nodes in Layers 3 and 4, and one node in Layer 5. Consequently, altogether 90 parameters for two joints are sent to the CMAC for learning, because three parameters are needed in each of the six input membership functions and nine output membership functions. The learning rate η and the tolerance ϵ_f were chosen to be 0.005 and 0.001, and gains in (9) chosen to $G_p = 10$ nt·m/rad and $G_d = 50$ nt·m/(rad/s), respectively. In general, several hundred learning trials were used for learning to govern one sampled motion. For each of the 90 parameters sent from the FNN, in the CMAC the sensory layer for input vectors (d, v, m) and the association layer for association cell vectors include 40 and 2000 cells, respectively. The number of weights contributing to the output vector was chosen to be $\underline{c} = (5, 5, 5)$ for (d, v, m). Thus, each output vector (C_m) in the response layer connects $125 = 5 \times 5 \times 5$ cells. The hash-coding technique was not used in the mapping. The learning rate β and the tolerance ϵ_c were chosen to be 0.01 and 0.001, respectively. The gains of the local controller in (1) were set to $K_p = 20$ and 2 nt·m/rad and $K_d = 8$ and 1 nt·m/(rad/s) for joints one and two, respectively.

Fig. 9 shows the simulation results under: 1) different distance requirements; 2) different velocity requirements; 3) different load requirements; and 4) different distance, velocity, and load requirements. In Fig. 9(a), motions of four different movement distances with the same movement velocity and load were simulated. The reference motions in the upper-left box are desired motions used for reference. Those in the upper-right box are motions generated by using fuzzy parameters from the CMAC: two of them are sampled motions used for learning, indexed by "L," and the other two are motions using fuzzy parameters generalized from those learned ones, indexed by "G." In Fig. 9(a), the motions generated using the proposed approach capture the behaviors of the reference motions well. The maximum position error for the "L" motions is about 0.4 cm and for the "G" motion is about 1 cm. Corresponding motion commands C_{m1} and C_{m2} are shown in the lower two boxes of Fig. 9(a). To note that, the proposed scheme bends the motion tracks near the ends, as shown at the upper-right

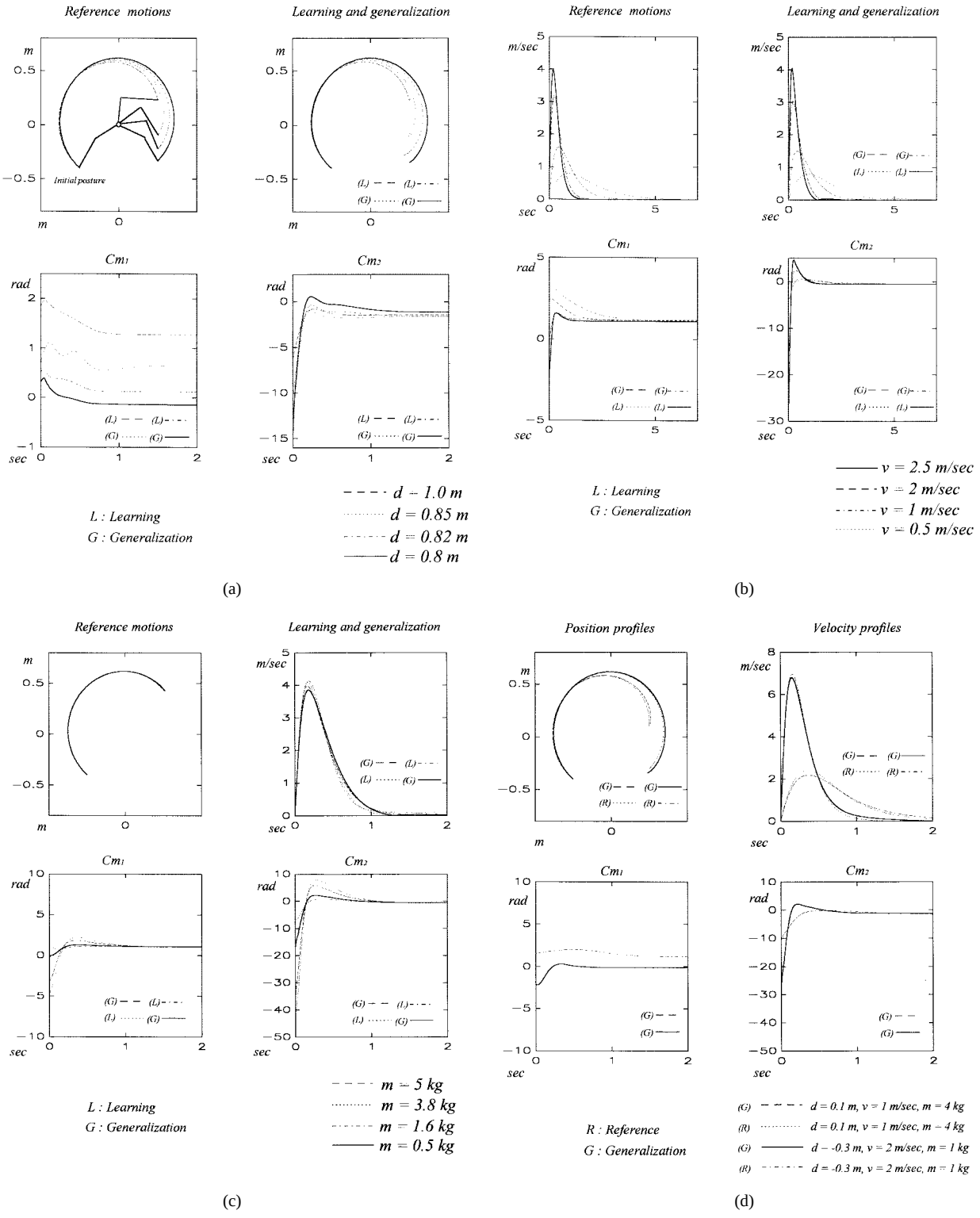


Fig. 9. Simulation results under (a) different distance requirements, (b) different velocity requirements, (c) different load requirements, and (d) different distance, velocity, and load requirements.

box in Fig. 9(a). This phenomenon occurs because the scheme did not fully accomplish a salient braking in the end of the motion tracking and could be diminished via further learning in motion governing. Analogously, the human also demonstrates a similar behavior in performing fast movements: an overshoot in the end of the movement [26]. In Fig. 9(b), motions of four different movement velocities with the same movement distance and load were simulated. The velocity traces of the motions generated using the proposed approach also emulated those of the reference motions well. The maximum velocity error for the "L" motions is about 10 cm/s and for the "G" motion is about 30 cm/s. In Fig. 9(c), motions of four different

loads with the same movement distance and velocity were simulated. As shown in the upper-right box of Fig. 9(c), the velocity traces of the four motions generated using the proposed approach are very close. And their corresponding position traces almost coincide with the reference motions shown in the upper-left box of Fig. 9(c). To further demonstrate the effect of generalization, we also simulated motions that are different from the sampled motions in movement distance, velocity, and load. As shown in Fig. 9(d), the two generalized motions approximate the reference motions well in both position and velocity.

V. Discussion and Conclusion

One major problem in applying learning controllers to govern general motions of multijoint robot manipulators is that the learning space is quite large. To tackle this, we propose an approach to enlarge learning space coverage by taking advantage of the merits of a fuzzy system and a CMAC-type neural network. The combination of the fuzzy system and the neural network in our design allows fuzzy rules to be generalized by a CMAC-type neural network. This means that abstract representations are generalized, and they tend to cover a larger learning space. Thus, the fuzzy parameters appropriate for a set of sampled motions can be generalized to deal with a whole class of motions. As the proposed scheme can be used to govern a class of motions with the same feature, possible industrial applications can be tasks that involve a number of workpieces with different loads, movement distances, etc. In addition, the scheme also provides the flexibility that workpieces in the same class can be added during task execution without repeating the learning process. Simulations performed verify the effectiveness of the proposed approach. At the current stage of the study, a robot learning control scheme based on the proposed approach is not available for application on general industrial robot manipulators. Nevertheless, with an appealing capability in learning space coverage, the proposed approach is with a potential to achieve the governing of general industrial robot manipulators by using the learning mechanism as the main control module.

One point concerning the proposed scheme that deserves discussion is the feature selection for motion classification. In the proposed scheme, the choices of distance, velocity, and load as trajectory descriptors are quite intuitive, although they are straightforward to serve as descriptors for motion indexing. Motion classification using these features cannot guarantee that motions in the same class will correspond to similar fuzzy parameters. We consider that motions may be classified from the viewpoint of learning. One possible strategy is to group into the same class those motions with the same number of fuzzy rules and similar shapes of membership functions when governed by using the FNN. Consequently, motions in the same class are guaranteed to correspond to very similar fuzzy parameters. Thus, the fuzzy parameters that the CMAC-type neural network needs to generalize will exhibit much less nonlinearity and then more classes of motions can be incorporated into the network with a reasonable size of network memory. From the preliminary simulation results based on this concept, we found that motions with different movement distances, velocities, and loads may belong to the same class. On the other hand, motions with the same distances or loads may be in different classes. It demonstrates that motion classification for learning does not correspond to only the kinematic or dynamic features. In future work, it may demand further investigations on the search of proper trajectory descriptor and on the classification of general motions in the entire learning space.

References

[1] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *ASME J. Dynamic Syst., Measurement, Contr.*, vol. 97, no. 3, pp. 220–227, Sept. 1975.

[2] S. Arimoto, "Robustness of learning control for robot manipulators," in *IEEE Int. Conf. Robot. Automat.*, Cincinnati, OH, May 1990, pp. 1528–1533.

[3] C. G. Atkeson and J. M. Hollerbach, "Kinematic features of unrestrained vertical arm movements," *J. Neurosci.*, vol. 5, no. 9, pp. 2318–2330, 1985.

[4] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, pp. 724–740, May 1992.

[5] E. Bizzi, N. Accornero, W. Chapple, and N. Hogan, "Posture control and trajectory formulation during arm movement," *J. Neurosci.*, vol. 4, no. 11, pp. 2738–2744, 1984.

[6] J. J. Craig, *Introduction to Robotics.* Reading, MA: Addison-Wesley, 1989.

[7] T. Flash, "The control of hand equilibrium trajectories in multijoint arm movements," *Biol. Cybern.*, vol. 57, pp. 257–274, 1987.

[8] C. C. A. M. Gielen and J. C. Houk, "A model of the motor servo: Incorporating nonlinear spindle receptor and muscle mechanical properties," *Biol. Cybern.*, vol. 57, pp. 217–231, 1987.

[9] S. Jung and T. C. Hsia, "On reference trajectory modification approach for Cartesian space neural network control of robot manipulators," in *IEEE Int. Conf. Robot. Automat.*, Nagoya, Japan, May 1995, pp. 575–580.

[10] A. Karakasoglu and S. I. Sudharsanan, "Identification and decentralized adaptive control using dynamical neural networks with application to robotic manipulators," *IEEE Trans. Neural Networks*, vol. 4, pp. 919–930, June 1993.

[11] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki, "Hierarchical neural network model for voluntary movement with application to robotics," *IEEE Contr. Syst. Mag.*, vol. 8, no. 2, pp. 8–16, Apr. 1988.

[12] S. W. Keele, "Movement control in skilled motor performance," *Psych. Bull.*, vol. 70, pp. 387–403, 1968.

[13] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Part I," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 404–418, Mar./Apr. 1990.

[14] C. M. Lim and T. Hiyama, "Application of fuzzy logic control to a manipulator," *IEEE Trans. Robot. Automat.*, vol. 7, no. 5, pp. 688–691, Oct. 1991.

[15] 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.

[16] W. T. Miller III, F. H. Glanz, and L. G. Kraft III, "Application of a general learning algorithm to the control of robotic manipulators," *Int. J. Robot. Res.*, vol. 6, no. 2, pp. 84–98, 1987.

[17] A. Nedungadi and D. J. Wenzel, "A novel approach to robot control using fuzzy logic," in *IEEE Int. Conf. Syst., Man, Cybern.*, Charlottesville, VA, Oct. 1991, pp. 1925–1930.

[18] T. Ozaki, T. Suzuki, T. Furuhashi, S. Okuma, and Y. Uchikawa, "Trajectory control of robotic manipulators using neural networks," *IEEE Trans. Indust. Electron.*, vol. 38, no. 3, pp. 195–202, June 1991.

[19] A. Polit and E. Bizzi, "Characteristics of motor programs underlying arm movements in monkeys," *J. Neurophysiol.*, vol. 42, no. 1, pp. 183–194, 1979.

[20] T. D. Sanger, "Neural network learning control of robot manipulators using gradually increasing task difficulty," *IEEE Trans. Robot. Automat.*, vol. 10, no. 3, pp. 323–333, June 1994.

[21] R. A. Schmidt, *Motor Control and Learning: A Behavioral Emphasis.* Champaign, IL: Human Kinetics, 1988.

[22] T. Shibata and T. Fukuda, "Hierarchical intelligent control for robotic motion," *IEEE Trans. Neural Networks*, vol. 5, pp. 823–832, May 1994.
[23] M. Takegaki and S. Arimoto, "A new feedback method for dynamic control of manipulators," *ASME J. Dynamic Syst., Measurement, Contr.*, vol. 103, no. 2, pp. 119–125, June 1981.
[24] B. A. M. Wakileh and K. F. Gill, "Use of fuzzy logic in robotics," *Comput. Industry*, vol. 10, pp. 35–46, 1988.
[25] P. D. Wasserman, *Neural Computing—Theory and Practice.* New York: Van Nostrand Reinhold, 1989.
[26] C. H. Wu, K. Y. Young, K. S. Hwang, and S. Lehman, "Voluntary movements for robotic control," *IEEE Contr. Syst. Mag.*, vol. 12, no. 1, pp. 8–14, Feb. 1992.
[27] K. Y. Young and C. C. Fan, "Control of voluntary limb movements by using a fuzzy system," in *IEEE Conf. Decision Contr.*, San Antonio, TX, Dec. 1993, pp. 1759–1764.
[28] H. J. Zimmermann, *Fuzzy Set Theory And Its Applications*, 3rd ed. Norwell, MA: Kluwer, 1996.

Kuu-Young Young (S'86–M'90) was born in Kaohsiung, Taiwan, on Dec. 22, 1961. He received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1983, and the M.S. and Ph.D. degrees in electrical engineering from Northwestern University, Evanston, IL, in 1987 and 1990, respectively.

From 1983 to 1985, he served as an Electronic Officer in the Taiwan Navy. Since 1990, he has been an Associate Professor in the Department of Control Engineering at National Chiao-Tung University, Hsinchu, Taiwan. His current research interests include biological control system, robot learning control, robot calibration and path planning, and CAD/robot integration.

Shaw-Ji Shiah was born in Taoyuan, Taiwan, 1969. He received the B.S. degree in electrical engineering from Chung Yuan Christian University, Chung-Li, Taiwan, in 1992, and the M.S. degree in control engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 1994. He is currently working toward the Ph.D. degree at the Institute of Control Engineering, National Chiao-Tung University.

His research interests are in robotics, neural networks, fuzzy systems, and intelligent control.