Data Storage in the Cerebellar Model Articulation Controller (CMAC)[1]

J. S. ALBUS

Project Manager,
Office of Developmental
Automation and Control Technology,
Institute for Computer
Sciences and Technology,
National Bureau of Standards,
Washington, D.C.

The storage of manipulator control functions in the CMAC memory is accomplished by an iterative process which, if the control function is sufficiently smooth, will converge. There are several different techniques for loading the CMAC memory depending on the amount of data which has already been stored and the degree of accuracy which is desired. The CMAC system lends itself to a "natural" partitioning of the control problem into manageable subproblems. At each level the CMAC controller translates commands from the next higher level into sequences of instructions to the next lower level. Data storage, or training, is accomplished first at the lowest level and must be completed, or nearly so, at each level before it can be initiated at the next higher level.

1 Introduction

The computation of control functions by table reference techniques has seldom been considered as a practical methodology for manipulator control. The fact that for N input variables with R distinguishable levels there are R^N possible inputs has been sufficient to discourage this line of research. It seems clear, however, that the control of mechanical devices with many degrees of freedom and with enormous numbers of input variables by means of table reference procedures is not only practical, but enormously successful. It almost certainly is the principle used by living creatures of every description. People do not consciously control each individual muscle or joint when manipulating their limbs, and there is no evidence to suggest that animals do either. Yet somehow high level decisions to perform specific movements result in coordinated control signals being sent to the individual muscles. Unless one is prepared to believe that the brain performs matrix inversions of the type used in resolved motion rate control [5],[2] it must be assumed that biological organisms use some form of memory driven control system for solving the muscle coordination problem. This fact alone suggests that the difficulties posed by R^N potential inputs may be considerably exaggerated. To begin with, it is clear that even a biological brain does not have sufficient cells to store R^N separate values of the control function any more than does an electronic computer. Furthermore, upon close examination it becomes evident that for any physical manipulator system, nowhere near R^N inputs are ever used or even needed for the execution of any single trajectory. In addition, for all trajectories within any given class, the sequences of joint actuator control signals are highly redundant. In fact, even for an entire repertoire of movements, only a tiny fraction of the R^N inputs are ever actually used and are thus not physically required for storing the control functions needed to operate a manipulator in a real environment. For example, in throwing a baseball or swinging a golf club, the proper command to each muscle actuator at each point in the throw or swing trajectory is strongly correlated with commands at neighboring points in the same or similar trajectories. If a memory management scheme could be devised to take proper advantage of this redundancy in the data being stored, then it might be quite practical to store the functions required for manipulator control in a memory of reasonable size [1]. This is the principle used by the Cerebellar Model Articulation Controller (CMAC).

The basic theory of how CMAC takes advantage of input redundancy and how the CMAC memory management algorithm functions are discussed in a companion paper [2]. Preliminary experimental results from a seven-degree-of-freedom manipulator operating under CMAC control have been published elsewhere [1]. This paper deals with algorithms by which data storage can be accomplished in the CMAC system.

2 Data Storage

CMAC uses a distributed storage system whereby the numerical contents of each address are distributed over a number of physical memory locations. The contents of these physical memory locations are referred to as weights, and the contents of an address is defined as the linear sum of all the weights selected by the CMAC addressing algorithm. The method used for storing numerical values in a CMAC memory is analogous to the procedure used for training a Perceptron [3]. In the Perceptron case, weights are typically adjusted by an iterative error-correction procedure which, under certain specific conditions, con-

[1]Contribution of the National Bureau of Standards. Not subject to copyright.

[2]Numbers in brackets designate References at end of paper.

Contributed by the Automatic Control Division for publication in the JOURNAL OF DYNAMIC SYSTEMS, MEASUREMENT, AND CONTROL. Manuscript received at ASME Headquarters, June 6, 1975.

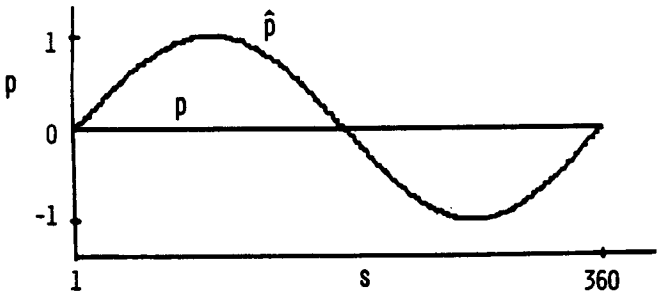

Fig. 1 p is the output from a one-input CMAC memory prior to any data being stored. $\hat{p}$ is the desired output. For this case the maximum error between p and $\hat{p}$ is 1.0 and the r.m.s. error is 0.707.

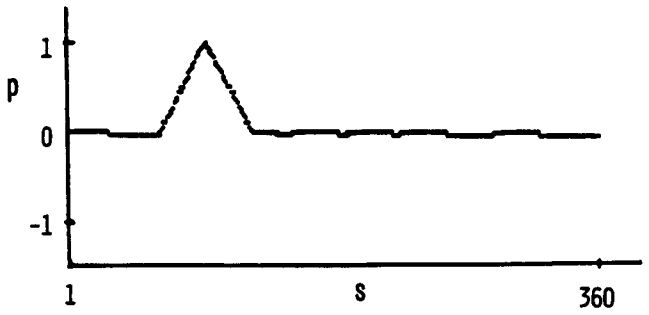

Fig. 2 The output of the CMAC memory after a single error correction data storage operation. p was set equal to 1.0 at s = 90. Maximum error is still 1.0 (at s = 270) and r.m.s error is now 0.625.

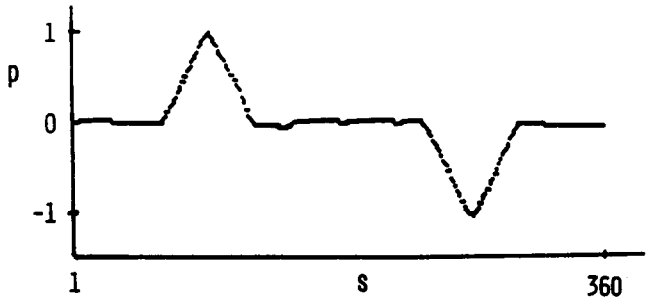

Fig. 3 After two data storage operations. Maximum error = 0.87 and r.m.s. error = 0.530.

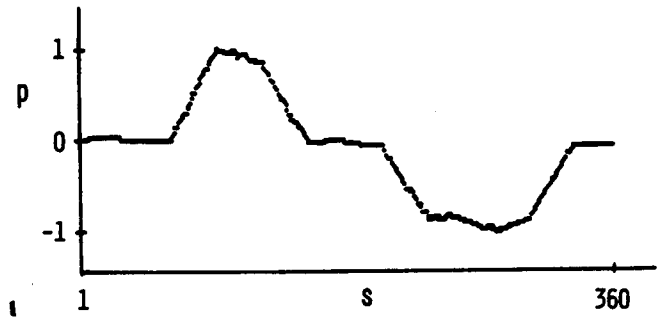

Fig. 4 After five data points are stored. Maximum error = 0.84 and r.m.s. error = 0.313.

verges. Similarly, storage of a function in the CMAC memory is an iterative procedure which, if the function being stored is sufficiently smooth, will converge.

2.1 Choosing the Weights. The procedure for storing a function in the CMAC memory is as follows:

1 Assume that $\hat{\mathbf{P}}$ is the desired value of the vector to be stored at address S.

2 Address the memory with S and retrieve whatever is currently stored. The current value of the function is $\mathbf{P} = h(\mathbf{S})$.

3 For every element in $\mathbf{P} = (p_1, p_2, \ldots, p_k, \ldots, p_L)$ and in $\hat{\mathbf{P}} = (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_k, \ldots, \hat{p}_L)$,

$$\text{if } |\hat{p}_k - p_k| \leq \xi_k$$

where ξ_k is an acceptable error, then do nothing; the desired value is already stored in the proper address. However, if

$$|\hat{p}_k - p_k| > \xi_k$$

then add Δ_k to every weight which contributed to p_k where

$$\Delta_k = \frac{\hat{p}_k - p_k}{|A^*|} \tag{1}$$

$|A^*|$ = the number of weights which contribute to p_k

At present, there exists no formal proof of the convergence of this procedure. However, it has been empirically demonstrated to be effective in loading a CMAC memory for controlling a manipulator in the performance of a number of coordinated movements [1]. The CMAC training algorithm is quite similar to the Perceptron error-correction algorithms. It thus seems reasonable to assume that the conditions of convergence would be roughly similar to those for convergence of the Perceptron training algorithms.

In any case, convergence of the CMAC training procedure is clearly related to the smoothness of the function within the neighborhood over which generalization takes place. If the value of the function varies slowly throughout the neighborhood of generalization, it should be quite easy to find a set of weights which can represent the function to within the acceptable error ξ_k everywhere in the neighborhood. However, if the value of the function changes significantly within the neighborhood of generalization, it may be difficult or impossible to find any set of weights which can represent the function to the desired accuracy at every point in the neighborhood. It therefore is very important that the neighborhoods over which the system is to generalize be small relative to expected variations in the value of the output function. In general, this may be accomplished by choosing the proper resolution for the $s_i \rightarrow m_i^*$ mapping. If a manipulator control signal is expected to be especially sensitive to a particular input variable, the $s_i \rightarrow m_i^*$ mapping for that variable should be high resolution. If the manipulator control signal is relatively insensitive to small changes in an input variable, the $s_i \rightarrow m_i^*$ mapping for that variable can be low resolution.

2.2 Examples of Data Storage. For the examples shown in Figs. 1–9 the number of elements in A^* is arbitrarily chosen equal to 32 and the size of A_p is 4096. In Figs. 1–6 both the input and output vectors are one-dimensional, i.e.,

$$\mathbf{S} = (s)$$

$$\text{and } \mathbf{P} = (p)$$

Assume that the value of the input variable s is defined on the interval 1 to 360 with a resolution of unity, i.e.,

$$\mathbf{S}_i = (i) \text{ where } i = 1, \ldots, 360$$

Also assume that the desired output $\hat{p} = \sin(2\pi s/360)$. If all the weights are originally zero, the value of the function $p = h(s)$ is zero for all s. The original condition of p and $\hat{p}$ is shown in Fig. 1.

Assume now that a data point is stored at $s = 90$. According to the data storage algorithm this implies that each of the 32 weights attached to an association cell in the set A_{90}^* will be incremented by an amount $\Delta = \frac{\hat{p} - p}{32}$, or $\frac{1}{32}$.

Fig. 2 shows the value of p over the entire range of s after a

Fig. 5 After nine data points are stored. Maximum error = 0.33 and r.m.s. error = 0.091.

Fig. 6 After sixteen data points are stored. Maximum error = 0.09 and r.m.s. error = 0.033.

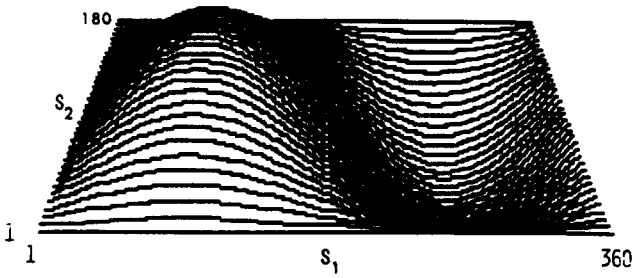

Fig. 7 A plot of a desired output $\hat{p}$ for a CMAC with two inputs. $\hat{p} = \sin\left(\frac{2\pi s_1}{360}\right)\sin\left(\frac{2\pi s_2}{360}\right)$

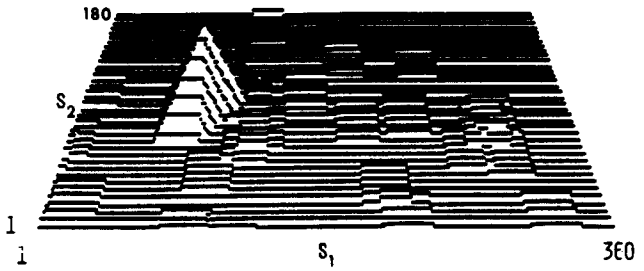

Fig. 8 The output of a two-input CMAC memory after a single error correction data storage operation. p was set equal to 1.0 at $s_1 = 90$, $s_2 = 90$.

single data storage at $s = 90$. At $s = 90$, $p = \hat{p} = 1$. As the input vector $\mathbf{S}_i$ moves away from the $s = 90$ point at which data were stored, the overlap $|A_{90}^* \wedge A_i^*|$ decreases until finally it reaches zero.

Thus, the influence of the value of p stored at $s = 90$ declines as a function of separation in input space. Outside of the range of $s = 90 \pm 32$, $p = 0$ except for spurious overlap due to hash coding. The effects of spurious overlap can be seen in the uneven baseline in Fig. 2.

Fig. 3 shows a plot of p after a second data storage at $s = 270$.

After each data storage operation, the maximum error and the root-mean-square error between p and $\hat{p}$ is computed over the entire range of s. In this series of examples, the point of maximum error is the value of s chosen for the next data storage. Fig. 4 is a plot of p after five data storage operations, Fig. 5 after nine, and Fig. 6 after sixteen.

As can be seen, the CMAC output p rapidly converges to the desired function $\hat{p}$.

For the examples shown in Figs. 7–9 the input vector is two-dimensional, i.e.,

$$\mathbf{S} = (s_1, s_2)$$

Assume that the desired output $\hat{p}$ at each point in input space is given by

$$\hat{p} = \sin\left(\frac{2\pi s_1}{360}\right)\sin\left(\frac{2\pi s_2}{360}\right)$$

over the range $1 \leq s_1 \leq 360$, $1 \leq s_2 \leq 180$.

The reference surface defined by $\hat{p}$ is shown in Fig. 7.

Fig. 8 is a plot of the CMAC output $p = h(s_1, s_2)$ after a single data storage at $s_1 = 90$, $s_2 = 90$. After sixteen data storage operations along the $s_2 = 90$ line, the CMAC output p looks like Fig. 9. In this particular example, a cross section through the $s_2 = 90$ plane is identical to Fig. 6.

If the strategy of correcting the maximum error is applied in storing the function shown in Fig. 7, a plot of rms error versus number of data points stored is a classical exponential learning curve with a $\frac{1}{e}$ point at 20 data storage operations.

Thus, at least for the type of curves shown in the preceding examples, CMAC does converge and rather rapidly.

From these examples it can be seen how CMAC tends to generalize over small neighborhoods in input-space and how quickly it can approximate a desired function after a relatively few data points are stored.

These examples also illustrate how increasing the dimensionality of the input vector affects the concept of an input-space neighborhood. The triangle of decreasing influence for one-dimensional inputs becomes a pyramid in two dimensions and a complex volume in n dimensions.

2.3 Training for Manipulator Control. CMAC, like the cerebellum in the brain after which it was modeled, does not operate by mathematically analyzing the dynamics of the control problem and then solving equations. Instead it operates by doing something, observing the results of the action, and then adjusting internal parameters in a direction calculated to improve the correspondence between what was called for and what actually occurred. The problem, of course, is to determine what internal parameters need adjustment and by what amount. The error correction algorithm requires that the observed output $\mathbf{P}$ be compared with a desired output $\hat{\mathbf{P}}$. In many cases, however, $\hat{\mathbf{P}}$ is not known explicitly. Typically, what is known is the desired end-point movement in the form of a particular $\dot{x}$, $\dot{y}$, $\dot{z}$ vector.

There are several possible techniques which can be employed to derive the desired output $\hat{\mathbf{P}}$. In general these can be classified into categories of coarse and fine. The coarse techniques typically are used to initiate the data storage, or training, procedure from the original starting state where all the CMAC weights are set to zero. By the time the "coarse" training process is complete, the CMAC system is capable of controlling a manipulator such that the output is roughly correct, although perhaps not as precise as would be desired. From this point on, "fine" training procedures can be utilized to optimize system performance to the theoretical limit set by $\mathbf{S} \rightarrow A_p^*$ mapping resolution and by memory capacity.

There are several possible coarse training techniques. One is training by analogy. For example, assume the CMAC is to be trained to respond to $\dot{x}$, $\dot{y}$, $\dot{z}$ velocity commands. First, the man-

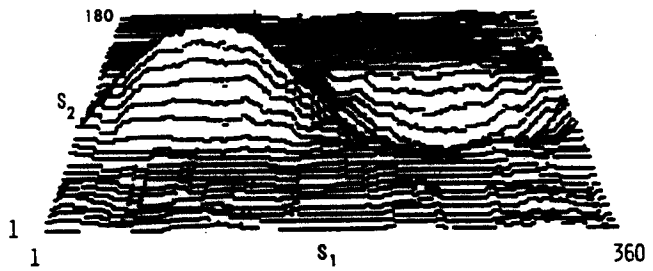

Fig. 9 The output of a two-input CMAC memory after sixteen data points were stored. A cross section of this figure in the $s_2 = 90$ plane is identical to Fig. 6.

ipulator can be led through several representative straight line trajectories corresponding to the appropriate $\dot{x}$, $\dot{y}$, $\dot{z}$ commands. This may be done either by master-slave techniques, or rate control techniques [4]. In either case, specific straight line trajectories are defined by a series of joint angle positions recorded along these trajectories. The manipulator can then be caused to execute the specific trajectories by simply stepping sequentially through these recorded points.

For example, assume that a point-by-point trajectory had been recorded starting from some specific initial point x_0, y_0, z_0, and proceeding along a specific trajectory defined by some $\dot{x}$, $\dot{y}$, $\dot{z}$ at each point. Now, at every point on the recorded trajectory two separate processes take place simultaneously:

1 The manipulator is physically controlled by a conventional servo system which drives each joint actuator at the proper velocity along its recorded trajectory.

2 The CMAC algorithm computes its own estimate of what the joint drive signals should be at each point by observing the appropriate feedback variables and assuming that the actual $\dot{x}$, $\dot{y}$, $\dot{z}$ at each point are the commanded velocities from higher level inputs. In other words, CMAC observes an input vector $\mathbf{S}$ and computes $\mathbf{P} = h(\mathbf{S})$.

Now, since the drive signals from the conventional servos are actually what is required to reach the next point on the recorded trajectory, these signals make up a desired output vector $\hat{\mathbf{P}}$. The CMAC training procedure may then be used at each point to adjust weights and null the difference between the $\mathbf{P}$ computed by CMAC and the $\hat{\mathbf{P}}$ derived from the conventional servo output signals. Thus the CMAC controller is trained at each point along the recorded trajectory by observing what is required to move the manipulator to the next point.

Due to the CMAC property of generalization, the CMAC controller can learn from observing a limited number of representative trajectories and generalize so as to operate effectively over a large number of similar trajectories. If the representative trajectories cover the relevant input set adequately, the overlap in the A_i^* sets generates output signals which are close enough to the proper speed and direction to produce generally satisfactory results. Of course, the definition of "satisfactory" is situation-dependent. The degree of accuracy which CMAC can produce within any volume of input-space depends upon the number of representative trajectories selected, and the precision of the $\mathbf{S} \rightarrow \mathbf{A}$ mappings used.

There are, of course, other techniques by which this initial phase of CMAC training can be accomplished. One is by analytical techniques similar to the resolved motion rate control system. The desired joint actuator signals $\hat{\mathbf{P}}$ may be computed from $\dot{x}$, $\dot{y}$, $\dot{z}$ commands by the inverse Jacobian technique [5]. The CMAC can then be trained to emulate the results of the analytic computations. Since only a rough approximation of the eventual desired performance is required at the end of the "coarse" training, numerous simplifications may be introduced so as to make analytical computations feasible for even rather complex manipulator systems. For instance, initial training might be accomplished at slow speeds so that higher order effects of velocity, acceleration, cross-products of inertia, and coriolis forces may be ignored.

2.4 Optimization or Fine Training. Once the initial phase of training is completed, it can be assumed that the CMAC control system can produce all of the desired elemental movements, but only in a crude and imprecise way. From this point it is possible for the CMAC to proceed in a boot-strapping manner. Again, there are several techniques by which this can be accomplished. One is by a time inversion technique.

Ordinarily the CMAC training algorithm proceeds by (1) observing an input $\mathbf{S} = (s_1, s_2, s_3, \ldots . \dot{x}, \dot{y}, \dot{z})$; (2) computing an output $\mathbf{P} = h(\mathbf{S})$; (3) comparing $\mathbf{P}$ against a desired $\hat{\mathbf{P}}$; and (4) adjusting weights so as to null the difference. In the process of training, the function h is modified to h' such that $\hat{\mathbf{P}} = h'(\mathbf{S})$. The critical factor in this conventional technique is finding the desired output $\hat{\mathbf{P}}$ corresponding to the actual input $\mathbf{S}$. In the time inversion technique this process is inverted, i.e., the computed output $\mathbf{P}$ is assumed to be the *desired* output for some unknown input $\hat{\mathbf{S}}$. The problem then is not to find the desired output $\hat{\mathbf{P}}$ corresponding to some actual input $\mathbf{S}$, but instead to find some input $\hat{\mathbf{S}}$ for which $\mathbf{P}$ is the desired output. This may be done in the following manner:

First, apply the computed output $\mathbf{P}$ to the joint actuators and observe the resulting movement $\hat{\dot{x}}$, $\hat{\dot{y}}$, $\hat{\dot{z}}$. Now, if the original input $\mathbf{S}$ had called for the observed movement $\hat{\dot{x}}$, $\hat{\dot{y}}$, $\hat{\dot{z}}$ instead of $\dot{x}$, $\dot{y}$, $\dot{z}$, then $\mathbf{P}$ would have been exactly the correct output. Therefore, the input $\hat{\mathbf{S}}$ for which $\mathbf{P}$ is the desired output, is merely the original input $\mathbf{S}$ with $\hat{\dot{x}}$, $\hat{\dot{y}}$, $\hat{\dot{z}}$ substituted for $\dot{x}$, $\dot{y}$, $\dot{z}$. In other words, $\hat{\mathbf{S}} = (s_1, s_2, s_3, \ldots . \hat{\dot{x}}, \hat{\dot{y}}, \hat{\dot{z}})$. If $\hat{\mathbf{S}}$ is now applied to the input, a new output will be computed $\mathbf{P}' = h(\hat{\mathbf{S}})$. This output need not be applied to the joint actuators but can be used solely in adjusting the weights. The weights should be adjusted so as to null the difference between the originally computed output $\mathbf{P}$ and the newly computed $\mathbf{P}'$.

This time inversion technique can be utilized while the manipulator is performing operational maneuvers under higher level commands. Thus, CMAC training can be performed "on the job" in an actual work environment. As training proceeds, the manipulator will respond more and more precisely to the commands being issued by the higher level centers. The time inversion technique automatically takes into account all the specific peculiarities resident in the individual manipulator being controlled. Variations and nonlinearities such as drive motor inefficiencies, joint friction, and gear backlash are automatically incorporated into the CMAC control system along with inertia cross-products, coriolis forces, and bending or twisting effects of the manipulator structure. The time inversion training algorithm can be used in real time while performing movements under actual operating conditions. No analytical computations are performed. The CMAC simply learns to do whatever is necessary to produce the output called for by the higher level input commands regardless of complications and irregularities existing in the electromechanical mechanisms. Since the CMAC training procedure is presumed to be convergent, the amount of change to the memory caused by this iterative training procedure is finite, and the rate of change approaches zero as performance is optimized.

Performance after training is complete is limited only by one of the following:

1 insufficient sensory input data

2 insufficient resolution on the various $\mathbf{S} \rightarrow A_p^*$ mapping functions

3 insufficient speed in the CMAC computational cycle

4 insufficient memory to prevent excessive noise caused by A_p^* overlaps due to hash coding

5 nonstationary processes and nonrecurring events such as random sticking in joints.

Of course, once a CMAC algorithm is trained to control a particular manipulator, the distribution of weights in its memory can be duplicated and used in CMAC controllers for other manipulators of similar configurations. Additional "fine" training will be necessary only if dissimilarities in manipulators or differences in task requirements cause one manipulator control function to differ appreciably from another.

3 Implications of CMAC

CMAC is essentially a sampled servo control system with a great number of very useful features. For example:

1 CMAC can handle arbitrarily large numbers of feedback variables with many different types of nonlinear cross-products.

2 Feedback variables can be expressed in arbitrary units derived from sensors arranged in unconventional coordinate systems.

3 Feedback sensors need not be linear, nor even free from hysteresis, if direction of approach is included as a variable. They merely need to be repeatable.

4 The designer of a CMAC control system need not be able to express the control function in terms of mathematical equations.

5 The CMAC control system can incorporate many different control functions, any one of which can be activated or switched into effect by changing the command signal from higher centers.

6 The CMAC controller not only is a servo controller, but is a coordinate system transformer which enables higher level input signals to be expressed in any convenient coordinate system.

CMAC has several unique features which distinguish it from other adaptive manipulator control techniques such as have been proposed by Freedy, et al. [6], or Lawrence and Lin [7]. First, CMAC represents a serious attempt to model the neurophysiological functioning of the mammalian cerebellum. Each mathematical transformation performed by CMAC has a specific analog in the structure and/or function of the living cerebellum.

Second, the complexity of accessing the CMAC memory increases only linearly as the number of input dimensions. Each $s_i \rightarrow m_i^*$ mapping is independent of all the others, and the computation of A^* is simply by concatenation of respective elements from each m^*. The number of cells in A^* and A_p need not change as the dimensionality of the input increases as long as $R^N < (V-1)^U$, where $V = \dfrac{|A_p|}{|A^*|}$ and $U = |A^*|$. See reference [2].

This implies that a CMAC controller with several tens or even hundreds of input variables is potentially feasible. Much of the processing of input variables can be done independently and in parallel, or if speed is not a critical requirement, the entire task can be handled by a single computer.

The computer program which implemented the examples in this paper was used to measure the time T required for a complete CMAC memory access cycle. The following formula was found to obtain:

$$T = (0.4N + 5.5)L \text{ millisec}$$

where N = number of inputs

L = number of outputs

This relationship was for a CMAC where $|A^*| = 32$. It was measured on a computer with a basic instruction cycle time of 1.2 microsec.

The number of data storage operations required to load the CMAC memory, of course, increases as some power of the number of inputs. Thus, the length of time required to store a function to some desired accuracy may be the limiting parameter in the case of multidimensional inputs.

Finally CMAC is structured so as to make it practical to partition the control problem into a hierarchy of CMAC control levels. Each CMAC input vector S is composed of two parts, feedback from the periphery, and command variables from higher level control centers. Each input vector $\mathbf{S}_1$ produces an output vector $\mathbf{P}_1$ which drives the joint actuators to a new position, thereby creating a new input vector $\mathbf{S}_2$. As this process is repeated, a string of input vectors $\mathbf{S}_1\mathbf{S}_2\mathbf{S}_3 \ldots$ and output driving signals $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3 \ldots$ is produced and the manipulator moves along some trajectory in space. If the command variables in each of the input vectors $\mathbf{S}_1\mathbf{S}_2\mathbf{S}_3 \ldots$ are held constant throughout the entire sequence, the resultant trajectory will be characteristic of that particular set of command variables. For example, if the command variables are set to $\dot{x} = 10$, $\dot{y} = -3$, $\dot{z} = 0$ for a sequence of inputs $\mathbf{S}_1\mathbf{S}_2\mathbf{S}_3 \ldots$, the manipulator end point will move along its entire trajectory with the velocities $\dot{x} = 10$, $\dot{y} = -3$, and $\dot{z} = 0$. The feedback variables, of course, reflect external conditions along the trajectory and the CMAC controller compensates for all disturbing influences so as to produce the trajectory commanded. The CMAC thus can be viewed as a form of function subroutine which accepts an input command and produces a string of outputs so as to carry out the input command. The input signal from a higher center is analogous to a subroutine call. The resulting series of output signals from the CMAC system to the joint actuators is analogous to the operational steps in the called subroutine.

This analogy to a computer subroutine suggests a possible hierarchical structure whereby the higher level input signals to a CMAC controller may be generated by another CMAC controller. For example, if a CMAC controller can be structured and trained to produce an appropriate string of output signals in response to higher level $\dot{x}$, $\dot{y}$, $\dot{z}$ inputs, it is possible for a second CMAC controller to produce a string of $\dot{x}$, $\dot{y}$, $\dot{z}$ commands in response to a still higher level input describing the type of tra-

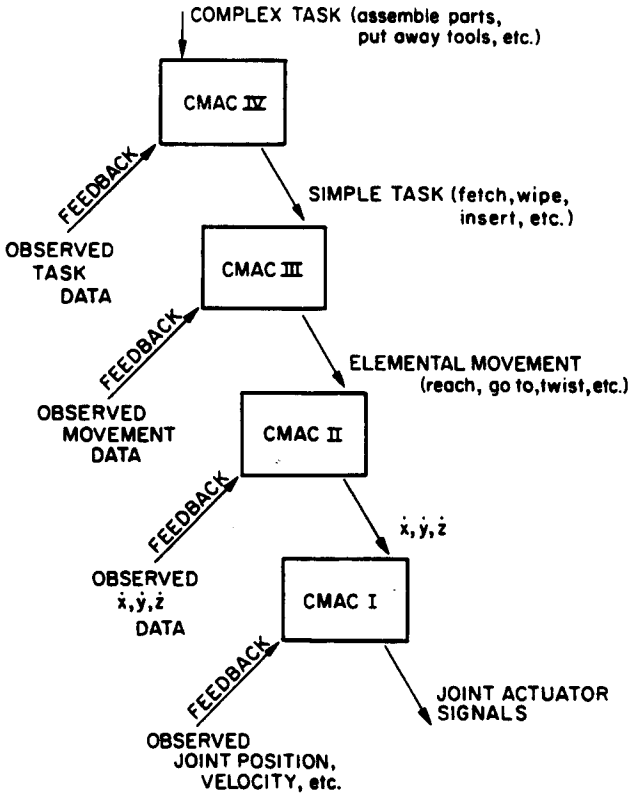

Fig. 10 A hierarchical structure of CMAC controllers. In response to each input command from a higher level, each CMAC computer generates a string of output commands to a lower level. These output commands are functions of feedback variables as well as higher level inputs. The CMAC concept can be used to partition the manipulator control problem into a manageable set of subproblems.

Table 1

CMAC levels	Analogous levels of computer language
complex task	main source program
simple task	source language subroutine
elemental movement command	source language statement
$\dot{x}$, $\dot{y}$, $\dot{z}$ command	assembly language instruction
joint actuator signals	internal machine code

An analogy between various steps in the CMAC control hierarchy and various levels of computer language coding.

jectory, or elemental movement, desired. This second CMAC controller can itself be commanded from above by a third level CMAC, and so on indefinitely as shown in Fig. 10. This type of hierarchical structure is a means of partitioning the control problem into manageable subproblems.

In a biological organism the highest level inputs may consist of commands like "fight," "flee," "build nest," "eat," etc. McCulloch and Kilmer [8] have hypothesized that the reticular formation of the brainstem functions so as to commit an animal to one or another of these "modes of behavior." Sutro and Kilmer [9], Friedman [10], Fikes, Hart, and Nilsson [11], and others have suggested that the same sort of higher level "selecting" or "planning" mechanism is essential in the control structure of an autonomous robot.

The decision of how to partition the control problem for a hierarchy of CMAC controllers depends on how complex are the highest level commands, how many and what type of feedback variables are available at each level, and what methods are to be used in training. For example, in the case of a simple manipulator where very few feedback variables are used, it is possible for a single CMAC to accept elemental movement commands and produce joint actuator signals directly. However, for a more complex manipulator with many feedback variables, it may be best to partition the control problem such that the lowest level CMAC is controlled by a second CMAC which generates $\dot{x}$, $\dot{y}$, $\dot{z}$ commands in response to elemental movement commands.

In many respects a CMAC controller is similar to a computer language translator which accepts input in a higher level language and generates strings of output statements in a lower level language. One might draw an analogy between each step in the CMAC hierarchy of Fig. 1 and a level of code in a computer program as shown in Table 1. At the very top, a high level goal (or main program) is selected and a command is issued to initiate the execution of this program. At each level the CMAC controller accepts a higher level command and generates a sequence of lower level instructions which carry out the higher goal. These instructions are generated on the basis of feedback variables together with prior training in how to respond to particular feedback patterns.

Training, of course, must begin at the bottom. Training must be complete, or nearly so, at each level before it can even begin at the next higher level in the hierarchy.

4 Summary and Conclusions

The storage of data in CMAC can be accomplished by one of several iterative training procedures. If a set of training trajectories is available, error correction techniques can be applied until performance reaches some predetermined level of acceptability. Otherwise, techniques are available by which the error correction information can be derived from observed discrepancies between input commands and output actions. The basic form of CMAC makes it amenable to a hierarchical structure whereby input commands from the next higher level are translated into strings of output commands to the next lower level.

Research on the CMAC control concept is still in its very early stages. Only preliminary experimental results are available even for the first level CMAC controller. The entire question of how to partition the control problem into a hierarchical structure has yet to be investigated in any depth. In particular, the question of how to structure the feedback for higher level CMAC controllers has not even been addressed. Undoubtedly, higher level feedback should undergo the preprocessing and pattern recognition procedures. Training techniques have not been fully explored, especially for the higher levels.

In spite of the very preliminary state of the present research, there is much about the CMAC system which suggests that it may be a valuable control concept for complex manipulator systems in the future. The CMAC approach is not limited to control problems where the relevant computations can be described in analytic mathematical form. Many different kinds of unconventional coordinate system transformations can be handled with equal ease. Furthermore, the computations fundamental to the CMAC algorithm are of such a nature that they seem readily amenable to construction in large-scale integrated circuit technology. Indications are that CMAC controllers could be constructed quite simply and cheaply as special purpose microcomputers. If this turns out to be feasible in practice, the CMAC hierarchical control technique may make it practical in the future to build extremely sophisticated manipulator systems which will be capable of operating far beyond the limits of the human arm in speed, strength, and even dexterity.

References

1 Albus, J. S., "Theoretical and Experimental Aspects of a Cerebellar Model," PhD thesis, University of Maryland, Dec. 1972.

2 Albus, J. S., "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," JOURNAL OF DYNAMIC SYSTEMS, MEASUREMENT, AND CONTROL TRANS. ASME, Series G, Vol. 97, No. 3, Sept. 1975.

3 Rosenblatt, F., *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington, D. C., 1961.

4 Corliss, W. R., and Johnsen, E. G., *Teleoperator Controls*, NASA SP-5070, 1968.

5 Whitney, D. E., "Resolved Motion Rate Control of Manipulators and Human Prostheses," *IEEE Trans. on Man-Machine Systems*, Vol. MMS-10, No. 2, June 1969, pp. 47–53.

6 Freedy, A., Hull, F. C., Lucaccini, L. F., and Lyman, J., "A Computer-Based Learning System for Remote Manipulator Control," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-1, No. 4, Oct. 1971, pp. 356–363.

7 Lawrence, P. D., and Lin, W. C., "Statistical Decision Making in the Real-Time Control of an Arm Aid for the Disabled," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-2, No. 1, Jan. 1972, pp. 35–42.

8 Kilmer, W., McCulloch, W. S., and Blum, J., "A Model of the Vertebrate Central Command System," *International Journal Man-Machine Studies*, 1, 1969, pp. 279–309.

9 Sutro, L. L., and Kilmer, W. L., "An Assembly of Computers to Command and Control a Robot," *Proceedings of Spring Joint Computer Conference*, AFIPS Press, 1969, p. 131.

10 Friedman, L., "Instinctive Behavior and Its Computer Synthesis," *Behavioral Science*, 12, 1967, pp. 85–108.

11 Fikes, R. E., Hart, P. E., and Nilsson, N. J., "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, Vol. 3, No. 4, Winter 1972, pp. 251–288.