Reprinted from September 1975 Journal of Dynamic Systems, Measurement, and Control

A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)[1]

J. S. ALBUS

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

CMAC is an adaptive system by which control functions for many degrees of freedom operating simultaneously can be computed by referring to a table rather than by mathematical solution of simultaneous equations. CMAC combines input commands and feedback variables into an input vector which is used to address a memory where the appropriate output variables are stored. Each address consists of a set of physical memory locations, the arithmetic sum of whose contents is the value of the stored variable. The CMAC memory addressing algorithm takes advantage of the continuous nature of the control function in a way which promises to make it possible to store the necessary data in a physical memory of practical size.

Introduction

Simply stated, the control problem for a manipulator is that of finding what each joint actuator should do at every point in time and under every set of conditions. In order to carry out any movement, it is necessary to drive the various joints through a sequence of positions as a function of time. The drive signal to each joint actuator is, in general, a function not only of time, but of many other variables as well. These include position, velocity, and acceleration loading in most, or all, of the joints; force and touch signals from various points on the manipulator; visual or other feedback data concerning the position of the end point; plus measurements of bending, twisting or backlash in various structural components. The drive signals depend on higher level input variables which identify the particular task which is being performed or end point movement which is being executed. If the manipulator is interacting with a dynamic environment, the forces in the various joints must also be functions of positions, velocities, and accelerations of external objects as well as forces imposed by external sources. Thus, although the manipulator control problem may be stated in rather simple terms, its solution is very complicated indeed.

In order to deal with a problem of this complexity without either sidestepping the computational difficulties, as in direct human control systems [1, 2],[2] or ignoring most of the relevant variables, as in point-to-point industrial robot control [3], it is necessary to partition the control problem into manageable subproblems [4]. For example, in order for a person to pick up an object such as a glass, it is first necessary to decide that "pick-up-glass" is a task to be accomplished, as opposed to "brush-teeth," "comb-hair," or any number of other potential tasks. Thus, the task name "pick-up-glass" is the first variable relevant to the manipulator control computation. It is also necessary to measure the position of the hand relative to the glass and compute what direction vector is required to move the hand into contact with the glass. This vector constitutes another input variable relevant to the manipulator control problem. In the human motor system, these first two variables are at the conscious level. Most of the subsequent computations are entirely subconscious. No one thinks about what their elbow or shoulder joints are doing during the "pick-up-glass" task, or how hard each individual muscle is pulling. They simply think in terms of what direction their hand should move. In the human manipulator control problem, the detailed computations of what each muscle must do in order to coordinate with other muscles so as to produce the desired movement are left up to lower level, subsconscious computing centers.

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

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.

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

For controlling mechanical manipulator systems, the computations required to coordinate individual joint rates so as to produce a particular motion of the end-effector are usually solved by computations based on trigonometric relationships between structural members of the manipulator itself. The resolved motion rate control system [5, 6] is illustrative of this technique. In the resolved motion rate control system, end-effector motion is expressed as a function of all the individual joint motions. A set of equations is written in the form

$$\dot{\mathbf{x}} = J(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{1}$$

where $\dot{\boldsymbol{\theta}}$ are velocities of individual joint angles and $\dot{\mathbf{x}}$ are components of the end-point velocity in some other coordinate system such as cartesian. $J(\theta)$ is the Jacobian matrix. If J is inverted, it is then possible to solve for $\dot{\boldsymbol{\theta}}$ in terms of $\dot{\mathbf{x}}$

$$\dot{\boldsymbol{\theta}} = J^{-1}(\boldsymbol{\theta})\dot{\mathbf{x}} \tag{2}$$

Thus, given a desired endpoint rate $\dot{\mathbf{x}}$, it is possible to use a small computer to solve for the required joint velocities $\boldsymbol{\theta}$. These $\dot{\boldsymbol{\theta}}$ are then converted to voltages and used to drive the joint actuators.

The type of computations performed by the resolved motion rate control system, and other similar systems, are typically based on more or less idealized mathematical formulations. Such systems usually take into account only joint angles and rates. With some difficulty other factors such as gravity loading and inertial forces can be included [4, 7]. However, as more and more real-world variables and nonlinearities are introduced into the problem, the trigonometric formalisms of systems of this type become less and less tractable. It is simply not possible to deal with many degrees of flexing and twisting or a very broad range of force, touch, and acceleration inputs by systems of simultaneous equations which can be solved by computer programs of practical speed and size.

When one examines the type of manipulation tasks routinely performed by biological organisms such as squirrels jumping from tree to tree, birds flying through the woods, and humans playing tennis or football, one is left with the distinct impression that the solution of trigonometric equations is a totally inadequate method for producing truly sophisticated motor behavior. It seems clear that the present mathematical formalisms for manipulator control are in deep trouble when addressing the type of mechanical control problems which are obviously trivial for the brain of the tiniest bird or rodent. If the fundamental principles of computation used by biological organisms were understood, it seems quite likely that an entirely new generation of manipulation control systems would be developed which would exhibit sensitivity and dexterity far beyond what is possible with present mathematical techniques. This is not to suggest that the proper course for research in the manipulator control field should be to attempt to model the structural properties of the biological brain. Early attempts along these lines were notoriously unsuccessful in producing any significant results and the subsequent disillusionment has strongly prejudiced the intellectual community against seeking any guidance from the numerous existence theorems provided by nature. There is good reason to believe, however, that it may be possible to duplicate the *functional* properties of the brain's manipulator control system without necessarily modeling the *structural* characteristics of the neuronal substrate.

One part of the brain that seems to be intimately involved in motor control processes is the cerebellum. Recent anatomical and neurophysical data has led to a detailed theory concerning the functional operations carried out by the cerebellum [8, 9]. Input to the cerebellum arrives in the form of sensory and proprioceptive feedback from the muscles, joints, and skin together with commands from higher level motor centers concerning what movement is to be performed. According to the theory, this input constitutes an address, the contents of which are the appropriate muscle actuator signals required to carry out the desired movement. At each point in time the input addresses an output which drives the muscle control circuits. The resulting motion produces a new input and the process is repeated. The result is a trajectory of the limb through space. At each point on the trajectory the state of the limb is sent to the cerebellum as input, and the cerebellar memory responds with actuator signals which drive the limb to the next point on the trajectory.

A neurophysiological theory of how the cerebellum accomplishes these tasks has been published elsewhere [10, 11]. This paper describes the mathematical concepts of how the cerebellum structures input data, how it computes the addresses of where control signals are stored, how the memory is organized, and how the output control signals are generated. These basic principles have been organized into a manipulator control system called CMAC (Cerebellar Model Articulation Controller).

The Cerebellum and the Perceptron

Certain features of the neurophysiological and anatomical structure of the cerebellum has led to the theory [9] that the cerebellum is analogous in many respects to a Perceptron [12]. The Perceptron is a member of a whole family of trainable pattern-classifying machines, or machines which distinguish between patterns on the basis of linear discriminate functions [13]. Physically, a Perceptron is structured as shown in Fig. 1. Because the Perceptron was originally inspired by attempts to model the brain, it embodies numerous neurophysiological terms. Input vectors are spoken of as sensory cell firing patterns. The input vectors (or sensory cell patterns) **S** may be either binary vectors or R-ary vectors. The appearance of an input vector **S** on the sensory cells produces an association cell vector **A** which also may be either binary or R-ary. In this paper, **A** will be a binary vector. This association cell vector multiplied by the weight matrix W produces a response vector **P**.

Mathematically the Perceptron may be represented by a pair of mappings

$$f\colon \mathbf{S} \rightarrow \mathbf{A} \tag{3}$$

$$g\colon \mathbf{A} \rightarrow \mathbf{P} \tag{4}$$

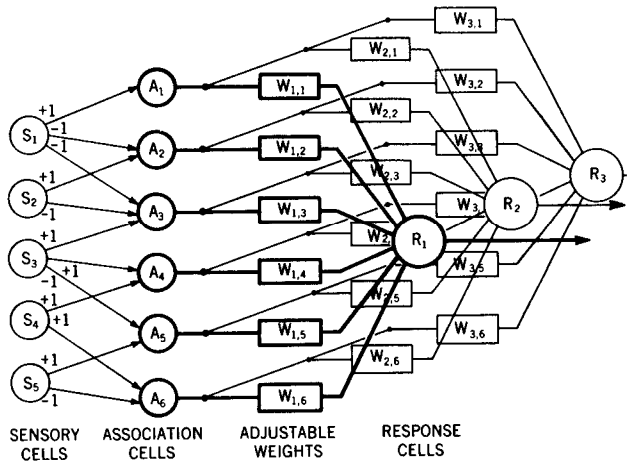

Fig. 1 Classical Perceptron. Each sensory cell receives stimulus either +1 or 0. This excitation is passed on to the association cells with either a +1 or −1 multiplying factor. If the input to an association cell exceeds 0, the cell fires and outputs a 1; if not, it outputs 0. This association cell layer output is passed on to response cells through weights $W_{i,j}$, which can take any value, positive or negative. Each response cell sums its total input and if it exceeds a threshold, the response cell R_j fires, outputting a 1; if not, it outputs 0. Sensory input patterns are in class 1 for response cell R_j if they cause the response cell to fire, in class 0 if they do not. By suitable adjustment of the weights $W_{i,j}$, various classifications can be made on a set of input patterns.

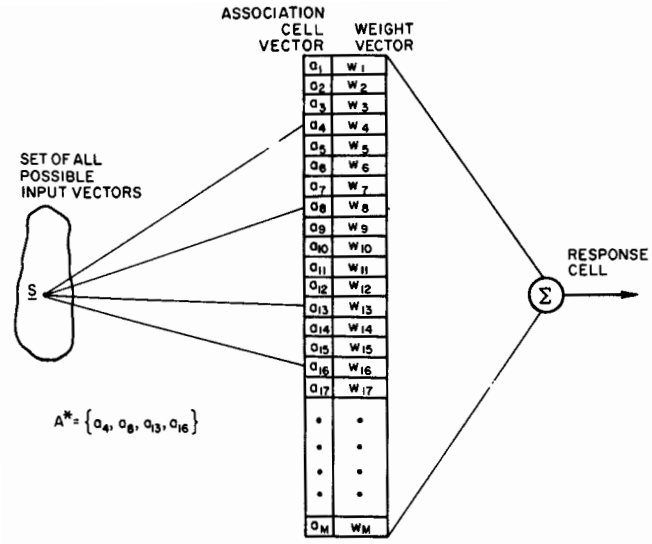

Fig. 2 A Perceptron in which an input vector S is mapped onto an association cell vector A. A^* is the set of non-zero elements in A. The response cell sums all weights attached to association cells in A^*.

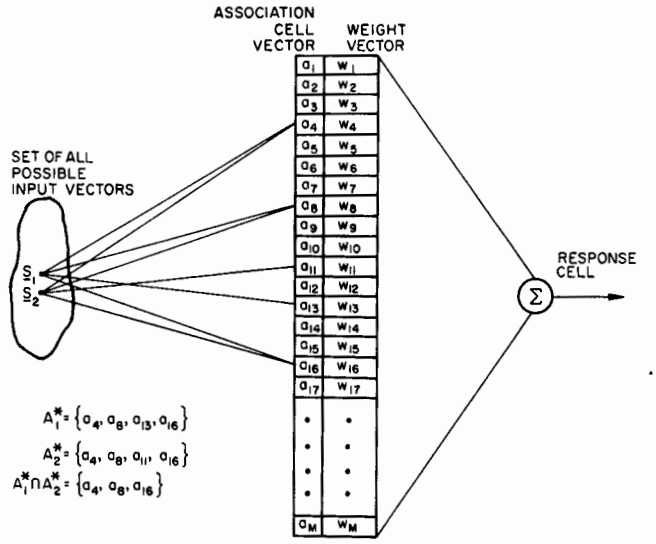

Fig. 3 The Perceptron's ability to generalize derives from the overlap, or intersection $A^*_1 \Lambda A^*_2$. If the intersection $A^*_1 \Lambda A^*_2$ is null, the response of the Perceptron to the two input vectors will be independent and generalization will not occur. If the intersection $A^*_1 \Lambda A^*_2$ is not null, the response cell will be affected in the same way for both input patterns S_1 and S_2 by all weights connected to association cells in $A^*_1 \Lambda A^*_2$.

$$\mathbf{S} = \{\text{sensory input vectors}\}$$

$$\mathbf{A} = \{\text{association cell vectors}\}$$

$$\mathbf{P} = \{\text{response output vectors}\}$$

The function f is generally fixed, but the function g depends on the values of weights which may be modified during the data storage (or training) process.

When an input vector $\mathbf{S} = (s_1, s_2, \ldots, s_N)$ is presented to the sensory cells, it is mapped into an association cell vector $\mathbf{A}$. Define A^* to be the set of active or nonzero elements of $\mathbf{A}$ as shown in Fig. 2. The response cell sums the values of the weights attached to active association cells to produce the output vector $\mathbf{P}$. Only the non-zero elements comprising A^* affect this sum. The input vector $\mathbf{S}$ can be considered an address, and the response vector $\mathbf{P}$ the contents of that address. If for any input $\mathbf{S}$, it is desired to change the contents $\mathbf{P}$, then one merely adjusts the weights attached to association cells in A^*.

Since a Perceptron does not have sufficient association cells so that a unique cell or group of cells can be reserved for every possible input pattern, individual association cells will typically be activated by many different input patterns. This leads to overlap, or cross-talk, between input patterns which in some cases is beneficial and in other cases leads to serious problems.

Consider, for example, the two input patterns $\mathbf{S}_1$ and $\mathbf{S}_2$ which activate two overlapping sets of association cells A^*_1 and A^*_2 as shown in Fig. 3. If it is desired that the output of the response cell for $\mathbf{S}_1$ be the same as for $\mathbf{S}_2$, then the overlap has decidedly beneficial properties. For example, if the proper response has been stored for $\mathbf{S}_1$, then $\mathbf{S}_2$ will elicit very nearly the same response (differing only by the difference between w_{11} and w_{13}) without any weight adjustment ever having been made for $\mathbf{S}_2$. This property is called generalization, because it is similar to the capacity of a biological organism to generalize from one learning experience to another.

However, if it is desired that the input vector $\mathbf{S}_2$ produce a markedly different response from $\mathbf{S}_1$, then the overlap $A^*_1 \Lambda A^*_2$ creates difficulty. Adjustment of all the weights attached to cells in A^*_2 so as to produce the proper response for $\mathbf{S}_2$ will upset most of the weights which contribute to the output for $\mathbf{S}_1$. This is called learning interference, and is similar to retroactive inhibition experienced by biological organisms when presented with highly similar stimuli for which different responses are required.

Learning interference can be overcome by repeated iteration of the data storage algorithm for $\mathbf{S}_1$ and $\mathbf{S}_2$. Repeated iteration eventually leads to sufficiently large weights being attached to the few association cells that are in A^*_1 or A^*_2 but *not* in the overlap $A^*_1 \Lambda A^*_2$ so that a highly dissimilar output can be obtained for $\mathbf{S}_2$ as opposed to $\mathbf{S}_1$.

In summary, the Perceptron's ability to generalize derives from the overlap, or intersection $A^*_1 \Lambda A^*_2$. If the intersection $A^*_1 \Lambda A^*_2$ is null, the response of the Perceptron to the two input patterns will be independent and generalization will not occur. If the intersection $A^*_1 \Lambda A^*_2$ is not null, the response cell will be affected in the same way for both input patterns $\mathbf{S}_1$ and $\mathbf{S}_2$ by all weights connected to association cells in $A^*_1 \Lambda A^*_2$. Thus, the tendency will be for the response cell to produce a similar output for both $\mathbf{S}_1$ and $\mathbf{S}_2$. The degree to which this tendency affects the response is the degree to which the Perceptron generalizes.

The Perceptron's ability to dichotomize, or produce dissimilar outputs for different input patterns, is derived from the difference between A^*_1 and A^*_2. In general, the smaller the intersection $A^*_1 \Lambda A^*_2$, the easier it is to find a set of weights which will produce a dissimilar output for $\mathbf{S}_1$ as opposed to $\mathbf{S}_2$.

The control function for a manipulator is typically a rather smooth continuous function. This means that for every point in input-space which requires a certain response, there is a small neighborhood of input-space points around that point for which very nearly the same response is required. Within that neighborhood, the control system should produce approximately the same response. However, what a particular joint of a manipulator should do at one of two widely separated points in input-space cannot be predicted from information associated with the other point. Thus, the control function for widely separated input-space points should be independent. This implies that a Perceptron-like controller which generalizes only over a small neighborhood of input-space, and which has good dichotomizing properties for points well separated in input-space would be a good controller for a manipulator.

In more precise terms, if two input vectors $\mathbf{S}_i$ and $\mathbf{S}_j$ have a small input-space distance, then the intersection $A^*_i \Lambda A^*_j$

Table 1

	m										
s	μ_a	μ_b	μ_c	μ_d	μ_e	μ_f	μ_g	μ_h	μ_i	μ_j	μ_k	μ_l
1	1	1	1	1	0	0	0	0	0	0	0	0
2	0	1	1	1	1	0	0	0	0	0	0	0
3	0	0	1	1	1	1	0	0	0	0	0	0
4	0	0	0	1	1	1	1	0	0	0	0	0
5	0	0	0	0	1	1	1	1	0	0	0	0
6	0	0	0	0	0	1	1	1	1	0	0	0
7	0	0	0	0	0	0	1	1	1	1	0	0
8	0	0	0	0	0	0	0	1	1	1	1	0
9	0	0	0	0	0	0	0	0	1	1	1	1

An example of a $s \rightarrow m$ mapping from the decimal variable s to the binary variable m.

Table 2

s	m^*
1	a, b, c, d
2	e, b, c, d
3	e, f, c, d
4	e, f, g, d
5	e, f, g, h
6	i, f, g, h
7	i, j, g, h
8	i, j, k, h
9	i, j, k, l

An abbreviated form of the $s \rightarrow m$ mapping in Table 1

should be large. Conversely if $\mathbf{S}_i$ and $\mathbf{S}_j$ have large input-space distance, $A^*_i \Lambda A^*_j$ should be small. Input-space distance is the R-ary equivalent of Hamming distance between binary vectors, and will be defined as

$$H_{ij} = \sum_{k=1}^{N} |s_{ij} - s_{jk}|$$

where s_{ik} are the components of the input vector $\mathbf{S}_i$ and N is the dimensionality of $\mathbf{S}_i$:

$$\mathbf{S}_i = (s_{i1}, s_{i2}, \ldots s_{iN})$$

$$\mathbf{S}_j = (s_{j1}, s_{j2}, \ldots s_{jN})$$

If H_{ij} is small, $A^*_i \Lambda A^*_j$ should be large. As H_{ij} grows larger, $A^*_i \Lambda A^*_j$ should get smaller until, at some value of H_{ij}, $A^*_i \Lambda A^*_j$ should be null.

This characteristic cannot be achieved in the classical Perceptron where for any input $\mathbf{S}$ the number of elements in A^* is usually a large percentage of the total number of association cells. In the classical Perceptron $A^*_i \Lambda A^*_j$ is large and generalization is good for almost every pair of input patterns $\mathbf{S}_i$ and $\mathbf{S}_j$. In order to achieve the situation where $A^*_i \Lambda A^*_j$ is null for almost all $\mathbf{S}_i$ and $\mathbf{S}_j$ except those with a small input-space distance, it is necessary to make the number of association cells much larger than $|A^*|$. $|A^*|$ is defined as the number of elements in A^*. In the cerebellum it is believed that for any input vector $\mathbf{S}$, $|A^*|$ is less than one percent of the total number of association cells [9]. $|A_p|$ is the number of association cells physically implemented by CMAC. $|A_p|$ is typically chosen as at least 100 times $|A^*|$.

This raises the question of whether it is possible to have a unique mapping $\mathbf{S} \rightarrow \mathbf{A}$ where $|A_p| = 100\,|A^*|$. If each variable in $\mathbf{S}$ can take on R different values, then there are R^N possible input patterns. If $|A_p| = 100\,|A^*|$, then the number of possible ways to select $|A^*|$ active cells out of $|A_p|$ potentially active cells is the number of combinations of $|A_p|$ things taken $|A^*|$ at a time,

$$\text{or } \binom{VU}{U} \text{ where } U = |A^*| \text{ and } V = \frac{|A_p|}{|A^*|}$$

$$\binom{VU}{U} = \frac{(VU)!}{U!(VU-U)!} > \frac{(VU-U)^U}{U!} > \frac{(VU-U)^U}{U^U} = (V-1)^U \qquad (6)$$

Therefore, so long as $R^N < 99^{|A^*|}$, it is theoretically possible to find a unique $\mathbf{S} \rightarrow \mathbf{A}$ mapping where $|A_p| = 100\,|A^*|$.

The CMAC System

Having established what kind of characteristics the mapping $\mathbf{S} \rightarrow \mathbf{A}$ should have, and having determined under what circumstances the mapping is possible, the question is then to devise an algorithm which will actually produce the desired results.

The CMAC Mapping Algorithm. The CMAC algorithm functions by breaking the $\mathbf{S} \rightarrow \mathbf{A}$ mapping into two sequential mappings

$$\mathbf{S} \rightarrow \mathbf{M} \qquad (7)$$

and then

$$\mathbf{M} \rightarrow \mathbf{A}$$

Each R-ary variable s_i in the input vector $\mathbf{S} = (s_i, s_2, \ldots s_N)$ is first converted into a binary variable m_i according to the following rule:

1 Each digit of the binary variable m_i must have a value of "1" over one and only one interval within the range of s_i and must be "0" elsewhere. For example, in Table 1, the digit μ_f is "1" over the interval $3 \leq s \leq 6$ and zero elsewhere.

2 There are always $|A^*|$ "1"s in the binary variable m_i for every value of the variable s_i. In other words, $|m^*| = |A^*|$ where m^* is the set of binary digits in m which are in the "1" state. In the mapping shown in Table 1, $|m^*| = 4$.

3 The names of the subscripts of the binary digits in m^* are then tabulated against the values of the variables s. The order of the subscripts is arbitrary except that a subscript must never change its position in the order. This is illustrated in Table 2.

For the one-dimensional case shown in Table 2, the relationship between input-space distance H_{ij}, and the number of elements in the intersection $A^*_i \Lambda A^*_j$ can be described by the formula

$$|A^*_i| - |A^*_i \Lambda A^*_j| = H_{ij} \text{ for } H_{ij} \leq |A^*_i| \qquad (9)$$

For example, if $\mathbf{S}_1 = (1)$ and $\mathbf{S}_2 = (3)$,
then $A^*_1 \Lambda A^*_2 = \{c, d\}$.
Now since $|A^*_1| = 4$
and $|\{c, d\}| = 2$, then

$$|A^*_1| - |A^*_1 \Lambda A^*_2| = 2 = H_{12}$$

Multidimensional Mappings. The complete mapping $\mathbf{S} \rightarrow \mathbf{M}$ consists of N individual mappings $s_i \rightarrow m^*_i$ for all the variables in the input vector $\mathbf{S} = (s_1, s_2, \ldots, s_N)$.

$$\mathbf{S} \rightarrow \mathbf{M} = \begin{cases} s_1 \rightarrow m_1^* \\ s_2 \rightarrow m_2^* \\ \cdot \quad \cdot \\ \cdot \quad \cdot \\ \cdot \quad \cdot \\ s_N \rightarrow m_N^* \end{cases} \qquad (10)$$

Consider for example a two-dimensional input vector $\mathbf{S} = (s_1, s_2)$. Assume $1 \leq s_1 \leq 5$ and $1 \leq s_2 \leq 7$. Again we will choose $|A^*| = 4$.
First, make two mappings

$$s_1 \rightarrow m^*_1 \text{ and } s_2 \rightarrow m^*_2$$

Table 3

s_1	m^*_1
1	*A, B, C, D*
2	*E, B, C, D*
3	*E, F, C, D*
4	*E, F, G, D*
5	*E, F, G, H*

s_2	m^*_2
1	*a, b, c, d*
2	*e, b, c, d*
3	*e, f, c, d*
4	*e, f, g, d*
5	*e, f, g, h*
6	*i, f, g, h*
7	*i, j, g, h*

An example of a pair of mappings $s_1 \rightarrow m^*_1$ and $s_2 \rightarrow m^*_2$ for a two-dimensional input vector $\mathbf{S} = (s_1, s_2)$

as shown in Table 3.

In the case where **S** has two or more dimensions, A^* is derived by concatenation of the corresponding elements in each of the m^*_i as shown in Table 4. For example, if $s_1 = 2$ and $s_2 = 4$, A^* is formed by concatenation of corresponding elements in $m^*_1 = \{E, B, C, D\}$ and $m^*_2 = \{e, f, g, d\}$. Thus $A^* = \{Ee, Bf, Cg, Dd\}$ for $\mathbf{S} = (2, 4)$. A complete representation of A^* for all values of **S** is shown in Table 4. Note that in the two-dimensional matrix, approximately the same relationship exists between input-space distance H_{ij}, and the number of elements in $|A^*| - |A^*_1 \Lambda A^*_2|$ as in the one-dimensional case. For example, between $\mathbf{S}_1 = (3, 5)$ and $\mathbf{S}_2 = (3, 2)$, the input-space distance $H_{12} = 3$. The intersection $A^*_1 \Lambda A^*_2$ has one element $\{Ee\}$, and $|A^*| = 4$. Thus

$$|A^*| - |A^*_1 \Lambda A^*_2| = 3 = H_{12}$$

However, examination of the matrix in Table 4 reveals that diagonally adjacent A^*'s sometimes differ by two elements and sometimes only one, whereas input-space distance for diagonally adjacent vectors always computes as two. For example, in Table 4 where $\mathbf{S}_1 = (4, 3)$ and $\mathbf{S}_2 = (3, 4)$, the input-space distance is 2, whereas $|A^*| - |A^*_1 \Lambda A^*_2| = 1$. Similarly, the vectors $\mathbf{S}_1 = (1, 1)$ and $\mathbf{S}_2 = (4, 4)$ have an input-space distance of 6, whereas they are only 3 positions distant along the diagonal, and $|A^*| - |A^*_1 \Lambda A^*_2| = 3$. On the other hand, along some diagonals, such as $\mathbf{S}_1 = (3, 3)$, $\mathbf{S}_2 = (4, 2)$, the input-space distance and $|A^*| - |A^*_1 \Lambda A^*_2|$ are the same. It should be noted, however, that $|A^*| - |A^*_1 \Lambda A^*_2|$ never decreases as the input-space distance from $\mathbf{S}_1$ to $\mathbf{S}_2$ increases. This implies that the number of elements in the intersection $A^*_1 \Lambda A^*_2$ decreases monotonically as the similarity between $\mathbf{S}_1$ and $\mathbf{S}_2$ decreases. This is precisely the behavior which is desirable for the $f:\mathbf{S} \rightarrow \mathbf{A}$ mapping in a manipulator control system because it produces conditions conducive to generalization between input-space vectors which are in the same neighborhood, and allows good dichotomizing between input-space vectors which are not in the same neighborhood.

Table 5

s_2						
7	0	0	0	0	0	
6	1	1	0	0	0	
5	1	2	1	1	1	
4	2	3	2	2	1	
3	3	4	3	2	1	
2	2	3	3	2	1	
1	2	2	2	1	0	
	1	2	3	4	5	s_1

A diagram of the amount of overlap $|A^*_1 \Lambda A^*_2|$ between the vector $\mathbf{S}_1 = (2, 3)$ and all other vectors $\mathbf{S}_2 = (i, j)$ within the range of the input variables. This overlap diagram corresponds to the particular $f:\mathbf{S} \rightarrow \mathbf{A}$ mapping defined in Table 4.

This $f:\mathbf{S} \rightarrow \mathbf{A}$ transformation can be executed on input vectors of any dimension $\mathbf{S} = (s_1, s_2, \ldots s_N)$. Each input variable s_i is first transformed into a set of subscript names m^*_i. Then a set of active associate cells A^* is formed by concatenation of the corresponding elements in all of the m^*_i. The result is that the number of elements in the intersection $A^*_1 \Lambda A^*_2$ is roughly proportional to the closeness in input-space of two input vectors $\mathbf{S}_1$ and $\mathbf{S}_2$ regardless of the dimensionality of the input.

Shaping Input-Space Neighborhoods. We can define two input vectors $\mathbf{S}_1$ and $\mathbf{S}_2$ to be in the same neighborhood if $A^*_1 \Lambda A^*_2$ is not null. For example, in Table 2, the size of a neighborhood in input-space is 3; i.e., $m^*_1 \Lambda m^*_2$ is not null for any two values of s_1 and s_2 such that $|s_1 - s_2| \leq 3$. In more than one dimension, the boundaries of a neighborhood become more complicated. For example, Table 5 is a diagram of the amount of overlap $|A^*_1 \Lambda A^*_2|$ between the vector $\mathbf{S}_1 = (2, 3)$ and $\mathbf{S}_2 = (i, j)$, i.e., any other vector within the range of the input variables. The neighborhood of $\mathbf{S}_1 = (2, 3)$ is composed of all points where $|A^*_1 \Lambda A^*_2| \neq 0$.

The size of a neighborhood obviously depends on the number of elements in the set A^*. It also depends on the resolution with which each $s_i \rightarrow m_i$ mapping is carried out. The resolution of each $s_i \rightarrow m_i$ mapping is entirely at the discretion of the control system designer. For example, the size of a neighborhood in the one-dimensional mapping in Table 6 is 1.5. (Compare Table 6 to Table 2.)

In multidimensional input-space, the neighborhood about any

Table 4

s_2						
i, j, g, h	7	*Ai, Bj, Cg, Dh*	*Ei, Bj, Cg, Dh*	*Ei, Fj, Cg, Dh*	*Ei, Fj, Cg, Dh*	*Ei, Fj, Gg, Hh*
i, f, g, h	6	*Ai, Bf, Cg, Dh*	*Ei, Bf, Cg, Dh*	*Ei, Ff, Cg, Dh*	*Ei, Ff, Gg, Dh*	*Ei, Ff, Gg, Hh*
e, f, g, h	5	*Ae, Bf, Cg, Dh*	*Ee, Bf, Cg, Dh*	*Ee, Ff, Cg, Dh*	*Ee, Ff, Gg, Dh*	*Ee, Ff, Gg, Hh*
e, f, g, d	4	*Ae, Bf, Cg, Dd*	*Ee, Bf, Cg, Dd*	*Ee, Ff, Cg, Dd*	*Ee, Ff, Gg, Dd*	*Ee, Ff, Gg, Hd*
e, f, c, d	3	*Ae, Bf, Cc, Dd*	*Ee, Bf, Cc, Dd*	*Ee, Ff, Cc, Dd*	*Ee, Ff, Gc, Dd*	*Ee, Ff, Gc, Hd*
e, b, c, d	2	*Ae, Bb, Cc, Dd*	*Ee, Bb, Cc, Dd*	*Ee, Fb, Cc, Dd*	*Ee, Fb, Gc, Dd*	*Ee, Fb, Gc, Hd*
a, b, c, d	1	*Aa, Bb, Cc, Dd*	*Ea, Bb, Cc, Dd*	*Ea, Fb, Cc, Dd*	*Ea, Fb, Gc, Dd*	*Ea, Fb, Gc, Hd*
		1	2	3	4	5
		A, B, C, D	*E, B, C, D*	*E, F, C, D*	*E, F, G, D*	*E, F, G, H*

s_1

The set A^* formed by concatenation of the corresponding elements of m^*_1 and m^*_2 from the two dimensional input vector $\mathbf{S} = (s_1, s_2)$ defined in Table 3. At each point in the two-dimensional input-space, the four element set A^* has a unique composition.

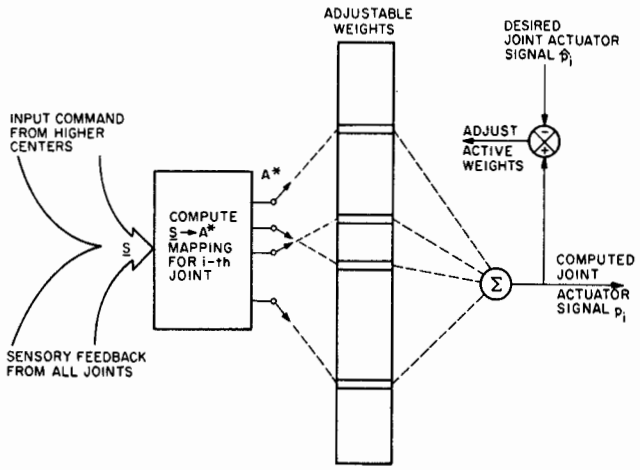

Fig. 4 A block diagram of the CMAC system for a single joint. The vector S is presented as input to all joints. Each joint separately computes an S → A* mapping and a joint actuator signal p_i. The adjustable weights for all joints may reside in the same physical memory.

input vector **S** may be elongated or shortened along different coordinate axes by using different resolution $s_i \rightarrow m^*_i$ mappings. If, for example, $s_x \rightarrow m^*_x$ is a high resolution mapping, and $s_y \rightarrow m^*_y$ is a low resolution mapping, then the input-space neighborhood about **S** will be elongated along the y dimension and shortened along the x dimension. A high resolution mapping $s_i \rightarrow m^*_i$ makes the composition of the set A^* strongly dependent on the input variable s_i (i.e., only a small change in s_i is required to produce a change in one or more of the elements in A^*). A low resolution mapping $s_j \rightarrow m^*_j$ makes the composition of the A^* set weakly dependent on the variable s_j (i.e., a large change in s_j is required to produce a change in any of the elements of A^*). If the resolution of the $s_j \rightarrow m_j$ mapping is made low enough, the composition of the set A^* will be independent of the value of s_j (i.e., no amount of change in s_j will affect A^*). In the case where A^* is independent of s_j, it can be said that the input-space neighborhood is infinite in extent along the j axis.

It is possible to construct $s_i \rightarrow m^*_i$ mappings which are non-uniform (i.e., high resolution over some portions of the range along the ith axis, and low resolution over other portions of the same axis). By this means, neighborhoods can be made different sizes and different shapes in various regions of the input-space. This feature has useful practical applications which will be discussed later.

Mapping into a Memory of Practical Size. Thus far, we have described a means of performing the mapping f:**S** → **A** in a manner which is well suited to producing generalization where generalization is desired, and dichotomization where that is desired. We have not, however, explained how this transformation can be accomplished with a reasonable number of association cells. The concatenation of subscript names m^*_i produces a potentially enormous number of association cell names. If each variable in **S** = (s_1, s_2, ... s_N) has R distinguishable values, then there are R^N distinguishable points in input-space. After the $s_i \rightarrow m^*_i$ mappings, concatenation of the m^*_i sets to obtain A^* yields a potential number of association cell names on the same order of magnitude as R^N. For any practical manipulator control problem, the number of input variables N is likely to exceed 10, and the number of distinguishable values R of each variable will probably be 30 or more. The number 30^{10} is clearly an impossibly large number of association cells for any practical control device.

If, however, it is not required for input vectors outside of the same neighborhood to have zero overlap, but merely a vanishingly small probability of significant overlap, then it is no longer necessary to have R^N association cells. Assume that an additional mapping $A \rightarrow A_p$ is performed such that the R^N association cells in the very large set A are mapped onto a much smaller, physically realizable set A_p. One way in which this can be done is by hash-coding [14].

Table 6

s	m^*
1.0	a, b, c, d
1.5	e, b, c, d
2.0	e, f, c, c
2.5	e, f, g, d
3.0	e, f, g, h
3.5	i, f, g, h
4.0	i, j, g, h
4.5	i, j, k, h
5.0	i, j, k, l

A mapping $s \rightarrow m^*$ where the resolution on the input variable s is 0.5 units

Hash-coding is a commonly used computer technique for reducing the amount of memory required to store sparse matrices and other data sets where a relatively small amount of data is scattered over a large number of memory locations. Hash-coding operates by taking the address of where a piece of datum is to be stored in the larger memory and using it as an argument in a routine which computes an address in the smaller memory. For example, any address in the larger memory might be used as an argument in a pseudorandom number generator whose output is restricted to the range of integers represented by the addresses in the small memory. The result is a many-into-few mapping of locations in the larger memory onto locations in the smaller. Any association cell name (address) in A can be used as the argument in a hash-coding routine to find its counterpart in A_p. The number of association cells in A_p can be chosen arbitrarily equal to the size of the physically available memory. In practice A_p may be orders-of-magnitude smaller than A. Thus, the $A \rightarrow A_p$ mapping is a many-into-few mapping.

For example in Table 4, at each point in input-space A^* is composed of four elements each of which could be represented by two BCD characters of six bits each. Thus, each element in A^* can be represented as a 12 bit number. Assume that the amount of memory available for A_p is only 16 locations. One method of hash-coding would then be to use these 12 bit numbers to address a table of 4096 four bit random numbers corresponding to addresses in A_p. The result would be an $A \rightarrow A_p$ mapping of the elements in Table 4 such that A_p contains only 16 locations.

The many-into-few property of the hash-coding procedure leads to "collision" problems when the mapping routine computes the same address in the smaller memory for two different pieces of data from the larger memory. Collisions can be minimized if the mapping routine is pseudorandom in nature so that the computed addresses are as widely scattered as possible. Nevertheless, collisions are eventually bound to occur, and a great deal of hash-coding theory is dedicated to the optimization of schemes to deal with them.

CMAC, however, can simply ignore the problem of hashing collisions because the effect is essentially identical to the already existing problem of cross-talk, or learning interference, which is handled by iterative data storage.

Assume, for example, that the actual number of memory locations available is 2000, i.e., $|A_p| = 2000$. Each association cell name in A^* is then mapped into one of the 2000 available addresses in A_p by a deterministic, but pseudorandom hash-coding routine. Each cell in A^* has equal probability of being mapped into any one of the 2000 cells in A_p. This, of course, makes it possible for two or more different cells in A^* to be mapped into the same cell in A_p. If, for example, $|A_p| = 2000$

and $|A^*| = 20$, then the probability of two or more cells in A^* being mapped into the same cell in A_p is approximately

$$\frac{1}{2000} + \frac{2}{2000} + \frac{3}{2000} + \cdots + \frac{19}{2000}$$

or about 0.1. In practice this is not a serious problem so long as the probability is rather low, since it merely means that any weight corresponding to a cell in A^*_p which is selected twice will be summed twice by the response cell. The loss is merely that of available resolution in the value of the output.

A somewhat more serious problem in the $A \rightarrow A_p$ mapping is that it raises the possibility that two input vectors $\mathbf{S}_1$ and $\mathbf{S}_2$ which are outside of the same neighborhood in input-space might have overlapping sets of association cells in A_p. This introduces interference in the form of unwanted generalization between input vectors which lie completely outside the same input-space neighborhood. The effect, however, is not significant so long as the overlap is not large compared to the total number of cells in A^*. In other words, spurious overlap is not a practical problem as long as $|A^*_{p1} \Lambda A^*_{p2}| << |A^*|$ when $A^*_1 \Lambda A^*_2 = \phi$. If, for example, we choose $|A^*| = 20$ and $|A_p| = 2000$, then for two input vectors chosen at random such that $A^*_1 \Lambda A^*_2 = \phi$, the probability of various amounts of overlap $A^*_{p1} \Lambda A^*_{p2}$ can be computed from the binomial expansion $(p + q)^{20}$, where $p = 1 - q$ and $q = \frac{20}{2000}$. The probability that

$|A^*_{p1} \Lambda A^*_{p2}| = 0$ is 0.818

$|A^*_{p1} \Lambda A^*_{p2}| = 1$ is 0.165

$|A^*_{p1} \Lambda A^*_{p2}| = 2$ is 0.016

$|A^*_{p1} \Lambda A^*_{p2}| \geq 3$ is 0.001

For practical purposes, two input vectors can be considered to be outside the same neighborhood if they have no more than one active association cell in common. Thus, in practice 100 $|A^*|$ association cells will perform nearly as well as R^N association cells. If $|A_p|$ is made equal to 1000 $|A^*|$, the overlap problem virtually disappears entirely. For example, if $|A^*| = 20$ and $|A_p| = 20{,}000$, the probability of two or more cells in A^* being mapped into the same cell in A_p is only 0.01, and the probability of overlap between random input vectors is as follows: The probability that

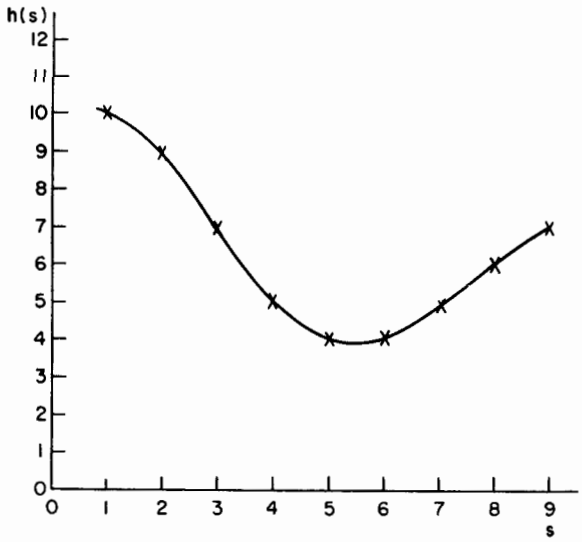

Fig. 5 A one-dimensional function $h(s)$

$|A^*_{p1} \Lambda A^*_{p2}| = 0$ is 0.9802

$|A^*_{p1} \Lambda A^*_{p2}| = 1$ is 0.0196

$|A^*_{p1} \Lambda A^*_{p2}| \geq 2$ is 0.0002

It is desirable to keep $|A^*|$ as small as possible in order to minimize the amount of computation required. It is also desirable to make the ratio $\frac{|A^*|}{|A_p|}$ as small as possible so that the probability of overlap between widely separated **S** patterns is minimized. $|A_p|$, of course, is limited by the physical size of the available memory. However, $|A^*|$ must be large enough so that generalization is good between neighborhood points in input-space. This requires that no individual association cell contribute more than a small fraction of the total output. If $|A^*| \geq 20$, each association cell contributes on the average 5 percent or less of the output.

Computing the Output. The $|A^*|$ addresses computed by the $A \rightarrow A_p$ hash coding procedure point to variable weights which are summed by the response cells. The linear sum of these weights (perhaps multiplied by an appropriate scaling factor) is then an output driving signal p_i used to power the ith joint actuator of the manipulator. The functional relationship

$$\mathbf{P} = h(\mathbf{S}) \tag{11}$$

is the overall transfer function of the CMAC controller. The individual components of $\mathbf{P} = (p_1, p_2, p_3, \ldots p_L)$ are the output drive signals to each individual joint. In general, each p_i is a different function of the input vector **S**

$$p_i = h_i(\mathbf{S}) \quad i = 1, \ldots L \text{ where } L \text{ is the number of joint actuators} \tag{12}$$

Fig. 4 shows a block diagram of the CMAC system for a single joint. The components in this diagram (except for the adjustable weights) are duplicated for each joint of the manipulator which needs to be controlled. Typically the $\mathbf{S} \rightarrow A^*$ mapping is different for each joint in order to take into account the different degrees of dependence of each p_i on the various input parameters s_j. For example, the elbow control signal is more strongly dependent on position and rate information from the elbow than from the wrist, and vice versa.

The values of the weights attached to the association cells determine the values of the transfer functions at each point in input-space. Consider, for example, the one-dimensional function shown in Fig. 5. If the mapping f:**S** $\rightarrow$ **A** is carried out as shown in Table 7, and the weights connected to the association cells have the values shown in Table 8, then $h(s)$ is the function in Fig. 5. A function of two variables can be represented in a similar way. Consider Table 4. Each square in the matrix corresponds to a location containing a value of the function $h(s_1, s_2)$. If, for example, $h(3, 4) = 7$, this would be satisfied whenever

Table 7

$\mathbf{S} = (s)$	A^*	$\mathbf{P} = h(\mathbf{S})$
1	*A, B, C, D*	10
2	*E, B, C, D*	9
3	*E, F, C, D*	7
4	*E, F, G, D*	5
5	*E, F, G, H*	4
6	*I, F, G, H*	4
7	*I, J, G, H*	5
8	*I, J, K, H*	6
9	*I, J, K, L*	7

A $\mathbf{S} \rightarrow A^*$ mapping for the one-dimensional input vector $\mathbf{S} = (s)$ and a table reference representation of the function $h(s)$ from Fig. 5

Table 8

Association cell	Weight
A	3
B	3
C	2
D	2
E	2
F	1
G	0
H	1
I	2
J	2
K	1
L	2

A set of weights attached to association cells *A* through *L* which produce the function $h(s)$ of Fig. 5, given the $\mathbf{S} \rightarrow A^*$ mapping shown in Table 7

the association cells *Ee*, *Ff*, *Cg*, *Dd* have attached to them weights summing to 7. For any function $h_i(s_1, s_2, \ldots, s_N)$ the problem then is to find a suitable set of weights which will represent the function over the range of the arguments.

Inputs From Higher Levels. *Commands from higher centers* are treated by CMAC in exactly the same way as input variables from any other source. The higher level command signals appear as one or more variables in the input vector **S**. They are mapped with an $s_i \rightarrow m^*_i$ mapping and concatenated like any other variable affecting the selection of A^*_p. The result is that input signals from higher levels, like all other input variables, affect the output and thus can be used to control the transfer function $\mathbf{P} = h(\mathbf{S})$. If, for example, a higher level command signal x changes value from x_1 to x_2, the set $m^*_{x_1}$ will change to $m^*_{x_2}$. If the change in x is large enough (or the $x \rightarrow m^*_x$ mapping is high enough resolution) that $m^*_{x_1} \wedge m^*_{x_2} = \phi$, then the concatenation process will make $A^*_{x_1} \wedge A^*_{x_2} = \phi$.

Thus, by changing the signal x, the higher level control signal can effectively change the CMAC transfer function. This control can either be discrete (i.e., x takes only discrete values x_i such that $m^*_{x_i} \wedge m^*_{x_j} = \phi$ for all $i \neq j$), or continuously variable (i.e., x can vary smoothly over its entire range). An example of the types of discrete commands which can be conveyed to the CMAC by higher level input variables are "reach," "pull back," "lift," "slap" (as in swatting a mosquito), "twist," "scan along a surface," etc. Experimental results of CMAC operating with a "slap" command are reported in reference [11].

An example of the types of continuously variable commands which might be conveyed to the CMAC are velocity vectors describing the motion components desired of the manipulator end-effector. Three higher level input variables might be $\dot{x}$, $\dot{y}$, z, representing the commanded velocity components of a manipulator end-effector in a coordinate system defined by some work space. If $\dot{x}$, $\dot{y}$, and z are all zero, the transfer function for each joint actuator should be whatever necessary to hold the manipulator stationary. If the higher center were to send $\dot{x} = 10$, $\dot{y} = 0$, $z = -3$, then the transfer function for each joint should be such that the joints would be driven in a manner so as to produce an end-effector velocity component of 10 in the x direction, 0 in the y direction, and -3 in the z direction.

The CMAC processor for each joint is thus a servo control system. The $\mathbf{S} \rightarrow A^*_p$ mapping, together with adjustment of the weights, define the effect of the various input and feedback variables on the control system transfer function. Inputs from higher centers call for specific movements of the end point. The CMAC weights are then adjusted so as to carry out those movements under feedback control.

Summary

CMAC computes control functions by referring to a table rather than by solution of analytic equations or by conventional analog servo techniques.

Functional values are stored in a distributed fashion such that the value of the function at any point in input-space is derived by summing the contents over a number of memory locations.

The unique feature of CMAC is the mapping algorithm which converts distance between input vectors into the degree of overlap between sets of addresses where the functional values are stored. CMAC is thus a memory management technique which causes similar inputs to tend to generalize so as to produce similar outputs; yet dissimilar inputs result in outputs which are independent.

There, of course, remains much work to be done in determining adequate memory size, computation cycle time, training requirements, and accuracy for practical applications. These parameters are all situation dependent and it remains to be seen which situations will be most suitable to the CMAC approach.

References

1 Johnsen, E. G., and Corliss, W. R., *Teleoperators and Human Augmentation*, NASA SP-5047, 1967.

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

3 Ashley, J. R., and Pugh, A., "Logical Design of Control Systems for Sequential Mechanisms," *International Journal of Production Research* (*I.P.E.*), Vol. VI, November 4, 1968.

4 Paul, R., "Modeling, Trajectory Calculation and Servoing of A Computer Controlled Arm," PhD thesis, Stanford University, Aug. 1972.

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

6 First Annual Report for the Development of Multi-Moded Remote Manipulator Systems, Charles Stark Draper Laboratory (Division of Massachusetts Institute of Technology), Report C-3790.

7 Kahn, M. E., and Roth, B., "The Near Minimum-Time Control of Open Loop Articulated Kinematic Chains," JOURNAL OF DYNAMIC SYSTEMS, MEASUREMENT, AND CONTROL, TRANS. ASME, Series G, Vol. XCIII, No. 3, Sept. 1971, pp. 164–172.

8 Grossman, S. P., "The Motor System and Mechanics of Basic Sensory-Motor Integration," *Textbook of Physiological Psychology*, Wiley, New York, 1967, Chapter 4.

9 Albus, J. S., "A Theory of Cerebellar Function," *Mathematical Biosciences*, Vol. X, 1971, pp. 25–61.

10 Albus, J. S., "A Robot Conditioned Reflex System Modeled After the Cerebellum," *Proceedings Fall Joint Computer Conference*, Vol. XLI, 1972, pp. 1095–1104.

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

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

13 Nilsson, N. J., *Learning Machines*, NcGraw-Hill, New, York, 1965.

14 Knuth, D., "Sorting and Searching," *The Art of Computer Programming*, Vol. 3, Addison Wesley, Menlo Park, Calif., 1973 p. 506.