Neighborhood Sequential and Random Training Techniques for CMAC

David E. Thompson, Associate, IEEE, and Sunggyu Kwon

Abstract—An adaptive control algorithm based on Albus' CMAC (Cerebellar Model Articulation Controller) was studied with emphasis on how to train CMAC systems. Two training techniques—neighborhood sequential training and random training, have been devised. These techniques were used to generate mathematical functions, and both methods successfully circumvented the training interference resulting from CMAC's inherent generalization property. In the neighborhood sequential training method, a strategy was devised to utilize the discrete, finite state nature of the CMAC's address space for selecting points in the input space which would train CMAC systems in the most rapid manner possible. The random training method was found to converge on the training function with the greatest precision, although it requires longer training periods than the neighborhood sequential training method to achieve a desired performance level.

I. INTRODUCTION

Albus proposed a control method that is based on the principles of the cerebellum's motor behavior; he called the control system CMAC [3]. CMAC as a control system does not operate by mathematically analyzing the dynamics of the control problem and then solving equations. It computes control functions by referring to a memory table where the control functions are stored. The memory table is established by storing the mapping between input values and the control function. In the procedure, CMAC learns the control functions over the input space and modifies or generalizes its internal data obtained from previous practice at other, similar, input vectors. Once the memory table is built, the output function value is obtained directly by arithmetic summation of the contents over a number of memory elements which are addressed by the input vector. These operations are both simple and fast—a major asset of CMAC.

The CMAC memory addressing algorithm is composed of a series of mappings. The mapping algorithm, based on the information processing characteristics of the cerebellum, addresses memory using a hashing function [9]. Also, the memory addressing algorithm takes advantage of the continuous nature of control functions whose values vary smoothly over the input space. The memory addressing algorithm causes similar inputs to tend to generalize and produce similar outputs; yet dissimilar inputs result in outputs which are independent. Thus, the memory addressing algorithm provides CMAC with the capabilities of learning and adaptation. In order to fully comprehend the CMAC memory addressing algorithm, it is very helpful to understand the biological and anatomical concepts of the brain which are then extended to the model. A comprehensive theory is described in detail in Albus' work [1].

The CMAC control method has been used in various applications since Albus formulated CMAC in 1975. In most CMAC applications to manipulator control, a trajectory is defined for a particular motion and the CMAC is trained along this path. The training sessions are continued along the trajectory until the desired performance is achieved. Albus applied CMAC to control a 3-axis master-slave arm [2]. The arm was trained until the error corrections for learning a specified motor task were sufficiently small. Camana studied the dynamic control of a two-degree-of-freedom biped model [6]. Training sessions were continued for each defined motion until the error was less than a prescribed value. Albus [5] drove a seven-degree-of-freedom manipulator arm a predetermined number of times along specific trajectories to train the manipulator. Rajadhyaksha [17] developed a unique learning algorithm and optimization technique for CMAC allowing a two-dimensional multilink manipulator to be trained to function in its workspace, avoiding collisions with obstacles to move from position to position on command. Mangelvedhakar [12] attempted to apply a hierarchal pattern recognizer to video pattern recognition, but with little success due to the lack of a standardized manner of classifying the patterns assumed to be the set of visual primitives. Miller et al. [14] proposed a training technique where the controller was trained along three different prescribed paths for controlling a 2-axis articulated robotic manipulator. Also, in the static positioning experiment of a 5-axis articulated robot, a training point for every learning cycle was randomly picked within a restricted region over the whole input space [13]. In 1990, Miller et al. [15] discussed an adaptive control technique using a CMAC neural network and applied the technique to real-time dynamic control experiments which involved learning the dynamics of a 5-axis industrial robot.

Among the many investigations and applications of CMAC, most have dealt with specific applications. Thus, the studies of training techniques for manipulator control have been for a subset of manipulator motions rather than that capable of producing all possible motions. There are several possible general iterative procedures for storage data in CMAC [4]. One simple strategy is to find the input vectors whose output has the greatest error and use these input vectors to correct the...
contents of the corresponding memory elements. The obvious drawback in the maximum-error-correcting method is that this technique requires long memory searches to find the maximum error point for every training session [4].

However, few studies have been done on developing general training techniques. Kwon [10], [11] developed a random training method as a general training method technique and applied the method to the simulation of training a six-axis robotic manipulator in a three-dimensional workspace. The technique requires relatively long training periods to reach a desired performance level. Parks and Milizer [16] discussed two training techniques—cyclic training and random training—in an investigation for the convergence properties of a CMAC learning algorithm. In cyclic training, a cycle defined by an ordered list of training points is repeated over and over again. However, the questions of how to choose the training points in the training cycle and how many times the cycle must be repeated were never discussed. In random training, the training points are chosen in a random fashion.

In this paper, the problems of CMAC training errors are studied and two efficient general CMAC training techniques are presented. The techniques are free from the adverse effects of CMAC generalization and also efficient in training CMAC systems for various applications.

II. CEREBELLAR MODEL ARTICULATION CONTROLLER (CMAC)

The CMAC algorithm is based on the biological structure of the brain [8] and was originally formulated by Albus [1], [3]. Various system parameters, data storage algorithms, and the application of the CMAC to mathematical functions are explained by Albus [4]. The following description is due to [5].

Fundamentally, CMAC is defined by a series of mappings:

\[ S \rightarrow M \rightarrow A \rightarrow p, \]

where \( S \) is the input vector, \( ^1C_j \) are the intermediate variables used to encode \( S, A \) are the elements of some multidimensional memory tables, and \( p \) is the resultant output value. The overall mapping \( S \rightarrow p \) is implemented to represent a function of the form \( p = h(S) \).

It is assumed that the overall mapping function, \( h \), for a CMAC is continuous on closed intervals of the \( N \) input variables. The closed intervals for \( s_1 \) and \( s_2 \) in Fig. 1 are \([0.0, 10.0]\) and \([5.0, 9.0]\). These intervals are discretized into a number of half-closed subintervals by introducing points. The number of subintervals are defined by the width of each subinterval. If the number of subintervals is \( n \), then \( n - 1 \) points, \( x_1, x_2, \ldots, x_{n-1} \), are introduced according to the width of each subinterval. The width of the subinterval is called the \textit{resolution}. In Fig. 1, the resolution of the \( i \)th input variable is shown as \( r_i \).

Generally, the resolution for an input variable is uniform and the \( n - 1 \) points are equally spaced. Any input variable may therefore be represented by one of those subintervals instead of by its own value.

The closed interval of each input variable axis is also spanned by an equal number of layers of a \textit{quantizing function} which is parallel with the input variable axis. Suppose the number of the quantizing functions to be \( K \). Let \( ^jC_j \) be the \( j \)th quantizing function of the \( i \)th input variable, where \( i = 1, 2, \ldots, N, j = 1, 2, \ldots, K \). There are three quantizing functions for each input variable in Fig. 1. Each quantizing function is composed of a number of \textit{intermediate variables}. Each intermediate variable of a quantizing function covers a portion of the closed interval of an input variable axis such that the quantizing function spans the interval completely. The width of each portion is called a \textit{quantizing interval} and is determined by \( (K \times r_i) \) for the \( i \)th input variable. In Fig. 1, the quantizing intervals of \( s_1 \) and \( s_2 \) are 3.0 and 1.5, respectively.

In this mapping, every value of the output variable is represented by \( K \) intermediate variables—one from each quantizing function. This corresponds to encoding the information content of an input value by a set of \textit{K} \textit{mosgy fibers} that are maximally active for this particular input value. These intermediate values are used as pointers into a set of \( K \) arrays which store the expected output from the CMAC.

The \( M \rightarrow A \) mapping is intended to emulate the mapping accomplished by sensory neurons in a biological system. In this mapping, all \( j \)th quantizing functions, \( ^1C_j, ^2C_j, \ldots, ^N C_j \), of the \( N \) input variables are combined to build the \( j \)th \( N \)-dimensional memory table such that \( K \) quantizing functions consist of \( KN \)-dimensional arrays. Each \( i \)th axis (for example, row and column for a 2-dimensional CMAC array) of the \( j \)th array is \( ^1C_j \) if \( A_j \) is the name of the \( j \)th array, then \( S_j(\text{layer}) \) is the \( j \)th \( N \)-dimensional array. The size of each array may be different according to the magnitude of each \textit{c}\textunderscore{}\textit{j}. Then every element of \( A_j \) is identified (or addressed) by an ordered concatenation of \( N \) labels from \( N \) intermediate variables. Thus, \( ^iA_{\text{bd-z}} \) denotes an element of the \( j \)th \( N \)-dimensional array \( A_j \) addressed by the label \( \text{bd-z} \) which is a concatenation of \( N \) labels, \( ^iA_{\text{bd-z}} = A_j(B, d, \ldots, z) \). The numerical value of such an element is referred to as its \textit{weight}.

The final step in this algorithm is the \( A \rightarrow p \) mapping. Numerical values of the \( K \) elements from \( K \) \( N \)-dimensional arrays \( A_j, (j = 1, K) \) are arithmetically summed to produce
an output \( p \)

\[
A \rightarrow p = \begin{cases} 
A_1 \\
A_2 \\
\vdots \\
A_K 
\end{cases} \rightarrow \sum_{j=1}^K a_{B_4\ldots z} = p
\]  

where \( J a_{B_4\ldots z} \) denotes an element of \( j \)th \( N \)-dimensional array \( A_j \). Therefore, \( K \) weights were selected by the input vector to produce an output value. This simulates the cerebellar axons from the granule cells contacting a large number of Purkinje cells through weighted synaptic connections and through inverting interneurons.

This series of mappings is the unique CMAC memory addressing algorithm patterned after the information processing characteristics of the cerebellum. If CMAC is to be used as a control function, the output control values for a finite set of output states are obtained by referring to the table stored in memory rather than by solving analytic equations. In this manner, CMAC requires only simple addressing and addition computations that execute with high speed since it uses stored tabular data.

III. TRAINING OF CMAC

In this work, the term “training” refers to the process of determining the specific relationships between the input vector and an output scalar. The term is not intended to infer that a CMAC can “learn” in the same sense that a biological organism does, but the method only emulates this character in a deterministic manner.

Since the output function values are stored in distributed memory tables in a unique fashion in CMAC systems, developing efficient training techniques is an important problem. In general, the CMAC memory table is initially empty and all the memory elements have a zero value. Such a CMAC would produce a zero response to any input until it has been trained. During training, the CMAC’s output for various inputs is compared to the desired output. The numerical values of all selected memory elements are then adjusted to reduce any errors appearing in the output and the result stored in the memory table. If a full adjustment is made such that the CMAC is trained in a single trial for a given input, then the training is said to have a gain of 1 [3]. Smaller gains produce smaller adjustments, and many more training sessions are required to reduce the error to zero. By repeating this process over and over, the values stored in memory gradually improve since the iterative data storage procedure has been empirically found to converge provided the control function is sufficiently smooth [4].

When a CMAC is trained, the contents of all memory elements whose input points are in the same neighborhood [3] of the training input point are affected. If subsequent training input points are chosen in the same neighborhood as any previous inputs, then some memory elements are repeated and those that were adjusted by the previous training sessions will be improperly altered. This is termed learning interference [5].

In Fig. 1, the input space of a two-dimensional CMAC is shown. For this CMAC, \( K \), the number of quantizing levels for each input variable, is 3. The resolutions of the input variables \( S_1 \) and \( S_2 \), are \( R_1 = 1.0 \) and \( R_2 = 0.5 \), respectively. Table I shows \( N_{ij} \), the numbers of intermediate variables for each level and for each input variable. Fig. 2 shows the planar neighborhood of an input point \( P' \) = (5.5, 7.2) of the two-dimensional CMAC. The number of common memory elements addressed by both the input point \( P' \) and other input points are specified. Three memory elements are addressed by the input point \( P' \) since \( K \) is 3. If a subsequent training input point, \( P'' \) = (7.2, 6.2), is selected, one of the three memory elements addressed by \( P'' \) is common to both and will be altered. Modification of memory elements common to both (in the same neighborhood) will then induce errors in the output for the input point \( P' \). As an example, consider a one-dimensional CMAC trained to generate an arbitrary mathematical function

\[
f(x) = 2 \left[ \sin(x) - \frac{1}{2} \sin(2x) + \frac{1}{3} \sin(3x) \right] \\
\cdot \left[ \cos(x) - \frac{1}{2} \cos(2x) + \frac{1}{3} \cos(3x) \right]
\]  

where \(-180^\circ \leq x \leq 180^\circ \). The choice of this function was merely one of convenience. The only constraint was that the function must remain smooth within the resolution of the CMAC. If training input points (\( x \)-values for this problem) are chosen such that they are in the same neighborhood as the previous training input points, the resulting function and errors are shown in Fig. 3. For this example, the manner in which the training input points were chosen made the CMAC subject to significant learning interference training errors.

Both avoiding learning interference and taking advantage of CMAC’s generalization are essential to the development of efficient CMAC training methods. This idea for developing
efficient CMAC training methods can be elucidated by answering the following questions: 1) where is training to be done in the input space? 2) what criteria should be used for selecting the subsequent training input points after one training session? and 3) how many training sessions must be provided to get a desired level of performance?

IV. NEIGHBORHOOD SEQUENTIAL TRAINING

One technique which we have devised to avoid learning interference is to choose training input points which lie just outside of the neighborhood of the previous training input point. This technique is termed Neighborhood Sequential Training since it is based on adjacent neighborhood sequential training. In the remainder of this work, it will simply be termed as Neighborhood training. Data for each memory element are adjusted only one time throughout the training period since \( A_n^m \cap A_m^n = 0 \) for any training session. \( A_m^n \) denotes the set of memory elements addressed by the \( m \)'th training input point (session) and \( A_n^m \) is for the \( n \)'th training session. This means that no learning interference occurs in this training technique. In addition, once all memory elements are initialized, there is no need to compare the desired output value to the CMAC output value for every new input point since the CMAC output will always be zero for the training session due to the initialization.

For the \( I \)-dimensional CMAC with \( I \) input variables, the input coordinate values of the \( n \)'th input variable, \( S_n \), are discretized for training without learning interference. These input values, \( s_{in} \), are computed by

\[
s_{in} = s_{i0} + (n - 1)Kr_i \quad \text{where } s_{i0} \text{ is the smallest input value of the } i \text{th input variable, } K \text{ is the number of quantizing functions for every input variable, } r_i \text{ is the resolution of } i \text{th input variable, and } N_i \text{ is the number of intermediate variables in the first quantizing level for the } i \text{th input variable.}
\]

For the Neighborhood technique, the total number of input values at which the CMAC is to be trained is determined by the dimension of the input space, the size and the resolution of each input variable axis, and the value of \( K \). For a one-dimensional CMAC, for example, the number of training sessions may be the same as the number of the intermediate variables of the first quantizing function. For any \( I \)-dimensional CMAC, the number of training sessions, \( N_{\text{train}} \), may be determined by

\[
N_{\text{train}} = \prod_{i=1}^{I} (N_{i1}).
\]
TABLE III
RELATIONSHIP AMONG \( K \), \( M \), \( N_{\text{train}} \) AND RMS ERRORS

<table>
<thead>
<tr>
<th>( K )</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M )</td>
<td>32,942</td>
<td>16,744</td>
<td>11,246</td>
<td>7,030</td>
<td>5,652</td>
<td>4,875</td>
<td>3,800</td>
</tr>
<tr>
<td>( N_{\text{train}} )</td>
<td>16,471</td>
<td>4,186</td>
<td>1,891</td>
<td>703</td>
<td>496</td>
<td>325</td>
<td>190</td>
</tr>
<tr>
<td>RMS Errors</td>
<td>0.0001</td>
<td>0.0005</td>
<td>0.0011</td>
<td>0.0031</td>
<td>0.0044</td>
<td>0.0069</td>
<td>0.0123</td>
</tr>
</tbody>
</table>

TABLE IV
RELATIONSHIP AMONG \( K \), \( M \), \( N_{\text{train}} \) AND RMS ERRORS

<table>
<thead>
<tr>
<th>( K )</th>
<th>4</th>
<th>6</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M )</td>
<td>16,744</td>
<td>11,246</td>
<td>7,030</td>
<td>4,875</td>
<td>3,800</td>
</tr>
<tr>
<td>( N_{\text{train}} )</td>
<td>2.8x10^8</td>
<td>2x10^8</td>
<td>600,000</td>
<td>500,000</td>
<td>180,000</td>
</tr>
<tr>
<td>RMS Errors</td>
<td>0.00018</td>
<td>0.00034</td>
<td>0.00086</td>
<td>0.0018</td>
<td>0.0033</td>
</tr>
</tbody>
</table>

Fig. 5. Neighborhood-trained CMAC for \( f(x, y) \) [see (4)].

The largest CMAC memory \((K = 2)\) required is 32,942 locations. If errors as large as 1.23% are acceptable, it is possible to reduce the required memory size to 3,800 locations. CMAC's having a smaller \( K \) required more training sessions and had larger memory requirements than those of a CMAC with a larger \( K \).

The required CMAC memory size, \( M \), can be determined by the relationship

\[
M = \sum_{j=1}^{K} \prod_{i=1}^{r} (N_{ij})
\]

with the resolutions of each input variable as predetermined constants, \( M \) is obviously dependent on the value of \( K \). As the value of \( K \) increases, \( N_{ij} \) decreases, leading to a decrease in the required system memory size.

V. RANDOM TRAINING

In the Random training method, random numbers are generated at each training session for every input variable. This produces input values that are uniformly distributed random numbers [7] for a large number of training locations. In this manner, repetitive training in the same neighborhood is greatly reduced to minimize learning interference. In the early stage of this training, however, some learning interference may be experienced. As the number of training sessions increase, the weight adjustments to the CMAC memory will continually decrease as the CMAC "learns" its function and asymptotically converges to its maximum accuracy.

As an example, \( f(x) \) of (3) was used as the training function for a one-dimensional CMAC. The resulting CMAC performance is depicted in Fig. 6. This illustrates that the use of this training technique minimizes the training errors.

As a second example, a two-dimensional CMAC was trained to produce the function \( f(x, y) \) of (6) using this technique. Table IV shows the influence of variations in the number of quantization levels, \( K \), on \( M \), \( N_{\text{train}} \), and rms errors for this two-dimensional CMAC.

VI. RESULTS AND DISCUSSION

In the neighborhood technique, the necessary number of training sessions is usually small compared to the number of required memory locations. Generally, all memory elements are not addressed during this method of training; therefore, some memory elements will not be modified and remain zero. This is especially true near the boundaries of the input space because the input is not always an integer fraction of the steps between the values at which the CMAC was trained. In Fig. 1, for example, memory elements \( l_d \), \( l_e \), \( l_f \), \( l_g \), \( l_h \), \( j, k, l_k \), and \( M_k \) were not addressed and not listed in Table II. The outputs generated when these memory elements are addressed will therefore contain errors, effectively degrading the overall CMAC performance. This problem will appear when the numbers of intermediate variables for each quantizing level for any input variable are not the same. In Table V, for example, the number of intermediate variables for each quantizing level are not the same (e.g., \( K = 7; N_{11} = 52 \) and \( N_{12} = 53; K = 8; N_{21} = 23 \) and \( N_{22} = 24 \)). Thus, as shown in Table VI, the accompanying rms errors of these CMAC's with \( K = 7 \) and \( K = 8 \) are greater than those of the CMAC with \( K = 9 \), while the performance of the CMAC's with smaller \( K \) is expected to be better as demonstrated previously in Table III. This problem can be avoided by adjusting both \( K \) and the resolution of the input variables such that the number of intermediate variables of each quantizing function of each input variable is the same. This is possible only to the extent that the minimum required memory size is within the available limit. If this condition is met, then all the memory elements will be addressed and the CMAC may be trained in one pass.

The Neighborhood technique will yield a CMAC which emulates a control function in the shortest possible period of training provided the numbers of the intermediate variables of each quantizing function for any input variable are the same. Although sequential selection is not necessary as long as the training input vectors are chosen outside of a neighborhood, sequential training is convenient from a pragmatic training standpoint. As a general guideline to be used in selecting a
TABLE V
THE NUMBER OF INTERMEDIATE VALUES FOR VARIOUS QUANTIZING LEVELS

<table>
<thead>
<tr>
<th>Neighborhood-trained CMAC for f(x, y) (see Equation (4))</th>
<th>K</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N_{I_1}</td>
<td>N_{I_2}</td>
<td>N_{I_3}</td>
<td>N_{I_4}</td>
<td>N_{I_5}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

TABLE VI
THE EFFECT OF VARYING K ON M, N_{train}, AND RMS ERRORS

<table>
<thead>
<tr>
<th>Neighborhood-trained CMAC for f(x, y) (see Equation (4))</th>
<th>K</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M</td>
<td>11,346</td>
<td>9,805</td>
<td>8,648</td>
<td>7,749</td>
</tr>
<tr>
<td></td>
<td>N_{train}</td>
<td>1,891</td>
<td>1,332</td>
<td>1,058</td>
<td>861</td>
</tr>
<tr>
<td></td>
<td>RMS Errors</td>
<td>0.0011</td>
<td>0.0028</td>
<td>0.0032</td>
<td>0.0025</td>
</tr>
</tbody>
</table>

Fig. 7. The rms error levels achieved by the Neighborhood and Random methods.

Fig. 8. A comparison of the data flow of the (a) Neighborhood and (b) random training methods.

VII. CONCLUSIONS AND RECOMMENDATIONS

The data flow during the Neighborhood and Random training methods is depicted in Fig. 8. The number of Neighborhood training points is fixed by the size of the neighborhood and the range of the input space—a value much smaller than the number of memory locations to be trained. In addition, the gain for modification of the CMAC weights is unity for this method, resulting in one-pass training. Because of the effect of interference in Random training, the gain is generally reduced to much less than 0.5 and the number of training sessions must greatly exceed the number memory locations. This was demonstrated in Table VII for a common CMAC application. In this comparison, the Neighborhood method trains the CMAC between 20 and 200 times faster than can be accomplished using the Random technique. If additional accuracy is desired, however, one can follow the Neighborhood training by a period of Random training until the errors are within the acceptable range.

As a general rule, the Neighborhood sequential training requires the least possible number of training periods for a CMAC to learn a continuous mathematical function whose values change smoothly. It accomplishes this without introducing the errors normally associated with common coding methods which introduce incremental (either linear or curvilinear) training. Random training requires many more training periods to be an effective CMAC training method. However, for the ultimate CMAC performance, any CMAC can achieve its ultimate accuracy by this technique over an extended training period.

Because of its speed advantage, the Neighborhood training method should be applied to all CMAC's prior to invoking
random training procedures. Keeping the value of $K$ smaller in these two training techniques is recommended as long as the required memory size is smaller than the available one. Both techniques have potential for generating simple continuous mathematical functions and training CMAC's for general control problems, real-world pattern recognition, computer vision, and other applications.

REFERENCES


David E. Thompson (A’86) received the B.S.M.E. degree from Lamar University in 1964, the M.S.M.E. degree from Louisiana Technological Institute in 1964, and the Ph.D. degree from Purdue University in 1969.

He currently holds the rank of Professor and Chairman of Mechanical Engineering at The University of New Mexico in Albuquerque. His interests include biomechanics, math modeling, CAD, and computer graphics.

Sungyu Kwon received the B.S.M.E. degree in 1980 and the M.S.M.E. degree in 1983, both from Yonsei University; he received a second M.S.M.E. degree in 1987 and the Ph.D. degree in 1990 from Louisiana State University.

He is currently employed as a Senior Researcher in the Remote Technology Department of the Korea Atomic Energy Research Institute in Taejon, Korea. His interests include robotics, telerobotics, CAD, and computer graphics.