

AN OVERVIEW OF THE CMAC NEURAL NETWORK

by

Filson H. Glanz, W. Thomas Miller, L. Gordon Kraft
Robotics Laboratory, Electrical Engineering Department
University of New Hampshire
Durham, N.H. 03824

Abstract

This paper describes the CMAC neural network which is an alternative to backpropagated multilayer networks. CMAC has properties of generalization, rapid algorithmic computation based on LMS training, functional representation, output superposition, and practical hardware realization, all of which are discussed. Data concerning CMAC capacity and generalization are shown. Brief descriptions of applications in pattern recognition, robot control, and signal processing are given.

I. Introduction

In this paper we will describe a neural network called CMAC, which stands for Cerebellar Model Arithmetic Computer. We at the UNH Robotics Laboratories have been using CMAC in real time control of an industrial robot, and in other applications, typically employing neural networks with hundreds of thousands of adjustable weights that can be trained to approximate nonlinearities which are not explicitly written out or even known. CMAC can learn nonlinear relationships from a very broad category of functions.

The CMAC neural network described in this paper is an alternative to the well known backpropagation-trained analog multilayer neural network. Here are a few reasons for using CMAC instead of a backpropagation network: 1) Backpropagation, the dominant algorithm for training multilayer networks, generally takes many iterations to converge and is therefore inappropriate for online real-time learning. This imposes the need for using small networks, and therefore many difficult problems of current interest can not be solved. Furthermore, the number of computations per iteration is large so that the algorithm runs slowly unless implemented in expensive custom hardware. 2) It is known that multilayer networks have an error surface which can have relative minima, and the training algorithms now in use (such as backpropagation) are based on gradient search techniques- not very comforting if one wants to be assured of finding the best, or even a good, solution! 3) Successful multilayer backpropagation usually does not allow incremental learning (if one has finite time to train) in the sense that to achieve reasonable convergence all inputs must be seen before any weight change can take place.

In the 1970's James Albus reported the work he had been doing on CMAC [1,2,3,4]. His neural model was a result of studying what was known about the cerebellum at that time. CMAC is an associative memory which has a built-in generalization.

II. CMAC Characteristics and Parameters

The CMAC technique has several important characteristics:

1) It takes integer valued vector inputs (mapped from possibly real valued vectors) and gives real valued vector outputs. Input vectors of 5 to 20 components each which can take on one of 200 values are typical in our applications. Our output vectors typically have 1 to 10 components.

- 2) It responds with approximately correct outputs, even to inputs that it has not seen as long as there has been training in the neighborhood in the input state space. This property is called generalization.
- 3) It is capable of learning any(?) single valued well behaved function to some degree of accuracy. The functions learned need not be linear.
- 4) It can be a very large network and still be trained in a practical length of time in comparison to other forms of neural networks. We often use several hundred thousand weights.
- 5) It requires a small number of computations per output. Typically 32 to 128 weights are summed to give a single output. Therefore software is very fast in comparison to many other neural models for the same number of total weights.
- 6) The algorithm can be implemented in relatively simple software or in high speed hardware. Our hardware realization [5] is now available through Shenandoah Systems Company, and gives control rates below one millisecond, with future rates possibly well below one microsecond.
- 7) We use the normal LMS adaption of Widrow and Hoff[6] to change the weights, so that being caught in a relative minimum is not possible as in the case of back propagation.
- 8) Input vectors to CMAC are associatively mapped to obtain the output. This explains why not all weights contribute to each output. This also explains why there is no direct input to output path for a signal so that scaled inputs do not give scaled outputs without training for that result.
- 9) CMAC obeys superposition in the output space, which means that a multidimensional discrete Fourier series can be used to show what functions can be learned. For example, if multidimensional sinusoids of various spacial frequencies (harmonics) can be learned, a whole class of functions is also learnable. Notice that the functions learned are still nonlinear, since the superposition is in the output space only.

Notice also that in spite of the spacial frequency constraints implied by the superposition discussed above, there is no such constraint in time response if consecutive inputs to CMAC are not neighbors in the state space as would be the case when learning a function whose input is the state of a shift register. In that case we have learned very sharp transitions, as for example, learning to deconvolve a random bit string convolved with a decaying sinusoid[7,8]. The CMAC inputs are a fixed number of current and past values of the convolution output so that there is not necessarily state space continuity from input to input.

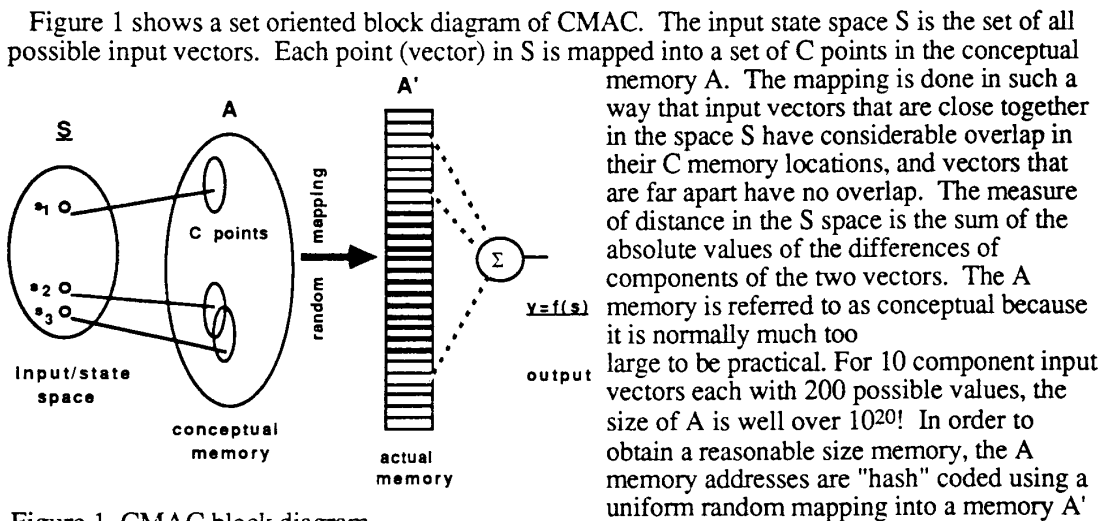


Figure 1. CMAC block diagram

of size M which is chosen as convenient for the particular application. This trick usually works well since normally only a small fraction of all possible input vectors are ever seen by the system. However, if the memory is too small for the number of unique input vectors that actually do appear, then there is a finite probability that a location in A' will be the mapping image of more than one location in A . This occurrence is called a "collision" and causes trouble only if there are a large number of them in which case they have the effect of noise on the output. Generally the number of collisions can be reduced by increasing the A' memory size.

III. CMAC as a Learning System

Because the CMAC system is generally not familiar, we present two simple simulation examples to demonstrate the statements asserted in the foregoing. These are both scalar output examples. The first has a only one input. It illustrates the ability of CMAC to generalize, the effects of collisions, and the ability to realize a nonlinear function. The inputs are single integers randomly taken from 1 to 100. The desired output is one if the integer input is between 30 and 70 inclusive and zero if the integer input is otherwise (i.e. <30 or >70). The threshold to determine the binary output is 0.5. Figure 2 shows several plots of the input integer on the abscissa versus the output analog value on the ordinate. The threshold line is shown at 0.5, and vertical lines define the inputs of 30 and 70. The lower line has tick marks showing the input values seen since the beginning of training. Figure 2a shows the system response after having seen 5 inputs, with $M=10k$, $C=10$ and $\beta=0.5$. Notice that even though only one integer in the interval 30 to 70 has been seen, roughly five on either side are generalized to be above the threshold. Figure 2b shows the same case but with $M=100$. The only difference is that the result is more "noisy" looking due to the large number of collisions as a result of the small memory size. Figure 2c shows the results

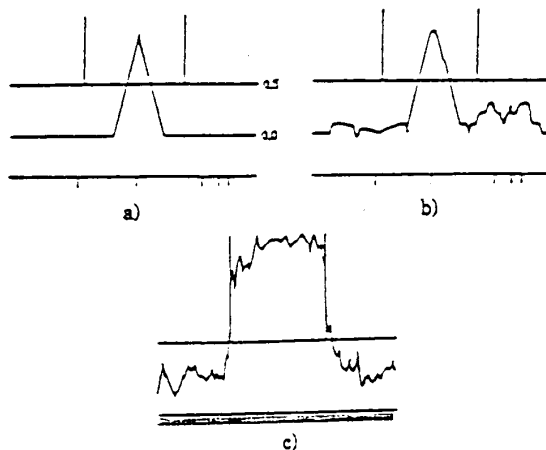


Figure 2. One dimensional example:

- a) $M=10k$, $C=10$, iters.=5
- b) $M=100$, $C=10$, iters.=5
- c) $M=50$, $C=30$, iters.=3000

after the 3000th iteration when $M=50$, $C=30$, and $\beta=0.5$. Notice the approximation to the ideal rectangular pulse analog output (a very nonlinear function).

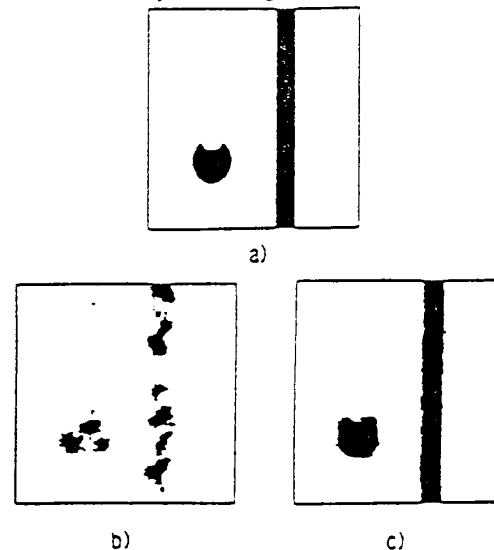


Figure 3. Two dimensional example:

- a) Desired classification
- b) Actual classification, 100 iters.
- c) Actual classification, 100k iters.

The second example has a two component input vector, each component being an integer between 1 and 500 and representing the two components along the sides of a square 500 on a side. Figure 3a shows the desired classification with black being a desired output of one and white being a

desired output of zero. Clearly, a single linear classifier can not solve this classification problem without a rather clever feature transformation. Figures 3b and c show the system output after training using 100 and 100,000 randomly generated training data points.

Simulations that trained CMAC on randomly generated inputs and outputs have shown some interesting results about how CMAC works. Figure 4 shows the normalized average LMS training error as a function of the ratio of the number of training samples to the "available" weights for various memory sizes, number of inputs, and other parameters. The number of "available" weights is the actual memory size or the maximum number of weights addressable for the given generalization size, C , range of input values, R , and number of inputs, N : i.e. max. weights available = $(C+R-1)N/C^{N-1}$. The results show that CMAC has a capacity equal to the number of weights available, since at that point (about 1.0) on the plot the LMS error starts to grow rapidly.

Figure 5 shows the normalized squared generalization error as a function of the number of training samples normalized by the ratio of the number of available weights to the generalization size. In order to measure generalization error it is necessary to have a means of generating what the output error should be for comparison. This requires knowing an output function over the input space, so randomly assigned outputs will not do. By the output superposition characteristic discussed above, we can study the learning of multidimensional sinusoids since any output function can be made up of a sum of these with frequencies that are harmonics of the fundamental frequency. The data shown in Figure 5 is for sinusoids with fundamental equal to the input range, R . The error is that obtained from 100 randomly generated inputs.

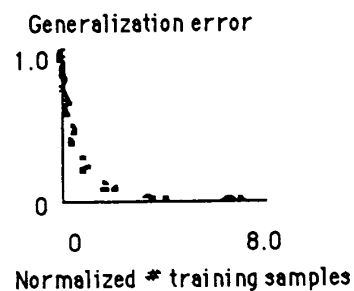
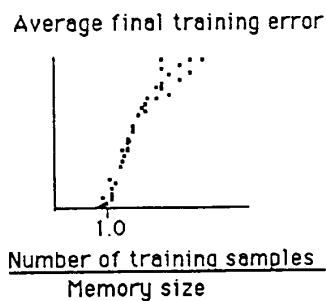


Figure 4. LMS training error versus normalized number of training samples.

Figure 5. Generalization error versus normalized number of training samples.

IV. Some Applications of CMAC

One of the advantages of CMAC at this stage of neural network development is that its realization in software is sufficiently fast and efficient for large networks that many applications are possible. We have used CMAC in a number of real time robotics[9,10,11,12,13,14], pattern recognition[15,16], and signal processing[7,8] demonstrations. In this section we briefly describe three applications in which we have used CMAC. We describe two pattern recognition applications, a robot control application, and a deconvolution/nonlinear inverse filtering application.

A. Pattern Recognition

The first example is a shape recognition system [16]. Features are extracted from an image and CMAC is then used to do the recognition. First the image is located in the frame buffer and the boundary is determined. Then the centroid of the boundary pixels is determined and 64 radius vectors uniformly spaced in angle are found. The ordered radius vectors form a discrete periodic

(circular) function or "time" series.

Then an autoregressive model of order m is found for the radius vector series using the Levinson algorithm. The order m is normally small, and is three in our example. The end result is a feature vector consisting of the $m+1$ parameters: $(\theta_1, \theta_2, \dots, \theta_m, \alpha/\sqrt{\beta})$. This feature vector has several important properties: it is approximately independent of the boundary point at which the radius vectors start, of translation, of rotation, and of size of the object.

The technique described above was applied to recognition of samples taken from two separate groups of eight shapes each, made of white paper placed on a black background. Group A consisted of the eight geometric shapes shown in Figure 6 and the same eight shapes reduced in size to about 67%. The industrial shapes of Group B are shown in Figure 7. These "industrial" shapes were adapted from the recognition literature. The average size of all of these shapes was approximately 3 inches.

For these experiments the image consisted of a 512 by 480 pixel frame buffer of eight bits per pixel. The CMAC input vector had $N=4$ components, each of which was converted to an integer from 1 to 200. Memory size was $M=5000$ and each input was mapped to $C=40$ locations. There were eight outputs from the CMAC, so that the output binary code had a one in the appropriate bit for the shape recognized and zeros elsewhere. Threshold for the output quantization was 0.5, and the weights in memory were real values.

Three training/test files were generated with a videosystem for each of shape groups A and B. For all but one of these files the total number of samples was 200, consisting of 25 samples of each of the eight shapes. The one remaining file totaled 400 samples of 25 large and 25 small examples of each of the eight geometric shapes. In all cases, the shapes were randomly placed in front of the camera so that many orientations and positions were obtained. These files were used in various combinations as training and/or test sets.

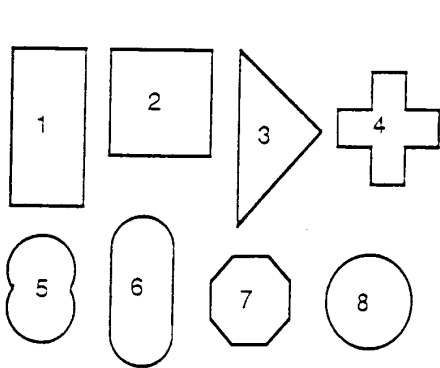


Figure 6. Eight geometric shapes.

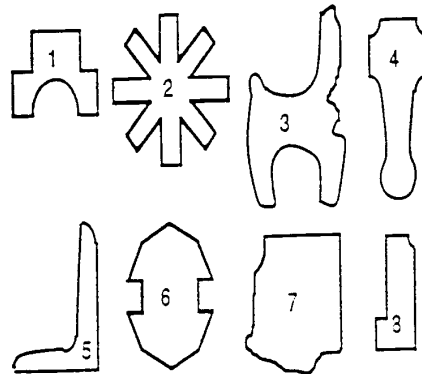


Figure 7. Eight industrial shapes.

The results are shown in Table 1. The table shows the training and test set names, the number of samples in each set and the percent of test shapes identified correctly for each of eight combinations of the six shape files. Note that increasing AR model order, m , might give better results.

Another application of CMAC shows its use in character recognition. In this example the first five letters of the alphabet, each in lower case and upper case, and each in 72 rotations 5 degrees apart

Table I. Shape Recognition Results

Training Set	No. Samples	Test Set	No. Samples	% Correct
Geom1	200	Geom2	200	94%
Geom1	200	Geom3	400	93%
Geom3	400	Geom1	200	91%
Geom3	400	Geom2	200	98%
Indust1	200	Indust2	200	98%
Indust1	200	Indust3	200	100%
Indust3	200	Indust1	200	98%
Indust3	200	Indust2	200	95%

around the circle, were trained into CMAC. The images were obtained by putting each letter on a calibrated rotating platform in front of a video camera which was interfaced through a frame buffer to an 80386 based personal computer. In order to obtain a reasonably sized feature vector and to create features that would not be deliberately easy to classify for most classification techniques, the computer averaged all pixels in each box of an 8 by 8 array centered on the letters. The rows of these averages were summed, as were the columns, giving the 16 features used in the pattern recognition. Files of these features for each letter (case and rotation) were made to use in the training. A separate file was made of new images to act as a test set after training.

Table I summarizes the results. Five other classifiers were included in order to compare the CMAC results with standard techniques. The table briefly lists these and also gives the notation used. The "random" letters were placed at purely random rotations with no attention paid to the 5 degree increments used for the training data. Note that, although the nearest neighbor technique did as well as CMAC, computation for the nearest neighbor approach increases linearly with the number of training samples whereas CMAC computation is independent of the number of training samples.

Table II. Character Recognition Results

Classification techniques shown: CMAC=A CMAC based classifier; TEMP=Standard template matching; F-WT=Feature weighing method; R-COORD=Rotated coordinate method; HYPER="Optimal" hyperplane separation; NNBR=Nearest neighbor method.

EXPT*	CMAC	Percent Correct on Test Set				
		TEMP	F-WT	R-COORD	HYPER	NNBR
U/U	100.0	78.3	75.0	100.0	82.8	100.0
U/RU	100.0	71.1	68.3	99.4	77.8	100.0
UL/U	100.0	56.4	63.3	93.3	70.3	100.0
UL/RU	100.0	47.2	57.8	93.3	71.7	100.0
L/L	100.0	85.3	60.0	99.2	73.6	100.0
L/RL	100.0	83.9	61.1	100.0	73.9	100.0
UL/L	100.0	38.6	46.9	83.3	36.1	100.0
UL/RL	100.0	38.3	50.0	83.3	35.6	100.0

*Experiments indicated by Train/Test where:

U - 72 upper case examples, 5 deg. rotations.

RU - 36 random upper case examples, random rotations.

L - 72 lower case examples, 5 deg. rotations.

RL - 36 random lower case examples, random rotations.

B. Robot Control

One robot tracking problem which we have used to demonstrate neural network control involved the control of a five axis industrial robot with a video camera attached to the fifth axis in the place of a gripper[10]. The camera looked at an object on a conveyor belt and the control problem was

to move the robot in such a way as to keep the object at a fixed orientation and size on the video monitor screen, and simultaneously have the centroid of the object follow a trajectory defined in the video image space rectangular coordinate system. No kinematics of the robot were known to the system, no height measurements other than the length of the object found by the image processing, and no camera-screen calibration were given to the system. All of these had to be learned by the neural network system. The object in this case was a white disposable razor.

Because of the slowness of the image processing, updates of the object image were too infrequent. A second CMAC was used to predict the object position in between image processing results. This CMAC learned to do its prediction with no initial knowledge of the system. The acquisition and orientation of the object and the trajectory following of the object centroid were all done as the object passed along the five foot long conveyor belt, having been placed (within limits) at a random orientation and position on the belt. The trajectories to be followed were either repeated or random, and were described by specifying both video image X and Y coordinate velocities and accelerations.

Training was done at the end of each control cycle based on the error between the desired position and the actual position of the object in the image. In order to initiate training, a simple fixed gain control loop was included in parallel in the system in order to keep the camera close enough to the object initially that the image could be found and the error determined for training. As the system learned more, the fixed gain controller had less and less influence on the control and the learning had more and more. The input vector had 12 components which describe the robot joint positions, the predicted image parameters, and the desired image parameter changes. There were 4 outputs from the inverse model, one to drive each of the four robot joint motors. The fifth motor was not driven in order to keep the camera vertical. The image prediction CMAC had as inputs the latest image parameters, the latest joint angles, and the latest motor voltages. The outputs of this CMAC were the predicted changes in image parameters. Each network consisted of $16,384 \times 4$ 16-bit weights, and the generalization parameter was $C = 64$.

The RMS control error for each of the four image parameters decreased each trial and generally converged to the sensor sensitivity of one pixel within 15 trials in the repeated trajectory case and about 50 trials in the random trajectory case. The average error was always below the error of the fixed gain controller without learning.

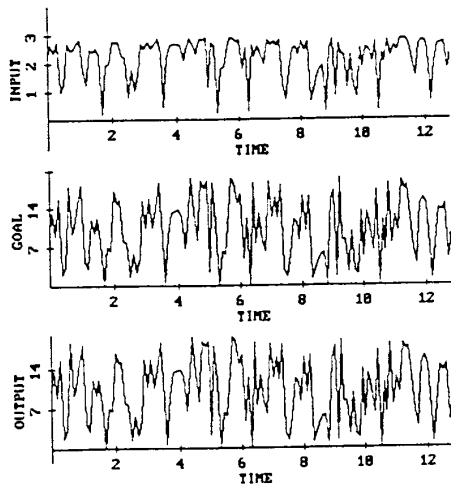


Figure 8. Nonlinear inverse filtering showing input time series, goal, and CMAC output.

C. Signal Processing

A third application involves a signal processing problem[8]: given the output of a nonlinear channel with memory, learn to generate the original input. Such a problem arises, for example, in equalization of a nonlinear communication channel. The nonlinearity used in this example is $y(n) = \arctan[x(n)] + 0.1 \cdot \arctan[x(n-1)]$. In this simulation, $y(n)$ is created by generating uniformly distributed pseudorandom noise, $x(n)$, and feeding it into the equation above, along with the last input value, $x(n-1)$.

The input to CMAC has two components: $y(n)$ and $y(n-1)$. The value of $x(n)$ is used as the desired output. On each presentation of the input 2-vector, training is done until the squared error is appropriately small. Note that there is no repeating of the input vectors (except by chance)

since they are generated from the $x(n)$ s which are not repeated, i.e. there is no fixed training set. Figure 8 shows the input sequence $y(n)$ to CMAC, the desired output(goal) sequence $x(n)$, and the CMAC output. As can be seen from the figure, the CMAC output is almost exactly the same as the goal; i.e. CMAC has learned to inverse the nonlinear output in order to obtain the input sequence (the goal), $x(n)$. If a new random number sequence is used to generate a test sequence, which is run through the trained CMAC, the output is of similar high quality.

V. Conclusions

The CMAC neural network is a viable alternative to a backpropagated multilayered perceptron for learning, especially in real-time situations. Fast and efficient software and hardware realizations exist, and much is known about parameter design for successful use.

VI. References

- [1] J. S. Albus, "A theory of cerebellar functions," *Mathematical Biosciences*, Vol. 10, pp. 25-61, 1971.
- [2] J. S. Albus, "Theoretical and experimental aspects of a cerebellar model," PhD. Dissertation, University of Maryland, 1972.
- [3] J. S. Albus, "Data storage in the cerebellar model articulation controller," *J. Dynamic Systems, Measurement and Control*, pp. 228-233, Sept. 1975.
- [4] J. S. Albus, "A new approach to manipulator control: the cerebellar model articulation controller(CMAC)," pp. 220-227, Sept. 1975
- [5] W. T. Miller, B. A. Box, and J. M. Glynn, "Design and implementation of a high speed CMAC neural network using programmable CMOS logic cell arrays," *IEEE Conference on Neural Information Processing Systems -Natural and Synthetic*, pp. 1990.
- [6] B. Widrow and M. E. Hoff, "Adaptive switching circuits," *IRE Western Electronic Show and Convention*, Vol. 4, pp. 96-104, 1960.
- [7] F. H. Glanz and W. T. Miller, "Deconvolution using a CMAC neural network," *Proc. of the First Annual Meeting of the International Neural Network Society*, pp. 440, September, 1988.
- [8] F. H. Glanz and W. T. Miller, "Deconvolution and nonlinear inverse filtering using a neural network," *International Conference on Acoustics and Signal Processing*, Vol. 4, pp. 2349-2352, May 23-29, 1989.
- [9] W. T. Miller and R. P. Hewes, "Real time experiments in neural network based learning during high speed nonrepetitive robotic operations," *Proceedings of the Third IEEE International Symposium on Intelligent Control*, pp. 513-518, August 24 -26, 1988.
- [10] W. T. Miller, "Real time application of neural networks for sensor-based control of robots with vision," *IEEE SMC*, Vol. 19, pp. 825-831, July-August 1989.
- [11] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft, "Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller," *IEEE Trans. Robotics Automat.*, Vol. 6, pp. 1-9, Feb. 1990.
- [12] W. T. Miller, F. H. Glanz, and L. G. Kraft, "CMAC: an associative neural network alternative to backpropagation," *IEEE Proceedings*, pp. 1561-1567, 1990.
- [13] W. T. Miller and C. M. Aldrich, "Rapid learning using CMAC neural networks: real time control of an unstable system," *5th. Symposium on Intelligent Control*, pp. 465-470, September, 1990.
- [14] L. G. Kraft and D. P. Campagna, "A summary comparison of CMAC neural network and traditional adaptive control systems," *IEEE-AC Mag.*, April 1990.
- [15] D. Herold, W. T. Miller, L. G. Kraft, and F. H. Glanz, "Pattern recognition using a CMAC based learning system," *Proc. of the SPIE: Automated Inspection and High Speed Vision Architectures II*, Vol. 1004, pp. 84-90, November 10-11, 1989.
- [16] F. H. Glanz and W. T. Miller, "Shape recognition using a CMAC based learning system," *SPIE Conference on Robotics and Intelligent Systems*, Vol. 848, pp. 294-298, November, 1987.