A Neural-Network Approach to Robotic Control

D P W Graham and G M T D'Eleuterio

Institute for Aerospace Studies
University of Toronto

Abstract

An artificial neural-network paradigm for the control of robotic systems is presented. The approach is based on the Cerebellar Model Articulation Controller created by James Albus and incorporates several extensions. First, recognizing the essential structure of multibody equations of motion, two parallel modules are used that directly reflect the dynamical characteristics of multibody systems. Second, the architecture of the proposed network is imbued with a self-organizational capability which improves efficiency and accuracy. Also, the networks can be arranged in hierarchical fashion with each subsequent network providing finer and finer resolution.

1. Introduction

The brain possesses a remarkable ability to learn and perform motor control functions without the apparent need to write out elaborate differential equations. Researchers have long been intrigued by this cerebral calculus and have made various attempts at replicating it. In an age of robotics, not only has this goal been pursued with more vigor than ever before, but the issue has also become of decidedly greater practical import.

After having had initially lost popularity, the concept of *artificial neural networks* has experienced a renaissance and the field is now flourishing. Although many of the proposed architectures bear little resemblance to the biological structures that motivated them, they have been successfully implemented in myriad applications from pattern recognition to real-time control.

In the field of robotics, one must deal with highly nonlinear systems which are in general not easily amenable to analysis. While the basic dynamics of a system can be had with a modicum of effort, accurate modeling of motor dynamics, joint friction, link damping and structural flexibility is, at the very least, an arduous task. Yet *model-based* control relies on a model and when that model is suspect one must rely on the robustness of the controller.

Artificial neural networks offer another approach to control, a *nonmodel-based* approach. Neural nets in particular have been demonstrated to be quite a viable strategy for robotic control. One of the main attractive features of neural nets is that they can 'learn.' In the present context, this means the ability to infer, through training, the dynamics of a robotic system including nonlinear characteristics that may be difficult to model by even modern techniques. Neural nets are furthermore inherently parallel, robust, fault tolerant and less susceptible to noise. Surveys of

neural-network architectures for robotic applications have been presented by Kung and Hwang (1989), Freeman and Kosko (1990) and Lee and Bekey (1991).

In debating model-based control *vs.* nonmodel-based control, it is important to observe that the choice is far from binary. Rather, these two broadly described approaches can be considered opposite ends of the same spectrum, a 'control' spectrum which is virtually continuous. Even many control approaches traditionally considered model-based depend on system identification, for example, which itself introduces an element of learning. Indeed, in many ways a neural-network approach may be regarded ultimately as an extensive form of system identification. But here too, there is much to gain by making a few simple and yet general observations about the 'model.'

This paper presents a new artificial neural-network (ANN) paradigm for robotic control. The architecture of the proposed network is founded on the work of Albus (1975, 1981) and it attempts to encode very basic knowledge about the dynamics of robotic manipulators. Several extensions to Albus's work are made including the development of a modular architecture of cooperative networks specifically tailored for mechanical systems. Our network is moreover given a self-organizing structure using the technique of Kohonen (1989). Finally, a hierarchy of these networks can be established to provide progressively more accurate representations of the system at hand. The theoretical development is complemented by simulation results showing improvements in the control of robotic systems over existing comparable methods.

2. Foundations

Our aim in the present work is to develop an ANN schema for the modeling of the inverse dynamics of a (rigid) multibody system that can be used in a computed-torque control procedure. In essence, we seek an ANN representation of motion equations of the form

$$\mathcal{M}(q)\ddot{q} + \eta(q, \dot{q}) = f_c \tag{1}$$

where q represents the system degrees of freedom and f_c is the column of (joint) control forces and/or torques. The mass matrix $\mathcal{M}(q)$ is configuration-dependent and $\eta(q, \dot{q})$ accounts for nonlinear inertial forces such as Coriolis and centrifugal forces. Gravity effects and friction are also assumed to be contained in η.

Let us, however, begin by considering the more generic mapping $f : x \mapsto y$ where $x = \mathrm{col}\{x_1, \ldots, x_N\}$. A relatively crude representation of f can be had by creating a 'look-up table,' directly reminiscent of 'trig' and 'log' tables. Michie and Chambers (1968) used essentially such a technique in their scheme, called BOXES, to control a broom-carriage system (an inverted pendulum on a cart). The value in each 'box,' *i.e.*, table element, was learned from the response of the system to various force stimuli. The direct table look-up method, however, possesses several problems. First, the number of table elements grows exponentially as the number of input states increases. Second, information is not shared; that is, similar inputs should produce similar responses, but with BOXES and other table look-up strategies the response for each set of inputs must be learned separately.

Albus's CMAC

James Albus sought to remedy these shortcomings by developing his CMAC—Cerebellar Model Articulation Controller. This architecture was motivated by the biological motor control functions of the human cerebellum (Albus 1975, 1981). The basic concept can be best represented diagrammatically for a two-dimensional problem (with inputs x_1 and x_2), as in Figure 1. Instead of using one layer of finely divided cells, essentially the structure of BOXES, the CMAC employs several overlapping layers of coarser cells with each layer progressively shifted relative to the previous one. The example shown in Figure 1 displays four layers of coarse cells. To evaluate the function $f(x_1, x_2)$ in this case requires 'activating' four cells and summing their 'encoded values.' This quantization process is an example of what are generally called coarse-coding techniques (Rumelhart and McClelland 1988).

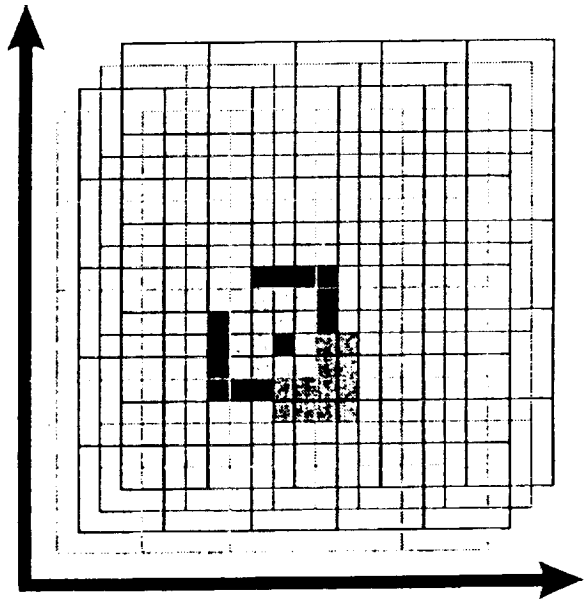

Figure 1: Two-dimensional example of CMAC discretization. Shaded areas represent the activated coarse cells for a point in the small black square.

As can be seen from Figure 1, the resolution attainable in this procedure is substantially better (depending on the number of coarse-cell layers used) than the actual size of the coarse cells. Indeed, with four layers, the CMAC can achieve the same resolution as BOXES but with only about one-third of the total number of cells. For larger problems, the savings factor could be orders of magnitude with, of course, a comparable saving in required memory space.

Albus moreover recognized that the entire input space is typically not necessary for applications to robotics. Hence, the coarse cells can be 'hashed' (randomly assigned) to a smaller

number of units, called 'granule' cells. The sparseness of the active input space will ensure (in probabilistic terms) that the number of collisions in hashing will be negligible. This hashing procedure makes more efficient use of the input space and further reduces memory requirements.

In addition, the overlapping layers in the CMAC permit the sharing of information. Returning briefly to the example of Figure 1, the value of f at any point must be reconstructed by the information (encoded value) contained in four coarse cells. However, evaluating f at a new point in a neighboring fine cell will activate three of the previous four coarse cells. Thus, some knowledge at the new point is already known from the learning done at the previous point. This process is called 'generalization' and enables one to acquire knowledge over large regions of the input domain by learning at only a relatively few points.

Although not done by Albus, the CMAC can be cast in the familiar ANN-like architecture as shown in Figure 2. Each coordinate is finely discretized into 'input units' and then coarsely discretized, according to the shifted layers or grids of coarse cells, into 'receptive units.' The receptive units activate a single coarse cell ('coarse cell unit') in each grid, which in turn can be hashed to a reduced set of 'granule cell units.' Outputs from these units are weighted and summed by the 'output unit' to yield the final output value. A complete description of this architecture can be found in Graham and D'Eleuterio (1991*a*).

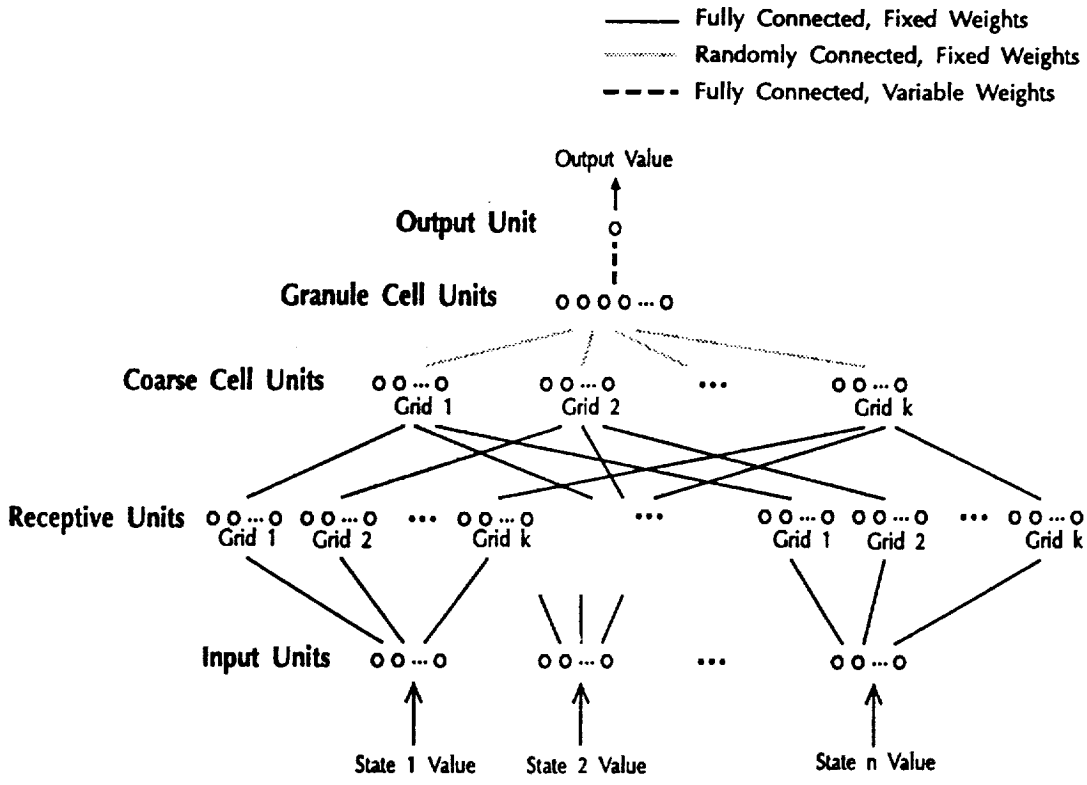

Figure 2: Architecture of the CMAC cast in the form of an ANN.

It is important to note that it is only the last layer which contains variable weights (*i.e.*, the encoded values) that must be learned. The rest of the network can be said to be 'hard-wired' or more precisely 'hard-coded.' Learning is therefore quite fast, typically orders of magnitude faster than comparable backpropagation networks.

Mathematical Formulation

The CMAC concept can in general be represented mathematically as follows:

$$\hat{f}(\boldsymbol{x}) = \sum_{\alpha} w_{\alpha} \psi_{\alpha}(\boldsymbol{x}) \tag{2}$$

where w_{α} are variable weights (encoded values) and $\psi_{\alpha}(\boldsymbol{x})$ may be viewed as basis functions or 'receptive fields.' Also, the notation $\hat{f}$ recognizes that the expansion (2) is in general only an approximation to f. In a CMAC, $\psi_{\alpha}(\boldsymbol{x})$ may be described as 'hyperbox' functions, *i.e.*, they would delineate the coarse cells:

$$\psi_{\alpha}(\boldsymbol{x}) \triangleq \begin{cases} 1, & \boldsymbol{x}_{\alpha} < \boldsymbol{x} < \boldsymbol{x}_{\text{next}(\alpha)} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $\boldsymbol{x}_{\alpha} < \boldsymbol{x} < \boldsymbol{x}_{\text{next}(\alpha)}$ is to be read as $x_{1,\alpha} < x_1 < x_{1,\text{next}(\alpha)} \cdots x_{N,\alpha} < x_N < x_{N,\text{next}(\alpha)}$. Note that we must write $\boldsymbol{x}_{\text{next}(\alpha)}$ instead of $\boldsymbol{x}_{\alpha+1}$, say, since we cannot index cellular divisions consecutively in N-space; however, a functional relation, 'next(α),' can be defined. Another example of possible basis functions are the Gaussian fields used by Moody and Darken (1989).

The learning rule, the well known Delta Rule, for w_{α} can be expressed, for general $\psi_{\alpha}(\boldsymbol{x})$, as

$$\Delta w_{\alpha} = \gamma \frac{\psi_{\alpha}(\boldsymbol{x})}{\sum_{\beta} \psi_{\beta}(\boldsymbol{x})} \Delta f(\boldsymbol{x}) \tag{4}$$

where

$$\Delta f(\boldsymbol{x}) \triangleq f(\boldsymbol{x}) - \sum_{\beta} w_{\beta} \psi_{\beta}(\boldsymbol{x})$$

is the error in the mapping. For the basis functions defined by (3), the denominator

$$k \triangleq \sum_{\beta} \psi_{\beta}(\boldsymbol{x}) \tag{5}$$

is the number of 'activated' cells and is furthermore constant; in fact, k is the number of coarse-cell layers. Thus, we can define the 'learning rate' or 'learning coefficient' as

$$\nu \triangleq \frac{\gamma}{k} \tag{6}$$

Note that in the CMAC architecture, the error is distributed uniformly among layers.

Application to Robotics

The CMAC was designed with manipulator control specifically in mind. An implementation of the CMAC scheme in the control of a two-link planar manipulator has been performed by Miller *et al.* (1987). The input variables in this application were the joint angles, rates and accelerations ($\theta = \text{col}\{\theta_1, \theta_2\}, \dot{\theta}, \ddot{\theta}$). Two CMAC networks are used, each requiring all six inputs. The outputs from networks are the two joint control torques.

The CMAC controller measures the state of the system (θ and $\dot{\theta}$) and a trajectory planner determines the required acceleration ($\ddot{\theta}$) to drive the actual trajectory towards the desired trajectory in a prescribed number of time steps. A position-error controller is superimposed onto the CMAC controller to deal with any residual error as may arise from the CMAC discretization. The position-error controller also provides nominal control during the initial learning phase. The control system is displayed in Figure 3.

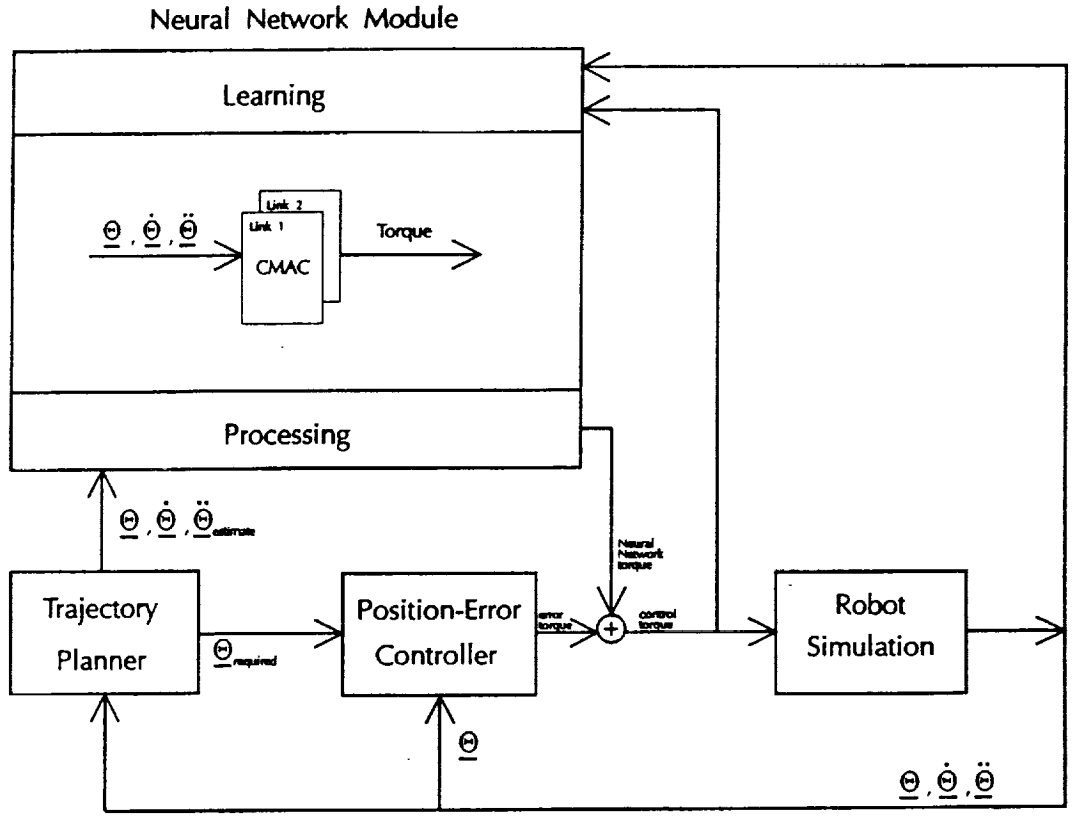

Figure 3: Robotic control system that includes a single CMAC module.

The CMAC is taught by presenting data on input torques and the corresponding measured state (plus joint accelerations) of the system. The CMAC is thus able to learn the inverse dynamics of the manipulator without direct supervised learning. Miller *et al.* show that the CMAC can learn to follow a path when presented with it only a few times. It was also found that the CMAC architecture was able to handle multiple paths, noise, different cell sizes and learning rates and it was able to adapt readily to changes in the manipulator's mass properties. Miller *et al.* (1990) have also successfully implemented the CMAC controller on a five degree-of-freedom manipulator.

3. Modular Architecture

The CMAC approach as developed by Albus and as implemented by Miller *et al.* does not assume anything about a robotic system apart from the selection of the input variables. A significant enhancement, however, can be achieved by making the merest note of the motion equations. Rewriting (1) slightly, we have

$$\boldsymbol{\mu}(\boldsymbol{q}, \ddot{\boldsymbol{q}}) + \boldsymbol{\eta}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{f}_c \tag{7}$$

(The purpose of doing this is to emphasize the fact that the structure of the first term is unimportant in the following.) It should be underscored that (7) still represents the most general form of motion equations for multibody systems.

Written as (7), it is clear that the equation of motion can be parsed into two distinct parts: One being a function of only position and acceleration, and another of only position and rate. This structure suggests a modular architecture consisting of two CMAC subnetworks, each defined on a different subset of the augmented state ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$) and, hence, each smaller than a single CMAC. Unlike other modular networks (*e.g.*, Jacobs *et al.* 1991), each subnetwork here operates cooperatively and on a different set of inputs. In addition to reducing the total memory requirements when compared to a single-CMAC implementation, this 'divide and conquer' approach possesses the very attractive feature that it captures the dynamical structure of a multibody system without explicitly encoding the motion equations. In fact, setting $\boldsymbol{\eta} = \mathbf{0}$ would yield the linearized equations of motion.

Learning Procedure

The problem in this architecture arises in learning. In application to a real robotic system, we cannot assume the separate parts of the motion equation are available to us to enable the modules to learn separately. Rather we would only have the total error produced by the network (Graham and D'Eleuterio 1991*b*). Thus, a technique to distribute the error between the two modules is needed.

In general, we can represent a modular architecture of cooperating CMACs as

$$\hat{f}(\boldsymbol{x}) = \sum_{\kappa} \sum_{\alpha} w_{\kappa,\alpha} \psi_{\kappa,\alpha}(\boldsymbol{x}_\kappa) \tag{8}$$

where κ is the modular index and $\boldsymbol{x}_\kappa \in$ span $\boldsymbol{x}$. For the problem at hand, there are only two modules which can be identified by $\kappa = \mu$ for the first (rate-linear) module and $\kappa = \eta$ for the second (rate-nonlinear) one. The proposed learning rule may be expressed as

$$\Delta w_{\kappa,\alpha} = \nu \rho_\kappa(\boldsymbol{x}) \psi_{\kappa,\alpha}(\boldsymbol{x}_\kappa) \Delta f(\boldsymbol{x}) \tag{9}$$

where $\Delta f(\boldsymbol{x})$ is the error given by the mapping (8) relative to the desired value and the 'gating coefficients' $\rho_\kappa > 0$ satisfy

$$\sum_\kappa \rho_\kappa(\boldsymbol{x}) = 1$$

which assures that all the error is distributed although ν can be adjusted to set the learning rate separately.

For robotic systems, we propose the following heuristic for determining the gating coefficients:

$$\rho_\mu = \frac{r_{\ddot{q}} \|\ddot{\boldsymbol{q}}\|}{r_{\ddot{q}} \|\ddot{\boldsymbol{q}}\| + r_{\dot{q}} \|\dot{\boldsymbol{q}}\|}, \qquad \rho_\eta = \frac{r_{\dot{q}} \|\dot{\boldsymbol{q}}\|}{r_{\ddot{q}} \|\ddot{\boldsymbol{q}}\| + r_{\dot{q}} \|\dot{\boldsymbol{q}}\|} \tag{10}$$

where $r_{\ddot{q}}$ and $r_{\dot{q}}$ are fixed weighting constant. The ratio of these parameters is set here as the ratio of the expected input limits of the joint rates and accelerations. An algorithm to determine the gating coefficients using reinforcement learning is under development (McGuire and D'Eleuterio 1992).

4. Self-Organized Hierarchical Architecture

Thus far, we have implied that the size and spacing of the coarse cells as well as the number of coarse-cell layers are fixed and moreover regular. However, there is ample reason to investigate the choice of these parameters and indeed the manner in which they may be changed. A trade-off exists, for example in the selection of the size of cells: Smaller cells may increase resolution at a cost of generalization; larger cells may overgeneralize and reduce resolution. A delicate balance must be struck. Miller *et al.* (1990) suggest that a broad range for these parameters exists that permits successful learning. In the spirit of artificial neural networks, it would make for an effective approach if, for example, the cell size and position could be automatically organized according to the input training data.

Several self-organizing neural networks have attempted to capture and exploit the spatial distribution of input data. The 'locally tuned network' by Moody and Darken (1989) and the 'self-organizing network' by Kohonen (1989) are two such approaches. Both are statistically based and organizes the neurons only; learning is a separate step.

One technique that uses differently sized cells is offer by Moody (1989). In this approach, levels of progressively finer CMACs are employed. The first CMAC uses coarse grids and is allowed to learn over the entire input space. Once this learning has achieved a precision within a prescribed tolerance level, the weights of this CMAC are fixed. A second CMAC with a finer grid is then added and learning continues, adjusting only the weights of the second CMAC. This procedure is repeated as required until the resulting hierarchy provides the desired resolution.

A disadvantage of the technique, however, is that subsequent grids must span the entire input space. As a consequence, the number of coarse cell units needed grows exponentially which in turn increases the number of granule cell units to prevent interference that may occur because of hashing. Also, the cell sizes of subsequent layers must be specified *a priori*.

Cell Organization Based on Kohonen's Network

Motivated by these efforts, we now present a concept for a self-organized hierarchical architecture compatible with the modular architecture described earlier and based on the CMAC network and Kohonen's self-organizing network (Graham and D'Eleuterio 1991*c*). Kohonen's network is well-suited to this application because of its simplicity and its nonoverlapping cell structure.

For explanatory purposes, let us consider a one-dimensional case. The Kohonen cells are precisely the fine cells which result from the overlapping coarse cells. Each cell, designated $\square_n$, has associated with it a real-valued weight v_n which is the position of the cell as measured from some reference point to the center of the cell. Without loss in generality, the weights can be ordered such that $v_n < v_{n+1}$. Our objective is to change gradually the position of the cells to reflect the probabilistic distribution of the input variable and thereby render the structure of the CMAC more efficient, providing greater accuracy in regions of the input space which is likely to display more activity.

Now consider a sample input value x. The cell that is 'activated' $\square_p$ is, in general, the one whose weight most closely matches to the input value. In this case, it is the cell which is closest in distance to x. This 'winner take all' activation can be represented by

$$|x - v_p| = \min_n |x - v_n| \tag{11}$$

where v_p is the weight of the activated cell. A neighborhood of cells $\mathcal{N}_p$, centered on $\square_p$, that is,

$$\mathcal{N}_p = \{\square_{p-r}, \ldots, \square_p, \ldots, \square_{p+r}\} \tag{12}$$

where the index r is a 'radius of activation,' is selected for weight adjustment. This adjustment is accomplished as follows:

$$\Delta v_n = \begin{cases} \lambda(x - v_n), & \square_n \in \mathcal{N}_p \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where the learning rate λ is chosen between 0 and 1. Both the learning rate and the radius of activation are gradually decreased to zero which allows the Kohonen network to converge to a stable, ordered distribution of cells.

As desired, the effect of (13) is to redistribute the cells incrementally with each input datum. Each new input value essentially acts as a magnet drawing the cells in a given neighborhood slightly towards itself. Repeated over a set of input data, the resulting distribution will bear the statistical signature of the input space.

An example of the result of Kohonen self-organization is given in Figure 4. This is a cross-section of the fine (joint angle) discretization used for a two-link robotic arm. The training data was distributed normally.

Multiresolution

Following Moody (1989), a further enhancement that can be made to the present technique is to stack subsequent CMAC modules with progressively finer grids. With each additional CMAC, the weights of the previous one would after sufficient learning be fixed. But instead of having to span the entire input space with these subsequent CMACs, the self-organizing results afforded by Kohonen's network permit us to identify those regions of greatest activity in the input space and thereby restrict subsequent CMACs to selected areas. Of course, Kohonen self-organization can be implemented with each CMAC.

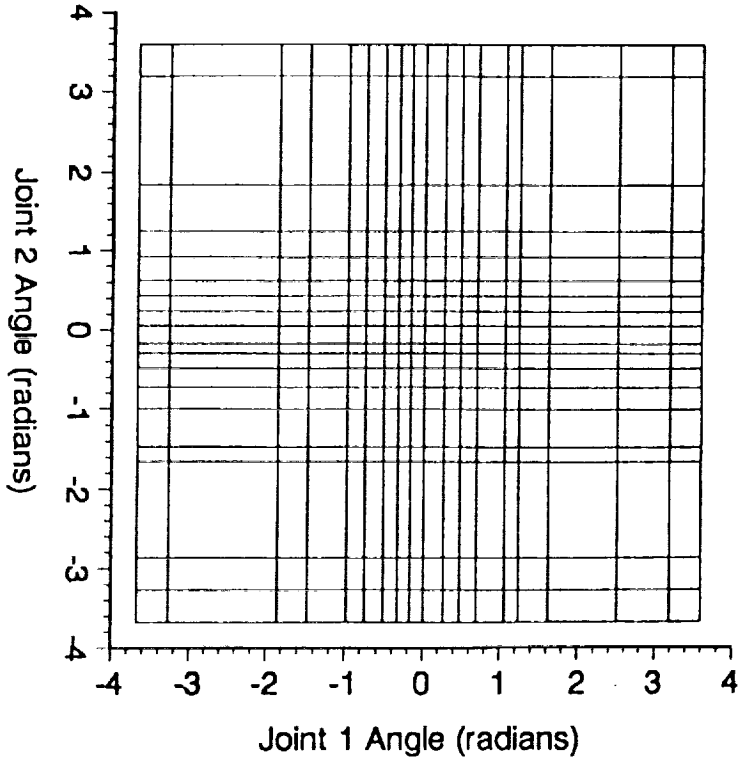

Figure 4: Cross-section of redistributed (fine) cells for a two-link manipulator after training on 1,000 normally distributed random samples (Mean: 0, Standard Deviation: $\frac{1}{6}$ of input range).

5. Simulation Results

We refer to the architecture resulting from these enhancements to the CMAC architecture as MOVE—Manipulator Operation using Value Encoding. We now present computer simulation results demonstrating the performance of MOVE and comparing it to the CMAC. The strawman system investigated here is the two-link manipulator used by Miller *et al.* (1987).

Modular Architecture. We begin by comparing MOVE, consisting only of the modular-architecture enhancement, with a single CMAC. Each variable is evenly discretized into 100 units and, to promote a reasonable amount of generalization, 30 layers (grids) of coarse cells are used. The single CMAC possesses 18,000 granule cells for each joint while each module in MOVE has 9,000 granule cells. Thus, the total memory requirements for each system are the same. A learning factor of $\gamma = 0.6$ was used.

For the first test, 100,000 uniformly distributed random sets of input data (θ, $\dot{\theta}$, $\ddot{\theta}$ and corresponding f_c). served as training data. For every set of 100 input samples, the networks were permitted to learn at only that sample which produced the worst error in the joint torques. Thus, actual learning was done on only 1,000 input samples. The average RMS error in the torques, however, was computed on each set of 100 samples. The results are plotted in Figure 5. As can be seen, not only does 'modular' MOVE learn significantly faster but it yields a lower final error.

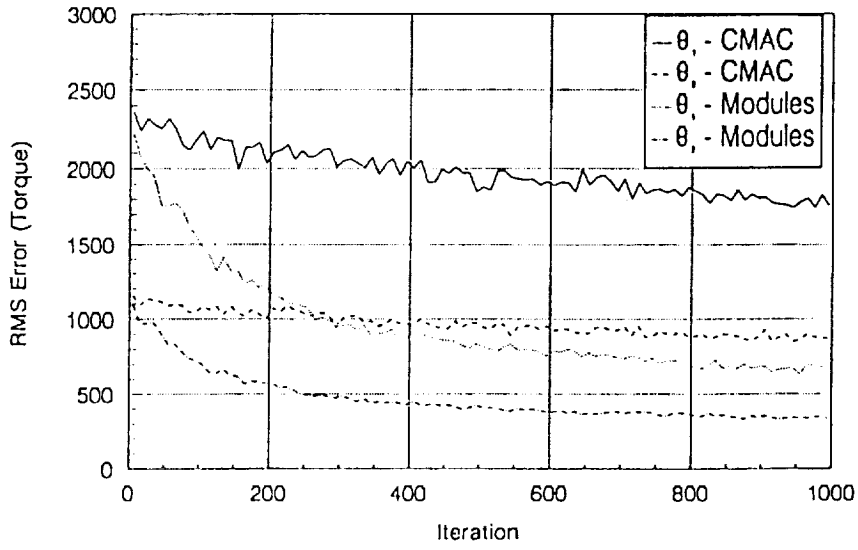

Figure 5: Comparison of learning trends of the CMAC and the modular version of MOVE.

The second test compares the CMAC and modular MOVE in a simulated control environment. For MOVE, the neural-network module in the control system of Figure 3 is replaced by the

modular architecture of MOVE represented diagrammatically in Figure 6. Representative control results, based on the random-sample learning explained above, are shown in Figure 7. The RMS error, as computed over the length of the trajectory, were 0.32 rad for the CMAC and 0.10 rad for modular MOVE.

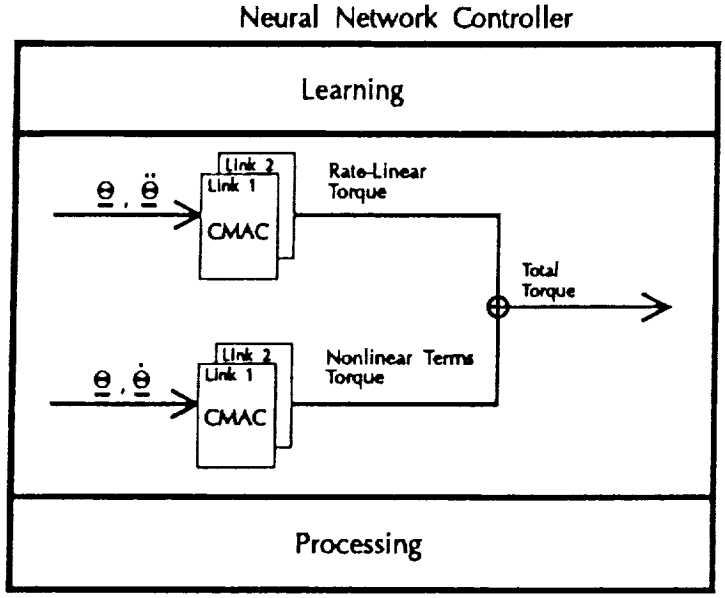

Figure 6: Neural-network module using modular MOVE

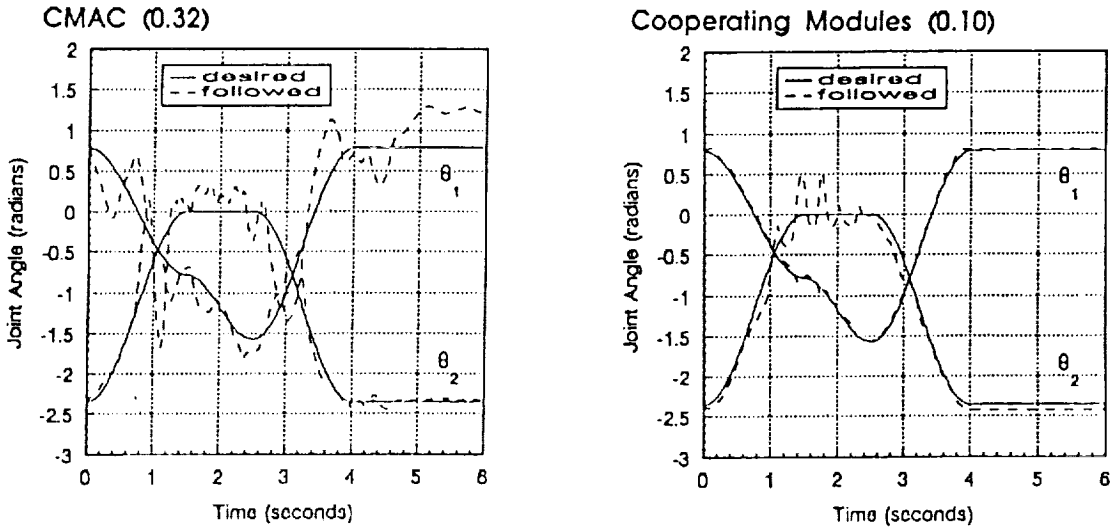

Figure 7: Comparison of control for CMAC and the modular version of MOVE

Self-Organized, Multiresolution Hierarchical Architecture. The hierarchical architecture of MOVE for self-organization and multiresolution was demonstrated and evaluated independently of the modularity enhancement. The neural-network module in the control system of Figure 3 in now replaced by the module in Figure 8. Two levels of CMACs are used in this example. The first level is trained on 1,000 normally distributed input samples (with zero mean and standard deviation of one-sixth of the input range). The weights of this level are then fixed and the second level is trained on a further set of 1,000 input samples.

Figure 9 shows the control results for a representative trajectory. The single CMAC, whose results are shown for comparison, was trained on all 2,000 input samples. The plots show the absolute errors in tracking for the two joints separately. The RMS error over the entire trajectory for both joints was 0.016 rad for the single CMAC and 0.011 rad for 'multiresolution' MOVE.

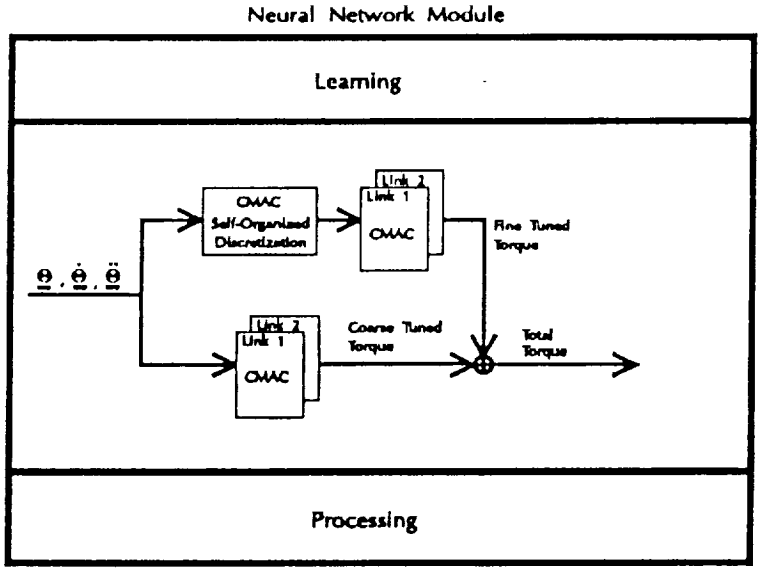

Figure 8: Neural-network module using self-organized, multiresolution hierarchical architecture of MOVE

6. Concluding Remarks

The basic concepts introduced by Albus in his CMAC provide a sound foundation for an artificial neural-network approach to the control of robotic systems. The enhancements incorporated in MOVE significantly improve on the performance of a CMAC robotic controller. The modular architecture of MOVE anticipates the form of the dynamical equations. By recognizing that all mechanical systems share this simple yet basic form, an appropriate structure can be

imposed on MOVE without compromising its applicability to robotics. This 'divide and conquer' technique results in faster learning and more efficient use of memory space.

The generalization property intrinsic to the CMAC has been enhanced by implementing a self-organization scheme based on the Kohonen network. This self-organization enables the cells in a CMAC to arrange themselves according to the statistical distribution of the training data. Furthermore, by creating a hierarchy of self-organized, multiresolution CMACs, one can also improve accuracy. This hierarchical architecture has been successfully employed in the modeling of chaotic systems as well.

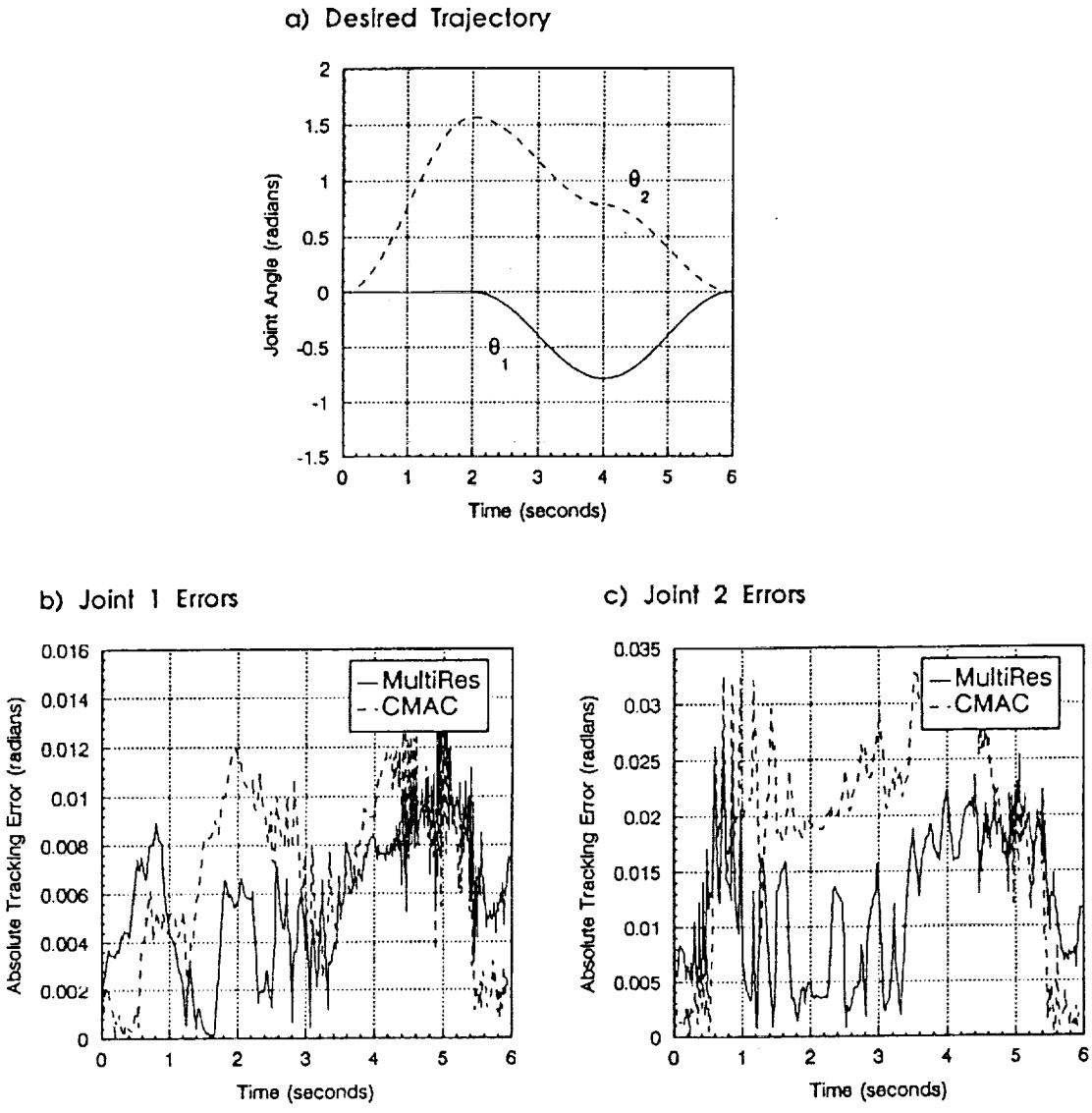

Figure 9: Comparison of a single CMAC and the self-organized, multiresolution hierarchical version of MOVE for a given trajectory.

It is evident that the concept of artificial neural networks holds considerable promise in the field of robotics. Neural networks allow us to dispense, at to least some extent, with carefully constructed system models. They moreover possess a characteristic highly desirable in the industrial workplace, that of being able to adapt to gradually deteriorating systems. For example, the dynamics of a new robotic manipulator will not be the same as the dynamics of that same manipulator when it is older. But neural networks continually learn, continually adapt.

The re-emergence of artificial neural networks also resonates with another current trend—that of parallel processing in computer technology. Like the brain, neural networks are inherently parallel which is of course a very desirable feature, particularly when considering real-time implementation. A hardware version of the the CMAC architecture is already commercially available and DiNardo and Graham (1992) have investigated the performance of MOVE on a parallel transputer platform.

The MOVE artificial neural-network architecture also exhibits considerable potential in other robotic application areas such vision, pattern recognition and analysis, sensor fusion, and flexible manipulators as well as a multitude of nonrobotic applications.

Acknowledgements

This research was supported by the Natural Science and Engineering Research Council of Canada, the Institute for Robotics and Intelligent Systems, and the Institute for Space and Terrestrial Science.

References

ALBUS J.S., "A New Approach to Manipulator Control: Cerebellar Model Articulation Controller (CMAC)," *ASME J. Dynamic Systems, Measurement, and Control*, September 1975, pp. 220–233.

ALBUS J.S., *Brains, Behavior, and Robotics*, BYTE Publications, 1981.

DINARDO G.D.M. & GRAHAM D.P.W., "The Efficiency of the MOVE Artificial Neural Network Architecture on a Multi-Transputer Platform," Seventh CASI Conference on Astronautics, Ottawa, ON, 4-6 November 1992.

FREEMAN W. & KOSKO B., "CHAIRs," 1990 International Joint Conference on Neural Networks, San Diego, CA, June 1990.

GRAHAM D.P.W. & D'ELEUTERIO G.M.T., "MOVE—A Neural-Network Paradigm for Robotic Control," *Canadian Aeronautics and Space Journal*, Vol. 37, No. 1, March 1991*a*, pp. 17-26.

GRAHAM D.P.W. & D'ELEUTERIO G.M.T., "Robotic Control Using a Modular Architecture of Co-operative Artificial Neural Networks," 1991 International Conference on Artificial Neural Networks, Helsinki, Finland, 22-26 June 1991*b*.

GRAHAM D.P.W. & D'ELEUTERIO G.M.T., "A Hierarchy of Self-Organized Multiresolution Artificial Neural Networks for Robotic Control," 1991 International Joint Conference on Neural Networks, Seattle, WA, 8-12 July 1991*c*.

JACOBS R.A., JORDAN M.I., NOWLAN S.J. & HINTON G.E., "Adaptive Mixtures of Local Experts," *Neural Computation*, Vol. 3, No. 1, 1991.

KOHONEN T., *Self-Organization and Associative Memory*, 3rd Edition, Springer-Verlag, 1989.

KUNG S-Y. AND HWANG J-N., "Neural Network Architectures for Robotic Applications," *IEEE Transactions on Robotics and Automation*, Vol. 5, No. 5, October 1989, pp. 641-657.

LEE S. & BEKEY G.A., "Applications of Neural Networks to Robotics," *Control and Dynamic Systems* (C.T. Leondes, ed), Volume 39: Advances in Robotic Systems, Academic Press, 1991.

MCGUIRE P.F. & D'ELEUTERIO G.M.T., "Active Control of Interference in CMAC/MOVE Neural Networks for Robotic Applications," Seventh CASI Conference on Astronautics, Ottawa, ON, 4-6 November 1992.

MICHIE D. & CHAMBERS R.A., "BOXES: An Experiment in Adaptive Control," *Machine Intelligence 2*, (Dale E. & Michie D., eds), Oliver and Boyd, Edinburgh, 1968, pp. 137-152.

MILLER III W.T., GLANZ F.H. & KRAFT III L.G., "Application of a General Learning Algorithm to the Control of Robotic Manipulators," *International J. Robotics Research*, Vol. 6, No. 2, 1987, pp. 84-98.

MILLER III W.T., HEWES R.P., GLANZ F.H. & KRAFT III L.G., "Real-Time Dynamic Control of an Industrial Manipulator Using a Neural-Network-Based Learning Controller," *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 1, February 1990.

MOODY J., "Fast Learning in Multi-Resolution Hierarchies," Research Report YALEU/DCS/RR-691, February 1989.

MOODY J. & DARKEN C., "Fast Learning Networks of Locally-Tuned Processing Units," Research Report YALEU/DCS/RR-654, March 1989.

RUMMELHART D.E. & MCCELLAND J.L., *Parallel Distributing Processing*, MIT Press, 1988.