Real-Time Application of Neural Networks for Sensor-Based Control of Robots with Vision

W. THOMAS MILLER III, MEMBER, IEEE

Abstract—A practical neural-network-based learning control system is described, which is applicable to complex robotic systems involving multiple feedback sensors and multiple command variables. In the controller, one network is used to learn to reproduce the nonlinear relationship between the sensor outputs and the system command variables over particular regions of the system state space. The learned information is used to predict the command signals required to produce desired changes in the sensor outputs. A second network is used to learn to reproduce the nonlinear relationship between the system command variables and the changes in the video sensor outputs. The learned information from this network is then used to predict the next set of video parameters, effectively compensating for the image processing delays. The results of learning experiments using a General Electric P-5 manipulator are presented. These experiments involved control of the position and orientation of an object in the field of view of a video camera mounted on the end of the robot arm, using moving objects with arbitrary orientation relative to the robot. No *a priori* knowledge of the robot kinematics or of the object speed or orientation relative to the robot was assumed. Image parameter uncertainty and control system tracking error in the video image were found to converge to low values within a few trials.

I. Introduction

THE SUCCESSFUL application of robotic systems to tasks in an uncertain environment typically requires the use of special sensors for task-related control feedback. Such sensors might, for example, provide tool position or force information relative to an object being manipulated or involve visual or acoustic navigation sensors for an autonomous vehicle. Unfortunately, for complex systems it is often difficult, in the controller, to relate the task relative error feedback directly to robot actuator commands. In addition, the use of special sensors often introduces undesirable feedback delays, as the result of either sensor data-processing requirements (e.g., video image processing time) or fundamental sensor limitations (e.g., acoustic path delays).

To make robot control decisions in a task-referenced space, it is typically necessary to transform the sensor outputs to the decision space, compute task relative error signals, and finally compute joint level commands using appropriate coordinate transformations and control algorithms. For joint- or wrist-mounted position, velocity or force sensors on a rigid link manipulator, necessary transformations can be derived systematically using kinematic and/or dynamic analyses of the manipulator structure [1]–[3]. Sanderson and Weiss [4], [5] described and compared various configurations for the control of systems involving the interaction of computer-based vision with robotic manipulators.

Adaptive control techniques have also been used for relating task-referenced control decisions to joint level commands, using multiple sensors [4]–[8]. Such adaptive systems typically assume a general form for the relationship between system variables (including sensor outputs) and the command variables, with equation parameters which are adapted in real time based on some performance criteria, rather than being specified *a priori* on the basis of an analytic model. A general drawback to adaptive controllers in such applications is that the computational requirements for real-time parameter identification and the sensitivity to numerical precision, sensor noise, and choice of model can increase undesirably as the number of system variables increases.

Several investigators have discussed possibilities for the application of neural networks in robot control [9]–[18]. The basic theme of all such discussions is that of using the network to learn the characteristics of the robot/sensor system, rather than having to specify explicit robot/sensor system models. While there seems to be wide spread interest in learning control systems within the neural network and robotics communities, past research has generally focused on conceptual representations of geometrical transformations, and relatively little has been reported in the nature of closed-loop control experiments using an actual robotic system. This is due, at least in part, to the computational speed and stability problems encountered when using typical neural models in networks of sufficient complexity to be useful for realistic robot control problems.

We have been investigating a learning control approach [18]–[21] that utilizes the cerebellar model arithmetic computer (CMAC) neural network model proposed by Albus [22]–[25]. In the controller, the network is used as a feedforward term in place of an explicit system model.

Manuscript received February 24, 1988; revised September 15, 1988 and January 22, 1989.

The author is with the Department of Electrical and Computer Engineering, Kingsbury Hall, University of New Hampshire, Durham, NH 03824.

IEEE Log Number 8927251.

0018-9472/89/0700-0825$01.00 ©1989 IEEE

Network training is performed at the end of each control cycle based on on-line observations of the system input/output relationships, and independent of the immediate control goals. The learned information is then used at the beginning of each control cycle to predict the commands required to produce desired changes in the feedback parameters. Information learned during previous movements is generalized automatically to achieve new control goals. As a result, the technique is applicable to the control of complex nonlinear systems performing nonrepetitive operations. In past experiments, the learning controller developed in our laboratory has been shown to be capable of learning the kinematics of robot/video camera systems interacting with parts on a moving conveyor [19]–[12], assuming only minimal qualitative *a priori* knowledge of the robot/video/conveyor system characteristics.

The major limitation on learning controller performance in our previous studies was found to result from image parameter uncertainties created by the image processing delays. Conventional parameter prediction techniques were not effective at reducing the effects of processing delays (generally leading instead to control system instability) because the future image parameters were related in a highly nonlinear way to the actuator positions and drive voltages, as well as to the previous images. This paper presents the results of learning experiments that utilized a cooperative arrangement of two CMAC neural networks to overcome the effects of image processing delays. One network was trained to represent an inverse model of the system being controlled. The output of this network was used as a feedforward term in the controller, compensating for the highly nonlinear system properties [16], [19]–[22]. The second neural network was trained on line to represent a forward model of the system. The output of this network was used to predict the values of the image parameters for the next control cycle, effectively compensating for the image processing delays. The resulting learning controller was shown to provide good control system performance, limited by the basic resolution of the video sensor, for both repetitive and random trajectory tracking tasks.

II. Methods

The experiments involved learning to intercept, track, and perform repetitive or random movements relative to a randomly oriented object on a moving conveyor. Fig. 1 shows the basic experimental configuration. The conveyor was 150 cm long, 45 cm wide, and had a velocity of 3.0 cm/s. The "part" was a familiar plastic disposable razor. The "task" was first to intercept the razor centroid, adjusting the camera elevation to achieve a predefined length from centroid to handle end and adjusting the camera orientation such that the razor handle was parallel to the horizontal (X) video axis. The task was then to move the camera such that the razor centroid (X and Y video parameters) followed a trajectory defined in the video image space rectangular coordinate system, maintaining a constant size (Z video parameter) and horizontal orientation (R video parameter) in the image.

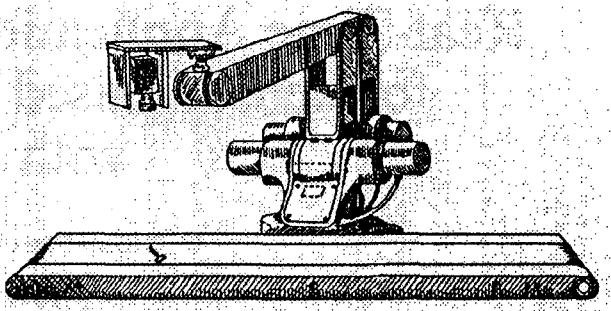


Fig. 1. P-5 robot and conveyor.

The control experiments presented two basic problems. The first was to determine the time sequences of motor drive voltages required to follow the desired trajectories in the image frame of reference. Given an accurate knowledge of the motor driver characteristics, robot kinematics, camera lens and sensor characteristics, and conveyor speed, height, and orientation, it would have been possible to compute the required voltages using conceptually straight forward (but computationally complex) geometric transformations. However, for these experiments the feedforward drive model was implemented using a CMAC neural network (see the Appendix) to represent the transformation from desired object velocities in the image space to actuator drive voltages, assuming no *a priori* knowledge of the above system characteristics.

In the learned feedforward network, the 12-component input vector s was considered to be composed of the vector θ of four joint positions at the beginning of the control cycle, the vector i of four image parameters (X, Y, Z, R) at the beginning of the control cycle (the parameters defining the control state), and the change in the image parameter vector di (dX, dY, dZ, dR) desired during the control cycle (the desired object velocity in the image space). The four-dimensional discrete network output was considered to be the four joint drive voltages in D/A converter units. The network used contained approximately $3 * 10^6$ virtual state space detectors, 16384 multiple field detectors and corresponding memory weight vectors (four 16-bit integer values each), with 64 active units per input state.

The second problem presented by the experiments was that of compensating for the feedback delay imposed by the long image-processing time. Without compensation, this delay corresponded to a significant uncertainty in the measured position of the object in the image. Similar to the feedforward prediction, the current position of the razor in the image field could be predicted accurately from the known previous position of the razor in the camera field and the current robot drive signals, assuming a detailed knowledge of the robot/camera/conveyor system characteristics. However, in these experiments a second CMAC network was used for image feedback predictions, representing the transformation from actuator drive volt-

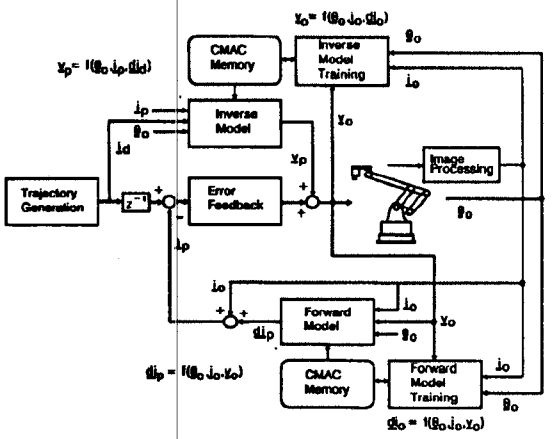


Fig. 2. Block diagram of learning controller with learned image parameter prediction.

ages to object velocities in the image space, again assuming no *a priori* knowledge of the system characteristics.

In the image parameter prediction network, the 12-component input vector s was considered to be composed of the vector θ of four joint positions at the beginning of a control cycle, the vector i of four image parameters (X, Y, Z, R) at the beginning of the control cycle, and the four motor drive voltages during the control cycle. The four-dimensional discrete network output was considered to be the vector change in the image parameters di (dX, dY, dZ, dR) during the control cycle (the object velocity in the image frame of reference). The network used contained approximately $6*10^6$ virtual state space detectors, 16384 multiple field detectors and corresponding memory weight vectors (four 16-bit integer values each), with 64 active units per input state.

The logical flow of the learning controller proceeded as follows (Fig. 2). At the beginning of each 350-ms control cycle, image acquisition was initiated, and the robot joint positions θ_0 were recorded. Image processing was then performed to obtain the four image parameters i_0 (requiring 280 ms). The vectors θ_0, i_0, and v_0 (the current motor drive voltages) were then input to the image parameter prediction network. The network output di_p was added to the most recent image parameters i_0 to form the predicted image parameter feedback vector i_p (the predicted position of the razor at the end of the current control cycle). The input to the control feedforward network was formed from the current motor positions θ_0, the predicted image parameters i_p, and the desired changes in the image parameters during the next control cycle di_d for the desired trajectory. The feedforward network output v_p (the predicted motor drive signal) was then computed.

A fixed-gain linear error feedback controller was implemented in parallel to the feedforward network to provide initial movements upon which the controller could learn and improve. The feedback equations were based on simple logic such as the observation that rotating the forearm tends to raise and lower the camera, changing the apparent handle length. Note in Fig. 2 that, as the wrist rotates by 90°, the X and Y video axes switch in their relationship to the robot structure. A simple test of the wrist axis position was used to divide the X and Y control terms into separate regions. The fixed gain controller was implemented as follows:

$$
\begin{aligned}
&\text{if } (-2950 < \theta 5 < 2950)\ \{\\
&\quad v1 = \ \ 10*(X_d - X_p)\\
&\quad v2 = -10*(Y_d - Y_p)\\
&\}\ \text{else if } (\theta 5 > 2950)\ \{\\
&\quad v1 = -10*(Y_d - Y_p)\\
&\quad v2 = -10*(X_d - X_p)\\
&\}\ \text{else}\ \{\\
&\quad v1 = \ \ 10*(Y_d - Y_p)\\
&\quad v2 = \ \ 10*(X_d - X_p)\\
&\}\\
&v3 = -10*(Z_d - Z_p)\\
&v5 = \ \ 10* - R_p
\end{aligned}
$$

where $\theta 5$ is the wrist rotation axis position in encoder count units. Clearly, the transformations inherent in the conveyor/video/articulated robot system being controlled were considerably more complicated than implied by the above control equations. Note that the error terms were computed as the difference between the desired image parameters at the end of the current control cycle and the predicted values of the image parameters at the end of the current control cycle. At the end of each control cycle, the drive signal from the fixed-gain controller was added to the output of the learned feedforward to form the total drive signal.

Each network memory was initialized to a null state. During image acquisition at the beginning of each control cycle, measured motor position θ_0, image parameter i_0, image parameter change di_0, and motor drive v_0 data from a previous control cycle were used to train each of the two networks, using a learning factor β of 0.2. The image parameter feedback network (forward system model) was trained by computing the CMAC memory adjustment

$$\Delta m = \beta(di_0 - f(\theta_0, i_0, v_0))$$

where $f(\cdot)$ represents the CMAC network vector output. This training adjustment was added to each of the 64 CMAC memory weight vectors selected by the input. The control feedforward network (inverse system model) was trained by computing the CMAC memory adjustment

$$\Delta m = \beta(v_0 - f(\theta_0, i_0, di_0))$$

where $f(\cdot)$ represents the CMAC network vector output. This training adjustment was added to each of the 64 CMAC memory weight vectors selected by the input.

Training was performed for both networks every control cycle throughout the experiments (there was no explicit training phase). The network weight adjustments were based solely on observations of the system inputs and outputs (consistent with the notion of building forward or inverse system models) and were not based on control system errors (valid system information was trained into the networks every control cycle, whether or not good control was being obtained). Finally, the training procedure required no *a priori* knowledge of the system characteristics, other than identifying appropriate input and output parameters.

The image processing and control algorithms were implemented on a VAX-11/730 minicomputer. The CMAC network was implemented on a TMS32010 microprocessor interfaced to the VAX as an auxiliary processor. The robot actuators (100-V dc motors) were controlled directly by analog joint velocity servo loops commanded from the VAX through five D/A converters. The learning controller commanded the robot in terms of a desired velocity for each of the three major robot axes and the wrist rotation axis. The wrist orientation axis was held stationary to keep the axis of the video camera perpendicular to the conveyor surface. The video image data (256×240-pixel format) were used to derive task related sensor outputs. The coordinates of the razor centroid (X and Y in pixel units) and the end of the handle were determined from each image and were used to define the task relative sensor outputs. The X and Y coordinates of the centroid were used to define the object location in the image. The distance from the centroid to the handle end (Z in pixel units) was used without transformation as an indirect measure reflecting camera elevation above the object. The difference between the centroid vertical (Y) coordinate and the handle end vertical coordinate in the image (R in pixel units) was used as a measure of razor orientation.

At the beginning of each trial, the robot was moved to a "home" position over the leading end of the conveyor. The razor was then placed on the conveyor. Each task attempt began as soon as the razor entered the field of view of the camera. The trial was terminated when the robot base rotation axis reached a predetermined position, signaling the end of the conveyor. At the end of each trial, the robot returned to the home position in preparation for the next tracking attempt.

The desired trajectory for the razor centroid in the image space was generated automatically as follows. The desired centroid was taken initially to be at the center of the image. This desired location was held constant for 10 s after the initial appearance for the razor in the image to allow time for the initial interception. Desired initial centroid velocities along the X and Y image axes were then chosen from a list of random velocities, uniform over the interval -12 pixels/cycle to $+12$ pixels/cycle. Constant centroid accelerations along the X and Y image axes were chosen from a list of random accelerations, uniform over the range of 0.0–1.2 pixels/cycle^2. The sign of the acceleration along each axis was set opposite that of the initial velocity. Whenever the desired centroid position approached an image boundary, new desired initial velocities and constant accelerations along both the X and Y image axes were chosen from the random lists, and the sign of either the X or Y velocity was adjusted to direct the desired centroid location away from the boundary. The desired location of the razor centroid thus traveled along curved trajectories around the image (with occasional abrupt changes in velocity near the image boundaries) until the end of the trial (approximately 50 s in duration).

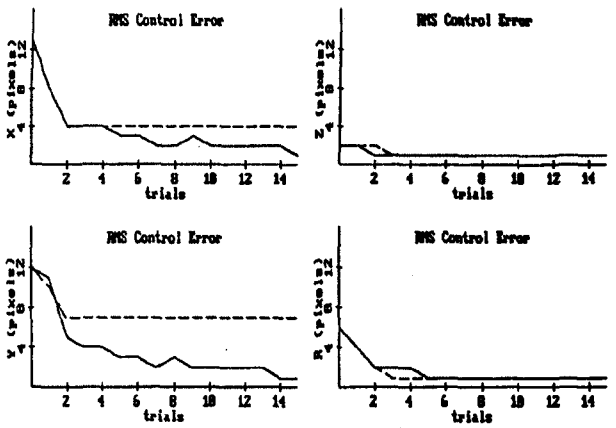


Fig. 3. RMS image parameter control errors as functions of training trial for 50-s repetitive exercise. Trial 0 corresponds to fixed-gain controller without learning. Control error curves are shown for each of four image parameters during experiments using delayed image parameter feedback (broken lines) and learned image parameter predictions (solid lines).

"Repetitive" trials were generated by placing the razor at the same initial position and orientation on the conveyor and starting the velocity selection at the beginning of the random list for each trial. "Random" trials were generated by placing the razor on the conveyor at random angles (uniform between $+45°$ and $-45°$ relative to the conveyor axis of motion), at random positions across the conveyor (within ± 10 cm of the belt center), and by proceeding from one trial to the next without resetting the random list pointer.

III. Results

Fig. 3 shows the root mean square (rms) control error in the image reference frame during 15 sequential repetitive trials when using the learning controller with either the delayed image parameter feedback (broken lines) or with the learned image parameter prediction (solid lines). In each case, the intersection with the vertical axis (trial 0) represents the error obtained using the fixed-gain error feedback controller alone. Note that the control error converged quickly for both cases, but reached significantly lower values for the X and Y image parameters when using the learned image parameter predictions.

The differences in control errors illustrated in Fig. 3 are consistent with the image parameter prediction errors shown in Fig. 4, for the same repetitive learning trials. Only the X and Y image parameters are shown because

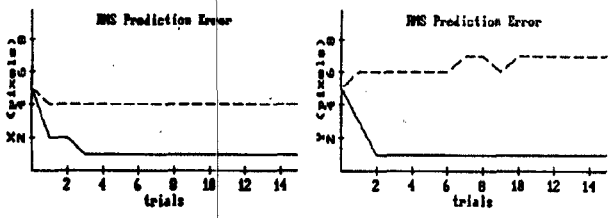


Fig. 4. RMS image parameter prediction errors as functions of training trial for 50-s repetitive exercise. Trial 0 corresponds to fixed-gain controller without learning. Prediction error curves are shown for X and Y image parameters during experiments using delayed image parameter feedback (broken lines), in which case last image was considered predictor of next, and learned image parameter predictions (solid lines).

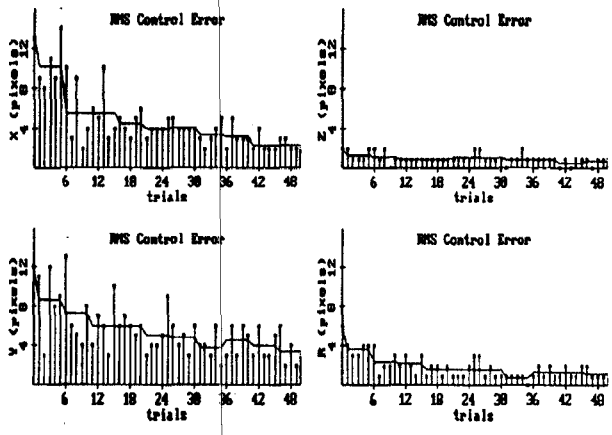


Fig. 5. RMS image parameter control errors as functions of training trial for 50 random exercises. Trial 0 corresponds to fixed-gain controller without learning. Vertical lines show individual trials, while continuous curves show ten average errors obtained from sequential sets of five trials.

the Z and R parameters changed little during each trial (the control goal was to maintain constant values), and thus both prediction methods were equally effective. The broken lines in this figure show the value of one image as a direct predictor of the next image. Since each trial involved the same desired image trajectory, it might be expected that these prediction errors should be constant. However, the actual image trajectory varied as the learning controller converged, with corresponding variation in the image prediction error. The solid lines show the efficacy of the neural network in predicting the image parameters from one cycle to the next. The rms prediction errors converged quickly to one pixel (the basic imaging system resolution) for both the X and Y image parameters. Note that the rms errors in Fig. 4 for both prediction cases match closely with the corresponding rms control errors after training in Fig. 3.

An important aspect of the learning controller being investigated is that its use is not limited to repetitive operations. Fig. 5 shows the rms trajectory tracking error during 50 random trials using both the motor drive feedforward and image prediction feedback networks. As described in Section II, these trials involved randomly generated curved razor trajectories in the image frame as well as

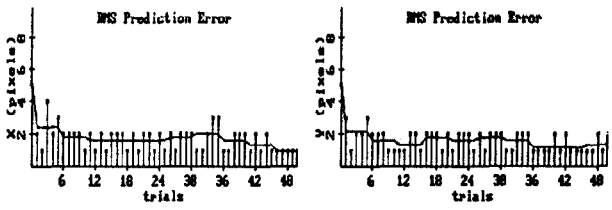


Fig. 6. RMS image parameter prediction errors as functions of training trail for 50 random exercises. Trial 0 corresponds to fixed-gain controller without learning. Vertical lines show individual trials while continuous curves show ten average errors obtained from sequential sets of five trials.

random placement and orientation of the razor on the conveyor. Significantly different movements of the robot were required from one trial to the next, yet the control error clearly converged to lower values during the 50 trials. While the most dramatic changes in performance occurred during the first few trials, it appears likely that the error would have continued to decrease had the experiment been extended beyond 50 trials.

Fig. 6 shows the rms image parameter prediction errors during the 50 random trials depicted in Fig. 5. The prediction errors converged rapidly during the first few trials and, after 35 trials, were nearly the same as those observed during the repetitive experiments (Fig. 4).

IV. Discussion

The experiments presented involved learning to position and orient the robot hand in three dimensions and to track repetitive and nonrepetitive trajectories relative to a moving object at arbitrary orientations and positions, using feedback from a video camera mounted on the robot wrist and assuming only minimal qualitative *a priori* knowledge of the robot kinematics and camera characteristics. Conveyor speed, height, and orientation relative to the robot could be changed (within reason) without affecting system performance. As the result of this feature, CMAC network-based learning controllers are easy to design for a particular application and easy to modify to accommodate system changes.

Many learning control approaches involve learning to perform a particular movement, restricting their use to strictly repetitive operations [26]–[28]. In the proposed controller, however, learning is based solely on on-line observations of the system being controlled, independent of immediate control objectives. Learned information can therefore be utilized automatically to achieve new control objectives, as long as similar regions of the system state space are involved. The learning control approach described is different than many other learning schemes in that the objective is to learn how to control the robot, rather than how to perform a specific movement. In the data presented, good controller and image parameter prediction performance were obtained for both repetitive and random trials, after sufficient training.

Some studies of neural-network-based control for robotics focus on open-loop representations of geometric

transformations rather than on closed-loop control [11]–[15]. Typically, a training exercise is used to train the network to approximate the desired transformation over the entire operating space of the system. While such exhaustive training leads to nonrepetitive operating possibilities, it is not practical for most robotic systems of reasonable complexity. For example, training a network to produce the correct drive signals to achieve every possible vector of desired joint accelerations, for every possible initial condition of joint velocities and positions, clearly would be difficult for a six axis industrial robot.

The neural-network-based controller being developed in our laboratory learns continuously during actual task execution without an explicit training phase. While the network is trained to represent the system characteristics, making it applicable in a nonrepetitive environment, it only learns the characteristics in the regions of the control space through which it has traveled. This is reasonable, since it is difficult to imagine a useful nonrepetitive task which truly spans the entire control space of a complex robotic system. No *a priori* limitation is placed on the extent of the control space by the network, but it is only necessary for it to learn the information required to complete the task, whether repetitive or nonrepetitive. A more complex task merely involves training over a larger region of the control space, possibly requiring a larger weight vector memory than for a simple task, but requiring no other adaptation of the control system.

The experiments presented in this paper demonstrate that practical real-time robot control systems using neural network learning techniques to develop nonlinear control transformations and sensor data processing can be implemented using low-cost microprocessors and practical amounts of memory. Learning control computations required approximately 12 ms for a 12-input, four output problem implemented on a low-cost TMS32010 microprocessor, acting as an auxiliary to the host VAX-11/730. The required computations increase only linearly with the number of input variables, even though the number of possible cross terms increases much faster. Networks containing 16384 weight vectors were found to be adequate for the control problems studied. These specifications should be adequate for application to many practical robot control problems.

Appendix
The CMAC Network

Fig. 7 shows a simple example of the CMAC network as implemented in our laboratory. Each variable in the input state vector s is fed to a series of input sensors with overlapping receptive fields. The width of the receptive field of each sensor produces input generalization, while the offset of the adjacent fields produces input quantization. Each input variable excites exactly C input sensors ($C = 4$ in Fig. 7, $C = 64$ in the experiments).

The outputs (on or off) of the input sensors are combined in a series of threshold logic units (called state space detectors) with thresholds adjusted to produce logical AND functions (the output is on only if all inputs are on). Each of these units receives one input from the group of sensors for each input variable, and thus its input receptive field is the interior of a hypercube in the input hyperspace. If the input sensors were fully interconnected, a very large number of state space detectors would be excited for each possible input. The input sensors are interconnected in a sparse and regular fashion, however, so that each input vector excites exactly C state space detectors.

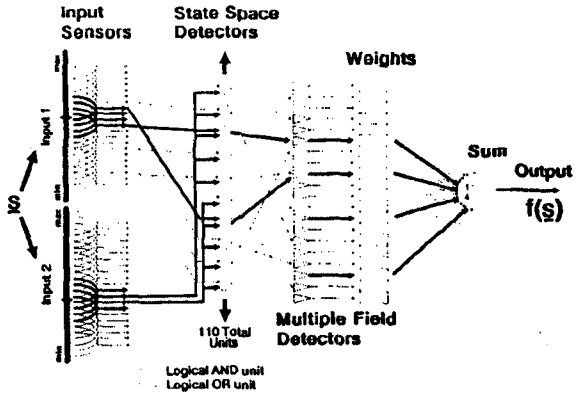


Fig. 7. Simple example of CMAC network architecture with two inputs, one output, and four active units per input ($C = 4$). Note that only subset of state space detectors is shown. CMAC networks used in experiments had 12 inputs, four outputs, 16384 weight vectors, and 64 active units per input.

The outputs of the state space detectors are connected randomly to a smaller set of threshold logic units (called multiple field detectors) with thresholds adjusted such that the output will be on if any input is on (a logical OR function). The receptive field of each of these units is thus the union for the fields of many of the state space detectors. Since exactly C state space detectors are excited by any input, at most C multiple field detectors will be excited by any input. The converging connections between the large set of state space detectors and the smaller set of multiple field detectors are referred to as "collisions."

Finally, the output of each multiple field detector is connected, through an adjustable weight, to an output averaging unit. The output for a given input is thus the average of the weights selected by the excited multiple field detectors.

Network training is typically based on observed training data pairs s_0 and f_0 (supervised learning), where f_0 is the desired network output in response to the input s_0, using a least mean square (LMS) training rule [29] as follows. The observed input state s_0 is input to the network and the network output $f(s_0)$ is determined. The weight adjustment value

$$\Delta w = \beta(f_0 - f(s_0))$$

is then added to each of the weights selected by the input state s_0. The influence of each such training cycle can be controlled by the training gain β (between zero and one). If β is chosen as 1.0, $f(s_0)$ is set equal to f_0 as the result of

the training step. If β is less than 1.0, $f(s_0)$ is changed in the direction of f_0. If β is chosen as zero, no learning occurs.

While the CMAC network can be trained to represent arbitrary nonlinear relationships between the input vector s and the output $f(s)$, convergence of the weights during training can be demonstrated by comparing the CMAC structure with well-known linear adaptive elements [29]. Let x represent the vector of binary outputs of the multiple field detectors in Fig. 7, and let w represent the vector of adjustable weights. For each training input s_0, there is a corresponding pattern of multiple field detector outputs x_0 that is dependent on the fixed pattern of the CMAC network interconnections but is independent of the values of the weights. The weight update described above can then be written as

$$\Delta w = \beta\left(f_0 - w^T x_0 / C\right) x_0$$

which is the well-known Widrow–Hoff rule for linear adaptive elements [30].

The nonlinear nature of the CMAC network is embodied in the interconnections of the input sensors, state space detectors, and multiple field detectors that perform a fixed nonlinear mapping of the continuous-valued input vector s to a many-dimensional binary-valued vector x (which had 16 384 dimensions in the experiments). The adaptation problem is linear in this many-dimensional space. All of the convergence theorems for linear adaptive elements [29] thus apply, and it can be expected that with repetitive training, using a value of β between zero and one, the networks weights will converge to values which minimize the rms error between the network predictions and the training data.

References

[1] R. P. Paul, *Robotic Manipulators — Mathematics, Programming and Control*. Cambridge, MA: MIT Press, 1981.

[2] C. S. Lee, "Robot arm kinematics, dynamics, and control," *Computer*, vol. 15, no. 12, pp. 62–80, Dec. 1982.

[3] F. Miyazaki and S. Arimoto, "Sensory feedback for robot manipulators," *J. Robotic Syst.*, vol. 2, no. 1, pp. 53–71, Spring 1985.

[4] A. C. Sanderson and L. E. Weiss, "Image-based visual servo control of robots," *Proc. SPIE*, vol. 360, pp. 164–169, Aug. 1982.

[5] ____, "Adaptive visual servo control of robots," in *Robot Vision*, A. Pugh, Ed. New York: Springer-Verlag, 1983, pp. 107–116.

[6] L. E. Weiss, A. C. Sanderson, and C. P. Neuman, "Dynamic visual servo control of robots: An adaptive image-based approach," in *Proc. 1985, IEEE Int. Conf. Robotics and Automation*, St. Louis, MO, Mar. 1985, pp. 662–668.

[7] T. H. Chiu, A. J. Koivo, and R. Lewezyk, "Experiments on manipulator gross motion using self-tuning controller and visual information," *J. Robotic Syst.*, vol. 3, no. 1, pp. 59–70, Spring 1986.

[8] L. E. Weiss, A. C. Sanderson, and C. P. Neuman, "Dynamic sensor-based control of robots with visual feedback," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 404–417, Oct. 1987.

[9] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-13, pp. 834–846, Sept./Oct. 1983.

[10] T. Kohonen, *Self-Organization and Associative Memory*. Heidelberg: Springer-Verlag, 1984, pp. 145.

[11] H. Ritter and K. Schulten, "Topology conserving mappings for learning motor tasks," in *AIP Conf. Proc. 151, Neural Networks for Computing*, Snowbird, UT, 1986, pp. 376–380.

[12] M. Kuperstein, "Adaptive visual-motor coordination in multijoint robots using parallel architecture," in *Proc. 1987 IEEE Int. Conf. Robotics and Automation*, Raleigh, NC, 1987, pp. 1595–1602.

[13] A. Sideris, A. Yamamura, and D. Psaltis, "Dynamical neural networks and their application to robot control," in *Proc. IEEE Conf. Neural Information Processing Systems — Natural and Synthetic*, Denver, CO, 1987, p. 29.

[14] B. W. Mel, "MURPHY: A robot that learns by doing," to appear in *AIP Proc. 1987 Neural Information Processing System Conf.*, Denver, CO, 1987.

[15] A. Pellionisz, "Tensor geometry: A language of brains and neurocomputers. Generalized coordinates in neuroscience and robotics," in *Neural Computers*, R. Eckmiller and Ch. V. D. Malsburg, Eds. Heidelberg: Springer-Verlag, 1988, pp. 381–391.

[16] M. Kawato, K. Furukawa, and R. Suzuki, "A hierarchical neural-network model for control and learning of voluntary movement," *Biol. Cybern.*, vol. 57, pp. 169–185, 1987.

[17] J. Barhen, W. B. Dress, and C. C. Jorgensen, "Applications of concurrent neuromorphic algorithms for autonomous robots," in *Neural Computers*, R. Eckmiller and Ch. V. D. Malsburg, Eds. Heidelberg: Springer-Verlag, 1988, pp. 321–333.

[18] W. T. Miller, F. H. Glanz, and L. G. Kraft, "Application of a general learning algorithm to the control of robotic manipulators," *Int. J. Robotics Res.*, vol. 6.2, pp. 84–98, Summer 1987.

[19] W. T. Miller, "A nonlinear learning controller for robotic manipulators," *Proc. SPIE: Intelligent Robots and Computer Vision*, vol. 726, pp. 416–423, October 1986.

[20] ____, "A learning controller for nonrepetitive robotic operations," in *Proc. Space Telerobotics*, Pasadena, CA, Jan. 1987, JPL publ. 87-13, vol. II, pp. 273–281.

[21] ____, "Sensor based control of robotic manipulators using a general learning algorithm," *IEEE J. Robotics Automt.*, vol. RA-3, pp. 157–165, Apr. 1987.

[22] J. S. Albus, "Theoretical and experimental aspects of a cerebellar model," Ph.D. dissertation, Univ. of Maryland, Dec. 1972.

[23] ____, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *Trans. ASME, J. Dynamic Syst. Meas. Contr.*, vol. 97, pp. 220–227, Sept. 1975.

[24] ____, "Mechanisms of planning and problem solving in the brain," *Math. Biosci.*, vol. 45, pp. 247–293, Aug. 1979.

[25] ____, *Brain, Behavior, and Robotics*. Peterborough, NH: BYTE Books, 1981, pp. 139–179.

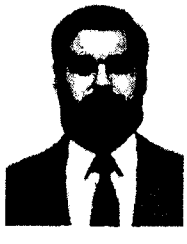
[26] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *J. Robotic Syst.*, vol. 1, pp. 123–140, 1984.

[27] C. G. Atkeson and J. McIntyre, "Robot trajectory learning through practice," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, Apr. 1986, pp. 1737–1742.

[28] M. Togai and O. Yamano, "Learning control and its optimality: Analysis and its application to controlling industrial robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, Apr. 1986, pp. 248–253.

[29] B. Widrow and S. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1985.

[30] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 Wescon Conv. Rec.*, 1960, pp. 96–104.



W. Thomas Miller, III (M'79) received the B.S. degree in electrical engineering in 1972, and the M.S. and Ph.D. degrees in bioengineering from the Pennsylvania State University, University Park, PA, in 1974 and 1977, respectively.

He served for two years as a Postdoctoral Fellow in biomedical engineering at the Duke Medical Center, and is currently an Associate Professor of Electrical and Computer Engineering at the University of New Hampshire, Durham, NH. During his graduate and postgraduate studies he specialized in bioelectric phenomena, including computer modeling, real time processing and automatic interpretation of bioelectric signals. His current research is focused on the design of neural network architectures for problems in control, pattern recognition, and signal processing.