Communicated by Michael Jordan

Learning Virtual Equilibrium Trajectories for Control of a Robot Arm

Reza Shadmehr
Department of Computer Science, University of Southern California, Los Angeles, CA 90089-0782 USA

The *cerebellar model articulation controller* (CMAC) (Albus 1975) is applied for learning the inverse dynamics of a simulated two joint, planar arm. The actuators were antagonistic muscles, which acted as feedback controllers for each joint. We use this example to demonstrate some limitations of the control paradigm used in earlier applications of the CMAC (e.g., Miller et al. 1987, 1990): the CMAC learns dynamics of the arm and not those of the feedback system. We suggest an alternate approach, one in which the CMAC learns to manipulate the feedback controller's input, producing a virtual trajectory, rather the controller's output, which is torque. Several experiments are performed that suggest that the CMAC learns to compensate for the dynamics of the plant, as well as the controller.

1 Introduction

Flash (1987) has shown that in the human arm, for moderate speed movements, the spring-like behavior of the neuromuscular system is such that by manipulating an *equilibrium point* model of the arm, the CNS can produce relatively accurate movements even without considering the dynamics of the moving limb. In order to produce a precise trajectory, however, the applied neuromuscular activity should take into account the dynamics of the skeleton, as well as those of the attached muscles and the segmental feedback system. In this paper we show how one might learn to produce such a *virtual equilibrium trajectory* (Hogan 1984a).

The learning paradigm is based on the *cerebellar model articulation controller* (CMAC) as proposed by Albus (1975), and demonstrated in the works of Miller et al. (1987, 1990) and Kraft and Campagna (1990). The CMAC is a coarse-coding technique that is implemented as a look-up table for approximating a piece-wise continous function with multiple input and output variables. In Miller et al. (1987), for example, the function was the inverse dynamics of a robot, the input variables described the desired state of the robot, the output variables were joint torques, and the coding was done by layers of perceptrons that mapped the immense input space into a much smaller output table. The output of the network (torque) was then added to the output of a fixed-gain, error-feedback controller.

In the application that we have considered here, the feedback controllers are the antagonistic muscles attached to the joints. This example will illustrate a limitation of the control scheme proposed by Miller et al. (1987, 1990): If the controller's response depends on something other than the error in the plant's output, then the CMAC will never be able to compensate for the dynamics of the controller, leading to persistent errors in the plant's behavior. We propose that this limitation can be addressed if the CMAC learns to modify the input to the controller rather than the controller's output. In effect, the CMAC will learn the dynamics of the skeleton, as well as the muscles that act on it.

1.1 Arm Dynamics. Consider a two joint arm, with a pair of antagonistic muscles attached to each joint (Fig. 1). In the idealized case, shoulder and elbow torques, $\boldsymbol{T} = [T_s\ T_e]$, can be written as a function of

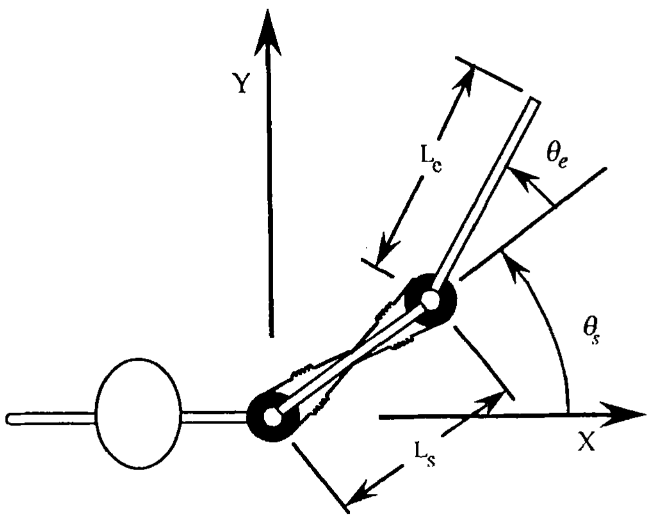

Figure 1: Schematic of the arm with the muscle-like actuators. End-effector positions in the text refer to a Cartesian coordinate system centered at the shoulder. Length of the upper-arm and forearm are 0.25 and 0.35 m, respectively.

joint position $\boldsymbol{\theta} = [\theta_s\, \theta_e]$, velocity $\dot{\boldsymbol{\theta}}$, and acceleration $\ddot{\boldsymbol{\theta}}$, where m, l, s, r, and b represent the mass, link length, distance from the center of mass to joint, rotary inertia, and viscosity of the joint:

$$\begin{aligned} T_s &= (r_s + r_e + 2m_e l_s s_e \cos\theta_e + m_e l_s^2)\ddot{\theta}_s \\ &\quad - m_e l_s s_e (2\dot{\theta}_s + \dot{\theta}_e)\dot{\theta}_e \sin\theta_e + b_s\dot{\theta}_s \\ &\quad + (r_e + m_e l_s s_e \cos\theta_e)\ddot{\theta}_e \qquad (1.1) \\ T_e &= (r_e + m_e l_s s_e \cos\theta_e)\ddot{\theta}_s + r_e\ddot{\theta}_e \\ &\quad + m_e l_s s_e \dot{\theta}_s^2 \sin\theta_e + b_e\dot{\theta}_e \qquad (1.2) \end{aligned}$$

We modeled the arm described in (1.1–1.2) using parameter values in Uno et al. (1989). Forward dynamics were calculated by solving (1.1–1.2) for $\ddot{\boldsymbol{\theta}}$. Given a torque vector $\boldsymbol{T}(t)$ at some $\boldsymbol{\theta}(t)$ and $\dot{\boldsymbol{\theta}}(t)$, the resulting $\ddot{\boldsymbol{\theta}}(t)$ was integrated to specify $\dot{\boldsymbol{\theta}}(t+\Delta)$ and $\boldsymbol{\theta}(t+\Delta)$.

We assumed that the force generated by a muscle can be essentially represented by considering its dependence on muscle length, velocity of contraction, and activation rate (Hogan 1984a). The torque generated by a pair of muscles acting on the shoulder joint, for example, was defined as $T_s = K(\phi_s - \theta_s) - B\dot{\theta}_s$, where K is joint stiffness, B is the joint's viscous coefficient, and ϕ_s is the equilibrium position of the joint (Flash 1987).

1.2 Trajectory Generation. Hogan (1984b) has suggested that for reaching movements, a trajectory is planned in which the change in acceleration of the hand (jerk) over the period of movement is minimized. For our case, in moving from an initial hand position $(x_i\, y_i)$ at $t = 0$ to $(x_j\, y_j)$ at $t = \alpha$, the function to be minimized is

$$E = \frac{1}{2}\int_0^{\alpha} (\frac{d^3x}{dt^3})^2 + (\frac{d^3y}{dt^3})^2 dt \qquad (1.3)$$

If motion begins and ends with zero velocity and acceleration, then it can be shown that hand trajectory always follows a straight line, and is described by

$$\begin{aligned} x(t) &= x_i + (x_i - x_j)(15\tau^4 - 6\tau^5 - 10\tau^3) \qquad (1.4) \\ y(t) &= y_i + (y_i - y_j)(15\tau^4 - 6\tau^5 - 10\tau^3) \qquad (1.5) \end{aligned}$$

where $\tau = t/\alpha$. Applying inverse kinematics to (1.4–1.5) leads to a trajectory in joint coordinates.

As an example, consider the case where the muscle parameters are set as follows: $K = 40$ N·m/rad, and $B = 2$ N·m/sec/rad, and the objective is to move the hand from $(-0.3\ 0.2)$ to $(0.1\ 0.5)$ in $\alpha = 0.7$ sec (see

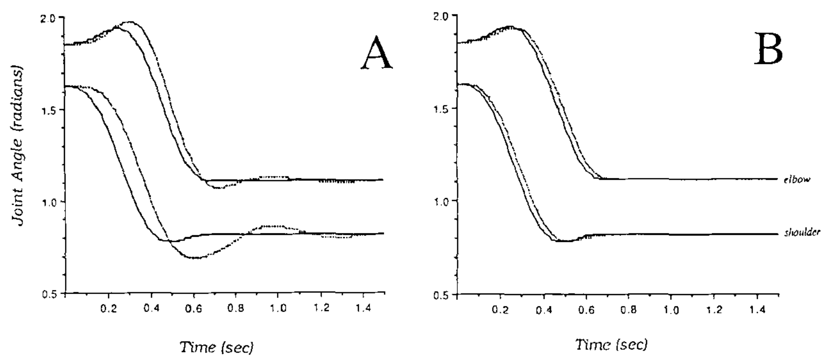

Figure 2: Desired and observed joint trajectories before and after learning. (A) Performance of the system before learning begun (RMSE = 0.1146 rad). (B) Performance of the system after the tenth learning iteration (RMSE = 0.073 rad).

Fig. 1). The desired and observed trajectories are plotted in Figure 2A. The RMS error (RMSE), averaged over 1.5 sec, was 0.1146 rad for this movement. Therefore by simply manipulating the equilibrium of the antagonist muscles, a reasonably accurate movement was accomplished. Our objective is to minimize this error.

2 CMAC and Adaptive Control

In learning the inverse dynamics of a manipulator, the CMAC has generally been used in control structures similar to Figure 3A: Here the CMAC maps a joint trajectory into torques, and this torque is then added to the controller's output (Miller et al. 1987, 1990). Our results will show the limitations of this approach, and an alternate approach will be presented where the CMAC learns a virtual equilibrium trajectory rather than joint torques (Fig. 3B).

Consider the control system of Figure 3A. At time t, the arm is at an observed state $Q_o(t) = [\theta(t)\ \dot{\theta}(t)\ \ddot{\theta}(t)]$. We would like the arm to be at $\phi(t)$, which is the equilibrium position generated by the minimum jerk trajectory. Actuators compare $\phi(t)$ to the currently observed position $\theta(t)$, and produce a torque $T_m(t) = K[\phi(t) - \theta(t)] - B\dot{\theta}(t)$. Next, a desired state $Q_d(t) = [\theta(t)\ \dot{\theta}(t)\ \ddot{\theta}_d(t)]$ is produced and given to the CMAC, which produces a torque $T_c(t)$. $T_c(t) + T_m(t)$ is applied to the arm. $\ddot{\theta}_d(t)$ is calculated as follows: From our current position $\theta(t)$ and *velocity* $\dot{\theta}(t)$,

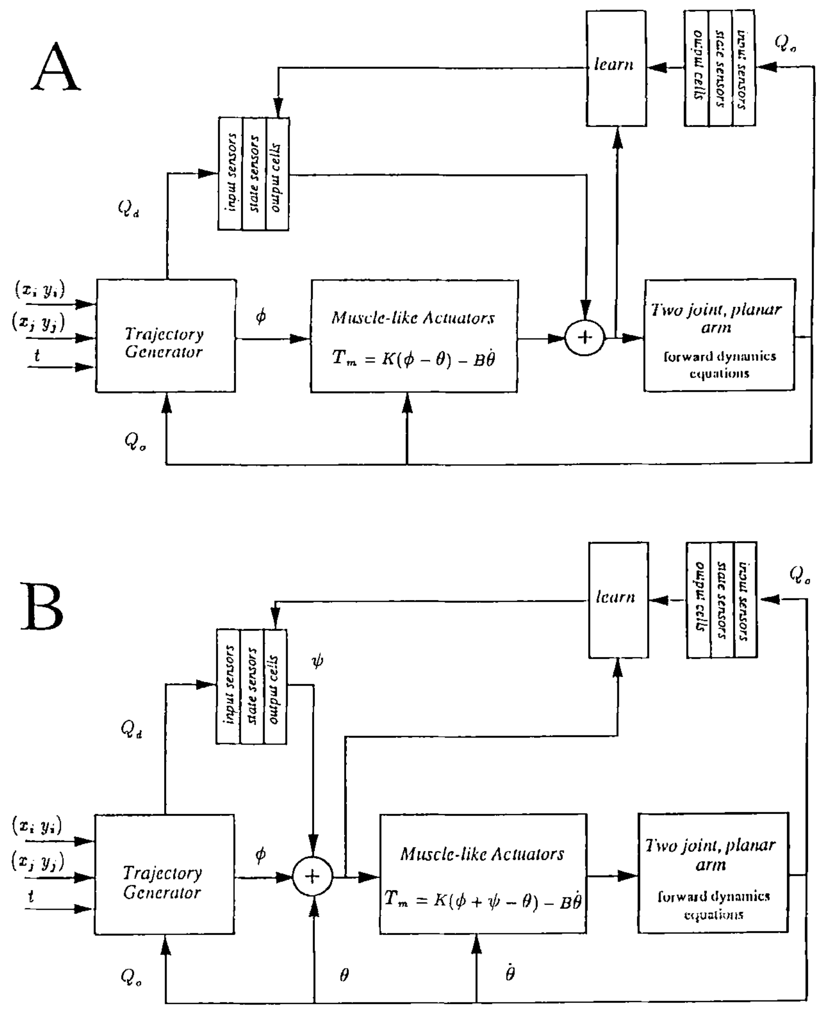

Figure 3: Two approaches to adaptive control. (A) This is the configuration that was used in Experiment 1. The task is to guide the arm along a minimum jerk equilibrium trajectory $\phi(t)$ from $(x_i\ y_i)$ to $(x_j\ y_j)$ in α seconds. Q_o is the observed state of the arm, and Q_d is the desired state. CMAC's output is a torque vector T_c, and it learns the dynamics of the arm but not the dynamics of the muscles. (B) The configuration used in the rest of the experiments. CMAC's output is a joint position vector ϕ, and $(\phi + \theta)$ is the *virtual equilibrium trajectory*. CMAC learns dynamics of the lumped system.

we wish to accelerate at a rate of $\ddot{\boldsymbol{\theta}}_d(t)$ for 50 msec to reach $\boldsymbol{\phi}(t)$. A second-order polynomial approximation method was used to solve for $\ddot{\boldsymbol{\theta}}_d(t)$. Next, the effects of the applied torques are calculated from the forward dynamics, and $\boldsymbol{Q}_o(t+\Delta)$ is observed. $\boldsymbol{Q}_o(t+\Delta)$ is given to the CMAC and the output $\boldsymbol{T}_c(t+\Delta)$ is compared to $\boldsymbol{T}_c(t)+\boldsymbol{T}_m(t)$. Finally, the contents of the CMAC's activated output cells are updated by an amount proportional to $\boldsymbol{T}_c(t)+\boldsymbol{T}_m(t)-\boldsymbol{T}_c(t+\Delta)$.

The feedback loop serves as the teacher to the CMAC in Figure 3A: It provides reasonably appropriate torques as the equilibrium trajectory $\boldsymbol{\phi}(t)$ deviates from $\boldsymbol{\theta}(t)$. But since the output of this controller never vanishes (due to the $B\dot{\boldsymbol{\theta}}$ term), the inverse dynamics map $\boldsymbol{Q}_o \rightarrow \boldsymbol{T}_c$ that is learned by the CMAC will always be "corrupted" by the controller's output. In Experiment 1 we will see that after learning, the CMAC's performance can be improved further if the feedback controller is disconnected.

The control scheme in Figure 3B is an alternate approach where the CMAC learns to control the controller, rather than augmenting its output. Here the output of the CMAC is not a torque, but a joint position vector $\boldsymbol{\psi}$. $\boldsymbol{\psi}+\boldsymbol{\phi}$ is the *virtual equilibrium position*, and $\boldsymbol{\psi}+\boldsymbol{\phi}-\boldsymbol{\theta}$ is the virtual error in position that is given to the controller. Experiments 2–4 are examples where the CMAC learns a $\boldsymbol{Q}_o \rightarrow \boldsymbol{\psi}$ mapping. In these experiments, the CMAC compensates for the dynamics of the plant as well as those of the controller.

2.1 Learning Inverse Dynamics.

For the arm in Figure 1, the state vector $\boldsymbol{Q}$ consists of six parameters: Shoulder and elbow position (bounded within -1 and 3 rad), velocity (bounded within -20 and 20 rad/sec), and acceleration (bounded within -100 and 100 rad/sec^2). CMAC (Albus 1975) is one method for mapping this six-dimensional space onto a two-dimensional space, which after learning will represent joint torque in the case of the control system in Figure 3A, and virtual joint position in the case of Figure 3B.

2.1.1 *Network Architecture.*

In the first layer of our CMAC, 400 input sensors encoded each parameter (so the input space was quantized into 400^6 segments and, for example, joint position was encoded at a resolution of 0.01 rad). Each input sensor had an 80-unit-wide receptive field, and each segment within this input space mapped onto a second layer of cells that acted as state sensors (there were 1.25×10^6 state sensors). For each input a unique set of 80 state sensor cells would be activated. Although this encoding reduced the memory requirements for representing the input space by a factor of 10^9, nevertheless, if each state sensor pointed to a memory location containing two real numbers, that is, a torque for each joint, then the required memory space would be 10 MB, exceeding the author's available machine memory. Following Miller et al.'s (1987) approach, a second mapping was done to overcome this problem: Initially, an output table containing 31,250 cells (each holding two real numbers) was constructed. Then a many-to-one mapping was performed from the state sensors to the output cells using a hashing function (an output cell had ~40 state sensors mapping onto it). For an input $\boldsymbol{Q}$, the contents of all activated output cells were summed to form $\boldsymbol{T}_c$.

2.1.2 Experiment 1. Here the CMAC attempted to learn the torque sequence necessary for moving the arm along the same trajectory as that shown in Figure 2A. We began with the output cells all set to zero. At this stage, the RMSE was 0.1146 rad. On the first attempt with the learning scheme in place, the RMSE was reduced to 0.0810. By the tenth such attempt, the RMSE was at 0.073 (Fig. 2B). Note that the overshoot and oscillatory behavior of the feedback controller have been eliminated, yet the arm still lags the desired trajectory.

To see the contribution of the CMAC as compared to the controller for this movement, we plotted the shoulder torque generated by each system before (Fig. 4A) and after (Fig. 4B) learning. In Figure 4A, the applied torque is the response of a damped-spring, and the CMAC's contribution is zero because its content has been initialized. After the tenth iteration (Fig. 4B), the CMAC's output has begun to dominate the output of the feedback controller. At this point, the controller was removed

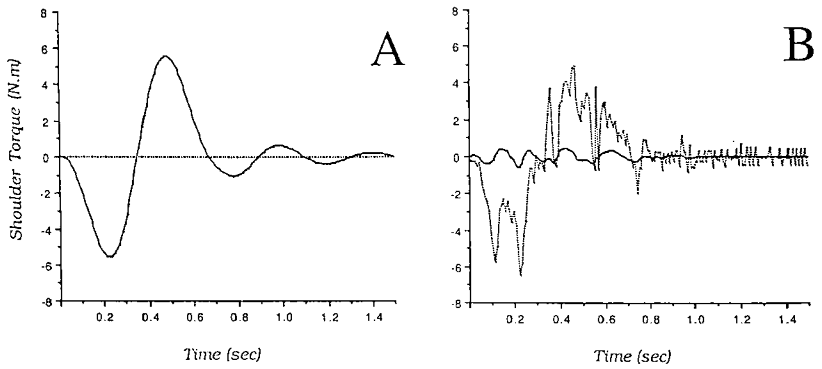

Figure 4: Torque contribution of the controller (solid line) and CMAC (dotted, more "noisy" line) to the shoulder joint during movement of Figure 2, using the control scheme of Figure 3A. (A) Before learning: All of the torque is due to the controller. (B) After the tenth learning iteration: Most of the torque is due to the CMAC.

from the loop and the same trajectory was repeated. This improved the performance of the CMAC by a factor of 5 (RMSE = 0.013 rad). We concluded that although the CMAC had essentially learned the inverse dynamics of the plant, it could not compensate for the effects of the controller because *it had not witnessed the input/output behavior of the lumped system*.

2.2 Learning Virtual Trajectories. We investigated the utility of the control structure depicted in Figure 3B by performing three experiments. In all cases, the CMAC was identical to the one used in the Experiment 1. The learning scheme was as follows: At time t, for a virtual position error $\psi(t)+\phi(t)-\theta(t)$, an observed state vector $Q_o(t+\Delta)$ was produced. $Q_o(t+\Delta)$ was given to the CMAC, which produced $\psi(t+\Delta)$, and the activated output cells were updated by an amount proportional to $\psi(t)+\phi(t)-\theta(t)-\psi(t+\Delta)$.

2.2.1 Experiment 2. The movement in Figure 2A was repeated with the control scheme of Figure 3B. Initially, the RMSE was 0.1146 rad. After the first iteration, RMSE fell to 0.0704. Following the tenth itera-

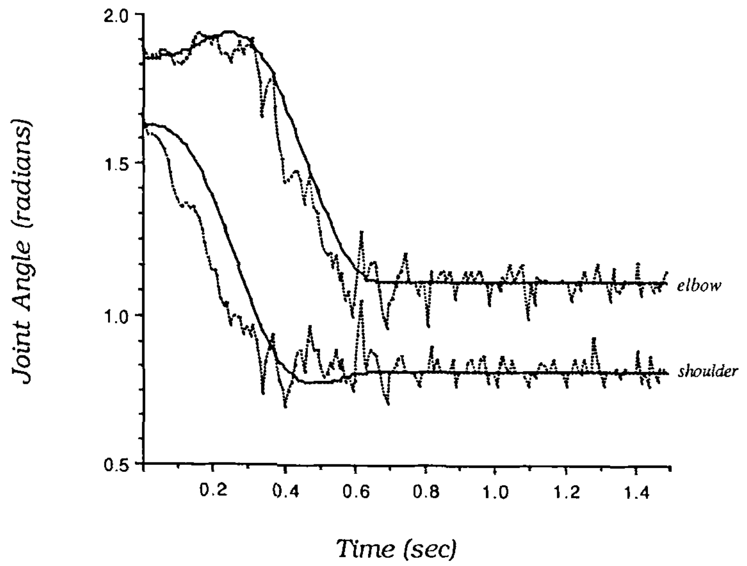

Figure 5: The equilibrium trajectory (solid lines) and virtual equilibrium trajectory (dotted line, the "noisy" signal) after the tenth learning iteration of the CMAC with the control structure of Figure 3B.

tion, RMSE was at 0.0086 rad. We have plotted the virtual equilibrium trajectory $\psi(t) + \phi(t)$ along with the equilibrium trajectory $\phi(t)$ in Figure 5. Intuitively, one would expect that in order to accelerate the arm along a particular trajectory, a larger than observed positional error would have to be presented to the spring-like muscles in order to start the movement. Using the same analogy, an early reversal in joint positional error terms needs to be implemented in order to brake the system at some position, and not have it oscillate there. Note that in Figure 5, the virtual trajectory is leading the equilibrium trajectory, and then braking and finally clamping it at the goal position.

2.2.2 *Experiment* 3. Here we tested the system on a much faster movement. In Figure 6A we have plotted the response of the system before learning begun (RMSE = 0.1571 rad). The performances of the CMAC after the first, tenth, and the fortieth iteration are plotted in Figures 6B, C, and D, respectively. The RMSE associated with these iterations were 0.1481, 0.0312, and 0.0089 rad.

2.2.3 *Experiment* 4. Can the system of Figure 3B learn the dynamics of the arm for a wide range of movements in a nonrepeating protocol? To investigate this, the initial position of the hand was fixed at (0.0 0.3), and the target position was randomly selected along the perimeter of a circle with radius of 0.25. The movement time was randomly selected within the range of 0.5 to 1.0 sec. After 500 such center-out movements, the average RMSE for the next 10 movements was 0.0089 rad, compared to an average RMSE of 0.1029 rad for the case where the CMAC was a *tabula rasa*.

3 Conclusions

In this work we have been concerned with the problem of learning inverse dynamics of a plant and its controller. We used the example of a robot arm with muscle-like actuators to illustrate the limitations of the learning/control system of Figure 3A. In Experiment 1 it was shown that after learning, the performance of the CMAC could be further improved if the control loop was disconnected and the CMAC allowed to run in a feedforward mode. It was suggested that instead of learning to modify the controller's output, the CMAC should be set up to augment the controller's input, therefore learning a virtual equilibrium trajectory (Experiments 2–4), rather than joint torques. The basic principle is to control a controlled system by supervised learning on the lumped system's input/output relationship.

Recent results on use of the CMAC have shown it to be a particularly useful approach for rapid learning of nonlinear functions (as compared to backpropagation, for example). This is because (1) the network uses local

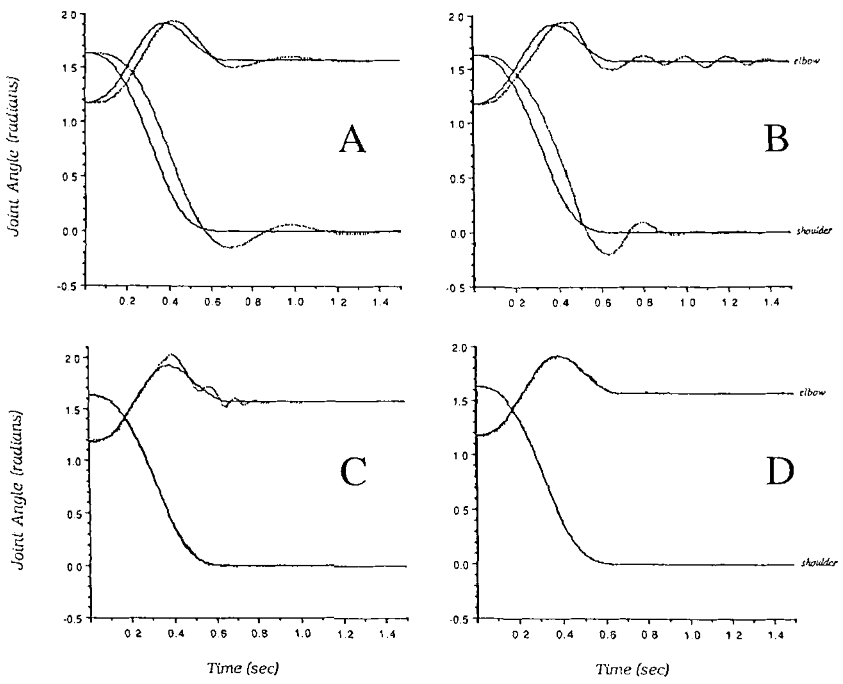

Figure 6: Learning a fast movement using the control scheme of Figure 3B. (A) Before learning begun (RMSE = 0.1571 rad). (B) The first learning iteration (RMSE = 0.1481). (C) The tenth iteration (RMSE = 0.0312). (D) The fortieth iteration (RMSE = 0.0089).

representation of the input space, thus requiring evaluation and modification of only a few output cells, and (2) the learning is a quadratic optimization procedure, avoiding the problem of local minimas. The mapping from the input space onto the output cells is the key to the CMAC: The idea is to not only activate a unique set of output cells for each input vector, but do so in such a way that similar input states share a large number of output cells, while far-away input states share no output cells. Recently, Moody (1989) has suggested two improvements to this basic network architecture. These include the use of a neighborhood function with graded response to overcome the potential problem of response discontinuity over state boundaries, and a multiresolution interpolation scheme where a hierarchy of CMACs work in parallel to provide high resolution along with good generalization abilities.

Acknowledgments

The author is supported by an IBM Graduate Fellowship in Computer Science. This work has been supported by a Grant-in-Aid of Research from the Sigma-Xi Foundation, and the NIH Grant 1R01-NS24926 (Prof. Michael Arbib, Principal Investigator). I am most grateful for the help of Prof. Tom Miller in constructing the CMAC.

References

Albus, J. S. 1975. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Trans. ASME J. Dynamic Syst. Meas. Contr.* **97**, 220–227.

Flash, T. 1987. The control of hand equilibrium trajectory in multi-joint arm movements. *Biol. Cybernet.* **57**, 257–274.

Hogan, N. 1984a. Adaptive control of mechanical impedance by coactivation of antagonist muscles. *IEEE Trans. Autom. Contr.* **AC-29**(8), 681–690.

Hogan, N. 1984b. An organizing principle for a class of voluntary movements. *J. Neurosci.* **4**(11), 2745–2754.

Kraft, L. G., and Campagna, D. P. 1990. A comparison between CMAC neural network control and two traditional adaptive control systems. *IEEE Control Syst. Magazine* **10**(3), 36–43.

Miller, W. T., Glanz, F. H., and Kraft, L. G. 1987. Application of a general learning algorithm to the control of robotic manipulators. *Int. J. Robotics Res.* **6**(2), 84–98.

Miller, W. T., Hewes, R. P., Glanz, F. H., and Kraft, L. G. 1990. Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller. *IEEE Trans. Robotics Automation* **6**(1), 1–9.

Moody, J. 1989. Fast learning in multi-resolution hierarchies. In *Advances in Neural Information Processing Systems*, D. S. Touretzky, ed., pp. 29–39. Morgan Kaufmann, San Mateo, CA.

Uno, Y., Kawato, M., and Suzuki, R. 1989. Formation and control of optimal trajectory in human multijoint arm movement: Minimum torque change model. *Biol. Cybernet.* **61**, 89–101.

Received 1 February 90; accepted 6 August 90.