A Multilayered Neural Network Controller

Demetri Psaltis, Athanasios Sideris, and Alan A. Yamamura

ABSTRACT: A multilayered neural network processor is used to control a given plant. Several learning architectures are proposed for training the neural controller to provide the appropriate inputs to the plant so that a desired response is obtained. A modified error-back propagation algorithm, based on propagation of the output error through the plant, is introduced. The properties of the proposed architectures are studied through a simulation example.

Introduction

The neural approach to computation has emerged in recent years [1]–[4] to tackle problems for which more conventional computational approaches have proven ineffective. To a large extent, such problems arise when a computer is asked to interface with the *real* world, which is difficult because the real world cannot be modeled with concise mathematical expressions. Problems of this type are machine vision, speech and pattern recognition, and motor control. Not coincidentally perhaps, these are precisely the types of problems that humans execute seemingly without effort. Therefore, people interested in building computers to tackle these problems have tried to adopt the existing understanding on how the brain computes. There are three basic features that identify a "neural computer": it consists of a very large number of simple processing elements (the neurons), each neuron is connected to a large number of others, and the functionality of the network is determined by modifying the strengths of the connections during a learning phase. These general characteristics are certainly similar to those evident in biological neural networks, however, the precise details of the operation of neural networks in a brain can be quite different from those in the abstract models used in the design of neural computers. Nevertheless, the general morphological features and the basic approach to computation are common in both natural and artificial neural networks. This can lead to computers that can tackle problems very effectively for which present approaches are relatively ineffective but natural intelligence is very good.

In this paper, we explore the application of the neural approach to control [5]–[15]. Both humans and machines perform control functions, however, there are sharp distinctions between machine and human control systems. For instance, humans make use of a much greater amount of sensory information in planning and executing a control action compared to industrial controllers. The reason for this difference is not as much sensor constraints but rather the inability of man-made controllers to efficiently absorb and usefully process such a wealth of information. The collective manner in which information is processed by neural networks is another distinction between human and machine control; it is this collective processing capability that provides neural networks with the ability to respond quickly to complex sensory inputs. Sophisticated control algorithms, on the other hand, are severely limited by the time it takes to execute them in electronic hardware. The third distinction, and perhaps the most important, is that human control is largely acquired through learning, whereas the operation of man-made controllers is specified by an algorithm that is written a priori. Therefore, in order to implement an effective algorithmic controller, we must have a thorough understanding of the plant that is to be controlled, something that is generally very difficult to achieve in practice. A neural controller performs a specific form of adaptive control, with the controller taking the form of a nonlinear multilayer network and the adaptable parameters being the strengths of the interconnections between the neurons. In summary, a controller that is designed as a neural network architecture should exhibit three important characteristics: the utilization of large amounts of sensory information, collective processing capability, and adaptation.

A general diagram for a control system is shown in Fig. 1. The feedback and feedforward controllers and the prefilter can all be implemented as multilayered neural networks. The learning process gradually tunes the weights of the neural network so that the error signal between the desired and actual plant responses is minimized. Since the error signal is the input to the feedback controller, the training of the network will lead to a gradual switching from feedback to feedforward action as the error signal becomes small. In this paper, we consider implementing only the feedforward controller as a neural network. During training, features of the plant that are initially unknown and not taken into account by the control algorithm are learned. In this manner, some of the model uncertainty is eliminated and, thus, improved control results. In other words, the feedforward controller is modified to compensate for the characteristics of the plant that are discovered during learning. When the error becomes small, training has been accomplished and only a relatively small feedback signal is necessary to compensate for random uncertainties that are unpredictable and, thus, cannot be learned. An immediate consequence of the increased use of feedforward control action is to speed up the response of the system.

In the remainder of this paper, we consider specific methods for training neural networks to minimize the error signal. In following sections, we propose three control learning methods, describe the error back propagation algorithm [16], which is the method used here to adapt the weights in the neural networks we use as controllers, and introduce a modification of the error back propagation algorithm that extends its utility to problems where the error signal used to train the network is not the error measured at the output of the network. We also present simulations for a very simple plant to demonstrate the operation of the proposed architectures and training methods.

Learning Control Architectures

Figure 2 shows the feedforward controller, implemented as a neural network architecture, with its output u driving the plant. The desired response of the plant is denoted

Presented at the 1987 IEEE International Conference on Neural Networks, San Diego, California, June 21-24, 1987. Demetri Psaltis, Athanasios Sideris, and Alan A. Yamamura are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125. This research was funded by DARPA and AFOSR and also in part by the Caltech President's Fund. Alan A. Yamamura was supported with a Fellowship from the Hertz Foundation.

0272-1708/88/0400-0017 $01.00 © 1988 IEEE

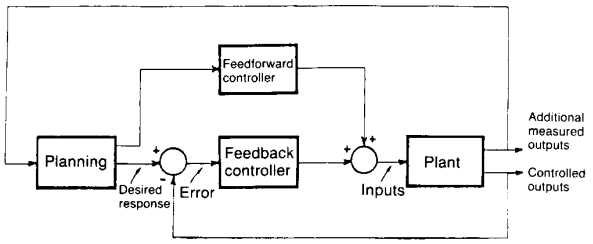


Fig. 1. General control system diagram.

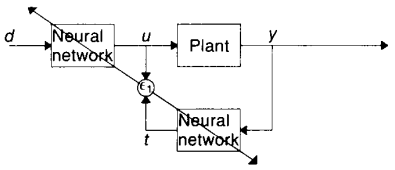


Fig. 2. Indirect learning architecture.

by d and its actual output by y. The neural controller should act as the inverse of the plant, producing from the desired response d a signal u that drives the output of the plant to $y \approx d$. The goal of the learning procedure, therefore, is to select the values of the weights of the network so that it produces the correct $d \mapsto u$ mappings at least over the range of d's in which we plan to operate the plant. We will consider three different learning methods. One possibility is suggested in Fig. 2. Suppose that the feedforward controller is successfully trained so that the plant output $y \approx d$. Then the network used as the feedforward controller will approximately reproduce the plant input from y (i.e., $t \approx u$). Thus, we might consider training the network by adapting its weights to minimize the error $\epsilon_1 = u - t$ using the architecture shown in Fig. 2, because, if the overall error $\epsilon = d - y$ goes to zero, so does ϵ_1. The positive features of this arrangement would be the fact that the network can be trained only in the region of interest since we start with the desired response d and all other signals are generated from it. In addition, as we will see in the next section, it is advantageous to adapt the weights in order to minimize the error directly at the output of the network. Unfortunately, this method, as described, is not a valid training procedure because minimizing ϵ_1 does not necessarily minimize ϵ. For instance, simulations with a simple plant showed that the network tends to settle to a solution that maps all d's to a single $u = u_0$, which, in turn, is mapped by the plant to $t = u_0$, for which ϵ_1 is zero but obviously ϵ is not. This training method remains interesting, however, because it could be used in conjunction with one of the procedures described below that minimize ϵ.

General Learning Architecture

The architecture shown in Fig. 3 provides a method for training the neural controller that does minimize the overall error ϵ^2. The training sequence is as follows. A plant input u is selected and applied to the plant to obtain a corresponding y, and the network is trained to reproduce u at its output from y. The trained network should then be able to take a desired response d and produce the appropriate u, making the actual plant output y approach d. This will clearly work if the input d happens to be sufficiently close to one of the y's that were used during the training phase. Thus, the success of this method is intimately tied to the ability of the neural network to generalize or learn to respond correctly to inputs it has not specifically been trained for. Notice that, in this architecture, we cannot selectively train the system to respond correctly in regions of interest because we normally do not know which plant inputs u correspond to the desired outputs d. Thus, we typically attempt to uniformly populate the input space of the plant with training samples so that the network can interpolate the intermediate points. In this case, the general procedure may not be efficient since the network may have to learn the responses of the plant over a larger operational range than is actually necessary. One possible solution to this problem is to combine the general

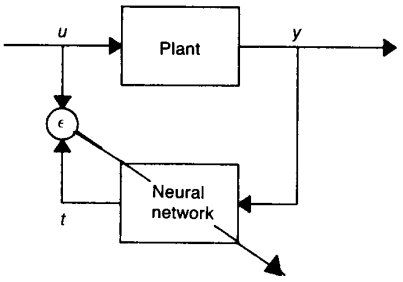


Fig. 3. General learning architecture.

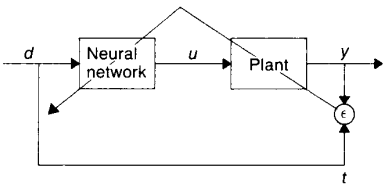


Fig. 4. Specialized learning architecture.

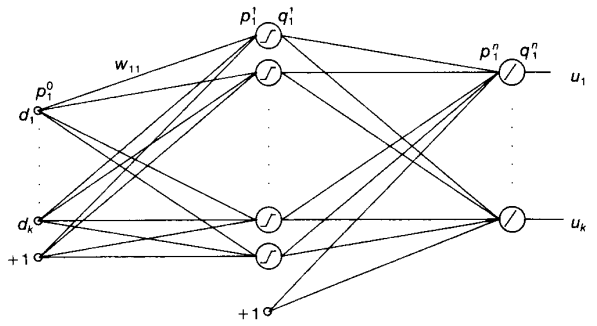
method with the specialized procedure described below.

Specialized Learning Architecture

Figure 4 shows an architecture for training the neural controller to operate properly in regions of specialization only. Training involves using the desired response, d, as input to the network. The network is trained to find the plant input, u, that drives the system output, y, to the desired d. This is accomplished by using the error between the desired and actual responses of the plant to adjust the weights of the network using a steepest descent procedure; during each iteration the weights are adjusted to maximally decrease the error. This procedure requires knowledge of the Jacobian of the plant. In the following section, we describe an iterative procedure for specialized training in which the plant derivatives are estimated continuously. This architecture can specifically learn in the region of interest, and it may be trained on-line—fine-tuning itself while actually performing useful work. The general learning architecture, on the other hand, must be trained off-line. Feedforward neural networks are nondynamical systems and, therefore, input–output stable. Consequently, off-line training of the neural network presents no stability problem for the control system. Intuitively, we expect no stability problems also in the case of on-line training, as long as the learning rate is sufficiently slower than the time constants of the other components of the control system.

Neural Net Training

The training method chosen may greatly affect the final operation of the system, its ability to adapt, and the time it takes to learn. Although neural networks come in many shapes and sizes, we consider here only networks that fit the form of Fig. 5. In this layered architecture, neurons in the same layer, m, perform the same function, f^m, on p^m, where the p_i^m's are the weighted sums of the outputs, q_j^{m-1}'s, of the previous layer. Training adjusts the connection strengths between the neurons, thus modifying the functional character of the network with the ob-



$[q_i^m = f_i^m(p_i^m),\ p_i^m = \Sigma_j\, w_{ij}^m q_j^{m-1}]$.

Fig. 5. Layered neural net.

jective of minimizing the training error, $\epsilon = t - o$, between the actual output, o, and target output, t, of the network.

Indirect Learning and General Learning Network Training

Training the first two proposed architectures is accomplished by error back propagation, a widely used algorithm for training feedforward neural networks. Error back propagation attempts to minimize the squared error for training sample input–output pairs; for every training sample, it modifies each weight using the partial derivative of the squared error with respect to that weight. Since error back propagation adjusts weights in the local direction of greatest error reduction, it is a gradient descent algorithm. For the network in Fig. 5, error back propagation would modify the weight, w_{ab}^m, between neurons a and b of the mth and $m-1$st layers, respectively, as follows:

$$\Delta w_{ab}^m \propto -\frac{\partial \|\bar{\epsilon}\|^2}{\partial w_{ab}^m}$$

$$= \rho \delta_a^m q_b^{m-1}$$

ρ is an acceleration constant that relates to the step size of the simulation; larger acceleration constants lead to lower accuracy but faster training. In the preceding equation, the output of the kth neuron is q_k, its input is p_k, and δ_k is the back-propagated error given for the final layer, n, by

$$\delta_a^n = f^{n\prime}(p_a^n)\,(u_a - t_a)$$

and for all other layers by

$$\delta_a^m = f^{m\prime}(p_a^m) \sum_i \delta_i^{m+1} w_{ia}^{m+1}$$

For the general learning network, we identify $o \equiv v$, where v is the output of the network to input y, and $t \equiv u$ which, when fed to the plant, produces y (see Figs. 2 and 3).

Specialized Learning Network Training

We cannot apply error back propagation directly to the specialized learning architecture because of the location of the plant. Referring to Fig. 4, the plant can be thought of as an additional, although unmodifiable, layer. Then the total error, $\epsilon = d - y$, is propagated back through the plant using the partial derivatives of the plant at its operating point:

$$\delta_a^n = f^{n\prime}(p_a^n) \sum_i \delta_i^p \frac{\partial P_i}{\partial u_a}$$

$$\delta_a^p = d_a - y_a$$

where $P_i(\bar{u})$ denotes the ith element of the plant output for plant input $\bar{u}$. The previously described error back propagation algorithm can then be applied. Therefore, *error back propagation through the plant* again amounts to a gradient descent search for weight combinations that minimize true total error.

If the plant is a function of unknown form, we can approximate its partial derivatives as

$$\frac{\partial P_i}{\partial u_j} \approx \frac{P_i(\bar{u} + \delta u_j \hat{i}_j) - P_i(\bar{u})}{\delta u_j}$$

This approximate derivative can be determined either by changing each input to the plant slightly at the operating point and measuring the change at the output or by comparing changes with previous iterations. The latter can be likened to a person comparing past and present experiences to determine how a system's behavior changes as its parameters change.

Generalized and Specialized Learning

A possible method for combining the two methods is to first perform general training to learn the approximate behavior of the plant followed by specialized training to fine-tune the network in the operating regime of the system. General training will have a tendency to create better initial weights for specialized training. Thus, starting with general training can speed the learning process by reducing the number of iterations of the ensuing specialized training. Another advantage of preliminary general learning is that it may result in networks that can adapt more easily if the operating points of the system change or new ones are added.

The distinction between general and specialized learning arises from the fact that different error functions are minimized. As a result, the general and specialized learning procedure will follow different paths to minima. Intuitively, we expect the general learning procedure to produce a network that approximates the inverse of the plant better over the entire state space but not as well in the regions as specialization as one produced by the specialized procedure. By adopting the strategy of switching back and forth between the two training methods, we can sometimes get out of local minima of one method by training with the other. Specifically, by performing general learning prior to specialized learning, we generally provide a better initial condition to the specialized procedure in that it has lower initial error. Switching from one training method to the other can also result in sudden changes of direction in weight space. This is clearly evident in the simulations presented in the next section.

Simulation Example

We considered as a simple example a plant that converts polar coordinates (r, θ) to Cartesian coordinates (x, y). The control network should convert Cartesian to polar coordinates. Desirable characteristics of this system include a well-behaved plant and a simple mathematical form for the desired network so that the performance of the neural network may be checked easily.

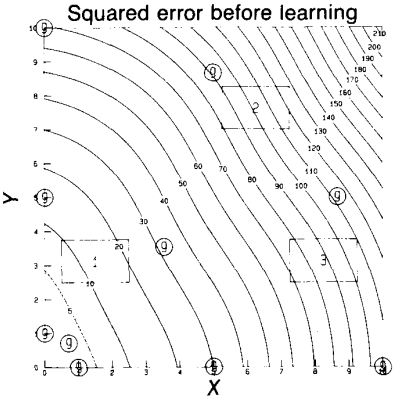
We chose a two-layer architecture with two inputs plus a fixed-unity input, 10 hidden neurons plus a fixed-unity hidden neuron, and two output neurons (see Fig. 5). The fixed units allow each neuron to find its own threshold value through the training procedure. The hidden neurons have a sigmoid transfer function, $f(x) = 1/[1 + \exp(-x)]$. We chose linear neurons, $f(x) = x$, for the output so that they have unlimited range. Initial weights were selected randomly as $\pm\frac{1}{2}$.

General Learning

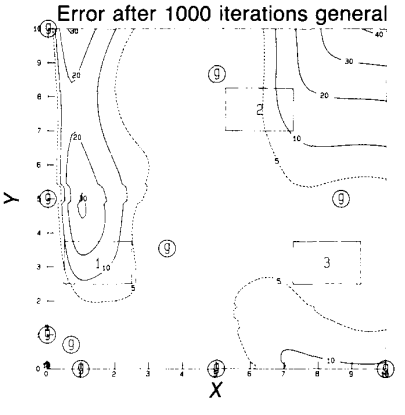
The general learning proceeded under the assumption that the region of specialization had unknown (r, θ) values, except that the input magnitude r is between 0 and 10 and

the input angle θ is between 0 and 90 deg so that the input values lie in a circular wedge in the first quadrant. The training samples were chosen as a 10-point grid spanning the known input space. Every training sample was presented once in each iteration of generalized learning. We used an acceleration constant of 0.1.

The diagrams in Fig. 6 are contour plots of $\epsilon^2 = \|\bar{u} - \bar{t}\|^2$ plotted as a function of the inputs to the network x and y. The points marked by a circled "g" are the training samples used for generalized training, whereas the three rectangular regions were selected for specialized learning. Figure 6(a) shows the error contour before training with the weights chosen randomly. Figure 6(b) is the error contour after 1000 iterations of general learning. The error has been suppressed at and around the training points. Increasing the number of iterations reduces the error at the training points, but, in general, we observe regions in which the error increases as the iterations increase. Since, with this training method, we cannot place the training samples in the regions of interest, we cannot guarantee what the error will be in these regions.



(a)



Region and points of specialized learning

Points of general learning

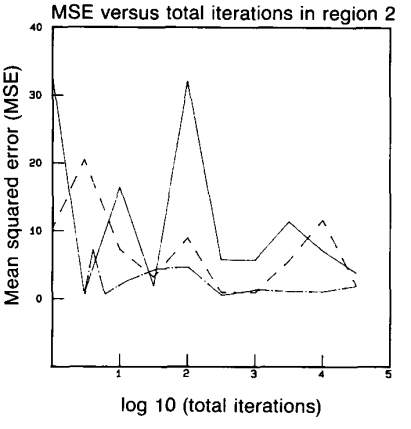
(b)

Fig. 6. Squared total error maps during general learning.

Specialized Learning

For each of the three specialization regions, we chose nine points spanning the domain of specialization. We then trained the network to specialize in each region alone, starting with the weight matrices after a varying number of iterations of general learning. We used an acceleration constant of 0.01.

Figure 7 is a plot of $\epsilon^2 = \|\bar{d} - \bar{y}\|^2$, averaged over the second region of specialization, versus the number of total iterations. Three separate curves are superimposed on the same diagram. The solid curve is for general learning only, the dashed curve specialized only, and the dash-dot curve is 10 iterations of general followed by specialized training. In this example, there seems to be a definite advantage to the hybrid learning method. However, in our simulations, with this simple plant, we were unable to determine conditions under which we could consistently observe improvement by performing general learning prior to specialized learning. An important topic for future research is to find ways to determine the properties of the plants that will allow us to specify the appropriate sequence of general and specialized learning.



Iterations

———General (G)

– – –Specialized (S)

——·——10 G + S

Fig. 7. Squared error versus total iterations.

Conclusions

We have begun to explore the idea of using neural networks for controlling physical systems. Specifically, we proposed three different methods for using error back propagation to train a feedforward neural network controller to act as the inverse of the plant. The general learning method attempts to produce the inverse of the plant over the entire state space, but it can be very difficult to use it alone to provide adequate performance in a practical control application. In order to circumvent this problem, we introduced the method of error propagation backwards through the plant, which allows us to train the network exactly on the operational range of the plant. Finally, we proposed using generalized training in conjunction with specialized training to gain their advantages and to avoid their potential disadvantages.

References

[1] J. Denker, ed., *AIP Conf. Proc. Neural Networks for Computing*, American Institute of Physics, New York, 1986.

[2] J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat. Acad. Sci. U.S.*, vol. 79, pp. 2554–2558, 1982.

[3] T. Kohonen, *Self-Organization and Associative Memory*, New York: Springer-Verlag, 1984.

[4] D. Rumelhart and J. McClelland, *Parallel Distributed Processing*, Cambridge, MA: MIT Press, 1986.

[5] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Synthesis of Nonlinear Control Surfaces by a Layered Associative Search Network," *Biol. Cybern.*, vol. 43, pp. 175–185, 1982.

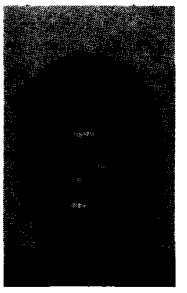
[6] D. Bullock and S. Grossberg, "A Neural Network Architecture for Automatic Trajectory Formation and Coordination of Multiple Effectors During Variable-Speed Arm Movements," presented at 1987 IEEE International Conference on Neural Networks.

[7] S. Grossberg and M. Kuperstein, *Neural Dynamics of Adaptive Sensory-Motor Control: Ballistic Eye Movements*, Amsterdam: Elsevier/North Holland, 1986.

[8] A. Guez, J. Eilbert, and M. Kam, "Neuromorphic Architecture for Fast Adaptive Robot Control," presented at 1987 IEEE International Conference on Neural Networks.
[9] 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, Feb. 1987.
[10] A. J. Pellionisz, "Tensor Network Theory and Its Applications in Computer Modeling of the Metaorganization of Sensorimotor Hierarchies of Gaze," in *AIP Conf. Proc. Neural Networks for Computing*, American Institute of Physics, New York, pp. 339–344, 1986.
[11] A. Pellionisz, "Sensorimotor Operations: A Ground for the Co-Evolution of Brain Theory with Neurobotics and Neurocomputers," presented at 1987 IEEE International Conference on Neural Networks.
[12] D. Psaltis, A. Sideris, and A. Yamamura, "Neural Controllers," presented at 1987 IEEE International Conference on Neural Networks.
[13] D. E. Rumelhart, presentation at the California Institute of Technology, Spring 1987.
[14] A. Sideris, A. Yamamura, and D. Psaltis, "Dynamical Neural Networks and Their Application to Robot Control," presented at the IEEE Conference on Neural Information Processing Systems Natural and Synthetic, 1987.
[15] K. Tsutsumi and H. Matsumoto, "Neural Computation and Learning Strategy for Manipulator Position Control," presented at 1987 IEEE International Conference on Neural Networks.
[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel Distributed Processing*, Chapter 8, "Learning Internal Representations by Error Propagation," vol. 1, Cambridge, MA: MIT Press, 1986.

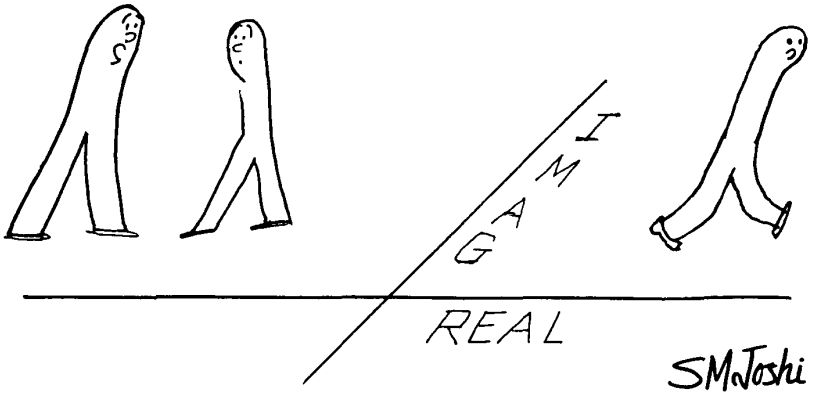
Demetri Psaltis received the B.Sc. degree in electrical engineering and economics in 1974 and the M.Sc. and Ph.D. degrees in electrical engineering in 1975 and 1977, respectively, all from Carnegie-Mellon University, Pittsburgh, Pennsylvania. After the completion of the Ph.D., he remained at Carnegie-Mellon for a period of three years as a Research Associate and later as a Visiting Assistant Professor. In 1980, he joined the faculty of the Electrical Engineering Department at the California Institute of Technology, Pasadena, where he is now Associate Professor and consultant to industry. His research interests are in the areas of optical information processing, acousto-optics, image processing, pattern recognition, neural network models of computation, and optical devices. He has over 130 technical publications in these areas. Dr. Psaltis is a Fellow of the Optical Society of America and Vice President of the International Neural Networks Society.

Athanasios Sideris received the diploma in electrical engineering from the National Technical University of Athens, Athens, Greece, in 1980, the M.S. degree in mathematics in 1986, and the M.S. and Ph.D. degrees in electrical engineering in 1981 and 1985, respectively, from the University of Southern California. From September 1980 until May 1986, he was a Teaching/Research Assistant and, from September 1985 until December 1986, a Lecturer with the Electrical Engineering Department at the University of Southern California. Since June 1986, he has been at the Department of Electrical Engineering at the California Institute of Technology, where he is an Assistant Professor. His general area of interest is control and systems theory; his current research interests include robust control theory, system optimization, and, in particular, L_∞ optimal control, nonlinear control systems, and applications of neural networks in control.



Alan A. Yamamura was born in Hampton, Virginia, in 1965. He received the S.B. degree in electrical engineering, the S.M. degree in electrical engineering and computer science, and the S.B. degree in physics from the Massachusetts Institute of Technology, Cambridge, Massachusetts, in 1986. He is currently a graduate student in the Department of Electrical Engineering at the California Institute of Technology, Pasadena, California.

Out of Control



"His defection is certain to have a destabilizing effect."