Fixed-Weight Controller for Multiple Systems

L. A. Feldkamp and G. V. Puskorius
Ford Research Laboratory, P.O. Box 2053, MD 1170 SRL
Dearborn, Michigan 48121–2053
lfeldkam@ford.com, gpuskori@ford.com

Abstract

We demonstrate here a perhaps unexpected result: the ability of a single fixed-weight time-lagged recurrent network, properly trained, to act as a stabilizing controller for multiple (here 3) distinct and unrelated systems, without explicit knowledge of system identity. This capability, which may be regarded as a challenge to the usual understanding of what constitutes an adaptive system, seemed plausible to us on the basis of our earlier results on both multiple time-series prediction and robust controller training. We describe our training method, which has been enhanced toward enforcing stability of the closed-loop system and dealing with process noise, and provide some results.

1 Introduction

In various papers we have reported the results of training time-lagged recurrent networks for various tasks and have noted what we regard as rather remarkable properties of such network architectures. In one application, a recurrent network was shown to solve a very difficult diagnostic problem (automotive engine misfire) [1, 2] that requires handling a wide range of system conditions. In a more abstract vein, a single recurrent network was trained to mimic an adaptive system by making accurate one-step predictions for 13 different time series [3], resuming good predictions following a switch between series, after a short period of accommodation, and performing well for new series whose parameters were modest variations of the parameters of the original series. In the context of control, we have shown [4, 5] that a recurrent network can be trained to act as a controller for difficult nonlinear systems ('plants' in the jargon of control theory), with robustness to plant variations made an explicit objective by training simultaneously for plants with a range of parameters or model structure.

The present work was motivated by a speculation that arose naturally from the experience mentioned above: that a single fixed controller could be trained to be effective for two or more different and unrelated plants, without the plant identity being provided by an external signal or being deduced by an explicit classifier. Note that we are not referring merely to a single *control structure*, whose parameters are then determined separately for each plant. Rather, we have in mind that the controller itself have a fixed set of parameters. We expect the controller to be effective only for those plants for which it has been trained, not for arbitrarily chosen plants. On the other hand, we expect the controller's performance to degrade gracefully for modest variations in the parameters of the chosen plants. Finally, due to the conflicting demands imposed by handling multiple plants, we recognize that the controller's performance, especially for transients, may not be as effective on a given plant as a controller specialized to that plant.

2 Problem Statement

The primary problem we consider here is that of finding a stabilizing controller for three plants, presented by Suykens et al. [6]. Each plant has the same structure, but their respective parameters result in quite different behavior. Each plant is nonlinear, third-order, and described by a difference equation of the following form:

$$x_i(k+1) = \sum_{j=1}^{3} W_{i,j} \tanh\Big(\sum_{l=1}^{3} V_{j,l} x_l(k) + 2(u_j(k) + b_j)\Big),$$

for $i = 1, 2, 3$. The three control inputs $u_j(k)$, augmented by fixed biases b_j, enter the system nonlinearly. We take the three states to be accessible as system outputs. The first system, which we label I, has multiple equilibria, i.e., the state variables x_i of the unforced system (control plus bias inputs set to zero) evolve asymptotically to fixed values which depend on their initial values. Under the same conditions, system II displays quasi-periodic behavior, while system III is chaotic. Numerical values for the matrices $\mathbf{W}$ and $\mathbf{V}$ and the vectors $\mathbf{b}$ are given in the Appendix. Suykens et al. showed that a (separate) stabilizing controller for each of these systems can be trained for the case of $\mathbf{b} = 0$.

0-7803-4122-8/97 $10.00©1997 IEEE

Our goal here is to train a single controller that can bring the state variables of all three systems asymptotically to zero from arbitrary but bounded initial values. As an additional complication, we choose non-zero values for the vectors $\mathbf{b}$ so that the asymptotic values of stabilizing control signals will be different in the three cases (for the nominal system definitions with $\mathbf{b} = 0$, stabilizing control of all three systems results in asymptotic control values of $\mathbf{u} = 0$). As mentioned above, the controller must deduce the identity of the system from the system outputs in combination with its own state. We follow Suykens et al. in paying attention to the *global asymptotic stability* of each closed-loop system, but we employ a quite different approach.

3 Procedure

To train the controller network we follow generally the multi-stream controller training procedure described in [4, 5]. This includes a generalization of the dynamic backpropagation method of Narendra and Parthasarathy [7] to the training of recurrent networks and involves controller weight updates with a node-decoupled extended Kalman filter method augmented with the multi-stream technique. Briefly stated, in multi-stream training each weight update attempts to satisfy simultaneously the potentially conflicting demands imposed by two or more streams of data. In the present case, each plant to be controlled is used to form an independent stream of data, so that each update is an attempt to reduce error on each plant by making the same change of weights on all copies of the controller.

A minor change from the procedure of [4] and [5] is that ordered derivatives are calculated using truncated backpropagation through time (BPTT) [8], as detailed in [9], rather than the more time-consuming forward method. Another difference is that here each stream has its own identification network, specific to the model, rather than an average identification network, as is adequate when the models differ only modestly.

We wish to be clear that the setting for the experiments described is controller training from a model rather than our usual training from input-output data from a physical system or a simulation thereof. Accordingly, we exploit the fact that the model structure is essentially that of a recurrent network, which can be used directly as both plant and identification network. We also make use of the ability, generally not possible with physical systems, to set plant states arbitrarily and to inject noise.

We performed several exercises of multi-stream controller training, two of which are reported here. In the first of these, we took $\mathbf{b} = 0$. In the second, $\mathbf{b}$ was nonzero (see Appendix) and we trained for 6 plants, half of which had state variables corrupted with process noise (i.e., $x_l(k)$ replaced by $x_l(k) + \epsilon$, where ϵ is taken from a Gaussian distribution of standard deviation 0.1). The first case was carried out using 3 streams, each of which has its respective model, corresponding identification network, and copy of the controller network. Similarly, the second case involved 6 streams, split between the three models, with and without noise. The truncation depth for BPTT was chosen to be 20, and a gradient discount factor of 0.95 was employed. The controller architecture was chosen as 3-8R-6R-3, i.e., 3 inputs, two fully recurrent hidden layers of 8 and 6 nodes, respectively, and 3 output nodes. (The use of a more complex architecture than that required to handle any of the models separately reflects the controller's dual tasks of determining the appropriate context and then evoking appropriate control values.) All node activation functions were bipolar sigmoids, equivalent to $\tanh(x/2)$. The identification network, equivalent to the model given above, is externally recurrent and of the form 6-3-3L. The input vector consists of the 3 controls and the 3 time-lagged recurrent connections from network outputs to network inputs. The weights from externally recurrent inputs to hidden nodes are elements of the matrix $2 \times \mathbf{V}$, while those between the hidden nodes and the linear output nodes are taken from $\mathbf{W}$. The hidden layer's bias weights are given by the vector $4 \times \mathbf{b}$ and each hidden node is connected to its respective controller input by a weight of value 4.

Training was carried out for 100,000 total weight updates for the first case and for 350,000 total weight updates for the second. Training was organized into trajectories of length 1000 for each stream. At the start of each trajectory, the state variables x_i of each plant were initialized by choosing uniformly distributed random values in the range [-1,1] for the outputs of the corresponding identification network's hidden layer nodes and propagating these random values to the output layer; these initialized state values are then used as controller inputs as well as externally recurrent inputs for the plant. To encourage stability of the closed-loop system, by forcing the controller to recover from arbitrary states, we also initialize the recurrent nodes of the controller to values in the range [-1,1] at the start of each trajectory. At randomly chosen intervals (in the range 30 to 50 time steps) during each trajectory the plant (but not the controller) is re-initialized with random values; this range was increased to 100 to 200

time steps for the last 100,000 weight updates of the second training case to enforce closed-loop system stability for long time spans. As is our usual practice for Kalman training, we begin with a relatively small effective learning rate and a relatively large training process noise parameter (not to be confused with plant or model process noise). As training proceeds, the learning rate is increased toward its maximum value and this process noise parameter is decreased.

4 Results

Results for the first case are present in Figure 1 and those for the second case in Figures 2 and 3. Each figure is organized as follows. The seven panels on the left-hand side show evolution of system states, while the right-hand side panels provide the corresponding controls. Time proceeds from left to right and from bottom to top. Each panel is 100 time units wide. A switch from one system to another occurs in every panel at the $25th$ time step without changing the system or controller state. The controlled system is perturbed at the $75th$ time step of each panel by instantiating random plant (but not controller) states in the same fashion as during training.

The excellent stabilization for all models after plant switches and plant re-initializations is evident, though the control sequences that follow plant re-initializations are somewhat more complicated than those of controllers trained separately for each plant. For the zero bias plants (Figure 1), the control is barely affected by plant switches, since each plant has the same steady state stabilizing control. When the biases are nonzero (Figures 2 and 3), a switch to another plant requires the controller output to change, since each plant definition requires different stabilizing steady-state controls. This seems to exclude the possibility of handling this problem with a pure feedforward network whose inputs are based on a finite number of previous plant outputs. It is interesting that the steady-state controller outputs are *not* merely negatives of the biases b_l, though this was the case in a similar training exercise carried out with bias but without noise.

The performance observed in Figure 1, both steady state and transient, is better than that in Figures 2 and 3. This is not particularly surprising, since the controller for Figure 1 was optimized for the case with neither bias nor noise, while the controller for Figures 2 and 3 had to deal with bias and with both noisy and noise-free plants. On the other hand, if the controller of Figure 1 is applied to a noisy plant, very poor performance results, while the controller trained in the presence of greater plant uncertainty handles the noisy plants almost as well as it does the noise-free plants.

Both controllers were subjected to extensive testing for global stability. For each of the three systems, we recorded the maximum absolute values of plant state variables in 25-step intervals beginning 100 time steps after each of one million random re-initializations of both plant and controller. For the zero bias, noise-free plants the maximum value was 0.0015 for plant III. For the noise-free biased plants the maximum value was 0.1294 and occurred for plant I. In the case of the noisy plants, we observe very occasional (mean interval of 12,500 time steps for the chaotic system, and substantially longer for systems I and II) episodes in which the noise interacts with the controller to produce large absolute values of the plant state variables (> 0.75), which then return quickly to small values. Each case was also tested for long-term instability by carrying out trajectories of one million time steps for 100 random re-initializations of both plant and controller states for each plant; no evidence of instability was found for the noise-free plants and no evidence of undesirable behavior beyond that just noted was observed for the noisy plants.

5 Discussion and Conclusions

The training experiments carried out here demonstrate that a single recurrent network can be trained to perform as an effective controller for multiple distinct plants. The controller exhibits a characteristic, context recognition, normally ascribed to adaptive systems. At this point it is probably best not to attempt to impose a label (e.g., robust or adaptive) but rather to regard the capability demonstrated here as an experimental result worthy of serious analytical study.

Our technique for enforcing closed-loop stability worked remarkably well, even though the space of possible initial system states was only sparsely sampled during training. When we executed the same training process without randomly initializing the controller at the start of each trajectory, the controller handled random plant state re-initializations without difficulty, but we were able to find combinations of controller and plant initial states for which the chaotic system was not stabilized.

The testing for stability performed here, though extensive, does not *prove* that the closed-loop system is stable. At present, this may be the best that can be done for these complex systems. (Even Suykens et al. [6] resorted to testing when the elegant analytical constraints they used to encourage global asymptotic stability were insufficiently satisfied to provide a stability guarantee.)

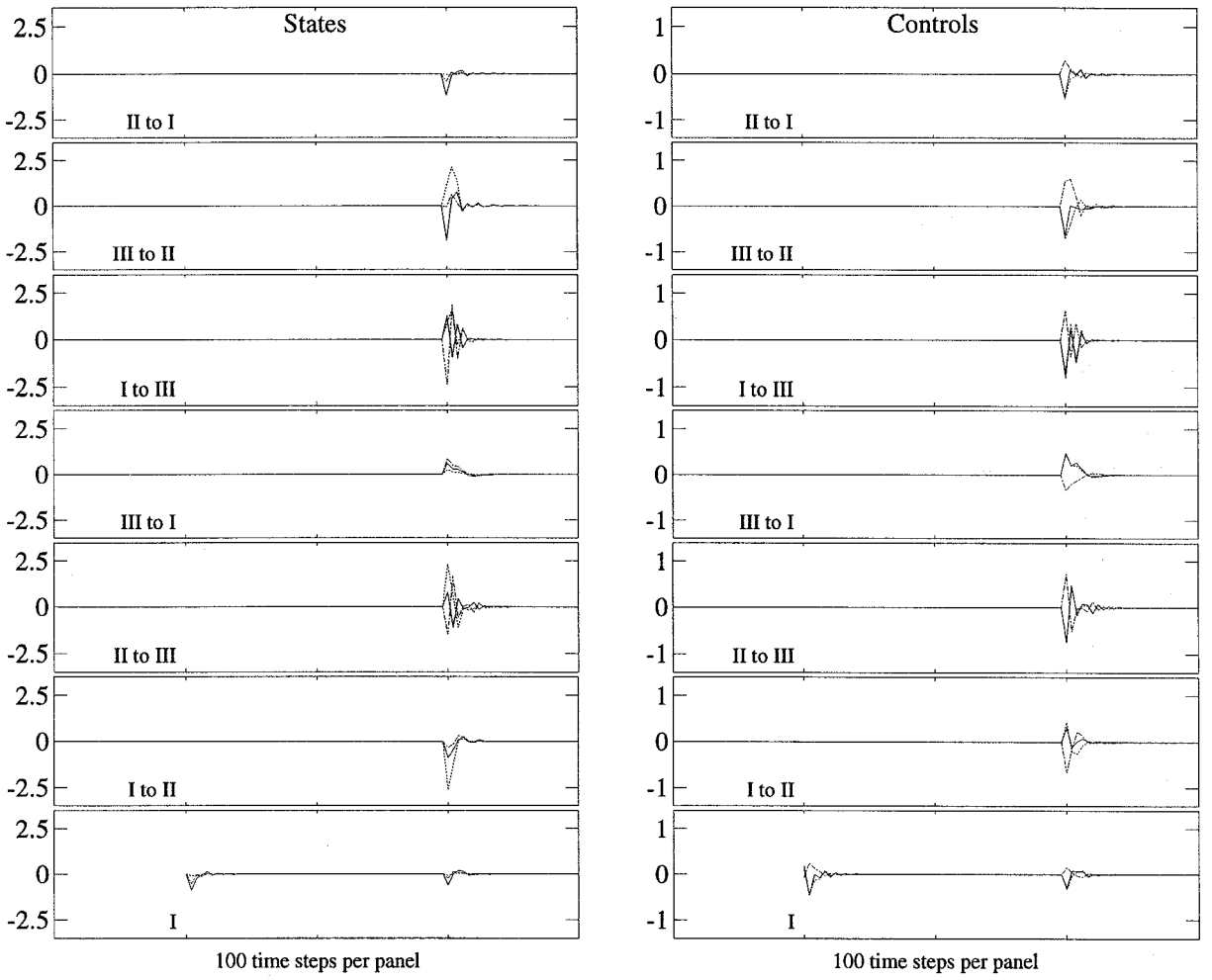

Figure 1: Testing trajectories (left) and controls (right) for each of the three zero bias plant models with the same controller. Plant switches occur at the $25th$ time step in each panel without reset of system state. Plant state initialization occurs at the $75th$ time step.

The multi-stream procedure also enabled us to train a controller that performs well with or without process noise. (We expect that even the occasional brief loss of control reported above could be reduced or eliminated by further procedural refinement.) In contrast, a controller trained only on noise-free plants performs very poorly when these plants are subjected to process noise; measurement noise degrades the performance much less. Interestingly, the network trained to control systems both with and without process noise was also significantly more robust to plant parameter variations (where each plant parameter α was modified by $\alpha(1+\epsilon)$ with ϵ taken from a Gaussian distribution of standard deviation 0.1) than was the controller trained without process noise. These results are just one illustration of the potential of the multi-stream procedure for tailoring neural network performance.

References

[1] L. A. Feldkamp, G. V. Puskorius, K. A. Marko, J. V. James, T. M. Feldkamp, and G. Jesion, "Unravelling dynamics with recurrent networks: Application to engine diagnostics," in *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, 1996, pp. 59–64.

[2] G. V. Puskorius and L. A. Feldkamp, "Signal processing by dynamic neural networks with application to automotive misfire detection," in *Proceedings of the 1996 World Congress on Neural Neural Networks,* San Diego, 1996, pp. 585–590.

[3] L. A. Feldkamp, G. V. Puskorius, and P. C. Moore, "Adaptation from fixed weight dynamic networks," in *Proceedings of the IEEE Interna-*

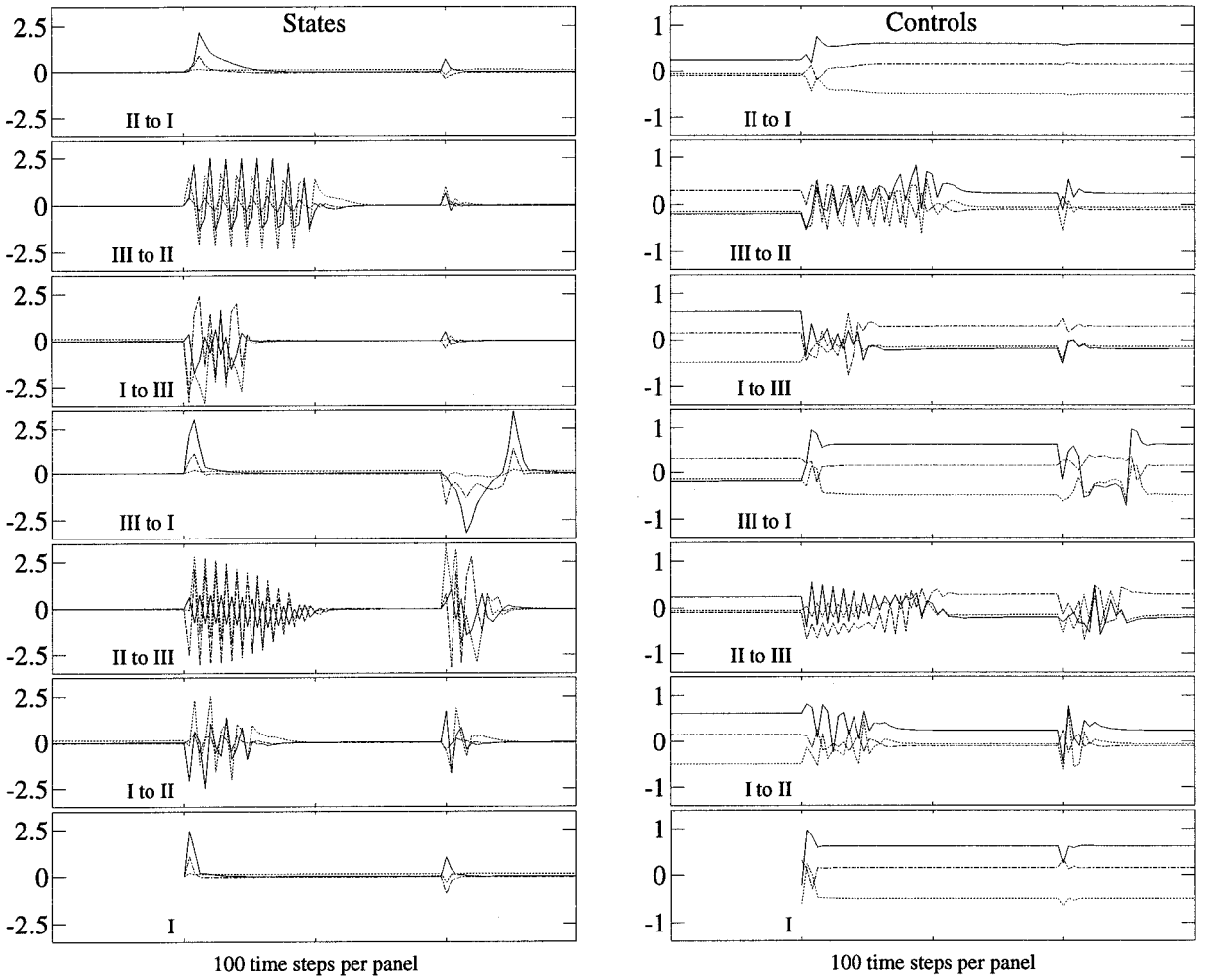

Figure 2: Testing trajectories (left) and controls (right) for each of the three noise-free, nonzero bias plant models with the same controller. Plant switches occur at the $25th$ time step in each panel without reset of system state. Plant state initialization occurs at the $75th$ time step.

tional Conference on Neural Networks, Washington, D.C., 1996, pp. 155–160.

[4] L. A. Feldkamp and G. V. Puskorius, "Training controllers for robustness: Multi-stream DEKF," in *Proceedings of the IEEE International Conference on Neural Networks,* Orlando, 1994, pp. 2377–2382.

[5] L. A. Feldkamp and G. V. Puskorius, "Training of robust neural controllers," in *Proceedings of the 33rd IEEE International Conference on Decision and Control,* Orlando, 1994, pp. 2754–2760.

[6] J. A. K. Suykens, J. P. L. Vandewalle, and B. L. R. De Moor, "Artificial neural networks for modelling and control of non-linear systems," Kluwer Academic Publishers, Boston, 1995. (The models used here are generalizations of models defined on pp. 157–161.)

[7] K. S. Narendra and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 252–262, 1991.

[8] P. J. Werbos. "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[9] G. V. Puskorius, L. A. Feldkamp, and L. I. Davis, Jr., "Dynamic neural network methods applied to on-vehicle idle speed control," *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1407–1420, 1996.

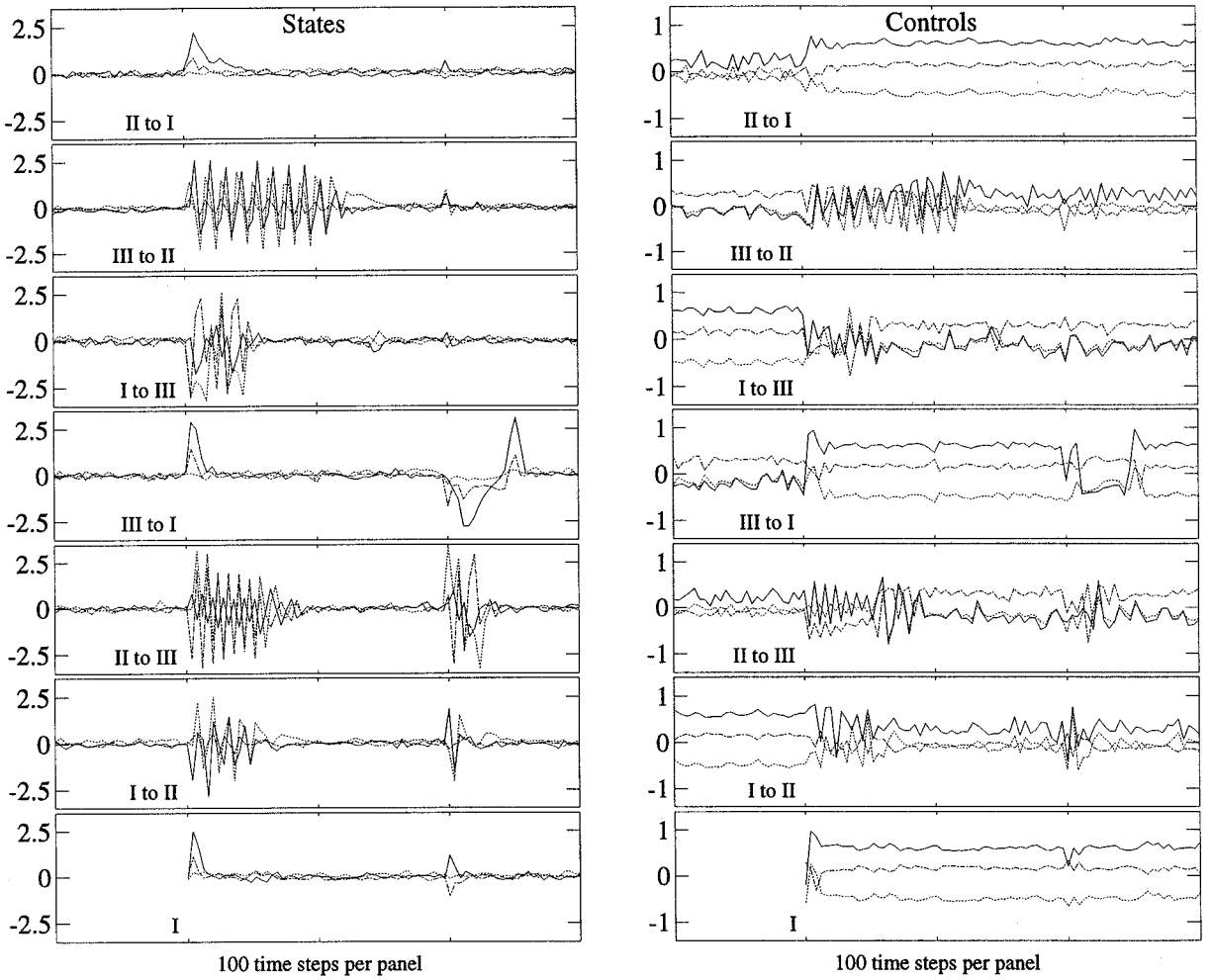

Figure 3: Testing trajectories (left) and controls (right) for each of the three noisy, nonzero bias plant models with the same controller as in Figure 2. Plant switches occur at the $25th$ time step in each panel without reset of system state. Plant state initialization occurs at the $75th$ time step.

Appendix

System I: Multiple Equilibria

$$\mathbf{W} = \begin{bmatrix} -1.6663 & -0.7588 & 1.1636 \\ -0.0571 & -0.3085 & -0.1793 \\ -0.8565 & -1.0656 & -0.5651 \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} -0.0542 & -0.4616 & -1.1189 \\ -1.1169 & 0.2699 & -0.6723 \\ 0.8167 & -0.1313 & 0.0912 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -0.1000 \\ -0.2000 \\ 0.1500 \end{bmatrix}$$

System II: Quasi-Periodic Behavior

$$\mathbf{W} = \begin{bmatrix} -1.8020 & 0.4875 & -1.2491 \\ -0.8841 & 0.4576 & 1.4070 \\ -0.2647 & -0.2385 & 0.3027 \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} 1.2887 & -0.2148 & -1.2468 \\ -0.2501 & 0.5668 & 0.2859 \\ -0.7716 & 0.0636 & -0.8706 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -0.2500 \\ 0.0500 \\ 0.1000 \end{bmatrix}$$

System III: Chaos

$$\mathbf{W} = \begin{bmatrix} 0.9690 & 0.6967 & 0.2985 \\ -0.7473 & 3.2069 & 0.2840 \\ -2.7960 & 0.5360 & 0.9597 \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} 2.0876 & 0.0173 & 1.1578 \\ 1.5247 & 0.2463 & 0.1619 \\ -0.1953 & -0.8545 & 1.5571 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0.2000 \\ 0.1500 \\ -0.3000 \end{bmatrix}$$