Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks

Gintaras V. Puskorius and Lee A. Feldkamp

Abstract—**Although the potential of the powerful mapping and representational capabilities of recurrent network architectures is generally recognized by the neural network research community, recurrent neural networks have not been widely used for the control of nonlinear dynamical systems, possibly due to the relative ineffectiveness of simple gradient descent training algorithms. Recent developments in the use of parameter-based extended Kalman filter algorithms for training recurrent networks may provide a mechanism by which these architectures will prove to be of practical value. This paper presents a decoupled extended Kalman filter (DEKF) algorithm for training of recurrent networks with special emphasis on application to control problems. We demonstrate in simulation the application of the DEKF algorithm to a series of example control problems ranging from the well-known cart-pole and bioreactor benchmark problems to an automotive subsystem, engine idle speed control. These simulations suggest that recurrent controller networks trained by Kalman filter methods can combine the traditional features of state-space controllers and observers in a homogeneous architecture for nonlinear dynamical systems, while simultaneously exhibiting less sensitivity than do purely feedforward controller networks to changes in plant parameters and measurement noise.**

I. Introduction

THE APPLICATION of artificial neural networks to problems in the control of nonlinear dynamical systems has been dominated largely by static network architectures trained by gradient descent. These neural control architectures can roughly be categorized as belonging to one of two classes. The most common architecture considered is the multilayer perceptron architecture (or simply MLP network), consisting of feedforward layers of nodes, such that the outputs of nodes in a layer form the inputs to nodes in succeeding layers. These networks are considered to encode *global* representations, since a change in a single weight of the network will generally result in a change in the network's outputs for all possible input signals. Recently, considerable attention has been focussed on radial basis function (RBF) networks for control. In contrast to MLP networks, RBF network architectures encode *local* representations such that a change in a single weight of the network will result in a change in its outputs for a limited range of input signals. In either case, these two network architectures lack internal or hidden state, so that the processing of input patterns does not depend upon the order of presentation during training or recall. Thus, the representation and processing of temporal information is not an intrinsic capability of MLP or RBF networks.

Manuscript revised September 14, 1993.

The authors are with Ford Motor Company's Research Laboratory, Dearborn, MI 48121.

IEEE Log Number 9214803.

A common approach to encoding temporal information for discrete time neural network processing is to form tapped delay line representations of applied inputs and measured outputs. However, tapped delay line representations are limited, since they can only encode a finite number of previously measured outputs or imposed inputs. We also note that a tapped delay line representation of inputs for RBF networks tends to require prohibitively large amounts of memory (growing exponentially in the dimensionality of the input space), thereby hindering these networks' use for all but relatively low-order dynamical systems. Hence, for the remainder of the paper, we assume network architectures that encode global representations.

An alternative to tapped delay line representations of temporal or sequential information is the use of recurrent network architectures, where feedback connections exist between nodes of the same layer or to nodes of preceding layers. In a time-lagged recurrent network, the output of a recurrent node is delayed by a unit time step before it is used as an input signal for nodes within the same or preceding layers. The temporal representational capabilities of recurrent networks have been shown to be considerably greater than those of purely static networks. Recurrent networks can be used to represent a number of conventional architectures used in control system design. For example, a tapped delay line can be constructed with linear recurrent nodes sparsely interconnected [1]. Derivative and integral terms are also conveniently represented with sparsely interconnected recurrent linear nodes [2], thereby providing a means to specify PID controllers. Recurrent networks with nonlinear output functions (e.g., sigmoidal nodes) have the capability to represent and encode *strongly hidden state* [3] in which a network's output is a function of an arbitrary number of previously imposed input signals. On the other hand, strongly hidden state cannot be represented by a tapped delay line of finite length.

Exploitation of the mapping and representational capabilities of recurrent networks has been hindered by two related difficulties in training. First, the calculation of dynamic derivatives (also known as total partial derivatives, ordered derivatives or recurrent derivatives) of a recurrent network's outputs with respect to its weights by the real-time recurrent learning algorithm (RTRL) [4] is computationally expensive, since these derivatives cannot be computed by the same backpropagation mechanism that is employed in training of MLP networks. For example, computing the derivatives for a time-lagged single-layer network of N completely interconnected processing elements requires $O(N^4)$ operations. A possible alternative is to use approximate methods that are

computationally less expensive (e.g., truncated backpropagation through time [5], [6]). The second practical difficulty is that training of recurrent networks with pure gradient descent methods tends not only to be slow but also ineffective at finding a good solution. This is attributable to correlations between node outputs at successive time steps. Highly correlated signals may result in ill-conditioned gradient descent training problems which require very low learning rates.

The second difficulty described above is partially addressed by using second-order training algorithms that process and use information about the shape of the training problem's underlying error surface. Among these algorithms are those based upon linearized recursive least squares or extended Kalman filtering [1], [7], [8], [9], [10]. Simulation results for problems in function approximation and pattern classification indicate that training algorithms based upon the extended Kalman filter (EKF) require significantly fewer presentations of training data than do the pure gradient descent algorithms. Using the same derivative information as do the gradient descent algorithms, the EKF algorithms with appropriate simplifications have only modest additional computational requirements for recurrent networks. Unlike some other higher-order methods, these EKF-based algorithms for recurrent networks do not require batch processing, making them more suitable for on-line use.

Narendra and Parthasarathy [11], [12] established a rigorous framework for the training of identification and controller networks for nonlinear dynamical systems by a gradient descent procedure called *dynamic backpropagation* (DBP). In the case of training controller networks by DBP, Narendra and Parthasarathy explicitly noted that the controller and plant form a closed-loop feedback system, and that the computation of derivatives used during training must evolve recursively as do the plant outputs. Because of this external recurrence, the training of controller networks may be difficult for some of the same reasons as is training of recurrent networks. Although Narendra and Parthasarathy treat the case of gradient computations for both static feedforward networks and dynamic recurrent networks, they did not present simulation results for nonlinear recurrent networks.

The work described in this paper explores the combination of three fundamental and interrelated claims. First, because of their representational and mapping capabilities, recurrent networks are more appropriate architectures for control than are purely feedforward networks. Second, the dynamic backpropagation framework developed by Narendra and Parthasarathy [11], [12] for computing gradients can be conveniently extended to include nonlinear recurrent networks as distinct components in a feedback control architecture. Third, second-order gradient training algorithms, such as EKF methods, are effective for training recurrent network controllers using gradients as computed by dynamic backpropagation.

Although we do not address in this paper the question of stability, either for adjustment of a controller's parameters or for use of a recurrent network controller as part of a closed-loop feedback system, we acknowledge the importance of rigorous analytical treatments of these problems. Parallel theoretical efforts are needed for determining stable adaptive laws for recurrent discrete-time controller networks, as well as for analyzing the stability of closed-loop feedback systems employing recurrent neural networks as controllers. In lieu of analytical stability methods for these closed-loop nonlinear dynamical systems, we resort to careful application of recurrent network training methods in both simulation and practice by drawing on whatever knowledge we may have concerning the system of interest, and to rigorous testing of the trained controllers.

The remainder of this paper is organized as follows. The general feedback control framework and the role of recurrent neural networks for control are described in Section II. A discussion of gradient computations for recurrent controller networks is provided in Section III. Section IV provides a detailed discussion of a parameter-based decoupled EKF (or DEKF) algorithm for updating the weight parameters of an arbitrary, feedforward or recurrent, controller network. We also include a qualitative comparison of the EKF procedure used in this work with one that estimates both weight parameters and network states (i.e., the outputs of recurrent nodes) as recently described by Williams [13], [14]. Finally, in Section V we demonstrate the application of the DEKF training algorithm for training recurrent controller networks and show representative results for three different problems.

II. Recurrent Network Architectures for Control

A. Feedback Control Architecture

For the purposes of this paper, we do not assume any special functional form for the input/output model of the plant. Rather, we address a more general class of control problems, precluding the use of constructive methods in which a plant's nonlinearities can be simply cancelled or subtracted by the use of an identified plant model. We are interested in multi-input/multi-output systems in which the plant's nonlinear model is assumed to be described by a set of differential or difference equations where the plant's states are coupled nonlinearly with the imposed controls. Additional complications may include unmeasured and random disturbances, a possibly non-unique plant inverse, and plant states that are not directly measured.

The general control structure is depicted in Fig. 1. The first form of recurrent network architecture considered in this paper is encountered in this feedback control architecture, since the controller network and the plant form a closed feedback loop. We refer to this neural control architecture as an *externally recurrent* network architecture, since a subset of the controller network's inputs are a function, through the evolution of the plant, of the controller's outputs from previous time steps.

In operation, the plant receives as input the discrete time control signals $\mathbf{u}_c$ along with asynchronously applied unmeasured disturbance inputs $\mathbf{u}_d$. The plant evolves as a function of these imposed inputs and its internal state $\mathbf{x}_p$. The output of the plant, which is a nonlinear function of its state, is sampled at discrete time intervals and may also be corrupted by measurement noise.

The controller receives as input time-delayed feedback measurements of the output of the plant $\mathbf{y}_p(n)$ along with

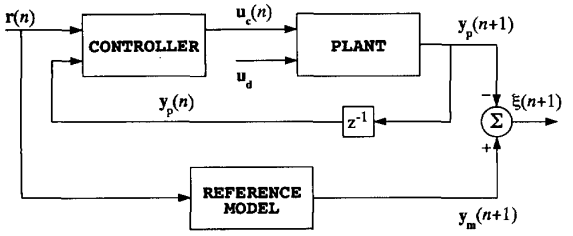


Fig. 1. Block diagram for model-reference control. The block labelled z^{-1} represents a delay of one time step.

externally specified reference signals $\mathbf{r}(n)$. In response to these inputs, and as a function of both its internal state $\mathbf{x}_c(n)$ and defined parameters $\mathbf{w}$, the controller produces a vector of signals at discrete time step n given by

$$\mathbf{u}_c(n) = \mathbf{f}_c(\mathbf{x}_c(n), \mathbf{y}_p(n), \mathbf{r}(n), \mathbf{w}), \tag{1}$$

where $\mathbf{f}_c(\cdot)$ is a function describing the controller's input/output behavior.

The desired output $\mathbf{y}_m(n+1)$ of the plant is given by the output of a stable reference model which is specified as a function of the externally imposed reference signals $\mathbf{r}(n)$ and the reference model's internal state. The goal is to train a controller network so that the squared differences (errors) between the outputs of the reference model and the plant, $\xi(n) = \mathbf{y}_m(n) - \mathbf{y}_p(n)$, are minimized over time.

The difficulty of synthesizing neural controllers by supervised training methods for this general framework lies in the fact that while the plant's desired behavior is known from the output of the reference model, the controls that lead to that behavior are not known *a priori*. Rather, these controls must be inferred indirectly. Following the work of Narendra and Parthasarathy [11], [12], a two-step procedure is employed to train controller networks by an indirect, gradient-based training scheme, as summarized below.

B. Internally Recurrent Network Architectures

We consider neural network architectures that are specified by a generalized version of the *recurrent multilayer perceptron* (RMLP) architecture described in [15]. A general RMLP architecture consists of layers of nodes arranged in a feedforward fashion. The nodes of a given layer may be connected to one another through trainable feedback connections with a unit time delay. The RMLP architecture subsumes both the conventional MLP and the single-layer, fully recurrent network. Our treatment does not consider feedback connections from the output of nodes in one layer to the inputs of nodes in preceding layers, although it can be extended to include such architectures. However, we do allow in some cases for an RMLP network's unit time-delayed outputs to be fed back as components of the network's input vector. In this case, the RMLP network would be considered to be both internally and externally recurrent.

It is convenient to think of an RMLP network as a cascaded set of subnetworks of the following form. Each subnetwork consists of layers of nodes, where the last layer of the subnetwork contains feedback connections between its nodes and where all preceding layers are restricted to have only feedforward connections. The last layer of a subnetwork can also be purely feedforward if it is the network's output layer. Then, a general RMLP network consisting of L layers, where all layers have internal feedback connections, will be considered to consist of $\mathcal{L} = L$ subnetworks, where each recurrent layer forms a distinct subnetwork. In the other extreme, a completely feedforward network architecture with L layers of nodes is composed of a single subnetwork, i.e., $\mathcal{L} = 1$. Let $\mathcal{N}_i$ refer to the number of nodes in the output layer of the ith subnetwork. Then, $\mathcal{N}_{i-1}$ is the number of inputs to the ith subnetwork, $\mathcal{N}_0$ is the number of inputs for the RMLP network, and $\mathcal{N}_{\mathcal{L}}$ is the number of nodes in the output layer of the RMLP network. The forward propagation of signals through an RMLP network is analogous to that of a conventional MLP. For recurrent layers, a recurrent node's net input is the weighted sum of feedforward signals (and bias input) from the preceding layer plus the weighted sum of the current layer's outputs from the previous time step.

Let the vector $\mathbf{y}_i(n)$ refer to the outputs of the ith subnetwork of an RMLP network. Then, the jth component of the ith subnetwork's output layer is given by

$$y_{i,j}(n) = \mathcal{F}_{i,j}(\mathbf{y}_{i-1}(n), \mathbf{y}_i(n-1), \mathbf{w}_i), \tag{2}$$

where $\mathbf{y}_{i-1}(n)$ is a vector of inputs from the preceding subnetwork, $\mathbf{y}_i(n-1)$ is a vector of the ith subnetwork's recurrent node outputs from the previous time step, $\mathbf{w}_i$ is the set of weights, both feedforward and recurrent, in the ith subnetwork, and $\mathcal{F}_{i,j}(\cdot)$ is a function that describes the mapping of the ith subnetwork's inputs to its jth output.

III. Dynamic Derivative Computations

The successful training of neural controllers by indirect, gradient-based methods often hinges critically on the proper computation of the dynamic derivatives of plant and controller outputs with respect to the controller's trainable parameters. We first provide an explicit equation for the real-time computation of dynamic derivatives for RMLP architectures in which there is no external recurrence. Next, we allow for output-to-input recurrence. Finally, the derivative computation is specialized to RMLP neural networks trained as controllers in an indirect, feedback control scheme.

A. Dynamic Derivatives for RMLP Networks

Assume that an RMLP network is being trained to model the input-output behavior of some dynamical system. In a series-parallel model [11], the actual outputs of the system being modeled, as opposed to system outputs as predicted by the RMLP network, are used as the network's inputs. The training process is said to employ *teacher forcing* [4]. In spite of the appearance of external recurrence, use of the series-parallel model allows us to ignore the external recurrent paths in derivative computations.

A general rule for computing the dynamic derivatives of an RMLP network's outputs with respect to its trainable weights can be formulated from three observations. First, the dynamic derivative of an output node in the ith subnetwork with respect

to any weight in the gth subnetwork is zero when $g > i$. Second, the dynamic derivative of an output node in the ith subnetwork with respect to a weight in the same subnetwork is obtained by a generalization of RTRL for a recurrent network with complete interconnectivity. Third, the dynamic derivative of an output node in the ith subnetwork with respect to a weight in the gth subnetwork when $g < i$ is a function of the dynamic derivatives of the outputs of both the $(i-1)$th and ith subnetworks with respect to the same weight. The dynamic derivative of the jth output node of the ith subnetwork with respect to the hth weight of the gth subnetwork for $g \leq i$ is found to be:

$$\frac{\bar{\partial} y_{i,j}(n)}{\bar{\partial} w_{g,h}} = (1-\delta_{g,i}) \sum_{k=1}^{\mathcal{N}_{i-1}} \frac{\partial y_{i,j}(n)}{\partial y_{i-1,k}(n)} \frac{\bar{\partial} y_{i-1,k}(n)}{\bar{\partial} w_{g,h}} + \gamma(n) \sum_{k=1}^{\mathcal{N}_i} \frac{\partial y_{i,j}(n)}{\partial y_{i,k}(n-1)} \frac{\bar{\partial} y_{i,k}(n-1)}{\bar{\partial} w_{g,h}} + \delta_{g,i} \frac{\partial y_{i,j}(n)}{\partial w_{g,h}}, \quad g \leq i, \tag{3}$$

where the dynamic derivative with respect to a parameter α is denoted by $\bar{\partial}/\bar{\partial}\alpha$, $\delta_{g,i}$ is the Kronecker delta (equal to one when $g = i$ and zero otherwise), and $\gamma(n)$ is a derivative discount factor imposed to decay exponentially the effects of past dynamic derivatives. The value of $\gamma(n)$ is usually set equal to or slightly less than unity.

In practice, the forward computation of derivatives through an RMLP network is carried out on a subnetwork-by-subnetwork basis in parallel with the forward propagation of signals. Consider the ith subnetwork of an RMLP network. After propagating signals from the $(i-1)$th subnetwork's outputs through the ith subnetwork, backpropagation is employed to compute the *partial* derivatives of the output nodes of the ith subnetwork with respect to the output nodes of the preceding subnetwork: $\frac{\partial y_{i,j}(n)}{\partial y_{i-1,k}(n)}$ for all j and k. Similarly, the partial derivatives of the outputs of the ith subnetwork at time step n with respect to the outputs of this subnetwork from the previous time step, $\frac{\partial y_{i,j}(n)}{\partial y_{i,k}(n-1)}$, are easily computed by application of the chain rule. Finally, backpropagation is also used to compute the partial derivatives of the ith subnetwork's outputs with respect to the subnetwork's weights. The dynamic derivatives are then computed as a function of these backpropagation-computed partial derivatives and as a function of dynamic derivatives from the preceding subnetwork and the previous time step. The sensitivity circuit of Fig. 2 illustrates the computation of dynamic derivatives of the ith subnetwork's outputs with respect to the hth weight of the gth subnetwork, where we assume that $g \leq i$. The derivative computations for a general RMLP network would be similarly illustrated by a cascaded set of these sensitivity circuits.

B. Dynamic Derivatives for Externally Recurrent Networks

The dynamic derivative formulation of (1) assumes that the network architecture has no external feedback connections from the output of a layer to the input of a preceding layer. We now relax this assumption and consider an RMLP network in which some or all of the inputs to the network are composed of a subset of the time-delayed outputs of the network. Similarly, the dynamic derivatives of network inputs with respect to the network's weights are the unit-time-delayed dynamic derivatives of network outputs with respect to these weights. Because of this external recurrence, the dynamic derivative of any subnetwork's output node with respect to *any* weight within an externally recurrent RMLP network is generally nonzero. Thus, (1) becomes

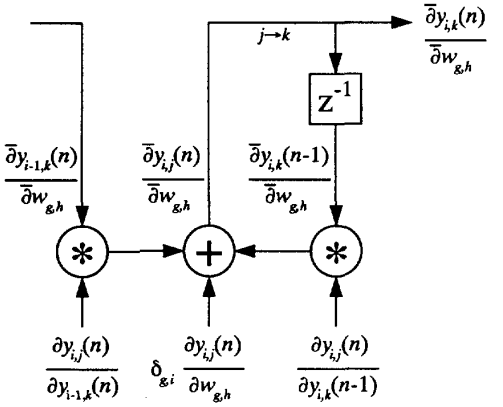


Fig. 2. Sensitivity circuit for the ith subnetwork of a recurrent multilayer perceptron (RMLP). The notation for dynamic derivatives employs the symbol $\bar{\partial}$. Ordinary partial derivatives (∂ without the bar) may be computed by static backpropagation. Summations are not shown explicitly. The time delay represented by z^{-1} can be accompanied by an optional discount factor $\gamma(n)$ [not shown here, but included in (1)]. The notation $j \to k$ indicates the change of index required to maintain consistency with the recursive equation.

$$\frac{\bar{\partial} y_{i,j}(n)}{\bar{\partial} w_{g,h}} = \sum_{k=1}^{\mathcal{N}_{i-1}} \frac{\partial y_{i,j}(n)}{\partial y_{i-1,k}(n)} \frac{\bar{\partial} y_{i-1,k}(n)}{\bar{\partial} w_{g,h}} + \gamma(n) \sum_{k=1}^{\mathcal{N}_i} \frac{\partial y_{i,j}(n)}{\partial y_{i,k}(n-1)} \frac{\bar{\partial} y_{i,k}(n-1)}{\bar{\partial} w_{g,h}} + \delta_{g,i} \frac{\partial y_{i,j}(n)}{\partial w_{g,h}}. \tag{4}$$

The computation of (2) is illustrated in the sensitivity circuit of Fig. 3, which consists of a cascaded set of circuits with an external feedback path reflecting the recurrence of time-delayed network outputs as network inputs. The derivative computations for externally recurrent RMLP networks generally require increased computational resources, since storage and computation of the dynamic derivatives of the outputs of all recurrent nodes with respect to all trainable weights of the network are required.

C. Dynamic Derivatives for Controller Networks

To use RMLP networks as feedback controllers for nonlinear dynamical systems, we follow the framework established in [11], [12]. We train controller networks by an indirect, gradient-based scheme, in two steps. The first step is to obtain a plant model to provide estimates of the differential relationships of plant outputs with respect to plant inputs, prior plant outputs, and prior internal states of the plant. This differential model is obtained by training a neural network to identify

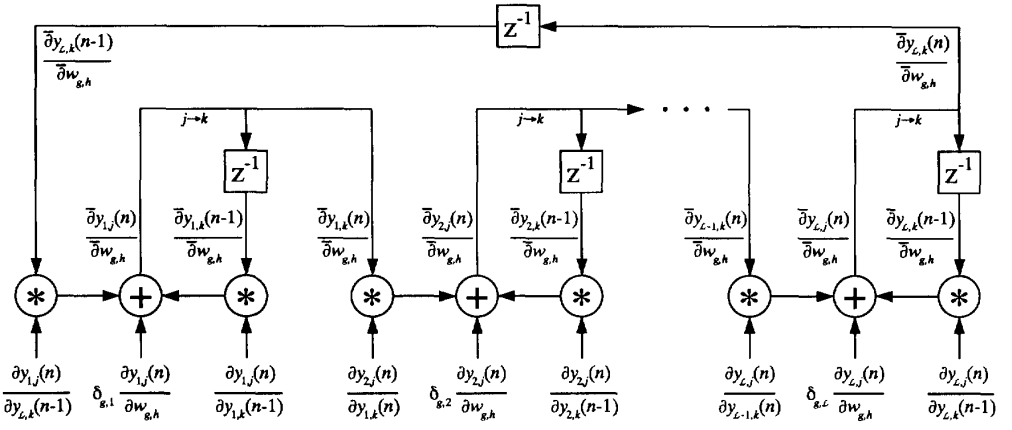


Fig. 3. Sensitivity circuit for a general RMLP with both internal and external (output to input) recurrence.

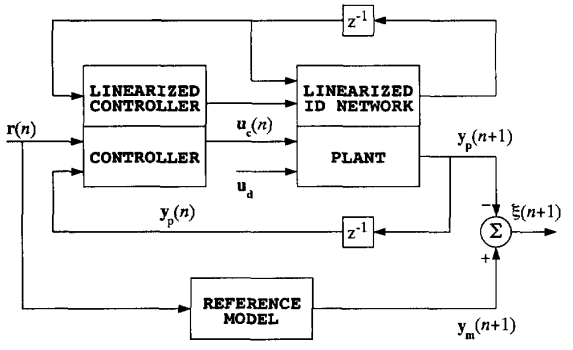


Fig. 4. Diagram of the overall flow of information in controller training with the dynamic gradient method. The blocks atop the controller and plant blocks represent sensitivity circuits for the controller and identification networks, respectively.

the plant's input/output characteristics; we call the resulting model an *identification network*. The trained identification network is then used in the second step to provide estimates of the dynamic derivatives of plant outputs with respect to the trainable controller parameters. For purposes of the derivative computations, we note that in addition to any internal recurrence within the identification and controller networks, there is also external recurrence in the form of measured time-delayed plant outputs as identification and controller network inputs. Fig. 4 shows a global representation of the flow of dynamic derivatives for the indirect controller training scheme. The blocks corresponding to the linearized controller and identification networks are composed of cascaded sensitivity circuits similar to those shown in Fig. 3.

A sensitivity circuit for computation of the dynamic derivatives of identification network outputs with respect to the controller's weight parameters is depicted in Fig. 5, where $\mathcal{L}$ and $\mathcal{M}$ are the number of subnetworks within the identification and controller networks, respectively. $y_{i,j}$ is the output of the jth node of the identification network's ith subnetwork, while $z_{i,j}$ is the corresponding output for the controller network. This figure consists of two connected circuits, one corresponding to the feedback of plant outputs as inputs to the controller, and a second corresponding to the feedback of plant outputs as identification network inputs.

A generalization of (2), based upon the sensitivity circuit of Fig. 5, mathematically describes the computation of the required dynamic derivatives for controller network training. For the identification network's ith subnetwork, where $i > 1$, the derivative computations of (2) are simplified to

$$\frac{\bar{\partial} y_{i,j}(n)}{\bar{\partial} w_{g,h}} = \sum_{k=1}^{\mathcal{N}_{i-1}} \frac{\partial y_{i,j}(n)}{\partial y_{i-1,k}(n)} \frac{\bar{\partial} y_{i-1,k}(n)}{\bar{\partial} w_{g,h}} + \gamma(n) \sum_{k=1}^{\mathcal{N}_i} \frac{\partial y_{i,j}(n)}{\partial y_{i,k}(n-1)} \frac{\bar{\partial} y_{i,k}(n-1)}{\bar{\partial} w_{g,h}}, \quad i > 1 \tag{5}$$

Since $w_{g,h}$ is a controller weight, the *partial* derivatives of the outputs of the identification network's subnetworks with respect to controller weights are all zero: $\frac{\partial y_{i,j}(n)}{\partial w_{g,h}} = 0$. When $i = 1$, the identification network receives feedback inputs from its own outputs or from the plant's outputs along with inputs from the controller. Then, the dynamic derivatives of the outputs of the identification network's first subnetwork with respect to controller parameter $w_{g,h}$ are given by

$$\begin{aligned}\frac{\bar{\partial} y_{1,j}(n)}{\bar{\partial} w_{g,h}} =& \gamma(n) \sum_{k=1}^{\mathcal{N}_\mathcal{L}} \frac{\partial y_{1,j}(n)}{\partial y_{L,k}(n-1)} \frac{\bar{\partial} y_{\mathcal{L},k}(n-1)}{\bar{\partial} w_{g,h}} \\ &+ \gamma(n) \sum_{k=1}^{\mathcal{N}_1} \frac{\partial y_{1,j}(n)}{\partial y_{1,k}(n-1)} \frac{\bar{\partial} y_{1,k}(n-1)}{\bar{\partial} w_{g,h}} \\ &+ \sum_{k=1}^{\mathcal{P}_\mathcal{M}} \frac{\partial y_{1,j}(n)}{\partial z_{\mathcal{M},k}(n)} \frac{\bar{\partial} z_{\mathcal{M},k}(n)}{\bar{\partial} w_{g,h}}, \end{aligned} \tag{6}$$

where $\mathcal{P}_i$ refers to the number of nodes in the output layer of the controller's ith subnetwork.

The dynamic derivatives of the jth output of the controller's ith subnetwork with respect to controller parameter $w_{g,h}$, for $i > 1$, is given by the same functional form as (2) with a simple change of variables to reflect that we are now dealing

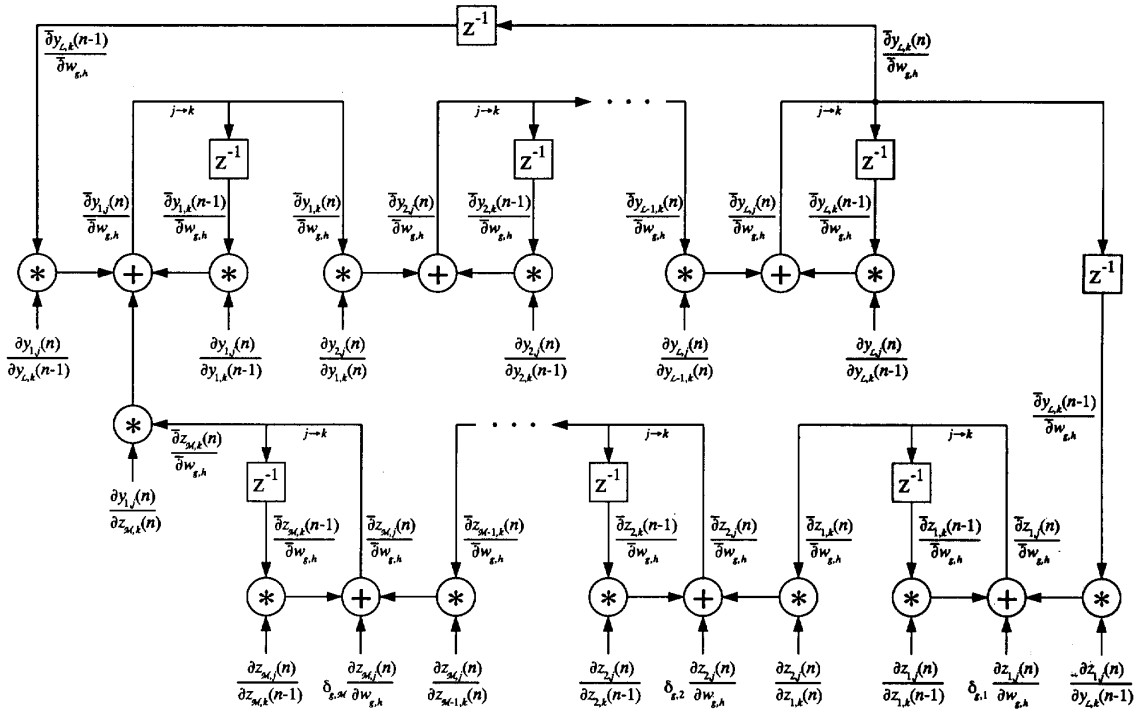


Fig. 5. Sensitivity circuit for controller training. The cascade for the identification network (overall flow left to right) is above that for the controller (overall flow right to left). The output of the circuit, $\frac{\partial y_{\mathcal{L},k}(n)}{\partial w_{g,h}}$ is time delayed and enters both cascades.

with the controller network:

$$\frac{\bar{\partial} z_{i,j}(n)}{\bar{\partial} w_{g,h}} = \sum_{k=1}^{\mathcal{P}_{i-1}} \frac{\partial z_{i,j}(n)}{\partial z_{i-1,k}(n)} \frac{\bar{\partial} z_{i-1,k}(n)}{\bar{\partial} w_{g,h}} + \gamma(n) \sum_{k=1}^{\mathcal{P}_i} \frac{\partial z_{i,j}(n)}{\partial z_{i,k}(n-1)} \frac{\bar{\partial} z_{i,k}(n-1)}{\bar{\partial} w_{g,h}} + \delta_{g,i} \frac{\partial z_{i,j}(n)}{\partial w_{g,h}}, \quad i > 1. \tag{7}$$

Since the controller's first subnetwork receives feedback signals from the plant's output, for $i = 1$, (5) becomes

$$\frac{\bar{\partial} z_{1,j}(n)}{\bar{\partial} w_{g,h}} = \gamma(n) \sum_{k=1}^{\mathcal{N}_{\mathcal{C}}} \frac{\partial z_{1,j}(n)}{\partial y_{\mathcal{L},k}(n-1)} \frac{\bar{\partial} y_{\mathcal{L},k}(n-1)}{\bar{\partial} w_{g,h}} + \gamma(n) \sum_{k=1}^{\mathcal{P}_1} \frac{\partial z_{1,j}(n)}{\partial z_{1,k}(n-1)} \frac{\bar{\partial} z_{1,k}(n-1)}{\bar{\partial} w_{g,h}} + \delta_{g,1} \frac{\partial z_{1,j}(n)}{\partial w_{g,h}}. \tag{8}$$

IV. Kalman Filter Based Training Algorithms

A. Background

The extended Kalman filter has served as the basis for a number of recent neural network training algorithm studies. Singhal and Wu [7] demonstrated that the EKF could be used to train MLP networks by treating the weights of the network as an unforced nonlinear dynamical system with stationary states. Alternatively, an identical training algorithm is derived by using the method of linearized recursive least squares [8]. Although this approach demonstrates fast learning as a function of number of presentations of training data, the training time per instance scales as the square of the number of weights in the network, thus rendering the algorithm impractical for many problems. A number of researchers have investigated simplifications to the EKF for training of MLP networks [9], [16]. In our own work [9], we show that an effective and efficient training algorithm can be achieved by ignoring the interdependencies of mutually exclusive groups of weights, thereby leading to lower computational complexity and storage per training instance. This decoupled EKF (DEKF) algorithm was demonstrated to exhibit faster training, both in terms of number of presentations of training data and in total training time on a serial processor, than a standard implementation of backpropagation for problems in pattern classification and function approximation. The classification and mapping performance of DEKF-trained networks was often found to be superior to that of networks trained by standard backpropagation.

The use of EKF for training recurrent networks was first explored by Matthews [17] with application to signal processing problems. Williams [13], [14] provides a detailed analytical treatment of EKF training of recurrent networks, and suggests that a ten-fold decrease relative to RTRL in the number of presentations of training data is achieved for

some simple finite state machine problems. In contrast to the EKF formulation of Singhal and Wu for MLP networks, the training problems as treated by Matthews and Williams provide for estimation of both network states (i.e., node outputs before they are recurrently used as input at the next time step) and parameters (i.e., the network's trainable weights). We refer to this class of training algorithm as *parallel* EKF to distinguish it from the *parameter-based* EKF algorithms in which only weight parameters are trained. Livstone *et al.* [18] have also explored the use of a parallel EKF for RBF networks with feedback connections. Although this development appears promising, we restrict the remainder of our discussion to recurrent networks with sigmoidal or linear node output functions.

In earlier work, independently of Matthews and Williams, we had extended the EKF formulation for MLP networks to architectures with recurrent connections [1], [19]. We describe below the parameter-based EKF and DEKF formulations, and discuss their application to training arbitrary neural networks with emphasis on neural controllers. We then discuss what we perceive to be the relative advantages and disadvantages of the parallel and parameter-based EKF formulations for training recurrent networks.

B. Parameter-Based EKF Algorithms

We view the neural network training problem as a parameter estimation problem, and treat the training of MLP and RMLP networks uniformly. The major difference in applying parameter-based EKF algorithms for the training of weights in MLP and RMLP networks lies in the computation of the derivatives of network outputs with respect to the trainable weights. For MLP networks, the derivatives are computed by application of backpropagation as described in [9]. However, for RMLP networks, the computation of dynamic derivatives in real time involves backpropagation of partial derivatives combined with dynamic derivatives from previous time steps (as shown in (1)–(6) above). Once the derivatives are computed, the same training algorithm applies to either class of network architecture.

The training problem is formulated as a weighted least squares minimization problem, where the error vector is the difference between functions of a network's output nodes and target values for these functions. Let the target vector at time step n be given by $\mathbf{d}(n) = [d_1(n)\, d_2(n) \cdots d_{N_c}(n)]^T$, and let $\mathbf{h}(n) = [h_1(\mathbf{y}(n))\, h_2(\mathbf{y}(n)) \cdots h_{N_c}(\mathbf{y}(n))]^T$ denote a vector of functions of the network's outputs $\mathbf{y}(n)$, where both $\mathbf{d}(n)$ and $\mathbf{h}(n)$ are of length N_c. The training cost function is given by $E(n) = \frac{1}{2}\xi(n)^T\mathbf{S}(n)\xi(n)$, where $\mathbf{S}(n)$ is a user-specified nonnegative definite weighting matrix, and where $\xi(n) = \mathbf{d}(n) - \mathbf{h}(n)$ is the error vector. The network's trainable weights are arranged into a M-dimensional vector $\mathbf{w}(n)$, and the estimate of the weight vector at time step n is denoted by $\hat{\mathbf{w}}(n)$. The EKF algorithm requires, in addition to the estimates of the network's weight vector, the storing and updating of an *approximate error covariance* matrix, $\mathbf{P}(n)$, which is used to model the correlations or interactions between each pair of weights in the network. The initial conditions at time step $n = 0$ for the weight vector are small random values (e.g., of magnitude less than 0.1), and the matrix $\mathbf{P}(0)$ is initialized as a diagonal matrix with large diagonal components (e.g., $O(100)$).

At the nth time step, the input signals and recurrent node outputs are propagated through the network, and the functions $\mathbf{h}(n)$ are computed. The error vector $\xi(n)$ is calculated, and the dynamic derivatives of each component of $\mathbf{h}(n)$ are formed with respect to the weights of the network, evaluated at the current weight estimates $\hat{\mathbf{w}}(n)$; these derivatives are arranged into the M-by-N_c matrix $\mathbf{H}(n)$. (In the case where $\mathbf{h}(n)$ is not differentiable, we approximate it with a differentiable form.) Then $\hat{\mathbf{w}}(n)$ and $\mathbf{P}(n)$ are updated by the following global EKF (GEKF) recursion:

$$\mathbf{A}(n) = \left[(\eta(n)\mathbf{S}(n))^{-1} + \mathbf{H}(n)^T\mathbf{P}(n)\mathbf{H}(n)\right]^{-1}, \quad (9)$$

$$\mathbf{K}(n) = \mathbf{P}(n)\mathbf{H}(n)\mathbf{A}(n), \quad (10)$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mathbf{K}(n)\xi(n), \quad (11)$$

$$\mathbf{P}(n+1) = \mathbf{P}(n) - \mathbf{K}(n)\mathbf{H}(n)^T\mathbf{P}(n) + \mathbf{Q}(n). \quad (12)$$

This parameter-based GEKF training algorithm includes a scalar learning rate parameter $\eta(n)$ which, in conjunction with the weighting matrix $\mathbf{S}(n)$, establishes the learning rate. The matrices $\mathbf{A}(n)$, for which a matrix of size N_c must be inverted, and $\mathbf{K}(n)$, the Kalman gain matrix, are computed at each time step, and are used to update the weight vector and error covariance matrix. Finally, $\mathbf{Q}(n)$ is a diagonal covariance matrix that provides a mechanism by which the effects of artificial process noise are included in the Kalman recursion. The presence of this matrix helps to avoid numerical divergence of the algorithm, and also helps the algorithm avoid poor local minima, as demonstrated in [9].

For the training of controller networks, $\mathbf{y}(n)$ refers to the output of the controller, while $\mathbf{h}(n)$ embeds the plant's input/output characteristics and, optionally, may include specified functions of controller and plant outputs. In the absence of the actual form of $\mathbf{h}(n)$, we approximate the components of the derivative matrix $\mathbf{H}(n)$ by using an input/output model of the plant, usually in the form of an identification network, together with the procedure described in Section III-C.

Jordan [20], [21] has described how the effects of constraints on a system's behavior are included in the cost function $E(n)$ for gradient descent training of controller networks by allowing the weighting matrix $\mathbf{S}(n)$ to be a function of the evaluations of the functions $\mathbf{h}(n)$. In general, if a *configurational* constraint is satisfied at time step n (e.g., the value of $h_i(\mathbf{y}(n))$ falls within a desired range), then the corresponding component of the error vector and derivatives of the function $h_i(\mathbf{y}(n))$ should not contribute to the update of $\hat{\mathbf{w}}(n)$. The constraint is imposed by introducing zero values into the appropriate columns and rows of the weighting matrix $\mathbf{S}(n)$ for time step n. This imposition of constraints via a weighting matrix that is a function of the system's behavior can also be employed with the GEKF algorithm. However, since (7) requires the inverse of $\mathbf{S}(n)$, the GEKF algorithm is reformulated to remove this mathematical difficulty. Defining $\xi^*(n) = \mathbf{S}(n)^{\frac{1}{2}}\xi(n) = \mathbf{S}(n)^{\frac{1}{2}}\mathbf{d}(n) - \mathbf{S}(n)^{\frac{1}{2}}\mathbf{h}(n) = \mathbf{d}^*(n) -$

$\mathbf{h}^*(n)$ yields the cost function $E(n) = \frac{1}{2}\xi^*(n)^T\xi^*(n)$. The GEKF algorithm is then given by

$$\mathbf{A}^*(n) = \left[\eta(n)^{-1}\mathbf{I} + \mathbf{H}^*(n)^T\mathbf{P}(n)\mathbf{H}^*(n)\right]^{-1}, \quad (13)$$

$$\mathbf{K}^*(n) = \mathbf{P}(n)\mathbf{H}^*(n)\mathbf{A}^*(n), \quad (14)$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mathbf{K}^*(n)\xi^*(n), \quad (15)$$

$$\mathbf{P}(n+1) = \mathbf{P}(n) - \mathbf{K}^*(n)\mathbf{H}^*(n)^T\mathbf{P}(n) + \mathbf{Q}(n), \quad (16)$$

where the scaling matrix $\mathbf{S}(n)$ is now distributed into both the scaled error vector $\xi^*(n) = \mathbf{S}(n)^{\frac{1}{2}}\xi(n)$ and the scaled derivative matrix $\mathbf{H}^*(n) = \mathbf{H}(n)\mathbf{S}(n)^{\frac{1}{2}}$. Although the two GEKF formulations are mathematically equivalent, the latter formulation gracefully handles the effects of configurational constraints.

The computational requirements of GEKF are dominated by the need to store and update the matrix $\mathbf{P}(n)$. GEKF's computational complexity is $O(N_cM^2)$ and its storage requirements are $O(M^2)$. The parameter-based DEKF algorithm is derived from GEKF by assuming that the interactions between certain weight estimates can be ignored. This simplification introduces many zeroes into the matrix $\mathbf{P}(n)$. If the weights are decoupled so that the weight groups are mutually exclusive of one another, then $\mathbf{P}(n)$ can be arranged into block-diagonal form. Let g refer to the number of such weight groups. Then, for group i, the vector $\hat{\mathbf{w}}_i(n)$ refers to the estimated weight parameters, $\mathbf{H}_i^*(n)$ is the submatrix of dynamic derivatives of the functions $\mathbf{h}^*(n)$ with respect to the weights, $\mathbf{P}_i(n)$ is the weight group's approximate error covariance matrix, and $\mathbf{K}_i^*(n)$ is its Kalman gain matrix. The concatenation of the vectors $\hat{\mathbf{w}}_i(n)$ forms the vector $\hat{\mathbf{w}}(n)$. Similarly, the global derivative matrix $\mathbf{H}^*(n)$ is composed of the individual submatrices $\mathbf{H}_i^*(n)$. The DEKF algorithm for the ith weight group is given by

$$\mathbf{A}^*(n) = \left[\eta(n)^{-1}\mathbf{I} + \sum_{j=1}^{g}\mathbf{H}_j^*(n)^T\mathbf{P}_j(n)\mathbf{H}_j^*(n)\right]^{-1}, \quad (17)$$

$$\mathbf{K}_i^*(n) = \mathbf{P}_i(n)\mathbf{H}_i^*(n)\mathbf{A}^*(n), \quad (18)$$

$$\hat{\mathbf{w}}_i(n+1) = \hat{\mathbf{w}}_i(n) + \mathbf{K}_i^*(n)\xi^*(n), \quad (19)$$

$$\mathbf{P}_i(n+1) = \mathbf{P}_i(n) - \mathbf{K}_i^*(n)\mathbf{H}_i^*(n)^T\mathbf{P}_i(n) + \mathbf{Q}_i(n). \quad (20)$$

A single matrix $\mathbf{A}^*(n)$, computed with contributions from all of the approximate error covariance submatrices and dynamic derivatives, is used with the scaled error vector $\xi^*(n)$ for updating the weight vectors and error covariance matrices for each weight group. In the limit of a single weight group ($g = 1$), the DEKF algorithm reduces exactly to the GEKF algorithm.

The computational complexity and storage requirements for DEKF can be significantly less than those of GEKF. For g disjoint weight groups, the computational complexity of DEKF becomes $O(N_c^2M + N_c\sum_{i=1}^{g}M_i^2)$, where M_i is the number of weights in group i, while the storage requirements become $O(\sum_{i=1}^{g}M_i^2)$. Note that this complexity analysis does not include the computational requirements for the matrix of dynamic derivatives. In particular, in the case of RMLP networks, the computational complexity of the derivative computations can be significant, as was discussed above.

For the purposes of this paper, we restrict ourselves to a DEKF algorithm for which the weights are grouped by node (i.e., the weights connecting inputs to a node are grouped together). We call this node-decoupled EKF (or NDEKF for short).

C. Comparison to Parallel EKF Training

The parallel EKF formulations of Matthews [17] and Williams [13], [14] have a number of significant differences from the parameter-based EKF algorithms described above. We summarize some of these differences for a recurrent network architecture consisting of a single layer of completely interconnected nodes, and discuss implications of these differences for practical applications. We concentrate on Williams' EKF formulation for comparative purposes.

The parallel EKF treats the state vector as the concatenation of the recurrent node outputs with the weight parameter vector. Thus, it is convenient to decompose the approximate global error covariance matrix into four distinct blocks: $\mathbf{P}_1(n)$ models the interactions between the recurrent node outputs, $\mathbf{P}_4(n)$ models the interactions between the weights of the network, and $\mathbf{P}_2(n) = \mathbf{P}_3(n)^T$ models the cross correlations between the network's weights and recurrent node outputs. The submatrices $\mathbf{P}_1(n)$ and $\mathbf{P}_4(n)$ lie along the diagonal of $\mathbf{P}(n)$, while $\mathbf{P}_2(n)$ and $\mathbf{P}_3(n)$ fill the off-block diagonal parts of $\mathbf{P}(n)$. The submatrix $\mathbf{P}_4(n)$ is equivalent to the entire matrix $\mathbf{P}(n)$ in the parameter-based GEKF recursion.

In applying the parallel EKF, there are two distinct steps, the *time* and *measurement* updates. The time update step involves propagating input signals and time-delayed recurrent node outputs through the recurrent network, computing the linearized dynamics, and propagating the error covariance. The linearized dynamics are provided by computing the *partial* derivatives of recurrent node outputs with respect to both the recurrent node outputs from the previous time step and the network's weight estimates. These partial derivatives, which are **not** recursively defined as a function of dynamic derivatives from the previous time step, are then used to propagate the error covariance matrix. The net effect of these steps is that the dynamic derivatives of recurrent node outputs with respect to the network's weights become embedded in the submatrices $\mathbf{P}_2(n)$ and $\mathbf{P}_3(n)$. These dynamic derivatives are coupled with the evolution of $\mathbf{P}(n)$ and do not appear to be directly accessible. In contrast, the parameter-based EKF computes the dynamic derivatives directly by the RTRL algorithm as a function of these derivatives from the previous time step.

The parallel EKF exhibits a number of appealing characteristics relative to the parameter-based formulation. First, the additional computational cost for state estimation is negligible, since the computational complexities and storage requirements of the two global EKF formulations along with the corresponding derivative computations are identical (for N nodes, $O(N^4)$). In addition, it may also be convenient to perform some decoupling of the weights in either formulation which

would lead to a block-diagonal structure for the submatrix $\mathbf{P}_4(n)$ in the parallel formulation. Although weight decoupling does not reduce the computational time complexity, the grouping of weights by node has the effect of reducing the storage requirements to $O(N^3)$. The parallel EKF may also exhibit performance superior to the parameter-based EKF because of cross coupling of estimated states and parameters. Finally, Williams [13], [14] points out that the parallel formulation has theoretical appeal since it provides a principled generalization of the *teacher forcing* mechanism that is often employed in recurrent network training by gradient descent.

For the parallel EKF, one potential difficulty for practical application involves training problems in which there are multiple trajectories or sequences: how to initialize the submatrix $\mathbf{P}_1(n)$ at the beginning of a new sequence of inputs. With each new trajectory, the submatrices $\mathbf{P}_2(n)$ and $\mathbf{P}_3(n)$ are initialized to null matrices, which initializes to zero the embedded dynamic derivatives of node outputs with respect to network weights. On the other hand, there is generally no need to initialize the submatrix $\mathbf{P}_4(n)$, although occasional resets may be useful for escaping poor local minima. Further, upon completion of training with the parallel formulation, it may be necessary in deployment to perform some aspect of Kalman filtering of the recurrent node outputs to achieve the same level of mapping performance as observed during training.

Perhaps the most critical obstacle in the practical application of a parallel EKF algorithm for real-time training of recurrent networks is the coupling of the dynamic derivatives with the approximate error covariance matrix. This coupling appears to preclude the use of computationally efficient methods for approximating or computing the derivatives in real time. A potentially useful alternative to the forward computation of dynamic derivatives as described by (1)–(6) is to approximate these derivatives by truncated backpropagation through time [5], [6]. The approximate derivatives computed by this method can be directly used by the parameter-based EKF, while it is not obvious how to do this in the parallel formulation.

V. Simulation Results

We have found that the parameter-based EKF controller training framework can be applied in simulation to a wide range of nonlinear control problems. In [19], we demonstrated the successful training of recurrent controller networks for a dynamical system with a multi-valued inverse. We have shown that recurrent networks are easily trained for the control of automotive subsystems such as active suspension [22] and anti-lock braking [23]. In both of these examples, the nonlinear systems were subjected to unmeasured disturbances. Furthermore, in addition to plant nonlinearities, phenomena such as actuator hysteresis and time delays were included for the anti-lock braking problem. For this paper, we have chosen three example problems in nonlinear control to demonstrate the effectiveness of training recurrent controller networks by EKF methods: 1) a variation of the well known cart-pole problem; 2) the bioreactor benchmark problem, for which the objective is to control a biochemical process; and 3) the control of an automotive engine operating at idle.

A. The Cart-Pole Problem

Although the cart-pole problem may be regarded as overemphasized in the neurocontrol and fuzzy logic control literature (akin to the XOR benchmark problem for pattern classification by neural networks), it has characteristics that make it a suitable benchmark for recurrent network controllers. The dynamical system equations and associated parameters as described in the landmark paper by Barto, Sutton and Anderson [24] form the basis of the simulation model of our work. The state variables of the system are the position and velocity of both the cart (x and $\dot{x}$) and the pole (θ and $\dot{\theta}$). The sampling rate of the original work [24], as well as most of the subsequent neural control studies for this problem, was 0.020 s. The training goal is to maintain the pole in an upright position ($\theta = 0$) while the cart simultaneously follows a commanded reference position; clearly it will not generally be possible to exactly satisfy both constraints simultaneously. Failure is defined to occur when the pole position exceeds 20 degrees in magnitude or when the cart position goes 2.4 m or more beyond the center of the track. The maximum magnitude of control applied to the cart is 10 Newtons (N).

We have found that controller training by parameter-based EKF easily handles the nominal cart-pole problem statement with bang-bang control if we use a cost function given by

$$E(n) = \frac{1}{2}\left[\left(\frac{x_r(n) - x(n)}{2.4}\right)^2 + \left(\frac{0 - \theta(n)}{20}\right)^2\right],$$

where $x_r(n)$ specifies the desired cart position at time step n. The identification network consists of a single layer of linear nodes, while the controller consists of a single nonlinear node which produces nearly saturated control outputs. Identification and controller training were repeatedly accomplished in less than 20 total failures of the cart-pole system. Furthermore, such a bang-bang controller can be trained almost as quickly under the more challenging conditions of sampling and control application at 0.1 s intervals. If we modify the problem to permit continuous control, the problem becomes even easier. We also observed no difficulty if the limiting pole position is extended from 20 degrees to 45 degrees. In this case, the system is required to operate over a more nonlinear region, but we found that use of a linear identification network continued to be satisfactory.

As the emphasis of this paper is on recurrent networks, we considered a modified problem statement that contains illustrative difficulties. First, we retained the more challenging sampling and control interval of 0.1 s. Next we assumed that only the positions (x and θ) are directly measurable. Thus, suitable control cannot be achieved without some observer capability, which we contend can be provided by a recurrent network controller. We permitted a number of disturbances to disrupt the behavior of the cart-pole system. For example, the cart experiences at each time step a small random force taken from a Gaussian distribution of zero mean and 0.1 standard deviation. In addition, a random force taken uniformly from the range ± 11 N is applied to the cart with 1% probability at each time step.

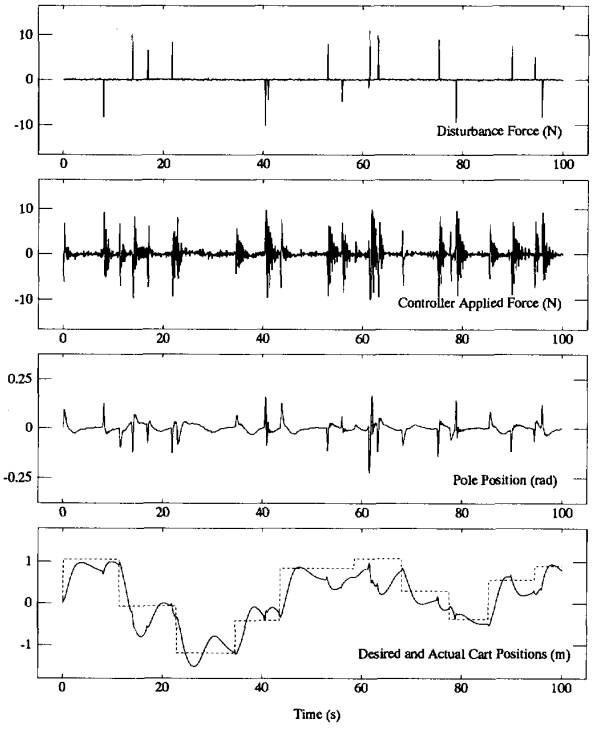


Fig. 6. Performance of a trained recurrent neural controller for the limited state information case (only cart and pole positions supplied). The cart position reference signal is given by the dashed pattern in the bottom panel.

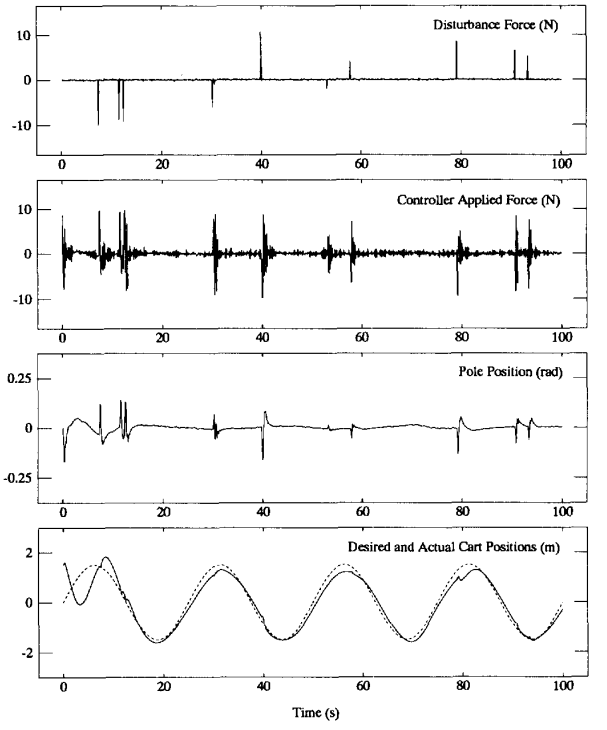


Fig. 7. Performance of a trained recurrent controller for a sinusoidal cart position reference signal. In this case, the controller's input vector is augmented with the outputs of the identification network's hidden recurrent nodes.

We considered both continuous and bang-bang control. For the continuous case, we conducted two training experiments. In the first, an identification network of six completely recurrent nodes with linear output functions was trained to predict cart and pole positions given a currently imposed control signal and the previously measured positions. For linear output functions, the recurrent weights are constrained to be less than unity in magnitude by treating them as the outputs of a bipolar sigmoid. Two of the six recurrent nodes of the network are taken as network outputs, and the four remaining nodes distributively encode unmeasured states of the system (i.e., velocity information). An RMLP controller network consisting of three inputs (reference input along with cart and pole positions), a hidden layer of six nonlinear recurrent nodes, and one self-recurrent output node was trained by the NDEKF algorithm. Performance of the trained RMLP controller network is demonstrated in Fig. 6 for a representative sequence of commanded reference positions and applied disturbance forces.

For the second experiment, the outputs of the four hidden nodes from the identification network were used as additional inputs to the controller network, since it is asserted that the identification network's hidden nodes encode useful state information. By augmenting the controller's input vector with the outputs of the identification network's hidden state nodes, we believed that controller training would be substantially easier. Using the same identification network and cost function as for the first simulation, an RMLP controller network of seven inputs, a hidden layer of four nonlinear recurrent nodes, and a self-recurrent output node was trained by NDEKF. Training was in fact easier and produced a somewhat smaller RMS error on a representative test sequence than did the unaugmented RMLP controller network. Fig. 7 shows the trained controller's ability to follow a sinusoidal reference signal while simultaneously balancing the pole in the face of random disturbances.

We have also repeated these training simulations for the 45-degree pole-angle limit, where we found that a six-node linear recurrent identification network sufficiently captured the input-output characteristics of the cart-pole system and produced dynamic derivatives that allowed for easy training of RMLP controller networks. However, we have not yet succeeded, even for the less challenging sampling interval of 0.02 s, in training a recurrent bang-bang controller when limited to cart and pole position measurements. Though we do not fully understand this difficulty, we hypothesize that the saturating behavior of the controller reduces our ability to compute accurately the dynamic derivatives that portray the long-term sensitivity of the plant output to controller weight changes.

B. The Bioreactor Benchmark Problem

The bioreactor problem is an adaptation of the problem as described in [25]. Coupled nonlinear differential equations describing a process involving a continuous flow stirred tank

reactor are given by

$$\dot{C}_1 = -C_1 u + C_1(1 - C_2)e^{C_2/\Gamma}$$
$$\dot{C}_2 = -C_2 u + C_1(1 - C_2)e^{C_2/\Gamma}\frac{1+\beta}{1+\beta - C_2}.$$

In these equations, the state variables C_1 and C_2 represent dimensionless forms of cell mass and amount of nutrients in a constant volume tank, bounded between zero and unity. The control u is the flow rate of nutrients into the tank, and is the same rate at which contents are removed from the tank. The constant parameters Γ and β determine the rates of cell formation and nutrient consumption; these parameters are set to $\Gamma = 0.48$ and $\beta = 0.02$ for the nominal benchmark specification. Practical restrictions are imposed on the rate at which the plant state can be measured and at which control can be imposed.

Following the benchmark specification, the dynamical equations are integrated with a forward Euler scheme using a step size of $\Delta = 0.01$ time units. The discrete time equations describing the evolution of the system are given by

$$C_1(t+1) = C_1(t) + \Delta\,(\,-C_1(t)u(t) + C_1(t)(1 - C_2(t))e^{C_2(t)/\Gamma})$$
$$C_2(t+1) = C_2(t) + \Delta\,(\,-C_2(t)u(t) + C_1(t)(1 - C_2(t))e^{C_2(t)/\Gamma}\,\tfrac{1+\beta}{1+\beta-C_2(t)})\,.$$

We define $k = 50\Delta$ (0.50 time units) as a *macro time step*, which defines the permissible intervals for sampling the system state, given by $(y_1(k), y_2(k))$, and for which the control $u(k)$ is applied. The overall goal for this problem is to adjust the control $u(k)$ at macro time step intervals so that the amount of cells produced is equal to a user-specified reference value, $y_r(k)$, which may be time-varying.

The bioreactor problem combines severe nonlinearity with a tendency to instability; active control must be exercised continuously to prevent divergent behavior. In the stable region of operation, a fixed value of control u leads to steady state behavior for both C_1 and C_2. As a level of control is increased beyond a certain point, the system begins to exhibit limit cycle behavior. As u is increased even further, the uncontrolled system ultimately becomes unstable. An example in the unstable region is illustrated in Fig. 8. Operation of the plant in the unstable region may be desired in order to maximize the yield of cells which is proportional to $C_1 u$. The plant is also characterized by a multi-valued inverse for which two different levels of control result in the same cell mass yield. The permissible intervals for state sampling and control as defined above are sufficiently large that it is difficult to infer the plant parameters by simple observation. Reference [25] suggests that it is difficult by standard methods to develop a controller that will guide the plant back and forth between stable and unstable regions, even with complete knowledge of plant parameters, and still harder to develop a controller that is robust to uncertainty in plant parameters.

We present in this paper results for a somewhat harder version of the benchmark problem than specified in [25], where it is suggested that controller networks be trained for three

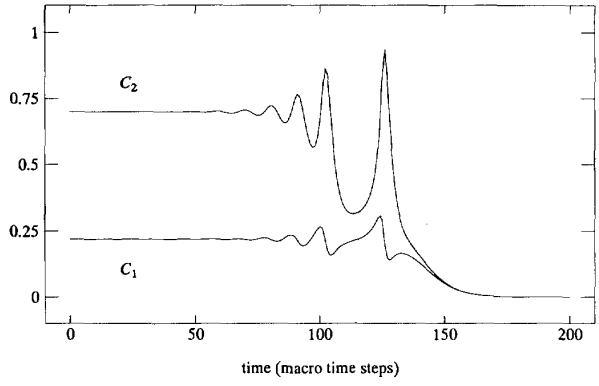


Fig. 8. Response of the uncontrolled bioreactor in the unstable region. Initial conditions are given by $C_1 = 0.22000$ and $C_2 = 0.69894$. A constant control of magnitude $u = 1.29133$ is applied. The bioreactor collapses after less than 200 macro timesteps.

cases: control of the plant 1) about a constant setpoint in the stable region, 2) about a constant setpoint in the unstable region, and 3) for a transition from a single setpoint in the stable region to one in the unstable region. Our version of the benchmark incorporates these individual cases into a single problem statement: train a single controller network that controls the plant about an arbitrary setpoint in either the stable or unstable region, and that also controls the system under transitions from one arbitrary setpoint (stable or unstable) to another. In addition, we also consider the same problem statement when only measurements of cell mass yield (C_1) are available; i.e., C_2 is not measured.

We assume that we have general knowledge of the system to be controlled, especially the fact that for target y_r above some value the behavior of the system becomes asymptotically unstable; i.e., exhibits limit cycles or instability. We also assume that collapse of the system, i.e., $C_1 \rightarrow 0$ and $C_2 \rightarrow 0$ as shown in Fig. 8, is undesirable, though not disastrous, and we attempt to minimize the number of system restarts encountered during the overall training procedure.

Our procedure proceeds in a bootstrap fashion. Knowing that a low value of control leads to stable behavior, we cautiously probe the system with momentarily increased control values in order to estimate the onset of limit cycle behavior, which we term the *limit cycle threshold*. We also observe both the minimum and maximum measured values of C_1 that seem to be attainable in the stable region. We then train identification networks, using a random pattern of control values extending up to the estimated limit cycle threshold. This procedure did not cause collapse of the bioreactor system. An MLP network with two hidden layers of 15 and 10 sigmoidal nodes was used to model the input/output behavior of the bioreactor for the full-state problem. The inputs to the network at time step k are measured state components $y_1(k)$ and $y_2(k)$ along with the imposed control $u(k)$. The last layer of the identification network consists of two linear summing nodes; the outputs of these nodes correspond to predictions of the state, $\hat{y}_1(k+1)$ and $\hat{y}_2(k+1)$, at the next macro time step. For the limited-state problem, an RMLP network consisting of two completely

recurrent hidden layers of 10 and 5 nodes and a single linear output node was used to model the bioreactor's input/output behavior. In this case, the inputs to the network at time step k are the measured cell mass yield $y_1(k)$ and the imposed control $u(k)$, while the output of the network is an estimate of the cell mass yield at the next macro time step, $\hat{y}_1(k+1)$.

The identification networks were trained with a fixed sequence of 10 000 data points. The control values were derived from a *skyline* function in which the dwell time τ_d was randomly chosen from $25 \leq \tau_d \leq 50$ macro time steps, and the control amplitude was chosen from an interval as determined by the initial system probing as described above. The series-parallel model was used for training of both networks so that the effects of external recurrence could be ignored, although the dynamic derivatives for training of the recurrent identification network evolved as specified by (1). Note that the identification networks as trained by this procedure can only be expected to provide good predictions of the bioreactor's evolution in the stable region, since the training set does not contain experiences outside this region.

Given the pre-trained identification networks, controller networks can be developed by a multi-stage procedure. For the full-state problem, the controller network is provided with three inputs, the measured state of the system, $y_1(k)$ and $y_2(k)$, along with a reference value for the desired cell mass yield, $y_r(k)$. For the limited-state problem, only $y_1(k)$ and $y_r(k)$ are provided as inputs. For each stage of training, the pattern of reference values is derived from a skyline function, with a dwell time chosen randomly from $100 \leq \tau_d \leq 400$ macro time steps. This range of dwell times was chosen so that the neural controller is trained to provide good performance for both steady-state and transient conditions. The minimum amplitude of $y_r(k)$ is chosen to be slightly larger than the estimated lowest value of C_1 as observed by probing the system. In the first stage of controller training, the maximum reference value is set at some fraction (e.g., 0.7) of the value of C_1 that was observed at the limit cycle threshold. The error signal used during training is given by the difference between an exponentially filtered value of $y_r(k)$ (filter coefficient 0.5) and the measured cell mass yield $y_1(k)$. After the error in following the filtered reference y_r has been reduced satisfactorily, the maximum value for y_r is increased a little and controller training is continued. The amount by which y_r is increased at each stage was chosen to be equal to 10% of the difference between the value of C_1 at the limit cycle threshold and the minimum observed stable value of C_1. This procedure is iterated; after each stage the trained network is saved and its performance is tested. We watch carefully for signs that C_1 is saturating, thus attempting to avoid carrying out substantial training with unattainable reference output. (For the nominal plant parameters, analysis of the plant equations discloses that no real-valued steady state solutions exist for $C_1 > 0.255$.) The number of stages of controller training before saturation of C_1 was found to be approximately 15.

We anticipated that some retraining of the identification networks would be required as training of controllers proceeded beyond the limit cycle threshold, since the original identification networks were only trained with control signals that did not explore these regions. Surprisingly, no retraining was necessary; rather, the identification networks appeared to be capable of providing reasonably good estimates of the dynamic derivatives as training proceeded into the unstable region, provided that a stabilizing control was being applied. This suggests that the dynamic derivatives of system outputs with respect to controller parameters, as estimated using an identification network trained with control signals applied in an open-loop fashion below the limit cycle threshold, are at least of the correct sign for a closed-loop system that is providing stabilizing control near or in the unstable region. The one-step predictions of the trained identification network for the *controlled* system with full-state information and with excursions beyond the limit cycle threshold are shown in Fig. 9, where it is seen that the prediction errors are large in the unstable region.

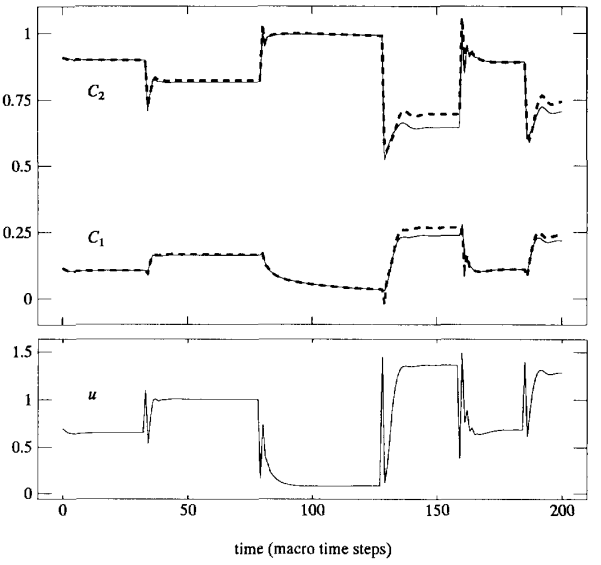


Fig. 9. One-step identification network predictions of states C_1 and C_2 for closed-loop control that takes the bioreactor beyond the limit cycle threshold. The actual state values are solid while the predictions are dashed.

For the full-state problem specification, we have successfully trained both MLP and RMLP controller networks, although we report in this paper only on results for recurrent controller networks. The chosen RMLP network consisted of three layers of nodes, with 3 inputs, two fully recurrent hidden layers of 7 and 4 nonlinear nodes respectively, and an output layer of a single nonlinear node without feedback. This RMLP network contains 130 trainable weights. For the limited-state problem specification, it was necessary to employ a recurrent controller in order to account for the missing state information. The RMLP network chosen for this problem consisted of three layers of nodes, with two fully recurrent hidden layers of 10 and 5 nodes respectively, and a single nonlinear node in the output layer; this controller network contains 216 trainable weights. In each case, we employed a derivative discount factor of $\gamma = 0.99$ during training with the NDEKF algorithm.

Upon completion of training, the overall performance that the final controller network achieves for the full-state case is

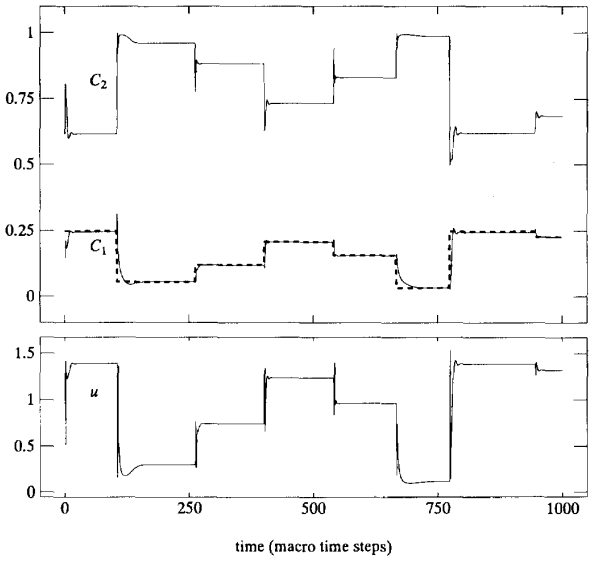


Fig. 10. Controller network performance for the nominal plant parameters with full-state information. The reference signal y_r is plotted as a dashed pattern.

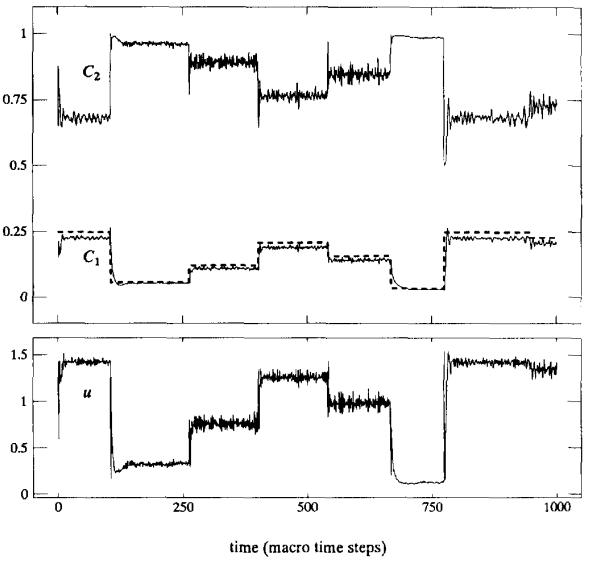


Fig. 11. Performance of the controller network for a bioreactor characterized by 5% change in Γ and a 20% change in β and added measurement noise. The traces reflect the true state values; the measurement noise reaches the system through the controller.

excellent, as shown in Fig. 10. We tested the robustness of this controller by applying it (without retraining) to a plant with altered parameters ($\Gamma = 0.456$ and $\beta = 0.016$) and with Gaussian measurement noise of standard deviation equal to 0.01 added to both state components. Representative results are shown in Fig. 11. (The measured state values are considerably noisier than the actual values shown.) While the tracking is not as good as for the nominal plant, it is reasonable considering the magnitude of the parameter alterations (5% for Γ and 20% for β).

The performance of the RMLP controller network for the limited-state problem with nominal parameters is reasonable, as shown in Fig. 12, although the applied control exhibits more ringing than did the controller for the full-state information problem (Fig. 10). In Fig. 13 we show that the limited-state controller exhibits good tracking performance in the presence of measurement noise. However, this controller is not particularly robust to the imposed plant parameter variations, as shown in Fig. 14. It is clear that the performance of this controller degrades more rapidly than did the full-state controller. This is not entirely surprising, since the implicit observer capability of the recurrent network may be expected to degrade when the plant changes, just as a traditional observer degrades when the plant ceases to be well represented by the model parameters on which it is based.

Though the mechanics of the training procedure are the same as we have used for many other problems, we wished to insure that our procedure does not depend, even implicitly, on prior mathematical analysis of the plant equations. Hence, to simulate dealing with plants with similar characteristics but not previously the subject of analysis, we repeated the procedure for other sets of plant parameters randomly chosen from the ranges $0.3 < \Gamma < 0.5$ and $0 < \beta < \frac{1}{4-1/\Gamma} - 0.5$ [26], and deferred mathematical analysis until after training. In each case

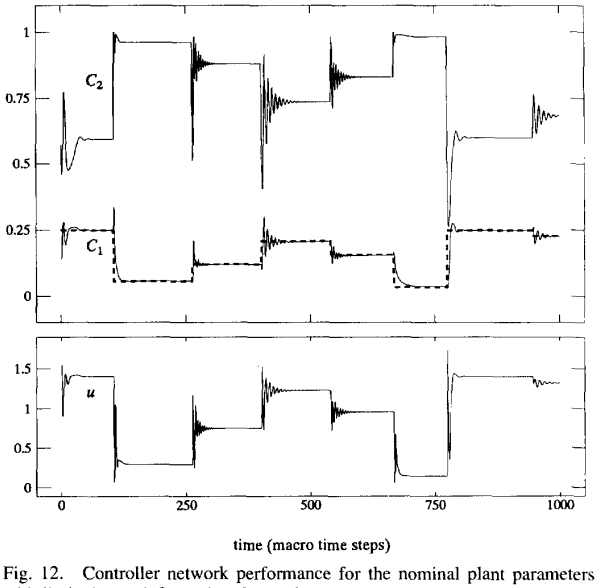


Fig. 12. Controller network performance for the nominal plant parameters with limited-state information. The reference signal y_r is plotted as a dashed pattern. There is considerably greater ringing in the system due to step changes in y_r than for the full-state information problem (compare to Fig. 10).

we were able to carry out identification and controller network training satisfactorily for both problem specifications. Results for these simulations are comparable to those described above.

C. Engine Idle Speed Control (ISC)

As a final example, we treat the control of an engine model operating under idle, for which the engine must operate far away from its optimal region of operation. The operation

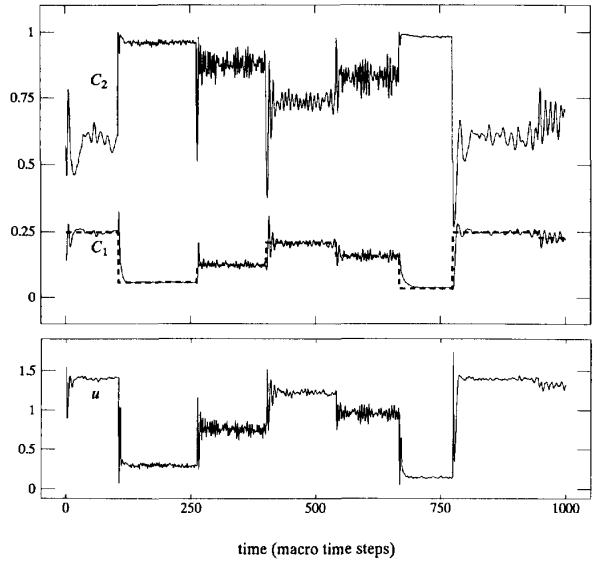


Fig. 13. Controller network performance for the nominal plant parameters with limited-state information and added Gaussian noise of zero mean and 0.01 standard deviation. The traces reflect the true state values.

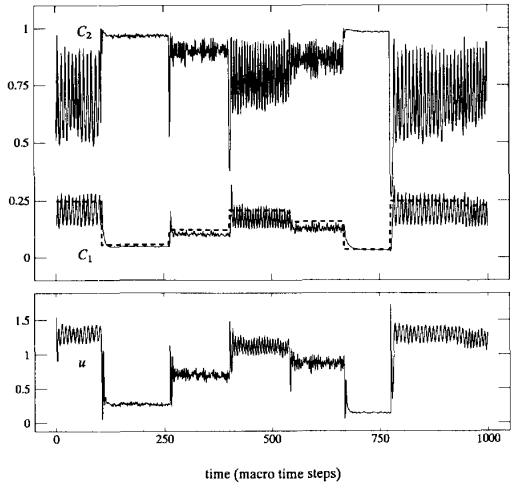


Fig. 14. Performance of the limited-state information controller network for a bioreactor characterized by 5% change in Γ and a 20% change in β and added measurement noise. Unlike the results of Fig. 11 for the full-state information controller, the limited-state information controller is not very robust to the effects of these parameter changes.

and regulation of an engine at idle is a highly nonlinear process. The process has time delays that vary inversely with engine speed and is time-varying due to aging of components and environmental changes such as engine warm-up after a cold start. The measurement of system outputs occurs asynchronously with the calculation of control signals. We assume that the occurrence of plant disturbances, such as engagement of air conditioner compressor, shift from neutral to drive in automatic transmissions, application and release of electric loads, and power steering lock-up, are not directly measured. Other random or periodic disturbances may affect the idling process as well. The principal goal of ISC is to regulate the engine speed at a desired set point in the face of these unmeasured disturbances.

The dynamic engine model employed in this study was derived from steady-state engine map data and empirical information by Powell *et al.* [27], with revisions as described by Vachtsevanos *et al.* [28]. The engine model parameters are for a 1.6 liter, 4-cylinder fuel injected engine. The model is a two-input/two-output system, given by the following differential equations:

$$\begin{aligned}
\dot{P} &= k_P(\dot{m}_{ai} - \dot{m}_{ao}), \quad k_P = 42.40 \\
\dot{N} &= k_N(T_i - T_L), \quad k_N = 54.26 \\
\dot{m}_{ai} &= (1 + 0.907\theta + 0.0998\theta^2)g(P) \\
g(P) &= \begin{cases} 1 & P < 50.6625 \\ 0.0197(101.325P - P^2)^{\frac{1}{2}} & P \geq 50.6625 \end{cases} \\
\dot{m}_{ao} &= -0.0005968N - 0.1336P \\
&\quad + 0.0005341NP + 0.000001757NP^2 \\
m_{ao} &= \dot{m}_{ao}(t - \tau)/(120N), \quad \tau = 45/N \\
T_i &= -39.22 + 325024m_{ao} - 0.0112\delta^2 + 0.635\delta \\
&\quad + (0.0216 + 0.000675\delta)N(2\pi/60) \\
&\quad - 0.000102N^2(2\pi/60)^2 \\
T_L &= (N/263.17)^2 + T_d \ .
\end{aligned}$$

For notational simplicity, explicit dependence on time is suppressed in these equations, except in the case of the delayed $\dot{m}_{ao}$. The system outputs are manifold pressure P expressed in kPa, with a range of [0,101.325], and engine speed N in rpm, with an approximate range of [300,1200] for an idling engine. The control inputs are throttle angle θ and spark advance δ in degrees, with ranges of [5,25] and [10,45], respectively. Disturbances act on the engine in the form of unmeasured accessory torque T_d in the range [0,61] N-m. In addition, the variables $\dot{m}_{ai}$ and $\dot{m}_{ao}$ refer to the mass air flow into and out of the manifold, respectively; m_{ao} is air mass in the cylinder; τ is a dynamic transport time delay; $g(P)$ is a manifold pressure influence function; T_i is the engine's internally developed torque; and T_L is the load torque. The references listed above present a more detailed discussion of these equations and engine dynamics. We also note that a recent treatment of this problem by fuzzy control methods [28] ignores the time delay for computing the m_{ao} altogether, thereby considerably simplifying the problem.

In simulation, a backward Euler scheme with a step size of 1 ms is used for numerical integration of the plant equations, since the differential equations describing the performance of the engine exhibit *stiff* behavior in certain regions of operation. Control signals for throttle θ and spark advance δ are computed in a simulated background loop of 20 ms intervals. However, application of the computed control signals occurs only twice per engine revolution. Hence, when the engine speed is low, certain computed control signals may not be used; conversely, when the engine speed is high, the cycle time of the background loop is greater than the time interval between applications of control signals, and the same control command may be applied more than once. Manifold pressure and engine

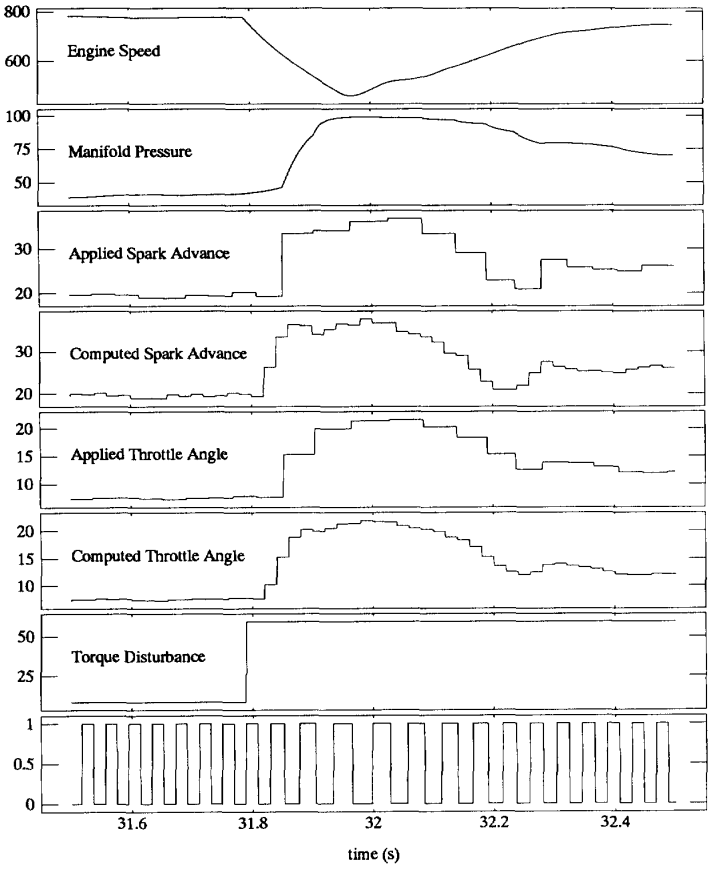


Fig. 15. Asynchronous sampling and application of control for the idle speed control problem. The transitions between low and high values in the bottom panel of pulses indicates the measurement of system outputs occurring four times per engine revolution. Control is applied only on the high-to-low transitions (twice every engine revolution). Note that the transitions occur asynchronously in time as a function of engine speed. The controls are computed once every 20 ms, from which the applied controls are derived asynchronously.

speed are sampled 4 times per engine revolution, where the current engine position (crank angle) is calculated from actual engine speeds. Measured manifold pressure P_m is corrupted by Gaussian noise of zero mean and standard deviation of 1 kPa, while measured engine speed N_m is corrupted by Gaussian noise of zero mean and standard deviation of 2.5 rpm. The application of control signals is delayed by at least 90 crank angle degrees (20 ms at 750 rpm) from the most recent measurements on which the control is based. Disturbances, ranging from 0 to 61 N-m, enter the system at a time resolution of 1 ms; hence, changes in disturbance input occur asynchronously with the application of control and the measurement of system outputs. The asynchronous nature of the output measurements with the computation and application of controls is illustrated in Fig. 15.

The two controls dynamically affect engine speed quite differently. The throttle command θ has a large dynamic range (thousands of rpm), but its effect is delayed by a time inversely proportional to engine speed (for a steady-state engine speed of 750 rpm, the delay is 60 ms). On the other hand, the spark advance δ has an immediate effect on the engine speed, but its dynamic range is an order of magnitude smaller than that of the throttle command. In addition to maintaining engine speed at a desired setpoint of 750 rpm, we would like to achieve a number of subjective secondary criteria. For example, it is desirable to maximize fuel efficiency while simultaneously minimizing vehicle vibrations. Fuel efficiency can be increased by operating the engine with an advanced spark angle. For the engine model described above, the optimal spark advance for fuel efficiency and torque output is at 30.9 degrees for a steady-state engine speed of 750 rpm. However, the spark advance must be retarded relative to this value if the spark advance control is to be useful in responding to torque load disturbances. A nominal spark advance of 22.9 degrees is chosen to allow rapid system response to unobserved torque disturbances while maintaining a reasonable level of fuel efficiency.

A recurrent identification network consisting of a single layer of 8 completely interconnected nodes with *linear* output functions was trained to model the input/output characteristics of the dynamical plant described above. The network has four inputs, consisting of linearly transformed values of measured system outputs, P_m and N_m, and linearly transformed values of control inputs, θ and δ. The output of the network is

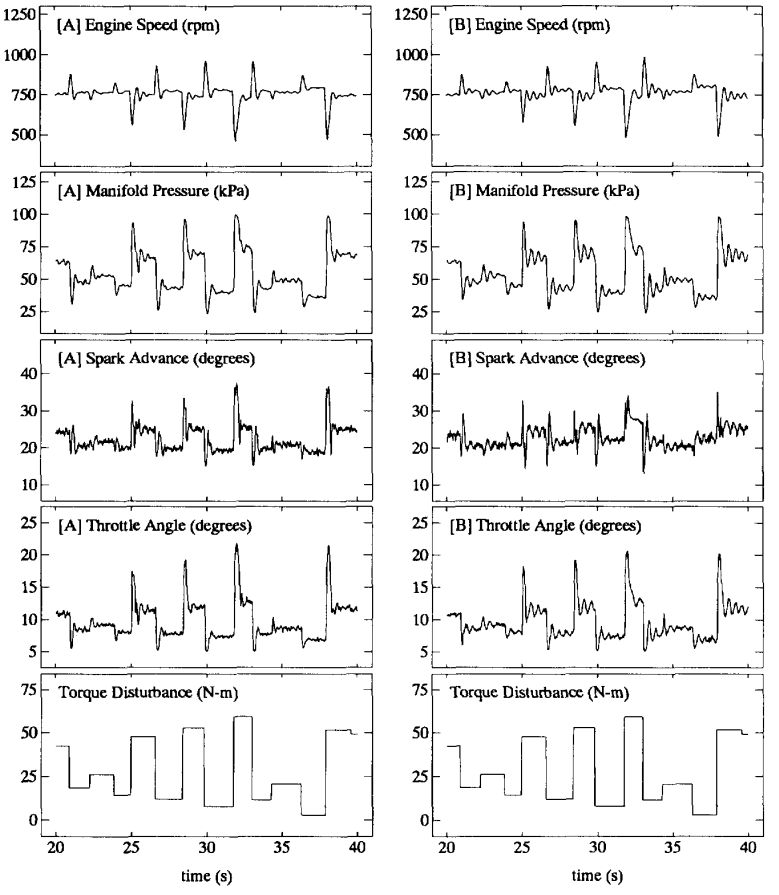


Fig. 16. Comparative behavior of the recurrent and generalized PID controller networks for a test sequence of load disturbances representative of those used during training. The imposed load disturbance is shown in the lowest panels. The panels on the left show the plant inputs and outputs for the recurrent controller network [A], while the panels on the right demonstrate the input/output behavior of the generalized PID controller network [B].

a prediction of the system output 20 ms into the future, corresponding to the background loop time for the calculation of control signals. The feedback weights of linear recurrent nodes are constrained to be less than unity in magnitude by treating them as the outputs of bipolar sigmoidal functions. The control inputs and torque disturbances were varied through their entire ranges during training of the identification network. Since the torque disturbances are not measured, the prediction of system outputs by the identification network will be in error, particularly during large transitions in torque disturbance. However, we require only that on average the derivatives of system outputs with respect to controller parameters be correct for controller training to be effective.

We chose a quadratic cost function consisting of four components. The first penalizes deviations of engine speed from a set point of 750 rpm. The next three components penalize certain behaviors of the control signals. We include a term in the cost function that penalizes deviations of the spark advance from its desired set point of 22.9 degrees. The third and fourth components penalize large changes in throttle and spark angle, respectively, between two successive control time steps. These two components implement a *smoothness* constraint [20], [21] that is observed to help avoid oscillatory behavior in the controls for dynamical systems with significant internal time delays. The contribution to the cost function at time step $n+1$ is given by

$$E(n+1) = \frac{1}{2}\left(\beta_1(750 - N_m(n+1))^2 + \beta_2(22.9 - \delta(n))^2 + \beta_3(\theta(n-1) - \theta(n))^2 + \beta_4(\delta(n-1) - \delta(n))^2 \right),$$

where the weighting factors are chosen empirically and are given by $\beta_1 = 1.6 \times 10^{-7}$ for $N_m(n+1) > 750$ and $\beta_1 = 2.4 \times 10^{-7}$ for $N_m(n+1) \leq 750$; $\beta_2 = 3.25 \times 10^{-7}$; $\beta_3 = 10^{-3}$; and $\beta_4 = 1.63 \times 10^{-5}$.

We present experimental results for two RMLP controller networks. The first network consists of two layers of nodes, with two inputs (P_m and N_m), a hidden layer of 8 sigmoidal recurrent nodes, and 2 recurrent output nodes with sigmoidal output functions, where we chose to include all possible connections for the two layers. This RMLP network has 110 trainable parameters. The second RMLP controller network contains a linear recurrent preprocessing layer followed by a standard feedforward network. The inputs to the preprocessing

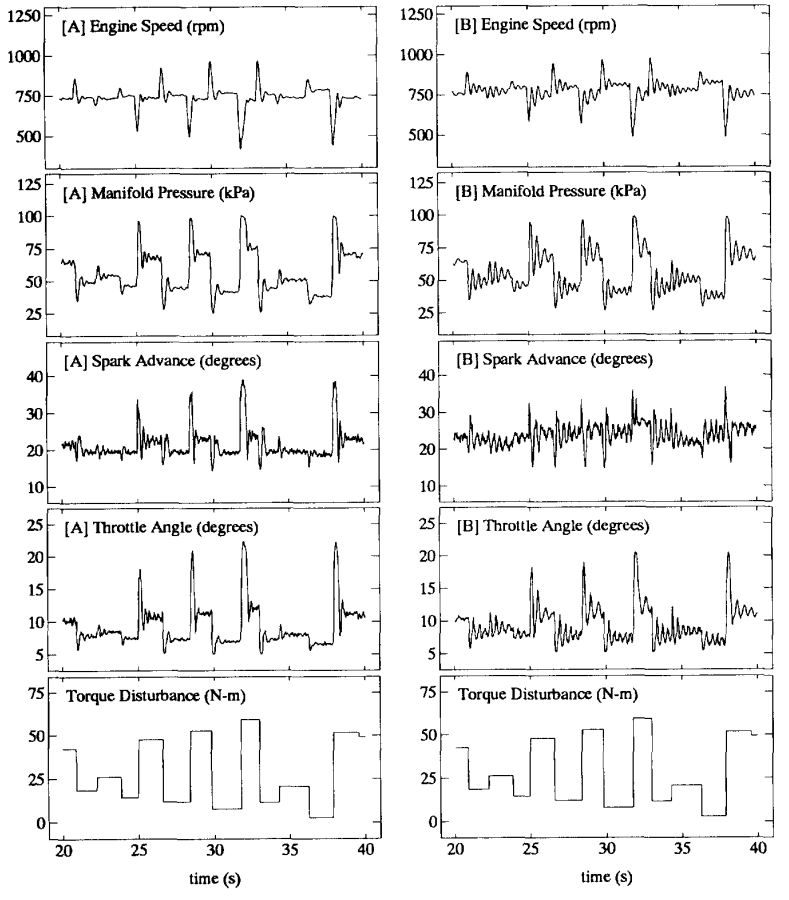


Fig. 17. Comparative behavior of the recurrent [A] and generalized PID [B] controller networks for a plant definition with altered parameters and an offset in measurement of manifold pressure. The organization of this figure is identical to that of Fig. 16.

layer are measured manifold pressure and error in engine speed. The preprocessing layer consists of seven linear nodes with limited interconnectivity. The first 3 nodes were chosen to encode proportional and derivative terms for manifold pressure, while the 4 remaining nodes construct proportional, integral and derivative terms for the error in engine speed. The remainder of the network consisted of three standard feedforward layers, with 10, 6, and 2 sigmoidal nodes from the preprocessing layer to the output layer. The 5 nodes of the preprocessing layer that encode PD terms for manifold pressure and PID terms for error in engine speed are connected to the following layer of 10 nodes. This network consists of 150 weights, and we allow all weights, including those of the preprocessing layer, to be trained. We will simply refer to the first architecture described above as a recurrent controller network and to the second network as a generalized PID controller network.

During controller training, torque disturbances are randomly selected from the range 0 to 61 N-m and are applied for a time ranging between 1 and 2 seconds of simulated real time. The training sequence of load disturbances was chosen to be identical for the two network architectures. The dynamic derivatives are computed as described in Section III, where we employ a derivative discount factor of $\gamma = 0.99$. Training was carried out for a total of 100 000 training instances (approximately 30 minutes of simulated real time), although we observed that both controller networks were capable of regulating the engine speed close to 750 rpm with minimal ringing in the applied control signals and measured system outputs after approximately 20 000 training instances.

The behavior of the trained controllers was examined with an independent test sequence of torque disturbances generated in a fashion similar to that used for training. Representative results are shown in Fig. 16. Qualitatively, the recurrent controller network is seen to provide smoother behavior than the generalized PID network, both for deviations from the desired set point of 750 rpm and for oscillations in the control signals.

We also examine the robustness properties of the two trained controllers. The plant definition provided above is altered by changing 5 parameters. The constants k_P and k_N are set to 38.16 and 59.69, respectively. The effect of throttle on the

plant is altered by changing the intake mass air flow equation to $\dot{m}_{ai} = (1.2 + 0.907\theta + 0.12\theta^2)g(P)$. The load torque is given by $T_L = (N/236.85)^2 + T_d$. As a final complication, the measured manifold pressure has an offset of -10 kPa. The behavior of the two controller networks, without any retraining, is shown in Fig. 17, from which it is clear that the recurrent controller network provides significantly more robust performance that does the generalized PID controller. For purposes of comparison, we also trained a simple linear controller network with 2 layers of nodes. The two inputs were manifold pressure and error in engine speed, the hidden layer consisted of 7 linear preprocessing nodes as used in the generalized PID controller network, and the output layer consisted of two linear nodes. This network provides a linear combination of PD terms for manifold pressure and PID terms for error in engine speed for both control signals. The behavior exhibited by this linear controller was substantially inferior to the behavior of either of the more complicated nonlinear controller networks.

We have tested the trained controller networks for load disturbance sequences that had longer intervals of relatively constant disturbance, and found that the steady state behavior in engine speed and manifold pressure was constant except for small variations due to measurement noise. Even in the unrealistic context of rapidly varying torque disturbances, e.g., switching back and forth between 5 and 55 N-m at a frequency of 1 Hz for 10 cycles, the recurrent controller network exhibits stable behavior and settles back to constant control after the disturbance is removed. We have also found that an RMLP controller network containing as few as four recurrent nodes in a hidden layer followed by two recurrent output nodes (42 trainable parameters) was capable of achieving a reasonable level of performance for these various test sequences.

VI. Summary and Conclusions

In this paper, we have attempted to clarify the computation of derivatives for recurrent systems with internal recurrence, external recurrence, or both, including recurrent multilayer perceptrons and systems for controller training. The subnetwork decomposition structures the computation of derivatives so as to make maximal use of conventional backpropagation and minimize the amount of computationally expensive forward propagation. We have shown how such dynamic derivatives are used to make weight updates with a parameter-based extended Kalman filter method, and have commented on alternative EKF formulations. We have illustrated our synthesis of derivative computation and decoupled EKF weight updates by treating three diverse control problems that benefit from use of recurrent controllers. Though we have not carried out rigorous comparisons between simple gradient updates and DEKF updates, we have made limited side-by-side comparisons, uniformly to the advantage of the latter.

References

[1] G. V. Puskorius and L. A. Feldkamp, "Recurrent network training with the decoupled extended Kalman filter algorithm," in *Proceedings of the 1992 SPIE Conference on the Science of Artificial Neural Networks*, Orlando, 1992, vol. 1710, pp. 461–473.

[2] G. V. Puskorius and L. A. Feldkamp, "Automotive engine idle speed control with recurrent neural networks," in *Proceedings of the 1993 American Control Conference*, San Francisco, 1993, pp. 311-316.

[3] R. J. Williams, "Adaptive state representation and estimation using recurrent connectionist networks," in *Neural Networks for Control*, W. T. Miller III, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1990, chap. 4, pp. 97–114.

[4] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[5] 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.

[6] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation*, vol. 2, pp. 490–501, 1990.

[7] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended Kalman algorithm," in *Advances in Neural Information Processing Systems 1*, Denver 1988, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 133–140.

[8] S. C. Douglas and T. H.-Y. Meng, "Linearized least-squares training of multilayer feedforward neural networks," in *International Joint Conference on Neural Networks*, Seattle, 1991, vol. I, pp. 307–312.

[9] G. V. Puskorius and L. A. Feldkamp, "Decoupled extended Kalman filter training of feedforward layered networks," in *International Joint Conference on Neural Networks*, Seattle 1991, vol. I, pp. 771–777.

[10] D. W. Ruck, S. K. Rogers, M. Kabrisky, P. S. Maybeck and M. E. Oxley, "Comparative analysis of backpropagation and the extended Kalman filter for training multilayer perceptrons," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 6, pp. 686–690, 1992.

[11] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[12] 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.

[13] R. J. Williams, "Some observations on the use of the extended Kalman filter as a recurrent network learning algorithm," Technical Report NU-CCS-92-1, Boston: Northeastern University, College of Computer Science, 1992.

[14] R. J. Williams, "Training recurrent networks using the extended Kalman filter," in *International Joint Conference on Neural Networks*, Baltimore 1992, vol. IV, pp. 241–246.

[15] B. Fernandez, A. G. Parlos, and W. K. Tsai, "Nonlinear dynamic system identification using artificial neural networks (ANNs)," in *International Joint Conference on Neural Networks*, San Diego 1990, vol. II, pp. 133–141.

[16] S. Shah, F. Palmieri, and M. Datum, "Optimal filtering algorithms for fast learning in feedforward neural networks," *Neural Networks*, vol. 5, pp. 779–787, 1992.

[17] M. B. Matthews, "Neural network nonlinear adaptive filtering using the extended Kalman filter algorithm," in *Proceedings of the International Neural Networks Conference*, Paris 1990, vol. I, pp. 115–119.

[18] M. M. Livstone, J. A. Farrell, and W. L. Baker, "A computationally efficient algorithm for training recurrent connectionist networks," in *Proceedings of the 1992 American Control Conference*, Chicago, 1992, vol. II, pp. 555–561.

[19] G. V. Puskorius and L. A. Feldkamp, "Model reference adaptive control with recurrent networks trained by the dynamic DEKF algorithm," in *International Joint Conference on Neural Networks*, Baltimore, 1992), vol. II, pp. 106–113.

[20] M. I. Jordan, "Supervised learning and systems with excess degrees of freedom," in *Proceedings of the 1988 Connectionists Summer School*, Pittsburgh, 1988, D. Touretzky, G. Hinton and T. Sejnowski, Eds. San Mateo, CA: Morgan Kaufmann, pp. 62–75.

[21] M. I. Jordan, "Generic constraints on underspecified target trajectories," in *International Joint Conference on Neural Networks*, Washington D.C., 1989, vol. I, pp. 217–225.

[22] L. A. Feldkamp, G. V. Puskorius, L. I. Davis, Jr., and F. Yuan, "Neural control systems trained by dynamic gradient methods for automotive applications," in *International Joint Conference on Neural Networks*, Baltimore, 1992, vol. II, pp. 798–804.

[23] L. I. Davis, Jr., G. V. Puskorius, F. Yuan, and L. A. Feldkamp, "Neural network modeling and control of an anti-lock brake system," in *Proceedings Intelligent Vehicle '92 Symposium*, Detroit, 1992, pp. 179–184.

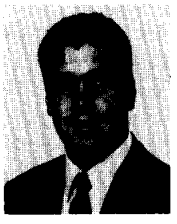
[24] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, no. 5, pp. 835–846, 1983).

[25] L. H. Ungar, "A bioreactor benchmark for adaptive network-based process control," in *Neural Networks for Control*, W. T. Miller, III, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, chap. 16, pp. 387–402 and Appendix A, 1990, pp. 476–480. [Note errors in the equations on page 477.]

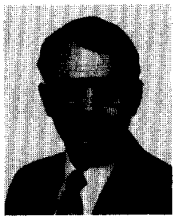
[26] P. Agrawal, C. Lee, H. C. Lim, and D. Ramkrishna, "Theoretical investigations of dynamic behavior of isothermal continuous stirred tank biological reactors," *Chemical Engineering Science* vol. 37, no. 3, pp. 453–462, 1982.

[27] B. K. Powell and J. A. Cook, "Nonlinear low frequency phenomenological engine modeling and analysis," *Proceedings of the 1987 American Control Conference*, vol. 1, pp. 336–340, 1987; J. A. Cook and B. K. Powell, "Modeling of an internal combustion engine for control analysis," *IEEE Control Systems Magazine*, vol. 8, no. 4, 20–26, 1988.

[28] G. Vachtsevanos, S. S. Farinwata, and D. K. Pirovolou, "Fuzzy logic control of an automotive engine," *IEEE Control Systems Magazine*, vol. 13, no. 3, 62–68, 1993.



Gintaras V. Puskorius received the B.S. degree in Engineering Physics (1980) and the M.S. degree in Physics (1982), both from John Carroll University. He joined Ford Motor Company's Physics Department in 1982, and moved to the Manufacturing Systems Department in 1989 where he is currently a Research Scientist Senior with the Artificial Neural Networks group. His past research activities have included studies of alkali metals, laser scanning devices, machine vision and robotics, and image processing for scanning tunneling microscopy. His current activities involve both fundamental studies of neural network learning algorithms as well as application of neural network methods to problems in control, diagnostics and computer aided engineering. His research has resulted in the publication of over 30 papers and technical reports.



Lee A. Feldkamp received the B.S.E. degree in Electrical Engineering (1964) and the M.S. (1965) and Ph.D. (1969) degrees in Nuclear Engineering, all from the University of Michigan. He is currently a Staff Scientist with Ford Motor Company's Research Laboratory. He was a member of Ford's Physics Department from 1969 to 1989, where his fields of interests included neutron scattering, electron energy-loss spectroscopy, theory of electronic spectroscopies, machine vision for robotics, image processing, x-ray tomography, and the biomechanics of bone (with the University of Michigan). In 1989, he joined the Manufacturing Systems Department in order to conduct both fundamental and applied research and to pursue application of neural networks, fuzzy logic and related computing technologies. He has published one book and has authored about 70 papers and reports. Dr. Feldkamp is a member of the American Physical Society and the International Neural Network Society.