Adaptive Inverse Control of Linear and Nonlinear Systems Using Dynamic Neural Networks

Gregory L. Plett, *Senior Member, IEEE*

***Abstract*—In this paper, we see adaptive control as a three-part adaptive-filtering problem. First, the dynamical system we wish to control is modeled using adaptive system-identification techniques. Second, the dynamic response of the system is controlled using an adaptive feedforward controller. No direct feedback is used, except that the system output is monitored and used by an adaptive algorithm to adjust the parameters of the controller. Third, disturbance canceling is performed using an additional adaptive filter. The canceler does not affect system dynamics, but feeds back plant disturbance in a way that minimizes output disturbance power. The techniques work to control minimum-phase or nonminimum-phase, linear or nonlinear, single-input–single-output (SISO) or multiple-input–multiple-ouput (MIMO), stable or stabilized systems. Constraints may additionally be placed on control effort for a practical implementation. Simulation examples are presented to demonstrate that the proposed methods work very well.**

***Index Terms*—Adaptive inverse control, disturbance canceling, disturbance rejection, feedforward control, system identification.**

I. Introduction

PRECISE control of dynamic systems ("plants") can be exceptionally difficult, especially when the system in question is nonlinear. In response, many approaches have been developed to aid the control designer. For example, an early method called *gain scheduling* [1] linearizes the plant dynamics around a certain number of preselected operating conditions and employs different linear controllers for each regime. This method is simple and often effective, but applications exist for which the approximate linear model is not accurate enough to ensure precise or even safe control. One example is the area of flight control, where stalling and spinning of the airframe can result if operated too far away from the linear region.

A more advanced technique called *feedback linearization* or *dynamic inversion* has been used in situations including flight control (e.g., see [2]). Two feedback loops are used: 1) an inner loop uses an inverse of the plant dynamics to subtract out all nonlinearities, resulting in a closed-loop linear system and 2) an outer loop uses a standard linear controller to aid precision in the face of inverse plant mismatch and disturbances.

Whenever an inverse is used, accuracy of the plant model is critical. So, in [3] a method is developed that guarantees robustness even if the inverse plant model is not perfect. In [4], a robust and adaptive method is used to allow learning to occur on-line, tuning performance as the system runs. Yet, even here, a better initial analytical plant model results in better control and disturbance rejection.

Nonlinear dynamic inversion may be computationally intensive and precise dynamic models may not be available, so [5] uses two neural-network controllers to achieve feedback linearization and learning. One radial-basis-function neural network is trained off-line to invert the plant nonlinearities, and a second is trained on-line to compensate for inversion error. The use of neural networks reduces computational complexity of the inverse dynamics calculation and improves precision by learning. In [6], an alternate scheme is used where a feedforward inverse recurrent neural network is used with a feedback proportional-derivative controller to compensate for inversion error and to reject disturbances.

All of these methods have certain limitations. For example, they require that an inverse exist, so do not work for nonminimum-phase plants or for ones with differing numbers of inputs and outputs. They generally also require that a fairly precise model of the plant be known *a priori*. This paper instead considers a technique called *adaptive inverse control* [7]–[13]—as reformulated by Widrow and Walach [13]—which does not require a precise initial plant model. Like feedback linearization, adaptive inverse control is based on the concept of dynamic inversion, but an inverse need not exist. Rather, an adaptive element is trained to control the plant such that model-reference based control is achieved in a least-mean-squared error optimal sense. Control of plant dynamics and control of plant disturbance are treated separately, without compromise. Control of plant dynamics can be achieved by preceding the plant with an adaptive controller whose dynamics are a type of inverse of those of the plant. Control of plant disturbance can be achieved by an adaptive feedback process that minimizes plant output disturbance without altering plant dynamics. The adaptive controllers are implemented using nonlinear adaptive filters. Nonminimum-phase and nonsquare plants may be controlled with this method.

Fig. 1 shows a block-diagram of an adaptive inverse control system. The system we wish to control is labeled the plant. It is subject to disturbances, and these are modeled additively at the plant output, without loss of generality. To control the plant, we first adapt a plant model $\widehat{P}$ using adaptive system-identification techniques. Second, the dynamic response of the system is controlled using an adaptive controller C. The output of the plant model is compared to the measured plant output and the differ-

Manuscript received December 5, 2001; revised October 2, 2002. This work was supported in part by the National Science Foundation under Contract ECS-9522085 and the Electric Power Research Institute under Contract WO8016-17.

The author is with the Department of Electrical and Computer Engineering, University of Colorado at Colorado Springs, Colorado Springs, CO 80933-7150 USA (e-mail: glp@eas.uccs.edu).

Digital Object Identifier 10.1109/TNN.2003.809412

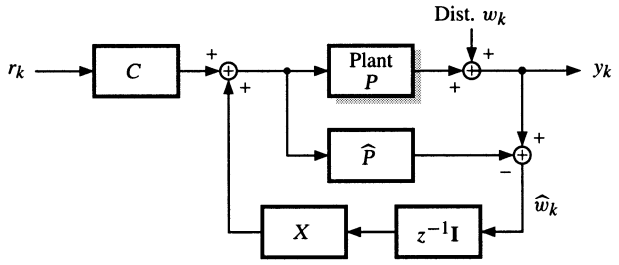


Fig. 1. Adaptive inverse control system.

ence is a good estimate of the disturbance. A special adaptive filter X is used to cancel the disturbances.

Control of linear systems require linear adaptive-filtering methods and control of nonlinear systems require nonlinear adaptive-filtering methods. However, even if the plant is linear, nonlinear methods will yield better results than linear methods if there are nonquadratic constraints on the control effort, or if the disturbance is generated from a nonlinear or chaotic source. Therefore, we focus on nonlinear adaptive inverse control (of linear or nonlinear plants) using nonlinear adaptive filters. We proceed by first reviewing nonlinear adaptive filtering and system identification. Next, we discuss adaptive inverse control of plant dynamics and adaptive disturbance canceling. We conclude with simulation examples to demonstrate the techniques.

II. Dynamic Neural Networks

A. Structure of a Dynamic Neural Network

An adaptive filter has an input x_k, an output y_k, and a "special input" d_k called the desired response. The filter computes a dynamical function of its input, and the desired response specifies the output we wish the filter to have at that point in time. It is used to modify the internal parameters of the filter in such a way that the filter "learns" to perform a certain function.

A nonlinear adaptive filter computes a nonlinear dynamical function of its input. It has a tapped delay line connected to its input and, possibly, a tapped delay line connected to its output. The output of the filter is computed to be a nonlinear function of these delayed inputs and outputs. The nonlinear function may be implemented in any way, but here we use a layered feedforward neural network.

A neural network is an interconnected set of very simple processing elements called neurons. Each neuron computes an internal sum that is equal to a constant plus the weighted sum of its inputs. The neuron outputs a nonlinear "activation" function of this sum. In this work, the activation function for nonoutput neurons is chosen to be the $\tanh(\cdot)$ function and all output neurons have the nonlinear function removed. This is done to give them unrestricted range.

Neural networks may be constructed from individual neurons connected in very general ways. However, it is sufficient to have *layers* of neurons, where the inputs to each neuron on a layer are identical and equal to the collection of outputs from the previous layer (plus the augmented *bias* value "1"). The final layer of the network is called the *output layer* and all other layers of neurons are called *hidden layers*. A layered network is a feedforward (nonrecurrent) structure that computes a static nonlinear function. Dynamics are introduced via the tapped delay lines at the input to the network, resulting in a *dynamic neural network.*

This layered structure also makes it easy to compactly describe the topology of a nonlinear filter. The following notation is used: $\mathcal{N}_{(a,\,b):\alpha:\beta:\ldots}$. This means: "The filter input is comprised of a tapped delay line with "a" delayed copies of the exogenous input vector x_k, and "b" delayed copies of the output vector y_k. Furthermore, there are "α" neurons in the neural network's first layer of neurons, "β"' neurons in the second layer, and so on." Occasionally, filters are encountered with more than one exogenous input. In that case, the "a" parameter is a row vector describing how many delayed copies of each input are used in the input vector to the network. To avoid confusion, any strictly feedforward nonlinear filter is denoted explicitly with a zero in that part of its description. For example, $\mathcal{N}_{(2,\,0):3:3:1}$.

A dynamic neural network of this type is called a Nonlinear AutoRegressive eXogenous input (NARX) filter. It is general enough to approximate any nonlinear dynamical system [14]. A nonlinear single-input–single-output (SISO) filter is created if x_k and y_k are scalars, and the network has a single output neuron. A nonlinear multiple-input–multiple-output (MIMO) filter is constructed by allowing x_k and y_k to be (column) vectors, and by augmenting the output layer with the correct number of additional neurons. Additionally, notice that since the output layer of neurons is linear, a neural network with a single layer of neurons is a linear adaptive filter (if the bias weights of the neurons are zero). Therefore, the results of this paper—derived for the general NARX structure—can encompass the situation where linear adaptive filters are desired if a degenerate neural-network structure is used.

B. Adapting Dynamic Neural Networks

A feedforward neural network is one whose input contains no self-feedback of its previous outputs. Its weights may be adapted using the popular *backpropagation algorithm*, discovered independently by several researchers [15], [16] and popularized by Rumelhart *et al.* [17]. An NARX filter, on the other hand, generally has self-feedback and must be adapted using a method such as *real-time recurrent learning* (RTRL) [18] or *backpropagation through time* (BPTT) [19]. Although compelling arguments may be made supporting either algorithm [20], we have chosen to use RTRL in this work, since it easily generalizes as is required later and is able to adapt filters used in an implementation of adaptive inverse control in real time.

We assume that the reader is familiar with neural networks and the backpropagation algorithm. If not, the tutorial paper [21] is an excellent place to start. Briefly, the backpropagation algorithm adapts the weights of a neural network using a simple optimization method known as gradient descent. That is, the change in the weights ΔW_k is calculated as

$$\Delta W_k^T = -\eta \frac{dJ_k}{dW}$$

where J_k is a cost function to be minimized and the small positive constant η is called the *learning rate*. Often, $J_k = (1/2)\,\mathbb{E}[e_k^T e_k]$ where $e_k = d_k - y_k$ is the network error computed as the desired response minus the actual neural-network output. A stochastic approximation to J_k may be made as $J_k \approx (1/2)\,e_k^T e_k$, which results in

$$\frac{dJ_k}{dW} = -e_k^T \frac{dy_k}{dW}.$$

For a feedforward neural network, $dy_k/dW = \partial y_k/\partial W$, which may be verified using the chain rule for total derivatives. The backpropagation algorithm is an elegant way of recursively computing $\partial J_k/\partial W$ by "backpropagating" the vector $-e_k$ from the output of the neural network back through the network to its first layer of neurons. The values that are computed are multiplied by $-\eta$ and are used to adapt the neuron's weights.

It is important for this work to notice that the backpropagation algorithm may also be used to compute the vector $v^T \partial y_k/\partial W$ (where v is an arbitrary vector) by simply backpropagating the vector v instead of the vector $-e_k$. Specifically, we will need to compute *Jacobian matrices* of the neural network. These Jacobians are $\partial y_k/\partial W$ and $\partial y_k/\partial X$, where X is the composite input vector to the neural network (containing all tap-delay-line copies of the exogenous and feedback inputs). If we backpropagate a vector v through the neural network, then we have computed the entries of the vector $v^T \partial y_k/\partial W$. If v is chosen to be a unit vector, then we have computed one row of the matrix $\partial y_k/\partial W$. By backpropagating as many unit vectors as there are outputs to the network, we may compose the Jacobian $\partial y_k/\partial W$ one row at a time. Also, if we backpropagate the vector v past the first layer of neurons to the inputs to the network itself, we have computed $v^T \partial y_k/\partial X$. Again, using the methodology of backpropagating unit vectors, we may simultaneously build up the Jacobian matrix $\partial y_k/\partial X$ one row at a time. Werbos called this ability of the backpropagation algorithm the "dual-subroutine." The primary subroutine is the feedforward aspect of the network. The dual subroutine is the recursive calculation of the Jacobians of the network.

Now that we have seen how to adapt a feedforward neural network and how to compute Jacobians of a neural network, it is a simple matter to extend the backpropagation algorithm to adapt NARX filters. This was first done by Williams and Zipser [18] and called real-time recurrent learning (RTRL). A similar presentation follows.

An NARX filter computes a function of the following form:

$$y_k = f(x_k, x_{k-1}, \ldots, x_{k-n}, y_{k-1}, y_{k-2}, \ldots, y_{k-m}, W).$$

To adapt using the familiar "sum of squared error" cost function, we need to be able to calculate

$$\frac{d\frac{1}{2}\|e_k\|^2}{dW} = -e_k^T \frac{dy_k}{dW}$$

$$\frac{dy_k}{dW} = \frac{\partial y_k}{\partial W} + \sum_{i=0}^{n} \frac{\partial y_k}{\partial x_{k-i}} \frac{dx_{k-i}}{dW} + \sum_{i=1}^{m} \frac{\partial y_k}{\partial y_{k-i}} \frac{dy_{k-i}}{dW}. \quad (1)$$

The first term $\partial y_k/\partial W$ is the direct effect of a change in the weights on y_k, and is one of the Jacobians calculated by the dual-subroutine of the backpropagation algorithm. The second term is zero, since dx_k/dW is zero for all k. The final term may be broken up into two parts. The first, $\partial y_k/\partial y_{k-i}$, is a component of the matrix $\partial y_k/\partial X$, as delayed versions of y_k are part of the network's input vector X. The dual-subroutine algorithm may be used to compute this. The second part, dy_{k-i}/dW, is simply a previously calculated and stored value of dy_k/dW. When the system is "turned on," dy_i/dW are set to zero for $i = 0, -1, -2, \ldots$, and the rest of the terms are calculated recursively from that point on.

Note that the dual-subroutine procedure naturally calculates the Jacobians in such a way that the weight update is done with simple matrix multiplication. Let

$$(d_w y)_k \triangleq \left[\left(\frac{dy_{k-1}}{dW}\right)^T \left(\frac{dy_{k-2}}{dW}\right)^T \cdots \left(\frac{dy_{k-m}}{dW}\right)^T\right]^T$$

and

$$(d_x y)_k \triangleq \left[\left(\frac{\partial y_k}{\partial y_{k-1}}\right) \left(\frac{\partial y_k}{\partial y_{k-2}}\right) \cdots \left(\frac{\partial y_k}{\partial y_{k-m}}\right)\right].$$

The latter is simply the columns of $\partial y_k/\partial X$ corresponding to the feedback inputs to the network, and is directly calculated by the dual subroutine. Then, the weight update is efficiently calculated as

$$\Delta W_k = \left(\eta e_k^T \left[\frac{\partial y_k}{\partial W} + (d_x y)_k (d_w y)_k\right]\right)^T.$$

C. Optimal Solution for a Nonlinear Adaptive Filter

In *principle*, a neural network can emulate a very general nonlinear function. It has been shown that any "smooth" static nonlinear function may be approximated by a two-layer neural network with a "sufficient" number of neurons in its hidden layer [22]. Furthermore, a NARX filter can compute any dynamical finite-state machine (it can emulate any computer with finite memory) [14].

In *practice*, a neural network seldom achieves its full potential. Gradient-descent based training algorithms converge to a local minimum in the solution space, and not to the global minimum. However, it is instructive to exactly determine the optimal performance that could be expected from any nonlinear system, and then to use it as a lower bound on the mean-square error (MSE) of a trained neural network. Generally, a neural network will get quite close to this bound.

The optimal solution for a nonlinear filter is from reference [23, Th. 4.2.1]. If the composite input vector to the adaptive filter is X_k, the output is y_k, and the desired response is d_k, then the optimal filter computes the conditional expectation

$$y_k = \mathbb{E}[d_k | X_k].$$

D. Practical Issues

While adaptation may proceed on-line, there is neither mathematical guarantee of stability of adaptation nor of convergence of the weights of a neural network. In practice, we find that if η is "small enough," the weights of the adaptive filter converge in a stable way, with a small amount of misadustment. It may be wise to adapt the dynamic neural network off-line (using premeasured data) at least until close to a solution, and

then use a small value of η to continue adapting on-line. Speed of adaptation is also an issue. The reader may wish to consider faster second-order adaptation methods such as dynamically decoupled extended Kalman filter (DDEKF) learning [24], where the gradient matrix $H(k) = dy_k^T/dW$, as computed above. We have found order-of-magnitude improvement in speed of adaptation using DDEKF versus RTRL [25].

III. Adaptive System Identification

The first step in performing adaptive inverse control is to make an adaptive model of the plant. The model should capture the dynamics of the plant well enough that a controller designed to control the plant model will also control the plant very well. This is a straightforward application of the adaptive filtering techniques in Section II.

A method for adaptive plant modeling is depicted in Fig. 2(a). The plant is excited with the signal u_k, and the disturbed output y_k is measured. The plant model $\widehat{P}$ is also excited with u_k, and its output $\hat{y}_k$ is computed. The plant modeling error $e_k^{(mod)} = y_k - \hat{y}_k$ is used by the adaptation algorithm to update the weight values of the adaptive filter.

NARX models have implicit feedback of delayed versions of their output to the input of the model. This feedback is assumed in all block diagrams, and is not drawn explicitly, except in Fig. 2(b). The purpose of Fig. 2(b) is to show that this feedback, when training an adaptive plant model, may be connected to either the model output $\hat{y}_k$ or the plant output y_k. The first method is called a *parallel* connection for system identification, and the second method is called a *series-parallel* connection for system identification. Networks configured in series-parallel may be trained using the standard backpropagation algorithm. Networks configured in parallel must be trained with either RTRL or BPTT. The series-parallel configuration is simple, but is biased by disturbance. The parallel configuration is more complex to train, but is unbiased by disturbance. In this work, nonlinear system identification is first performed using the series-parallel configuration to initialize weight values of the plant model. When the weight values converge, the plant model is reconfigured in the parallel configuration and training is allowed to continue. This procedure allows speedy training of the network, but is not compromised by disturbance.

We now confirm that the adaptive plant model $\widehat{P}$ converges to P. From Section II-C we can find the optimal solution for $\widehat{P}$

$$\begin{aligned}\widehat{P}^{(opt)}(\vec{u}_k) &= \mathbb{E}[y_k|\vec{u}_k] \\ &= \mathbb{E}[P(\vec{u}_k) + w_k|\vec{u}_k] \\ &= \mathbb{E}[P(\vec{u}_k)|\vec{u}_k] + \mathbb{E}[w_k|\vec{u}_k] \\ &= P(\vec{u}_k) + \mathbb{E}[w_k] \\ &= P(\vec{u}_k)\end{aligned}$$

where $\vec{u}_k$ is a vector containing past values of the control signal $u_k, u_{k-1}, \ldots, u_0$ and where two assumptions were made: 1) in the fourth line, we assume that the disturbance is statistically independent of the command input signal u_k and 2) in the final line, we assume that the disturbance is zero mean. Under these two assumptions, the plant model converges to the plant despite the presence of disturbance.

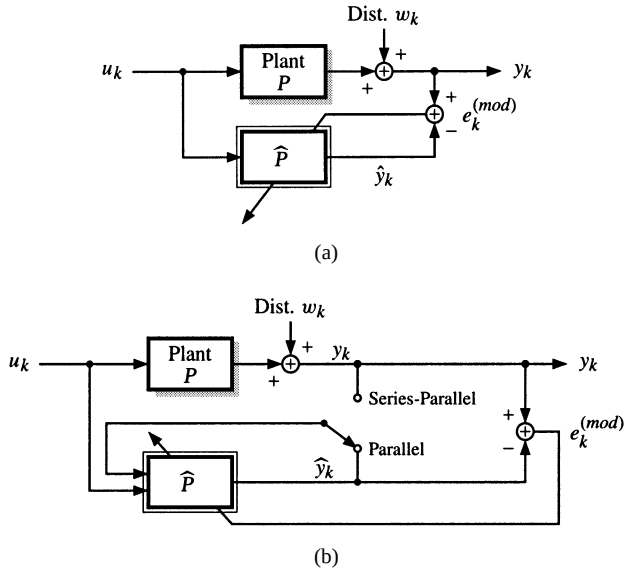


Fig. 2. Adaptive plant modeling. (a) Simplified and (b) detailed depictions.

A final comment should be made regarding the relative timing of the u_k and y_k signals. In order to be able to model either strictly- or nonstrictly-proper plants, we output u_k at time $t = (kT)^-$ and measure y_k at time $t = (kT)^+$, T being the sampling period. We will see that this assumption can limit the extent to which we can effectively cancel disturbance. If we know *a priori* that the plant is strictly proper, then we may instead measure y_k at time $t = (kT)^-$ and use its value when computing u_k (which is output at time $t = (kT)^+$), since we know that there is no immediate effect on the plant output due to u_k.

IV. Adaptive Feedforward Control of Plant Dynamics

To perform adaptive inverse control, we need to be able to adapt the three filters of Fig. 1: the plant model $\widehat{P}$, the controller C, and the disturbance canceler X. We have seen how to adapt $\widehat{P}$ to make a plant model. For the time being we set aside consideration of the disturbance canceling filter X and concentrate on the design of the feedforward controller C.[1] The goal is to make the dynamics of the controlled system PC approximate the fixed filter M as closely as possible, where M is a user-specified *reference model*. The input reference signal r_k is filtered through M to create a desired response d_k for the plant output. The measured plant output is compared with the desired plant output to create a system error signal $e_k^{(sys)} = d_k - y_k$. We will adapt C to minimize the mean-squared system error while constraining the control effort u_k.

The reference model M may be designed in a number of ways. Following traditions of control theory, we might design M to have the step response of a second-order linear system

[1]We shall restrict our development to apply only to stable plants. If the plant of interest is unstable, conventional feedback should be applied to stabilize it. Then the combination of the plant and its feedback stabilizer can be regarded as an equivalent stable plant.

that meets design specifications. However, we can often achieve even better tracking control if we let M be simply a delay corresponding to the transport delay of the plant. The controller C will adapt to a delayed inverse of the plant dynamics.[2]

A. Adapting a Constrained Controller Via the BPTM Algorithm

Recall that an adaptive filter requires a desired response input signal in order to be able to adapt its weights. While we do have a desired response for the entire system, d_k, we do not have a desired response for the output of the controller C. A key hurdle that must be overcome by the algorithm is to find a mechanism for converting the system error to an adaptation signal used to adjust C.

One solution is to regard the series combination of C and $\widehat{P}$ as a single adaptive filter. There *is* a desired response for this combined filter: d_k. Therefore, we can use d_k to *compute* weight updates for the conglomerate filter. *However, we only apply the weight updates to the weights in* C; $\widehat{P}$ is still updated using the plant modeling error $e_k^{(\mathrm{mod})}$. Fig. 3 shows the general framework to be used. We say that the system error is *backpropagated* through the plant model, and used to adapt the controller. For this reason, the algorithm is named "backprop through (plant) model" (BPTM). This solution allows for control of nonminimum-phase and nonsquare plants.

The algorithm is derived as follows. We wish to train the controller C to minimize the squared system error and to simultaneously minimize some function of the control effort. We construct the following cost function that we will minimize

$$J_k = \mathbb{E}\left[\tfrac{1}{2}\, e_k^{(\mathrm{sys})\mathrm{T}} Q e_k^{(\mathrm{sys})} + h\left(u_k,\, u_{k-1},\, \ldots,\, u_{k-r}\right)\right].$$

The differentiable function $h(\cdot)$ defines the cost function associated directly with the control signal u_k and is used to penalize excessive control effort, slew rate, and so forth. The system error is the signal $e_k^{(\mathrm{sys})} = d_k - y_k$, and the symmetric matrix Q is a weighting matrix that assigns different performance objectives to each plant output.

If we let $g(\cdot)$ be the function implemented by the controller C, and $f(\cdot)$ be the function implemented by the plant model $\widehat{P}$, we can state without loss of generality

$$\begin{aligned} u_k &= g\left(u_{k-1},\, u_{k-2} \ldots u_{k-m},\, r_k,\, r_{k-1} \ldots r_{k-q},\, W_C\right) \\ y_k &\approx \hat{y}_k = f\left(\hat{y}_{k-1},\, \hat{y}_{k-2} \ldots \hat{y}_{k-n},\, u_k,\, u_{k-1} \ldots u_{k-p}\right) \end{aligned} \quad (2)$$

where W_C are the adjustable parameters (weights) of the controller.

[2]We note that nonlinear plants do not, in general, have inverses. So, when we say that C adapts to a (delayed) inverse of the plant dynamics, we mean that C adapts so that the cascade PC approximates the (delayed) identity function as well as possible in the least-mean-squared error sense.

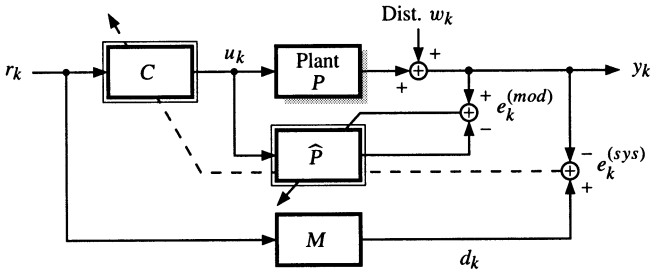


Fig. 3. Structure diagram illustrating the BPTM method.

The controller weights are updated in the direction of the negative gradient of the cost functional

$$\begin{aligned} (\Delta W_C)_k^T &= -\eta \frac{dJ_k}{dW_C} \\ &= -\eta \frac{d}{dW_C}\left\{ e_k^{(\mathrm{sys})\mathrm{T}} Q e_k^{(\mathrm{sys})} \right. \\ &\qquad \left. + h\left(u_k,\, u_{k-1},\, \ldots,\, u_{k-r}\right)\right\} \end{aligned}$$

where η is the adaptive learning rate. Continuing

$$\begin{aligned} \frac{(\Delta W_C)_k^T}{\eta} &= e_k^T Q \frac{d\hat{y}_k}{dW_C} \\ &\quad - \sum_{j=0}^{r} \left(\frac{\partial h\left(u_k,\, \ldots,\, u_{k-r}\right)}{\partial u_{k-j}}\right)^T \left(\frac{du_{k-j}}{dW_C}\right). \end{aligned}$$

Using (2) and the chain rule for total derivatives, two further substitutions may be made at this time

$$\frac{du_k}{dW_C} = \frac{\partial u_k}{\partial W_C} + \sum_{j=1}^{m} \left(\frac{\partial u_k}{\partial u_{k-j}}\right)\left(\frac{du_{k-j}}{dW_C}\right) \quad (3)$$

$$\begin{aligned} \frac{d\hat{y}_k}{dW_C} &= \sum_{j=0}^{p} \left(\frac{\partial \hat{y}_k}{\partial u_{k-j}}\right)\left(\frac{du_{k-j}}{dW_C}\right) \\ &\quad + \sum_{j=1}^{n} \left(\frac{\partial \hat{y}_k}{\partial \hat{y}_{k-j}}\right)\left(\frac{d\hat{y}_{k-j}}{dW_C}\right). \end{aligned} \quad (4)$$

With these definitions, we may find the weight update. We need to find three quantities: $\partial h(\cdot)/\partial u_{k-j}$, du_k/dW_C and $d\hat{y}_k/dW_C$.

First, we note that $\partial h(\cdot)/\partial u_{k-j}$ depends on the user-specified function $h(\cdot)$. It can be calculated given $h(\cdot)$. Second, we consider du_k/dW_C as expanded in (3). It is the same in form as (1) and is computed the same way.

Third, we consider $d\hat{y}_k/dW_C$ in (4). The first term in the first summation is the Jacobian $\partial \hat{y}_k/\partial u_{k-j}$, which may be computed via the backpropagation algorithm. The next term, du_{k-j}/dW is the current or a previously computed and saved version of du_k/dW_C, computed via (3). The first term in the

begin {Adapt C}

Update du_k/dW_C:

- Shift dU_k down N_i rows, where N_i is the number of plant inputs.
- Backpropagate N_i unit vectors through C to form $\partial_U U_k$ and $\partial u_k/\partial W_C$. Each backpropagation produces one row of both matrices.
- Compute top N_i rows of dU_k to be $\partial u_k/\partial W_C + (\partial_U U_k)(dU_k)$.

Update dy_k/dW_C:

- Backpropagate N_o unit vectors through $\widehat{P}$ to form $\partial_U Y_k$ and $\partial_Y Y_k$, where N_o is the number of plant outputs. Each backpropagation produces one row of both matrices.
- Compute $d_W Y_k = (\partial_U Y_k)(dU_k) + (\partial_Y Y_k)(dY_k)$.
- Shift dY_k down N_o rows and save $d_W Y_k$ in the top N_o rows.

Compute dH_k.

Update Weights:

- Compute $(\Delta W_C)_k^T = \eta e_k^T Q(d_W Y_k) - \eta(dH_k^T)(dU_k)$.
- Adapt, enumerating weights in the same order as when computing $\partial_W U_k$.

end {Adapt C}

Fig. 4. Algorithm to adapt a NARX controller for a NARX plant model.

second summation, $\partial\hat{y}_k/\partial\hat{y}_{k-j}$ is another Jacobian. The final term, $d\hat{y}_{k-j}/dW_C$, is a previously computed and saved version of $d\hat{y}_k/dW_C$.

A practical implementation is realized by compacting the notation into a collection of matrices. We define

$$dU_k \triangleq \left[\left(\frac{\partial u_k}{\partial W_C}\right)^T \left(\frac{\partial u_{k-1}}{\partial W_C}\right)^T \cdots \left(\frac{\partial u_{k-p}}{\partial W_C}\right)^T\right]^T$$

$$dH_k \triangleq \left[\left(\frac{\partial h(\cdot)}{\partial u_k}\right)^T \left(\frac{\partial h(\cdot)}{\partial u_{k-1}}\right)^T \cdots \left(\frac{\partial h(\cdot)}{\partial u_{k-r}}\right)^T\right]^T$$

$$dY_k \triangleq \left[\left(\frac{\partial y_{k-1}}{\partial W_C}\right)^T \left(\frac{\partial y_{k-2}}{\partial W_C}\right)^T \cdots \left(\frac{\partial y_{k-n}}{\partial W_C}\right)^T\right]^T$$

$$\partial_U Y_k \triangleq \left[\left(\frac{\partial y_k}{\partial u_k}\right)^T \left(\frac{\partial y_k}{\partial u_{k-1}}\right)^T \cdots \left(\frac{\partial y_k}{\partial u_{k-p}}\right)^T\right]^T$$

$$\partial_Y Y_k \triangleq \left[\left(\frac{\partial y_k}{\partial y_{k-1}}\right)^T \left(\frac{\partial y_k}{\partial y_{k-2}}\right)^T \cdots \left(\frac{\partial y_k}{\partial y_{k-n}}\right)^T\right]^T$$

$$\partial_U U_k \triangleq \left[\left(\frac{\partial u_k}{\partial u_{k-1}}\right)^T \left(\frac{\partial u_k}{\partial u_{k-2}}\right)^T \cdots \left(\frac{\partial u_k}{\partial u_{k-p}}\right)^T\right]^T.$$

The algorithm is summarized in Fig. 4. Any programming language supporting matrix mathematics can very easily implement this algorithm. It works well.

A similar algorithm may be developed for linear systems, for which convergence and stability has been proven [7]. We have no mathematical proof of stability for the nonlinear version, although we have observed it to be stable if η is "small enough." As a practical matter, it may be wise to adapt the controller off-line until convergence is approximated. Adaptation may then continue on-line, with a small value of η. Faster adaptation may be accomplished using DDEKF as the adaptation mechanism (BPTM-DDEKF) versus using RTRL (BPTM-RTRL). To do so, set $H(k) = dJ_k^T/dW_C$ [25].

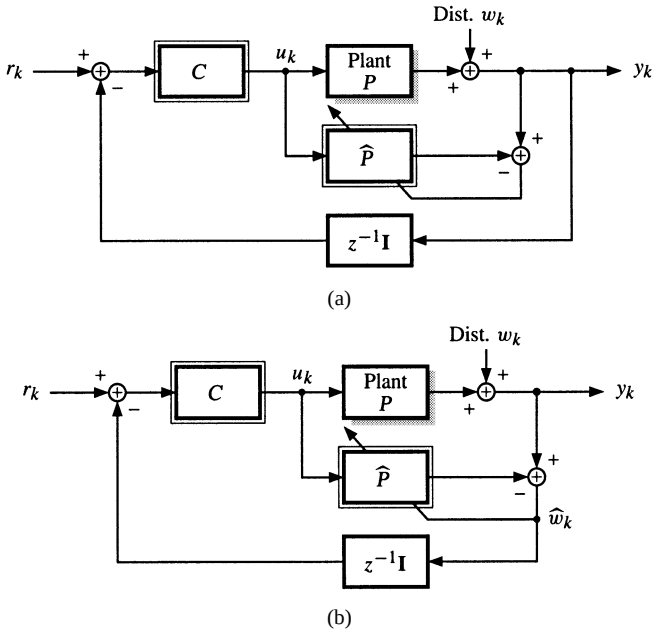


Fig. 5. Two methods to close the loop. (a) The output, y_k, is fed back to the controller. (b) An estimate of the disturbance, $\hat{w}_k$, is fed back to the controller.

V. Adaptive Disturbance Canceling

Referring again to Fig. 1, we have seen how to adaptively model the plant dynamics, and how to use the plant model to adapt a feedforward adaptive controller. Using this controller, an undisturbed plant will very closely track the desired output. A disturbed plant, on the other hand, may not track the desired output at all well. It remains to determine what can be done to eliminate plant disturbance.

The first idea that may come to mind is to simply "close the loop." Two methods commonly used to do this are discussed in Section V-A. Unfortunately, neither of these methods is appropriate if an adaptive controller is being designed. Closing the loop will cause the controller to adapt to a "biased" solution. An alternate technique is introduced that leads to the correct solution if the plant is linear and has substantially less bias if the plant is nonlinear.

A. Conventional Disturbance Rejection Methods Fail

Two approaches to disturbance rejection commonly seen in the literature are shown in Fig. 5. Both methods use feedback; the first feeds back the disturbed plant output y_k, as in Fig. 5(a), and the second feeds back an estimate of the disturbance $\hat{w}_k$, as in Fig. 5(b). The approach shown in Fig. 5(a) is more conventional, but is difficult to use with adaptive inverse control since the dynamics of the closed-loop system are dramatically different from the dynamics of the open-loop system. Different methods than those presented here are required to adapt C. The approach shown in Fig. 5(b) is called *internal model control* [26]–[31]. The benefit of using this scheme is that the dynamics of the closed-loop system are equal to the dynamics of the open-loop system if the plant model is identical to the plant. Therefore, the methods found to adapt the controller for feedforward control may be used directly.

Unfortunately, closing the loop using *either* method in Fig. 5 will cause $\widehat{P}$ to adapt to an incorrect solution. In the following analysis the case of a plant controlled with internal model control is considered. A similar analysis may be performed for the conventional feedback system in Fig. 5(a), with the same conclusion.

B. Least-Mean-Squared Error Solution for $\widehat{P}$ Using Internal Model Control

When the loop is closed as in Fig. 5(b), the estimated disturbance term $\hat{w}_{k-1}$ is subtracted from the reference input r_k. The resulting composite signal is filtered by the controller C and becomes the plant input signal, u_k. In the analysis done so far, we have assumed that u_k is independent of w_k, but that assumption is no longer valid. We need to revisit the analysis performed for system identification to see if the plant model $\widehat{P}$ still converges to P.

The direct approach to the problem is to calculate the least-mean-squared error solution for $\widehat{P}$ and see if it is equal to P. However, due to the feedback loop involved, it is not possible to obtain a closed form solution for $\widehat{P}$. An indirect approach is taken here. We do not need to know exactly to what $\widehat{P}$ converges—we only need to know whether or not it converges to P. The following indirect procedure is used.

1) First, remove the feedback path (open the loop) and perform on-line plant modeling and controller adaptation. When convergence is reached, $\widehat{P} \rightarrow P$.
2) At this time, $\widehat{P} \approx P$, and the disturbance estimate $\hat{w}_k$ is very good. We assume that $\hat{w}_k = w_k$. This assumption allows us to construct a feedforward-only system that is equivalent to the internal-model-control feedback system by substituting w_k for $\hat{w}_k$.
3) Finally, analyze the system, substituting w_k for $\hat{w}_k$. If the least-mean-squared error solution for $\widehat{P}$ still converges to P, then the assumption made in the second step remains valid, and the plant is being modeled correctly. If $\widehat{P}$ diverges from P with the assumption made in step 2), then it cannot converge to P in a closed-loop setting. We conclude that the assumption is justified for the purpose of checking for proper convergence of $\widehat{P}$.

We now apply this procedure to analyze the system of Fig. 5(b). We first open the loop and allow the plant model to converge to the plant. Second, we assume that $\hat{w}_k \approx w_k$. Finally, we compute the least-mean squared error solution for $\widehat{P}$

$$\begin{aligned}\widehat{P}^{(\mathrm{opt})}(\vec{u}_k) &= \mathbb{E}\left[y_k \,|\vec{u}_k\right] \\ &= \mathbb{E}\left[P(\vec{u}_k) + w_k \,|\vec{u}_k\right] \\ &= \mathbb{E}\left[P(\vec{u}_k)\,|\vec{u}_k\right] + \mathbb{E}\left[w_k \,|\vec{u}_k\right] \\ &= P(\vec{u}_k) + \mathbb{E}\left[w_k \,|\vec{u}_k\right]\end{aligned}$$

where $\vec{u}_k$ is a function of w_k since $u_k = C(\vec{r}_k - \vec{w}_{k-1})$ and so the conditional expectation in the last line is not zero. The plant model is biased by the disturbance.

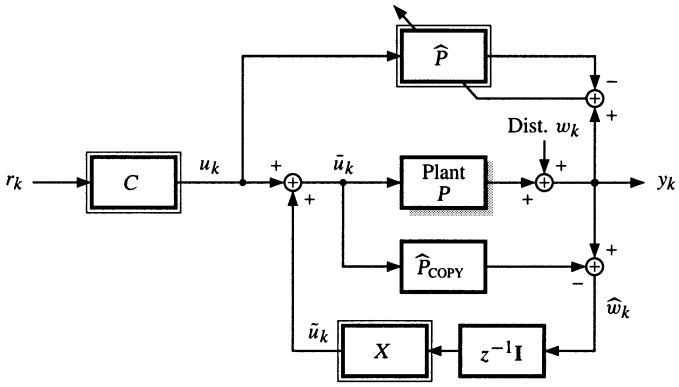


Fig. 6. Correct on-line adaptive plant modeling in conjunction with disturbance canceling for linear plants. The circuitry for adapting C has been omitted for clarity.

Assumptions concerning the nature of the r_k and w_k signals are needed to simplify this further. For example, if w_k is white, then the plant model converges to the plant. Under almost all other conditions, however, the plant model converges to something else. In general, an adaptive plant model made using the internal model control scheme will be biased by disturbance.

The bottom line is that if the loop is closed and the plant model is allowed to continue adapting after the loop is closed, the overall control system will become biased by the disturbance. One solution, applicable to linear plants, is to perform plant modeling with dither signals rather than with the command input signal u_k [13, schemes B, C]. However, this will increase the output noise level. A better solution is presented next, in which the plant model is allowed to continue adaptation and the output noise level is not increased. The plant disturbances will be handled by a separate circuit from the one handling the task of dynamic response. This results in an overall control problem that is partitioned in a very nice way.

C. A Solution Allowing On-Line Adaptation of $\widehat{P}$

The only means at our disposal to cancel disturbance is through the plant input signal. This signal must be computed in such a way that the plant output negates (as much as possible) the disturbance. Therefore, the *plant* input signal must be statistically dependent on the disturbance. However, we have just seen that the *plant model* input signal cannot contain terms dependent on the disturbance or the plant model will be biased. This conundrum was first solved by Widrow and Walach [13], as shown in Fig. 6.

By studying the figure, we see that the control scheme is very similar to internal model control. The main difference is that the feedback loop is "moved" in such a way that the disturbance dynamics do not appear at the input to the adaptive plant model, but do appear at the input to the plant. That is, the controller output u_k is used as input to the adaptive plant model $\widehat{P}$; on the other hand, the input to the plant is equal to $u_k + \tilde{u}_k$, where $\tilde{u}_k$ is the output of a special disturbance-canceling filter, X. $\widehat{P}$ is not used directly to estimate the disturbance; rather, a filter whose weights are a digital copy of those in $\widehat{P}$ is used to estimate the disturbance. This filter is called $\widehat{P}_{\mathrm{COPY}}$.

In later sections, we will see how to adapt X and how to modify the diagram if the plant is nonlinear. Now, we proceed to show that the design is correct if the plant is linear. We compute the optimal function $\widehat{P}$

$$\begin{aligned}\widehat{P}^{(\mathrm{opt})}(\vec{u}_k) &= \mathbb{E}\left[y_k \,|\vec{u}_k\right] \\ &= \mathbb{E}\left[P\left(\vec{u}_k + \vec{\tilde{u}}_k\right) + w_k \,|\vec{u}_k\right] \\ &= \mathbb{E}\left[P\left(\vec{u}_k + \vec{\tilde{u}}_k\right)|\vec{u}_k\right] + \mathbb{E}\left[w_k \,|\vec{u}_k\right] \\ &= \mathbb{E}\left[P\left(\vec{u}_k + \vec{\tilde{u}}_k\right)|\vec{u}_k\right] \\ &= \mathbb{E}\left[P\left(\vec{u}_k\right)|\vec{u}_k\right] + \mathbb{E}\left[P\left(\vec{\tilde{u}}_k\right)|\vec{u}_k\right] \\ &= P\left(\vec{u}_k\right)\end{aligned}$$

where $\vec{u}_k$ is a vector containing past values of the control signal u_k, u_{k-1}, …, u_0, and $\vec{\tilde{u}}_k$ is a vector containing past values of the disturbance canceler output, $\tilde{u}_k$, $\tilde{u}_{k-1}$, …, $\tilde{u}_0$. In the final line, we assume that $\vec{\tilde{u}}_k$ is zero-mean (which it would be if w_k is zero mean and the plant is linear).

Unfortunately, there is still a bias if the plant is nonlinear. The derivation is the same until the fourth line

$$\widehat{P}^{(\mathrm{opt})}(\vec{u}_k) = \mathbb{E}\left[P\left(\vec{u}_k + \vec{\tilde{u}}_k\right)|\vec{u}_k\right]$$

which does not break up as it did for the linear plant. To proceed further than this, we must make an assumption about the plant. Considering a SISO plant, for example, if we assume that P: $\mathbb{R}^k \rightarrow \mathbb{R}$ is differentiable at "point" $\vec{u}_k$, then we may write the first-order Taylor expansion [32, Th. 1]

$$P\left(\vec{u}_k + \vec{\tilde{u}}_k\right) = P\left(\vec{u}_k\right) + \sum_{i=0}^{k} \tilde{u}_i \frac{\partial P}{\partial \overline{u}_i}\left(\vec{u}_k\right) + R_1\left(\vec{\tilde{u}}_k, \vec{u}_k\right)$$

where $\overline{u}_i$ is the plant input at time index i. Furthermore, we have the result that $R_1(\vec{\tilde{u}}_k, \vec{u}_k)/||\vec{\tilde{u}}_k|| \rightarrow 0$ as $\vec{\tilde{u}}_k \rightarrow 0$ in $\mathbb{R}^k$. Then

$$\begin{aligned}\widehat{P}^{(\mathrm{opt})}(\vec{u}_k) &= \mathbb{E}\left[P\left(\vec{u}_k\right) + \sum_{i=0}^{k} \tilde{u}_i \frac{\partial P}{\partial \overline{u}_i}\left(\vec{u}_k\right) + R_1(\vec{\tilde{u}}_k, \vec{u}_k)\middle|\vec{u}_k\right] \\ &= P\left(\vec{u}_k\right) + \mathbb{E}\left[\sum_{i=0}^{k} \tilde{u}_i \frac{\partial P}{\partial \overline{u}_i}\left(\vec{u}_k\right)\middle|\vec{u}_k\right] \\ &\quad + \mathbb{E}\left[R_1(\vec{\tilde{u}}_k, \vec{u}_k)\middle|\vec{u}_k\right].\end{aligned}$$

Since w_k is assumed to be independent of u_k, then $\vec{\tilde{u}}_k$ is independent of $\vec{u}_k$. We also assume that $\vec{\tilde{u}}_k$ is zero-mean

$$\begin{aligned}\widehat{P}^{(\mathrm{opt})}(\vec{u}_k) &= P\left(\vec{u}_k\right) + \sum_{i=0}^{k} \underbrace{\mathbb{E}\left[\tilde{u}_i\right]}_{=0} \frac{\partial P}{\partial \overline{u}_i}\left(\vec{u}_k\right) \\ &\quad + \mathbb{E}\left[R_1\left(\vec{\tilde{u}}_k, \vec{u}_k\right)\middle|\vec{u}_k\right] \\ &= P\left(\vec{u}_k\right) + \mathbb{E}\left[R_1\left(\vec{\tilde{u}}_k, \vec{u}_k\right)\middle|\vec{u}_k\right] \\ &\approx P\left(\vec{u}_k\right). \qquad (5)\end{aligned}$$

The linear term disappears, as we expect from our analysis for a linear plant. However, the remainder term is not eliminated. In *principle*, if the plant is very nonlinear, the plant model may be quite different from the plant. In *practice*, however, we have seen that this method seems to work well, and much better than an adaptive version of the internal-model control scheme.

So, we conclude that this system is not as good as desired if the plant is nonlinear. It may not be possible to perform on-line adaptive plant modeling while performing disturbance canceling while retaining an unbiased plant model. One solution might involve freezing the weights of the plant model for long periods of time, and scheduling shorter periods for adaptive plant modeling with the disturbance canceler turned off when requirements for output quality are not as stringent. Another solution suggests itself from (5). If the $\tilde{u}_k$ terms are kept small, the bias will disappear. We will see in Section V-E that X may be adapted using the BPTM algorithm. We can enforce constraints on the output of X using our knowledge from Section IV to ensure that $\tilde{u}_k$ remains small. Also, since the system *is* biased, we use u_k as an additional exogenous input to X to help X determine plant state to cancel disturbance.

D. Function of the Disturbance Canceler

It is interesting to consider the mathematical function that the disturbance canceling circuit must compute. A little careful thought in this direction leads to a great deal of insight. The analysis is precise if the plant is minimum phase (that is, if it has a stable, causal inverse), but is merely qualitative if the plant is nonminimum phase. The goal of this analysis is not to be quantitative, but rather to develop an understanding of the function performed by X.

A useful way of viewing the overall system is to consider that the control goal is for X to produce an output so that $y_k = M(\vec{r}_k)$. We can express y_k as

$$y_k = w_k + P\left(C\left(\vec{r}_k\right) + X\left(\vec{\tilde{w}}_{k-1}, \vec{u}_k\right)\right).$$

Note that X takes the optional input signal u_k. This signal is used when controlling nonlinear plants as it allows the disturbance canceler some knowledge of the plant state. It is not required if the plant is linear.

Next, we substitute $y_k = M(\vec{r}_k)$ and rearrange to solve for the desired response of X. We see that

$$\begin{aligned}X^{(\mathrm{opt})}\left(\vec{\tilde{w}}_{k-1}, \vec{u}_k\right) &= P^{-1}\left(M\left(\vec{r}_k\right) - \vec{w}_k\right) - C\left(\vec{r}_k\right) \\ &= P^{-1}\left(P\left(\vec{u}_k\right) - \vec{w}_k\right) - u_k\end{aligned}$$

assuming that the controller has adapted until $P(\vec{u}_k) \approx M(\vec{r}_k)$. The function of X is a deterministic combination of known (by adaptation) elements P and P^{-1}, but also of the unknown signal w_k. Because of the inherent delay in discrete-time systems, we only know $\hat{w}_{k-1}$ at any time, so w_k must be estimated from previous samples of $\hat{w}_{k-1}$, …, $\hat{w}_0$. Assuming that the adaptive plant model is perfect, and that the controller has been adapted to convergence, the internal structure of X is then shown in Fig. 7(a). The w_k signal is computed by estimating its value

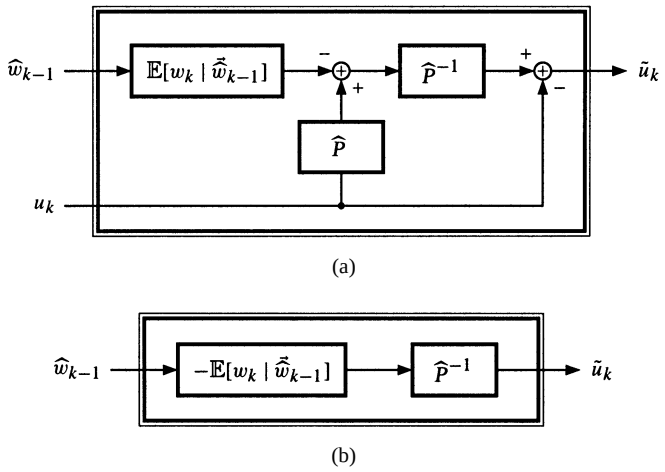


Fig. 7. Internal structure of X. (a) For a general plant. (b) For a linear plant.

from previous samples of $\hat{w}_k$. These are combined and passed through the plant inverse to compute the desired signal $\tilde{u}_k$.

Thus, we see that the disturbance canceler contains two parts. The first part is an estimator part that depends on the dynamics of the disturbance source. The second part is the canceler part that depends on the dynamics of the plant. The diagram simplifies for a linear plant since some of the circuitry cancels. Fig. 7(b) shows the structure of X for a linear plant.

One very important point to notice is that the disturbance canceler still depends on both the disturbance dynamics and the plant dynamics. If the process generating the disturbance is nonlinear or chaotic, then the estimator required will in general be a nonlinear function. The conclusion is that the disturbance canceler should be implemented as a nonlinear filter *even if the plant is a linear dynamical system*.

If the plant is generalized minimum phase (minimum phase with a constant delay), then this solution must be modified slightly. The plant inverse must be a delayed plant inverse, with the delay equal to the transport delay of the plant. The estimator must estimate the disturbance one time step into the future plus the delay of the plant. For example, if the plant is strictly proper, there will be at least one delay in the plant's impulse response, and the estimator must predict the disturbance at least two time steps into the future.[3]

It was stated earlier that these results are heuristic and do not directly apply if the plant is nonminimum phase. We can see this easily now, since a plant inverse does not exist. However, the results still hold qualitatively since a delayed approximate inverse exists; the solution for X is similar to the one for a generalized minimum phase plant. The structure of X consists of a part depending on the dynamics of the system, which amounts to a delayed plant inverse, and a part that depends on the dynamics of the disturbance generating source, which must now predict farther into the future than a single time step. Unlike the case of the generalized minimum phase plant, however, these two parts do not necessarily separate. That is, X implements some combination of predictors and delayed inverses that compute the least-mean-squared error solution.

[3]However, if the plant is known *a priori* to be strictly proper, the delay $z^{-1}\mathbf{I}$ block may be removed from the feedback path. Then the estimator needs to predict the disturbance one fewer step into the future. Since estimation is imperfect, this will improve performance.

E. Adapting a Disturbance Canceler via the BPTM Algorithm

We have seen how a disturbance canceling filter can be inserted into the control-system design in such a way that it will not bias the controller for a linear plant, and will minimally bias a controller for a nonlinear plant. Proceeding to develop an algorithm to adapt X, we consider that the system error is composed of three parts.

- One part of the system error is dependent on the input command vector $\vec{r}_k$ in C. This part of the system error is reduced by adapting C.
- Another part of the system error is dependent on the estimated disturbance vector $\vec{\hat{w}}_k$ in X. This part of the system error is reduced by adapting X.
- The minimum-mean-squared error, which is independent of both the input command vector in C and the estimate disturbance vector in X. It is either irreducible (if the system dynamics prohibit improvement), or may be reduced by making the tapped-delay lines at the input to X or C larger. In any case, adaptation of the weights in X or C will not reduce the minimum-mean-squared error.
- The fourth possible part of the system error is the part that is dependent on both the input command vector and the disturbance vector. However, by assumption, r_k and w_k are independent, so this part of the system error is zero.

Using the BPTM algorithm to reduce the system error by adapting C, as discussed in Section IV, will reduce the component of the system error dependent on the input r_k. Since the disturbance and minimum-mean-squared error are independent of r_k, their presence will not bias the solution of C. The controller will learn to control the feedforward dynamics of the system, but not to cancel disturbance.

If we were to use the BPTM algorithm and backpropagate the system error through the plant model, using it to adapt X as well, the disturbance canceler would learn to reduce the component of the system error dependent on the estimated disturbance signal. The component of the system error due to unconverged C and minimum-mean-squared error will not bias the disturbance canceler.

BPTM for the disturbance canceler may be developed as follows. Let $g(\cdot)$ be the function implemented by X, $f(\cdot)$ be the function implemented by the plant model copy $\widehat{P}_{\mathrm{COPY}}$, and W_X be the weights of the disturbance canceler neural network. Then

$$\tilde{u}_k = g\left(\tilde{u}_{k-1}\cdots\tilde{u}_{k-m}, \hat{w}_{k-1}\cdots\hat{w}_{k-q}, u_k\cdots u_{k-r}, W_X\right)$$

$$y_k \approx \hat{y}_k = f\left(\hat{y}_{k-1}\cdots\hat{y}_{k-n}, \overline{u}_k\cdots\overline{u}_{k-p}\right)$$

where $\overline{u}_k = u_k + \tilde{u}_k$ is the input to the plant, u_k is the output of the controller, and $\tilde{u}_k$ is the output of the disturbance canceler. BPTM computes the disturbance-canceler weight

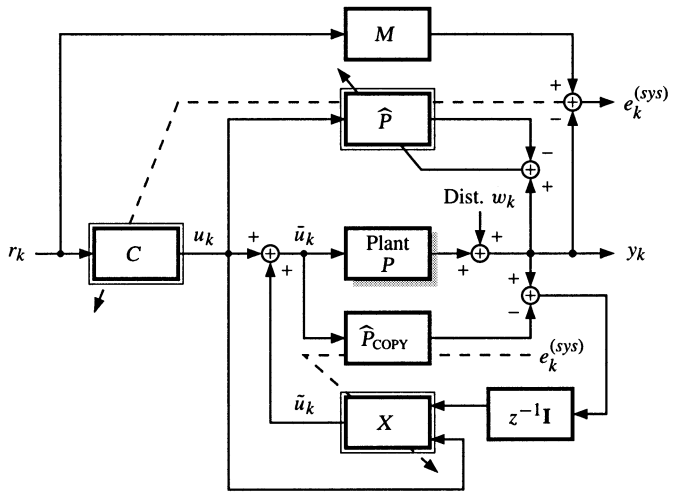


Fig. 8. Integrated nonlinear MIMO system.

update $\Delta(W_X)_k^T = -\eta e_k^T d\hat{y}_k / dW_X$. This can be found by the following:

$$\frac{d\overline{u}_k}{dW_X} = \frac{du_k}{dW_X} + \frac{d\tilde{u}_k}{dW_X} = \frac{d\tilde{u}_k}{dW_X}$$
$$= \frac{\partial \tilde{u}_k}{\partial W_X} + \sum_{j=1}^{m} \left(\frac{\partial \tilde{u}_k}{\partial \tilde{u}_{k-j}}\right)\left(\frac{d\overline{u}_{k-j}}{dW_X}\right)$$
$$\frac{d\hat{y}_k}{dW_X} = \sum_{j=0}^{p} \left(\frac{\partial \hat{y}_k}{\partial \overline{u}_{k-j}}\right)\left(\frac{d\overline{u}_{k-j}}{dW_X}\right)$$
$$+ \sum_{j=1}^{n} \left(\frac{\partial \hat{y}_k}{\partial \hat{y}_{k-j}}\right)\left(\frac{d\hat{y}_{k-j}}{dW_X}\right).$$

This method is illustrated in Fig. 8 where a complete integrated MIMO nonlinear control system is drawn. The plant model is adapted directly, as before. The controller is adapted by backpropagating the system error through the plant model and using the BPTM algorithm of Section IV. The disturbance canceler is adapted by backpropagating the system error through the *copy* of the plant model and using the BPTM algorithm as well. So we see that the BPTM algorithm serves two functions: it is able to adapt both C and X.

Since the disturbance canceler requires an accurate estimate of the disturbance at its input, the plant model should be adapted to near convergence before "turning the disturbance canceler on" (connecting the disturbance canceler output to the plant input). Adaptation of the disturbance canceler may begin before this point, however.

VI. Examples

Four simple nonlinear SISO systems were chosen to demonstrate the principles of this paper. The collection includes both minimum-phase and nonminimum-phase plants, plants described using difference equations and plants described in state space, and a variety of ways to inject disturbance. Since the plants are not motivated by any particular "real" dynamical system, the command signal and disturbance sources are artificial as well. In each case, the command signal is uniform independently identically distributed (i.i.d.), which was chosen since it is the most difficult to follow. The raw disturbance source is a first-order Markov process. In some cases the Markov process is driven by i.i.d. uniform random variables, and in other cases by i.i.d. Gaussian random variables. The disturbance is added to either the input of the system, to the output of the system, to a specific state in the plant's state–space representation or to an intermediate stage of the processing performed by the plant.

System 1: The first plant was introduced to the literature by Narendra and Parthasarathy [33]. The difference equations defining its dynamics are

$$s_k = \frac{s_{k-1}}{1 + s_{k-1}^2} + u_{k-1}^3$$
$$y_k = s_k + \mathrm{dist}_k.$$

The plant model was a $\mathcal{N}_{(2,1):8:1}$ network, and system identification was initially performed with the plant input signal u_k being i.i.d. uniformly distributed between $[-2, 2]$. Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were uniformly distributed in the range $[-0.5, 0.5]$. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added directly to the *output* of the system. Note that the disturbance is added *after* the nonlinear filter, and hence it does not affect the internal state of the system.

System 2: The second plant is a nonminimum-phase (meaning that it does not have a stable causal inverse) nonlinear transversal system. The difference equation defining its dynamics is

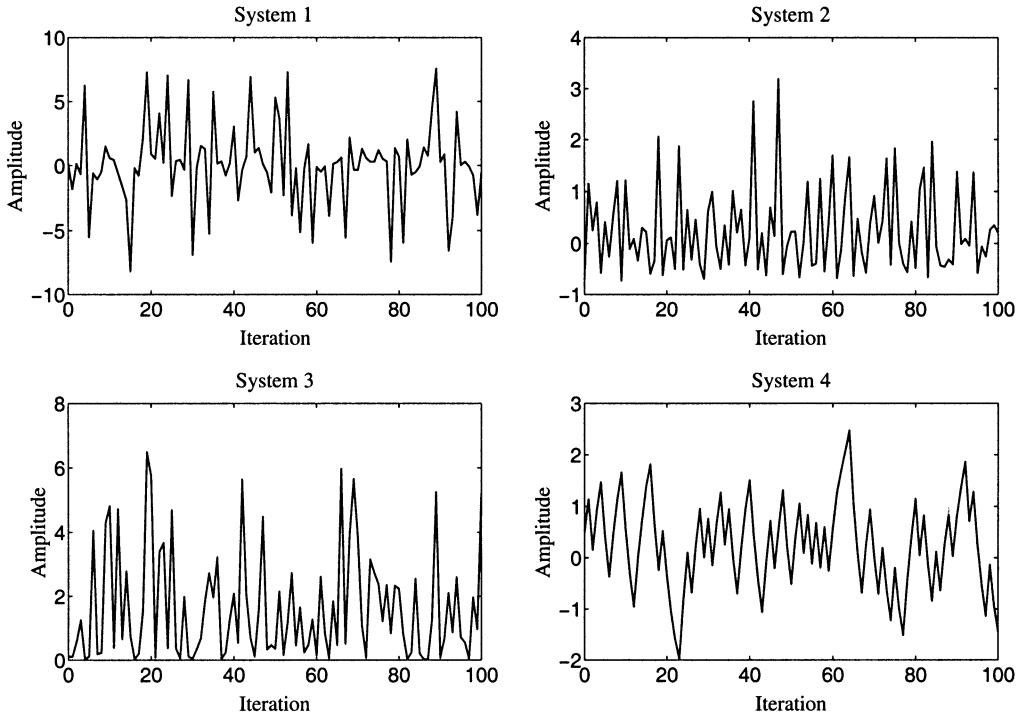
$$y_k = \exp(u_{k-1} - 2u_{k-2} + \mathrm{dist}_{k-1}) - 1.$$

The plant model was a $\mathcal{N}_{(3,0):3:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-0.5, 0.5]$. Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were distributed according to a Gaussian distribution with zero mean and standard deviation 0.03. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added to the *input* of the system.

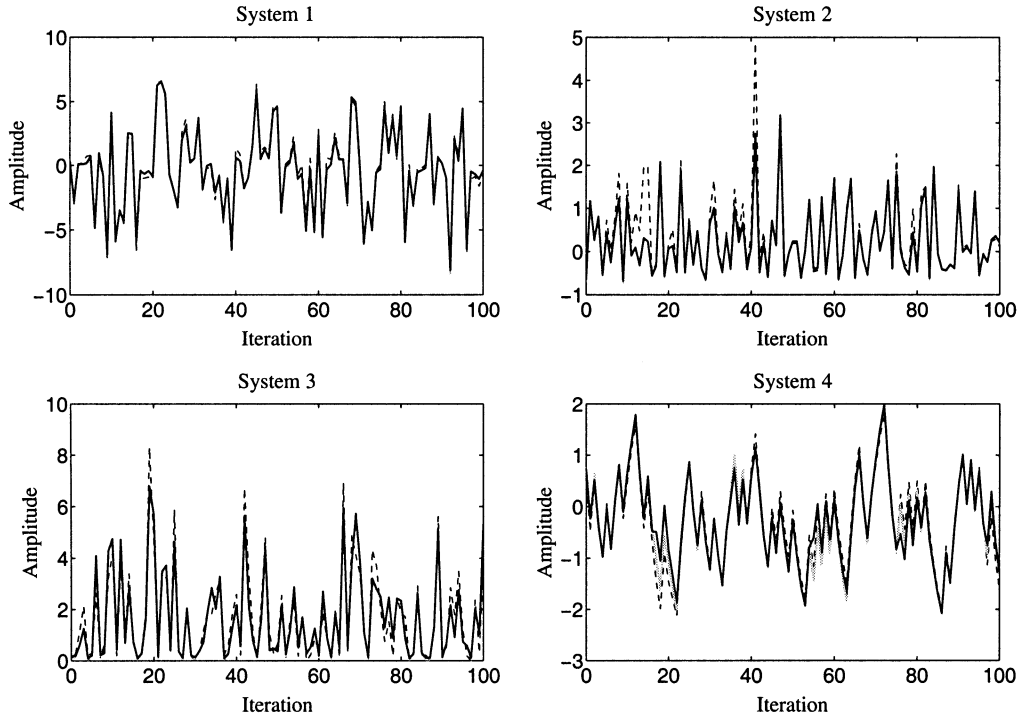
System 3: The third system is a nonlinear plant expressed in state–space form. The system consists of a linear filter followed by a squaring device. The difference equations defining this plant's dynamics are

$$x_k = \begin{bmatrix} 0 & 1 \\ -0.2 & 0.2 \end{bmatrix} x_{k-1} + \begin{bmatrix} 0.2 \\ 1 \end{bmatrix} u_{k-1} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathrm{dist}_{k-1}$$
$$s_k = [1 \quad 2]\, x_k$$
$$y_k = 0.3\,(s_k)^2.$$

The plant model was a $\mathcal{N}_{(10,5):8:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1, 1]$.



(a)

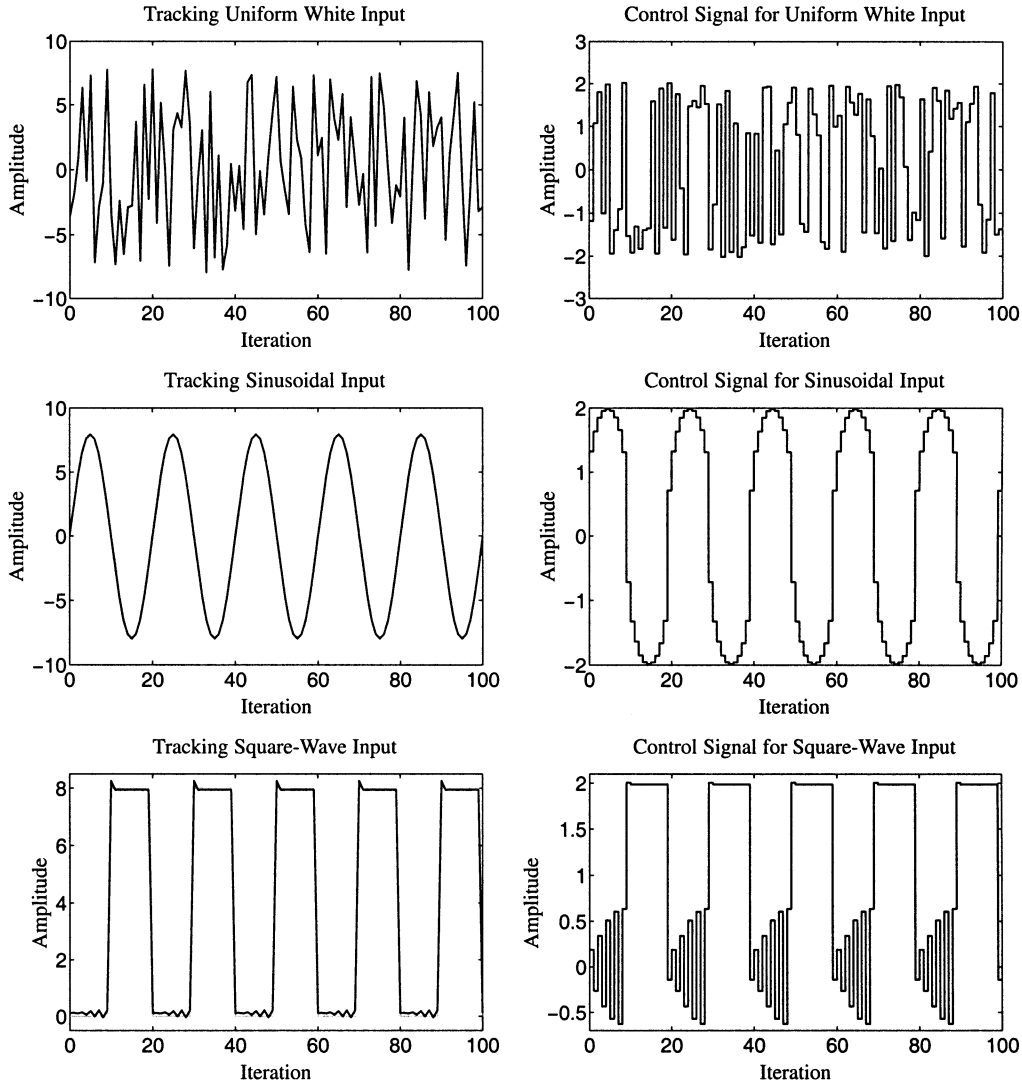


(b)

Fig. 9. (a) System identification of four nonlinear SISO plants *in the absence of disturbance*. The gray line (when visible) is the true plant output, and the solid line is the output of the plant model. (b) System identification of four nonlinear SISO plants *in the presence of disturbance*. The dashed black line is the disturbed plant output, and the solid black line is the output of the plant model. The gray solid line (when visible) shows what the plant output would have been if the disturbance were absent. This signal is normally unavailable, but is shown here to demonstrate that the adaptive plant model captures the dynamics of the true plant very well.

Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were uniformly distributed in the range $[-0.5, 0.5]$. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.99$. The resulting disturbance was added directly to the first state of the system.

System 4: The final system is a generalization of one in reference [13]. The nonlinearity in this system has memory or "phase," and is a type of hysteresis device. The system has two



(a)

Fig. 10. (a) Feedforward control of system 1. The controller, $\mathcal{N}_{(2,1):6:1}$, was trained to track uniformly distributed (white) random input, between $[-8, 8]$. Trained performance and generalization are shown.

equilibrium states as opposed to the previous plants which all had a single equilibrium state. The difference equations defining its dynamics are

$$s_k = 0.4 s_{k-1} + 0.5 u_k$$

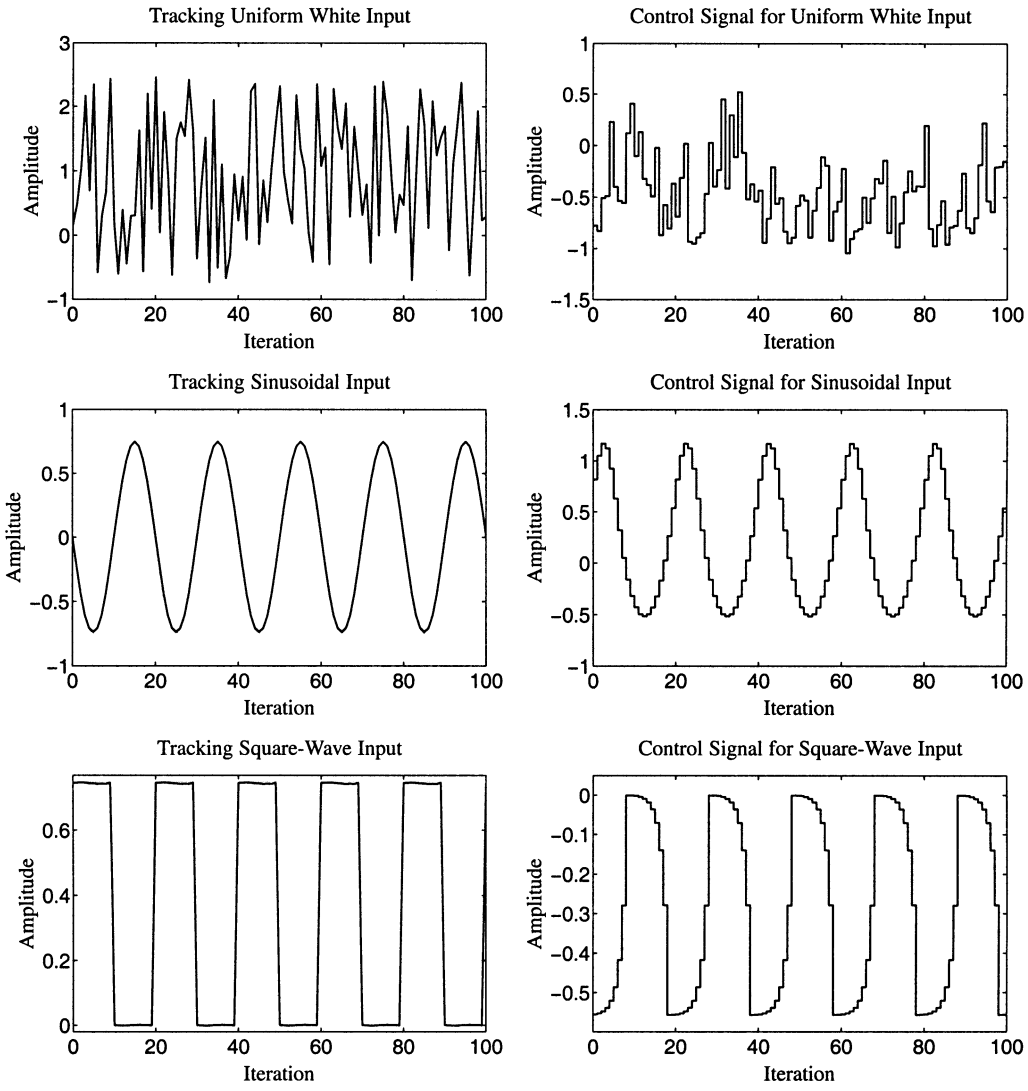
$$y_k = \text{dist}_k + \begin{cases} 0.8 y_{k-1} + 0.8 \tanh(s_k - 2), & \text{if } s_k > s_{k-1}; \\ 0.8 y_{k-1} + 0.8 \tanh(s_k + 2), & \text{if } s_k \leq s_{k-1}. \end{cases}$$

The plant model was a $\mathcal{N}_{(10,10):30:1}$ network, with the plant input signal u_k being i.i.d. uniformly distributed between $[-1, 1]$. Disturbance was a first-order Markov process, generated by filtering a primary random process of i.i.d. random variables. The i.i.d. random variables were distributed according to a Gaussian distribution with zero mean and standard deviation 0.01. The filter used to generate the first-order Markov process was a one-pole filter with the pole at $z = 0.95$. The resulting disturbance was added to an intermediate point in the system, just before the output filter.

A. System Identification

System identification was first performed, starting with random weight values, in the *absence* of disturbance, and a summary plot of is presented in Fig. 9(a). This was repeated, starting again with random weight values, in the *presence* of disturbance and a summary plot is presented in Fig. 9(b). Plant models were trained using a series-parallel method first to accomplish coarse training. Final training was always in a parallel connection to ensure unbiased results. The RTRL algorithm was used with an adaptive learning rate for each layer of neurons: $\mu_k^i = \min_{0 \leq j \leq k} 1/||X_{i,j}||^2$ where i is the layer number and $X_{i,k}$ is the input vector to that layer. This simple rule-of-thumb, based on stability results for LMS, was very effective in providing a ballpark adaptation rate for RTRL.

In all cases, a neural network was found that satisfactorily identified the system. Each system was driven with



(b)

Fig. 10. (*Continued*). (b) Feedforward control of system 2. The controller, $\mathcal{N}_{(20,\,1):20:1}$, was trained to track uniformly distributed (white) random input, between $[-0.75,\,2.5]$.

an i.i.d. uniform control signal. This was not characteristic of the control signal generated by the trained controller in the next section, but was a starting point and worked quite well to initialize the plant model for use in training the controller. Each plant produced its own very characteristic output for the same input, as seen in Fig. 9, but it was shown that neural networks could be trained to identify each system nearly perfectly.

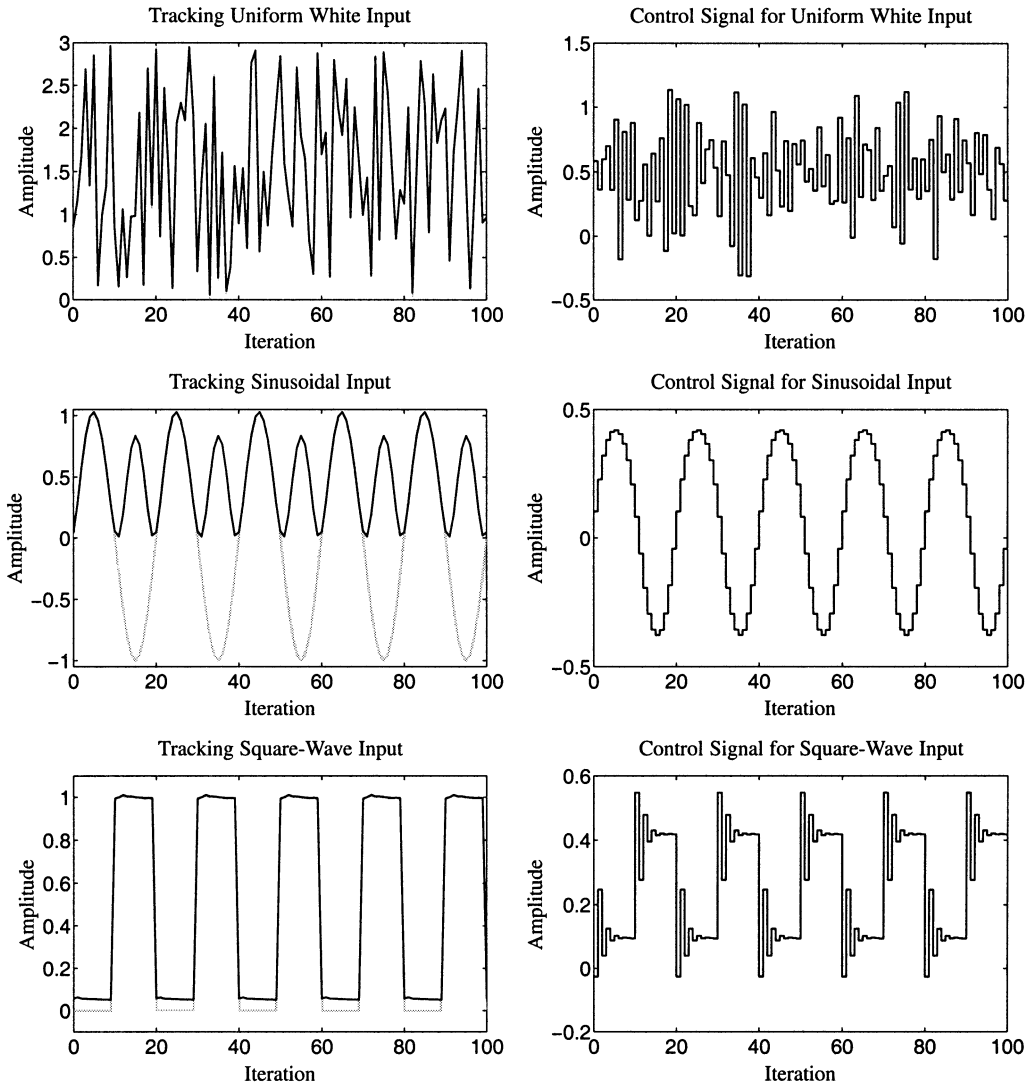
When disturbance is added, it is useful to think of the "disturbed plant dynamics" and the "nominal plant dynamics." In each case, the system identification process matched the nominal dynamics of the plant, which is what theory predicts and what we would like.

B. Feedforward Control

After system identification was done, the controller was trained to perform feedforward control of each system. The control input signal was always uniform i.i.d. random input. Since the plants themselves are artificial, this artificial control signal was chosen. In general, it is the hardest control signal to follow. The plants were undisturbed; disturbance canceling for disturbed plants is considered in the next section.

The control signals generated by the trained controllers are not i.i.d. uniform. Therefore, the system identification performed in the previous section is not sufficient to properly train the controller. It provides a very good initial set of values for the weights of the controller, however, and system identification continues on-line as the controller is trained with the BPTM algorithm. Again, the adaptive learning rate $\mu_k^i = \min_{0\le j\le k} 1/\|X_{i,\,j}\|^2$ proved effective.

First, the controllers were trained with an i.i.d. uniform command input. The reference model in all cases (except for system 2) was a unit delay. When the weights had converged, the values of the network weights were frozen, and the controller was tested with an i.i.d. uniform, a sinusoidal and a square wave to show the performance and generalization of the system. The results are presented in Fig. 10. The desired



(c)

Fig. 10. (*Continued.*) (c) Feedforward control of system 3. The controller, $\mathcal{N}_{(2,8):8:1}$, was trained to track uniformly distributed (white) random input, between $[0, 3]$.

plant output (gray line) and the true plant output (solid line) are shown for each scenario.

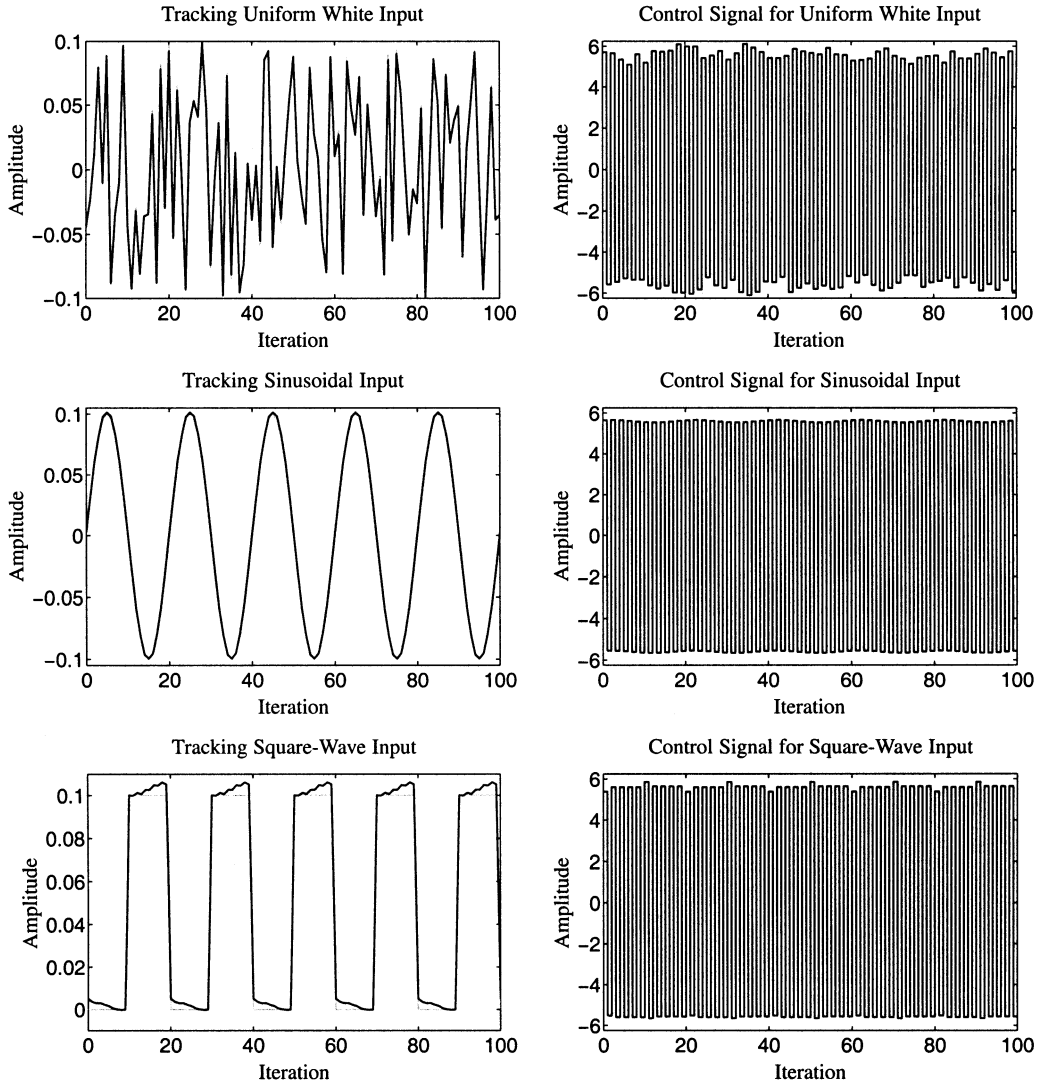
Generally (specific variations will be addressed below) the tracking of the uniform i.i.d. signal was nearly perfect, and the tracking of the sinusoidal and square waves was excellent as well. We see that the control signals generated by the controller are quite different for the different desired trajectories, so the controller can generalize well. When tracking a sinusoid, the control signal for a linear plant is also sinusoidal. Here, the control signal is never sinusoidal, indicating in a way the degree of nonlinearity in the plants.

Notes on System 2: System 2 is a nonminimum-phase plant. This can be easily verified by noticing that its linear-filter part has a zero at $z = 2$. This plant cannot follow a unit-delay reference model. Therefore, reference models with different latencies were tried, for delays of zero time samples up to 15 time samples. In each case, the controller was fully trained and the steady–state mean-squared-system error was measured. A plot of the results is shown in Fig. 11. Since both low MSE and low latency is desirable, a reference model for this work was chosen to be a delay of ten time samples: $M(z) = z^{-10}$.

Notes on System 3: The output of system 3 is constrained to be positive due to the squaring device in its representation. Therefore, it is interesting to see how well this system generalizes when asked to track a zero-mean sinusoidal signal. As shown in Fig. 10(c), the result is something like a full-wave rectified version of the desired result. This is neither good nor bad—just curious.

Simulations were also done to see what would happen if the system were *trained* to follow this kind of command input. In that case, the plant output looks like a half-wave rectified version of the input. Indeed, this is the result that minimizes MSE—the training algorithm works!

Notes on System 4: As can be seen from Fig. 10(d), the control signal required to control this plant is extremely harsh. The hysteresis in the plant requires a type of modulated bang-bang control. Notice that the control signals for the three different inputs are almost identical. The plant is very sensitive to its input,



(d)

Fig. 10. (*Continued.*) (d) Feedforward control of system 4. The controller, $\mathcal{N}_{(10,\,1):30:1}$, was trained to track uniformly distributed (white) random input, between $[-0.1,\,0.1]$. *Notice that the control signals are almost identical for these three very different inputs!*

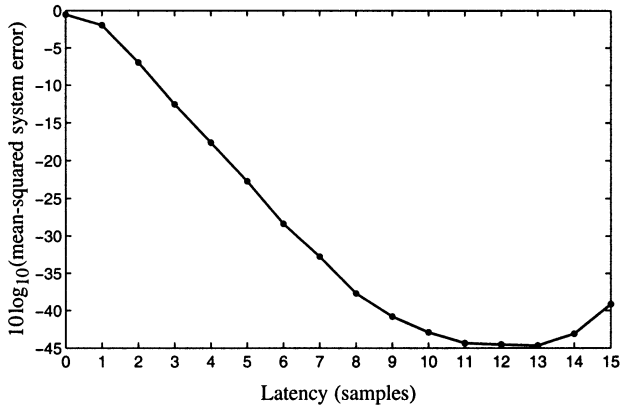


Fig. 11. Mean-squared system error (in decibels) plotted versus control-system latency for System 2.

yet can be controlled precisely by a neural network trained with the BPTM algorithm.

C. Disturbance Canceling

With system identification and feedforward control accomplished, disturbance cancellation was performed. The input to the disturbance canceling filter X was chosen to be tap-delayed copies of the u_k and $\hat{w}_k$ signals.

The BPTM algorithm was used to train the disturbance cancelers with the same adaptive learning rate $\mu_k^i = \min_{0\le j\le k} 1/\|X_{i,\,j}\|^2$. After training, the performance of the cancelers was tested and the results are shown in Fig. 12. In this figure, each system was run with the disturbance canceler turned off for 500 time samples, and then turned on for the next 500 time samples. The squared system error is plotted. The disturbance cancelers do an excellent job of removing the disturbance from the systems.

VII. Conclusion

Adaptive inverse control is very simple yet highly effective. It works for minimum-phase or nonminimum-phase, linear or

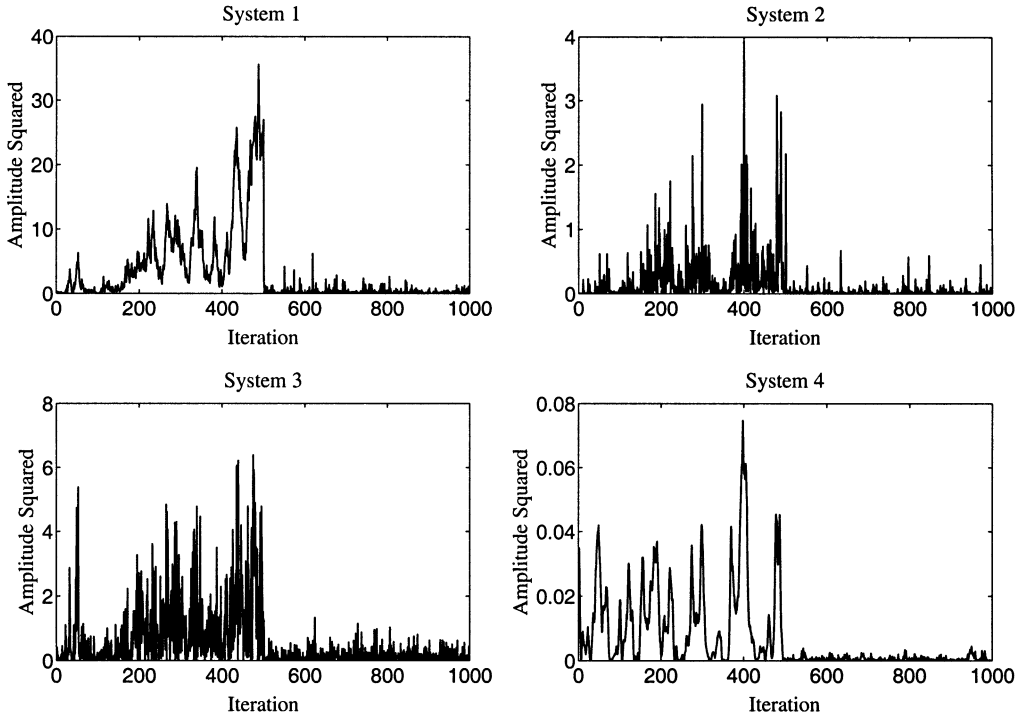


Fig. 12. Plots showing disturbance cancellation. Each system is run with the disturbance canceler turned off for 500 time steps. Then, the disturbance canceler is turned on and the system is run for an additional 500 time steps. The square amplitude of the system error is plotted. The disturbance-canceler architectures were: $\mathcal{N}_{([5,5],2):10:1}$, $\mathcal{N}_{([5,5],1):10:1}$, $\mathcal{N}_{([5,5],4):10:1}$, and $\mathcal{N}_{([5,5],1):30:1}$, respectively.

nonlinear, SISO or MIMO, stable or stabilized plants. The control scheme is partitioned into smaller subproblems that can be independently optimized. First, an adaptive plant model is made; second, a constrained adaptive controller is generated; finally, a disturbance canceler is adapted. All three processes may continue concurrently, and the control architecture is unbiased if the plant is linear, and is minimally biased if the plant is nonlinear. Excellent control and disturbance canceling for minimum-phase or nonminimum-phase plants is achieved.

REFERENCES

[1] K. J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed. Reading, MA: Addison-Wesley, 1995.

[2] S. H. Lane and R. F. Stengel, "Flight control design using nonlinear inverse dynamics," *Automatica*, vol. 24, no. 4, pp. 471–83, 1988.

[3] D. Enns, D. Bugajski, R. Hendrick, and G. Stein, "Dynamic inversion: An evolving methodology for flight control design," *Int. J. Contr.*, vol. 59, no. 1, pp. 71–91, 1994.

[4] Y. D. Song, T. L. Mitchell, and H. Y. Lai, "Control of a class of nonlinear uncertain systems via compensated inverse dynamics approach," *IEEE Trans. Automat. Contr.*, vol. 39, pp. 1866–71, Sept. 1994.

[5] B. S. Kim and A. J. Calise, "Nonlinear flight control using neural networks," *J. Guidance, Contr., Dyn.*, vol. 20, no. 1, pp. 26–33, Jan.–Feb. 1997.

[6] L. Yan and C. J. Li, "Robot learning control based on recurrent neural network inverse model," *J. Robot. Syst.*, vol. 14, no. 3, pp. 199–212, 1997.

[7] G. L. Plett, "Adaptive inverse control of unknown stable SISO and MIMO linear systems," *Int. J. Adaptive Contr. Signal Processing*, vol. 16, no. 4, pp. 243–72, May 2002.

[8] G. L. Plett, "Adaptive inverse control of plants with disturbances," Ph.D. dissertation, Stanford Univ., Stanford, CA, May 1998.

[9] B. Widrow, G. L. Plett, E. Ferreira, and M. Lamego, "Adaptive inverse control based on nonlinear adaptive filtering," in *Proc. 5th IFAC Workshop Algorithms Architectures for Real-Time Contr. AARTC'98*, Cancun, Mexico, Apr. 1998, (invited paper), pp. 247–252.

[10] B. Widrow and G. L. Plett, "Nonlinear adaptive inverse control," in *Proc. 36th IEEE Conf. Decision Contr.*, vol. 2, San Diego, CA, Dec. 10–12, 1997, pp. 1032–1037.

[11] M. Bilello, "Nonlinear adaptive inverse control," Ph.D. dissertation, Stanford Univ., Stanford, CA, Apr. 1996.

[12] B. Widrow and G. L. Plett, "Adaptive inverse control based on linear and nonlinear adaptive filtering," in *Proc. World Congr. Neural Networks*, San Diego, CA, Sept. 1996, pp. 620–27.

[13] B. Widrow and E. Walach, *Adaptive Inverse Control*. Upper Saddle River, NJ: Prentice-Hall, 1996.

[14] H. T. Siegelmann, B. B. Horne, and C. L. Giles, "Computational capabilities of recurrent NARX neural networks," *IEEE Trans. Syst., Man, Cybern. B*, vol. 27, pp. 208–215, Apr. 1997.

[15] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard Univ., Cambridge, MA, Aug. 1974.

[16] D. B. Parker, "Learning logic," Stanford Univ., Stanford, CA, Tech. Rep. Invention Rep. S81–64, File 1, Office Technol. Licencing, Oct. 1982.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, vol. 1, ch. 8.

[18] R. J. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm," *Connection Sci.*, vol. 1, no. 1, pp. 87–111, 1989.

[19] D. H. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Contr. Syst. Mag.*, vol. 10, pp. 18–23, Apr. 1990.

[20] S. W. Piché, "Steepest descent algorithms for neural network controllers and filters," *IEEE Trans. Neural Networks*, vol. 5, pp. 198–212, Mar. 1994.

[21] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: Perceptron, Madaline, and backpropagation," *Proc. IEEE*, vol. 78, pp. 1415–42, Sept. 1990.

[22] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition" (in Russian), *Dokl. Akad. Nauk USSR*, vol. 114, pp. 953–56, 1957.

[23] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1992.
[24] G. V. Puskorius and L. A. Feldkamp, "Recurrent network training with the decoupled extended Kalman filter algorithm," in *Proc. SPIE Sci. Artificial Neural Networks*, vol. 1710, New York, 1992, pp. 461–473.
[25] G. L. Plett and H. Böttrich. DDEKF learning for fast nonlinear adaptive inverse control. presented at *Proc. 2002 World Congr. Comput. Intell.*
[26] C. E. Garcia and M. Morari, "Internal model control. 1. A unifying review and some new results," *Ind. Eng. Chemistry Process Design Development*, vol. 21, no. 2, pp. 308–23, Apr. 1982.
[27] ——, "Internal model control. 2. Design procedure for multivariable systems," *Ind. Eng. Chem. Process Design Development*, vol. 24, no. 2, pp. 472–84, Apr. 1985.
[28] ——, "Internal model control. 3. Multivariable control law computation and tuning guidelines," *Ind. Eng. Chem. Process Design Development*, vol. 24, no. 2, pp. 484–94, Apr. 1985.
[29] D. E. Rivera, M. Morari, and S. Skogestad, "Internal model control. 4. PID controller design," *Ind. Eng. Chem. Process Design Development*, vol. 25, no. 1, pp. 252–65, Jan. 1986.
[30] C. G. Economou and M. Morari, "Internal model control. 5. Extension to nonlinear systems," *Ind. Eng. Chem. Process Design Development*, vol. 25, no. 2, pp. 403–44, April 1986.
[31] ——, "Internal model control. 6. Multiloop design," *Ind. Eng. Chem. Process Design Development*, vol. 25, no. 2, pp. 411–19, Apr. 1986.
[32] J. E. Marsden and A. J. Tromba, *Vector Calculus*, 3rd ed. San Francisco, CA: Freeman, 1988, pp. 243–7.
[33] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Mar. 1990.

Gregory L. Plett (S'97–M'98–SM'02) was born in Ottawa, ON, Canada, in 1968. He received the B.Eng. degree in computer systems engineering (with high distinction) from Carleton University, Ottawa, in 1990, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1992 and 1998, respectively.

He is currently an Assistant Professor of Electrical and Computer Engineering with the University of Colorado, Colorado Springs.