

Training Recurrent Neurocontrollers for Robustness With Derivative-Free Kalman Filter

Danil V. Prokhorov, *Senior Member, IEEE*

Abstract—We are interested in training neurocontrollers for robustness on discrete-time models of physical systems. Our neurocontrollers are implemented as recurrent neural networks (RNNs). A model of the system to be controlled is known to the extent of parameters and/or signal uncertainties. Parameter values are drawn from a known distribution. For each instance of the model with specified parameters, a recurrent neurocontroller is trained by evaluating sensitivities of the model outputs to perturbations of the neurocontroller weights and incrementally updating the weights. Our training process strives to minimize a quadratic cost function averaged over many different models. In the end, the process yields a robust recurrent neurocontroller, which is ready for deployment with fixed weights. We employ a derivative-free Kalman filter algorithm proposed by Norgaard *et al.* and extended by Feldkamp *et al.* (2001) and Feldkamp *et al.* (2002) to neural network training. Our training algorithm combines effectiveness of a second-order training method with universal applicability to both differentiable and nondifferentiable systems. Our approach is that of model reference control, and it extends significantly the capabilities proposed by Prokhorov *et al.* (2001). We illustrate it with two examples.

Index Terms—Derivative-free Kalman filter, neurocontroller, training for robustness, recurrent neural network (RNN).

I. INTRODUCTION

GROWING power of computers permits implementation of increasingly sophisticated models and algorithms for control of various technological processes and embedded systems. Large investments in modeling of physical systems to be controlled termed plants lead to development of high-fidelity dynamical models featuring complex, highly nonlinear, and often discontinuous relationship between variables. For example, a plant model can consist of many modules implemented in Matlab/Simulink each of which may contain multitudes of lookup tables, dead zones, saturations, and other nonsmooth and discontinuous elements. In spite of substantial time and money invested in their development, many plant models are still known only to within parametric and/or signal disturbances. Adaptive controllers may not always be applicable, and, additional, often costly model calibration efforts seem unavoidable, especially if high quality control is desired.

Available literature surveys on the use of neural networks to control show that the majority of papers deal with designing adaptive neurocontrollers mostly in the form of feedforward and time-delay neural networks (e.g., [5]–[10] and references therein). To compensate for uncertainties in plant dynamics,

such neural networks are trained either directly on the plant (direct adaptive control) or with the help of its model (indirect adaptive control, e.g., [11]).

Another direction of research on neurocontrol stems from neural networks with output-to-input and/or internal feedback called recurrent neural networks (RNNs); see, e.g., [12]–[14]. RNNs are very powerful nonlinear operators capable of accurate modeling [15], [16], and control [13] in the presence of disturbances. Of special interest to this paper are RNNs with internal feedback. These are known to be able to imitate adaptive algorithms, even with their weights fixed, as demonstrated in [20] and [21]. Due to such remarkable behavior, RNNs seem especially appealing as an alternative to adaptive neurocontrollers, provided that RNNs are trained appropriately for robustness to uncertainties or changes in plant parameters.

Our approach is to train an RNN controller on instances of plant models to handle our imprecise knowledge about the plant. It has been shown in [22] that it is possible to train a recurrent neurocontroller on several instances of an automotive engine model (idle speed control problem: maintaining the desired speed of the crankshaft in spite of various disturbances), where each of the instances is different from others in values of the model parameters. The goal of [22], as well as the goal of this paper, is to create a robust recurrent neurocontroller, i.e., a recurrent neurocontroller which is capable of delivering an acceptable performance regardless of the true (unknown) values of the plant parameters. After its comprehensive testing on many plant model instances, the recurrent neurocontroller is supposed to be deployed with fixed weights and no postdeployment adaptation. This bypasses a delicate issue of preserving stability while training neurocontrollers *online*. Establishing theoretical guarantees of performance including those for closed-loop stability is beyond the scope of this paper. We assume that performance can always be verified in practice, similar to the viewpoint expressed in [9].

To illustrate a setting in which we train a recurrent neurocontroller to cope with plant uncertainties, we consider the following example. Let us have a family of plants

$$y(t) = \phi(z_{pt}(t-1), u(t), \theta(k)) \quad (1)$$

which can be represented as

$$\begin{aligned} z_{pt}(t) &= \phi_1(z_{pt}(t-1), u(t), \theta(k)) \\ y(t) &= \phi_2(z_{pt}(t)) \end{aligned} \quad (2)$$

where $\phi = \phi_2 \circ \phi_1$, ϕ_1 , and ϕ_2 are smooth scalar functions. Parametric uncertainty $\theta(k)$ is bounded and time-varying, with the time scale different from that of the plant. (For example, $\theta(k)$ is piecewise constant, switching from one level to the next with

Manuscript received September 7, 2003; revised March 12, 2006.

The author was with the Ford Research and Advanced Engineering, Dearborn, MI 48124 USA. He is now with Toyota Technical Center, Ann Arbor, MI 48105 USA (e-mail: dvprokhorov@gmail.com).

Digital Object Identifier 10.1109/TNN.2006.880580

changing k ; k changes much less often than t , e.g., after $T \gg 1$ steps.) We wish to develop a *nonadaptive* controller capable of stabilizing all plants of the family (1) around the origin $y = 0$. We propose the following structure for such controller

$$u(t) = \pi(\mathbf{z}_{cr}(t-1), y(t-1), \mathbf{W}) \quad (3)$$

which can be represented as

$$\begin{aligned} \mathbf{z}_{cr}(t) &= \boldsymbol{\pi}_1(\mathbf{z}_{cr}(t-1), y(t-1), \mathbf{W}_1) \\ u(t) &= \pi_2(\mathbf{z}_{cr}(t), \mathbf{W}_2) \end{aligned} \quad (4)$$

where $\pi = \pi_2 \circ \boldsymbol{\pi}_1$, π_2 , and $\boldsymbol{\pi}_1$ are smooth functions and \mathbf{W}_1 and \mathbf{W}_2 are adjustable parameters of the controller¹ (\mathbf{W} is formed by merging \mathbf{W}_1 and \mathbf{W}_2). The structure (4) can be interpreted as an RNN with weights \mathbf{W} , with one recurrent layer of nodes \mathbf{z}_{cr} (internal feedback) and one feedforward node π_2 .

Suppose for (1) there exists bounded $u^*(t)$ such that the perfect stabilization of all plants is possible

$$0 = \phi(z_{pt}(t-1), u^*(t), \theta(k)). \quad (5)$$

This ideal control $u^*(t)$ can be computed only if $z_{pt}(t-1)$ and $\theta(k)$ are known. However, the full plant state including *both* z and θ is not accessible (hidden); therefore, the controller (3) must develop its internal representation \mathbf{z}_{cr} by observing y and its history such that

$$u^*(t) \approx \pi(\mathbf{z}_{cr}(t-1), y(t-1), \mathbf{W}). \quad (6)$$

Assuming sufficient number of nodes in layer $\boldsymbol{\pi}_1$ of (4), we create an adequate representation through training of weights \mathbf{W} .

At this point, it is worth recalling representation differences between the RNN controller (3) and other neurocontrollers. In our nonadaptive setting (*fixed* weights \mathbf{W}), a feedforward neurocontroller, i.e., a controller such as (4) but with the feedforward $\mathbf{z}_{cr} = \boldsymbol{\pi}_1(y(t-1), \mathbf{W}_1)$, may be inadequate to control the family (1) because the same $y(t-1)$ can require the controller to produce different values of $u(t)$, depending on different combinations of z_{pt} and θ . What about a time-delay neurocontroller? Such a controller can be represented by $\mathbf{z}_{cr} = \boldsymbol{\pi}_1(y(t-1), y(t-2), \dots, y(t-d), \mathbf{W}_1)$ in (4), and it has clearly much more information about the plant than the feedforward controller inputting any particular $y(t-i)$. However, its complexity in terms of the required number of inputs $y(t-i)$, $i = 1, 2, \dots, d$, can be significant. Furthermore, the time-delay neurocontroller may not be sufficient for some plant families. If we have the plant family $y(t) = \phi(z_{pt}(t-1), u(t) - \theta(k))$ (note $u(t) - \theta(k)$), then its stabilization around the origin requires an RNN controller (3) because control u must be different for varying $\theta(k)$ to keep the plant stabilized at $y = 0$.

Any successful process for training a recurrent neurocontroller should cover a sufficient number of variations of θ . This requirement is similar to training a neural network to approximate an unknown function from its samples. It is well known that a neural network can easily overfit when trained on a small number of samples, demonstrating very low errors on all of

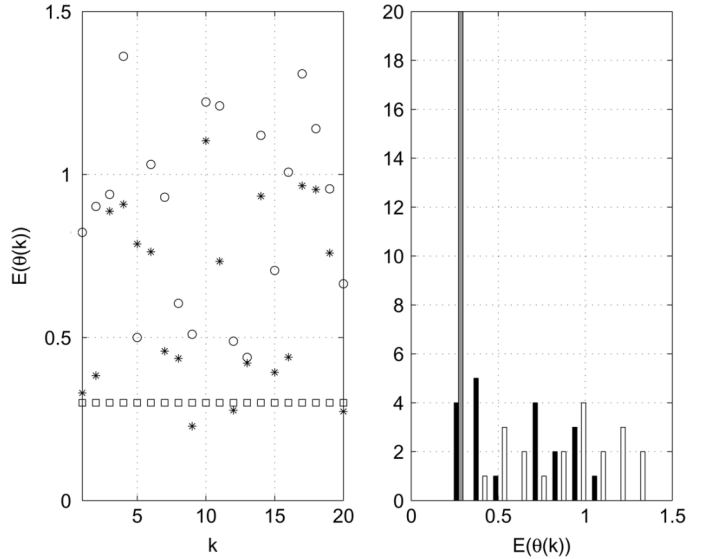


Fig. 1. Performance differences between three hypothetical controllers. The left panel shows the performance $E(\theta(k))$ of each controller while varying the plant index k . The performance of the ideal (perfectly robust) controller is denoted with squares at $E(\theta(k)) = 0.3$. The performance of the recurrent neurocontroller trained for robustness (stars) is better than that of the specialized neurocontroller (trained on just one plant $k = 5$ and denoted with circles) for all k except $k = 5$. The right panel is the histogram of the left panel (squares, stars, and circles on the left correspond to gray, black, and white bars on the right, respectively). The advantage of the robust neurocontroller (black bars) over the specialized neurocontroller (white bars) is apparent.

them. The same neural network may exhibit large errors on test samples, which is not acceptable. Suppose that we now train (4) to control (1) on a set of $\theta(k)$, $k = 1, 2, \dots, N_{pt}$, i.e., we *sample* the space of control tasks. For the performance measure

$$E(\theta(k)) = \frac{1}{T} \sum_{t=1}^T (0 - y(t, \theta(k)))^2 \quad (7)$$

where T is the trajectory length [T is chosen to be sufficiently large so that the effect of initial states $z_{pt}(0)$ and $\mathbf{z}_{cr}(0)$ on (7) is negligible], we assume that our trained recurrent neurocontroller exhibits the performance on the set of $\theta(k)$ as shown in Fig. 1 for $N_{pt} = 20$ (marked by stars). For comparison, the ideal (perfectly robust) controller is the one with performance $E = 0.3$ (marked by squares), whereas a controller specialized to just one of the plants, e.g., $\theta(5)$, exhibits the performance over all θ as marked by circles. This specialized controller overfits in the space of control tasks because in training it only had to deal with one particular example of the control task. [Its performance $E(\theta(5))$ is significantly better than that of the RNN controller trained for robustness, but worse for all other $\theta(k)$.] We would like to strive for the ideal controller insensitive to variations of θ , but in practice we can only achieve some (problem-specific) reduction of the maximum and the mean of $E(\theta)$ by training for robustness due to the limited size of RNN. Though reducing the variance of $E(\theta)$ is also a useful goal in robustness training, we recognize that sacrificing the minimum of $E(\theta)$ significantly for the sake of the variance reduction should be considered a last resort.

In the approach discussed in [4], plant models consist of differentiable components to enable training robust neurocontrollers via the extended Kalman filter (EKF) algorithm.

¹We henceforth denote vectors and matrices by letters in bold.

The EKF algorithm was first applied to RNNs more than ten years ago [22]. Since then, its effectiveness in training neural networks has independently been verified by many researchers, e.g., [17], [28], [19], and [18]. The important element of the approach in [4] is the use of backpropagation through time (BPTT) [23] to compute derivatives of plant outputs with respect to the neurocontroller weights. When the plant equations are not available, the authors of [4] resort to system identification, the well-known step consisting of training a separate neural network to act as a plant model and estimate the unavailable derivatives via BPTT. In many cases, a fairly accurate plant model is available since it is common in industry to invest a lot of time and money into system identification. Thus, it is highly desirable to use the already developed plant models directly for neurocontroller synthesis (training), rather than replace them with yet another set of models based on differentiable neural networks only because such networks enable the use of BPTT.

In contrast to [4], the approach described in this paper extends the applicability of RNN controllers to broader classes of plant models, e.g., those including lookup tables, dead zones, etc., since it does not use BPTT and thus can be applicable to nondifferentiable elements. Our approach employs the derivative-free Kalman filter algorithm derived in [1]. Our earlier work extended this algorithm which we called Kalman filter by Norgaard, Poulsen, and Ravn (*nprKF*) to training RNN in [2] and [3]. In this paper, we propose to use a computationally efficient form of the *nprKF* to training recurrent neurocontrollers for robustness.

This paper consists of four sections. In Section II, we describe our method. First, we discuss specifics of model reference control, then describe the derivative-free Kalman filter algorithm employed. In Section III, we illustrate its application with two examples. Our first example is one of the problems described in [4]. We show that our approach can achieve improved results on robustness, as compared to previous results. Our second example is an electronic throttle control problem for an internal combustion engine. We demonstrate that our approach is applicable to this practically important problem which includes non-differentiable elements. Section IV concludes our paper.

II. METHOD

A. Controller Training With Reference Model

We follow the framework for neurocontrol established in [24] and [25], and we adopt the viewpoint of model reference control with the controller being an RNN. An example of the general structure for model reference control is shown in Fig. 2. The plant is connected to the controller. The controller goal is to make the plant outputs track the reference (or desired) signals. The plant evolves as a function of the input vector $\mathbf{y}_{I_{pt}}(t)$ and its internal state $\mathbf{z}_{pt}(t)$. The input vector $\mathbf{y}_{I_{pt}}(t)$ consists of control signals $\mathbf{y}_{O_{cr}}(t)$ (outputs of the controller) and any other external variables including unmeasured disturbances (e.g., those corrupting the plant outputs).

The controller receives the time-delayed output of the plant along with the reference signals $\mathbf{y}_{I_{rm}}(t)$. The feedback loop is closed through a time-delay operator $\mathbf{z}^{-d} = \text{diag}(z^{-1}, z^{-2}, \dots, z^{-d})$. The appropriate delays are applied

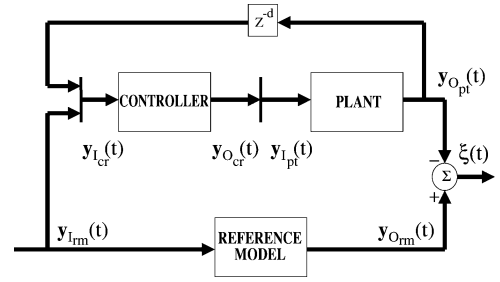


Fig. 2. Block diagram of model reference control [4]. The notation \mathbf{y}_{O_*} and \mathbf{y}_{I_*} is used to distinguish between different elements of the diagram (outputs and inputs, respectively). The symbol $*$ stands for plant, controller, or reference model, each of which may have its own set of state variables. Implicit in this diagram are various disturbances and noise always present in real control systems. The plant block denotes both the actual system to be controlled and its model(s) used to synthesize a controller. Our approach employs plant models to train a neurocontroller for robust control of the plant so that the average of (10) is reduced.

component-wise to elements of the vector $\mathbf{y}_{O_{pt}}(t)$. The controller produces the vector $\mathbf{y}_{O_{cr}}(t)$ as a function of the input vector $\mathbf{y}_{I_{cr}}(t)$, its internal state $\mathbf{z}_{cr}(t)$, and the vector of controller weights \mathbf{W} .

In Fig. 2, the plant block has a dual meaning. It is not only the actual system to be controlled but also its models employed to train a recurrent neurocontroller for robustness, as discussed in this paper.

The desired output $\mathbf{y}_{O_{rm}}(t)$ of the plant is given by the output of a stable reference model which is specified as a function of the reference signals $\mathbf{y}_{I_{rm}}(t)$ and the internal state $\mathbf{z}_{rm}(t)$ of the reference model. The simplest possible reference model is just the time-delay operator \mathbf{z}^{-d} , providing time shifts of components of the vector $\mathbf{y}_{I_{rm}}(t)$, in order to correctly compute the error vector $\boldsymbol{\xi}(t) = \mathbf{y}_{O_{rm}}(t) - \mathbf{y}_{O_{pt}}(t)$. Such shifts are necessary to reflect causality and properly account for internal delays always present in the plant model.

Our neurocontroller is implemented as an RNN with `n_nodes` in the pseudocode below. The pseudocode is general, and it can be used to describe a large variety of computational structures including networks with layers, nonrecurrent networks, polynomials, etc. After setting up several required vectors and matrices describing the network architecture (i.e., `n_con` specifies the number of connections per node, $c_{i,j}$ is an element of the connectivity matrix \mathbf{c} specifying which node j is connected to node i , with the corresponding delay $d_{i,j}$ of the link specified in the delay matrix \mathbf{d} ; often $d_{i,j} \in \{0, 1\}$), the network execution is compactly expressed in pseudocode 1 [26]:

```

for i = 1 to n_nodes {
  if n_con(i) > 0 {
    a_i(t) = [
      sum_{j=1}^{n_con(i)} W_{i,j}(t) y_{c_{i,j}}(t - d_{i,j}),
      if i is additive;
      prod_{j=1}^{n_con(i)} (W_{i,j}(t) + y_{c_{i,j}}(t - d_{i,j})),
      if i is multiplicative.
    ]
    y_i(t) = f_i(a_i(t))
  }
}

```

Most commonly, we take the activation function $f_i(\cdot)$ to be a bipolar sigmoid, though we also can make use of other functions, e.g., linear or sinusoids, for special purposes. Furthermore, some $y_{c_{i,j}}$ are controller inputs $\mathbf{y}_{I_{cr}}(t)$, others form its internal state $\mathbf{z}_{cr}(t)$, and still others serve as controller outputs $\mathbf{y}_{O_{cr}}(t)$.

The neurocontroller weights can be adjusted using either gradient-based methods (less effective, first-order methods, e.g., the gradient descent, or more effective, second-order methods, e.g., EKF with BPTT truncated after h_d time steps [27]) or derivative-free methods. We choose the second-order derivative-free method to be described because: 1) it obviates the need for computing derivatives in (possibly) complex and not readily differentiable closed-loop systems, and 2) it provides more accurate treatment of nonlinearities than either the EKF or any other estimation technique available today (see [1]), while remaining computationally competitive. The recurrent neurocontroller is trained to minimize the root-mean-square (rms) error

$$\begin{aligned} & \sqrt{\frac{1}{T} \sum_{t=1}^T \xi^T(t) \Lambda \xi(t)} \\ &= \sqrt{\frac{1}{T} \sum_{t=1}^T \sum_{i \in N_{\text{out}}} \lambda_i \left(y_{i,O_{rm}}(t) - y_{i,O_{pt}}(t, \mathbf{W}, \boldsymbol{\theta}(k)) \right)^2} \end{aligned} \quad (10)$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_{N_{\text{out}}})$ is the output weighting matrix (often set as the identity \mathbf{I} , as in our examples of Section III), $\boldsymbol{\theta}(k)$ is the vector of plant parameters, and T is sufficiently large [cf. (7) in Section I]. (A penalty for excessive control efforts can be imposed by employing feedthrough connections: $y_{j,O_{pt}}(t) = y_{j,O_{cr}}(t)$.) In an adaptive problem setting, the controller weights \mathbf{W} are time-varying to compensate for uncertain $\boldsymbol{\theta}$. In our robust setting, the controller weights \mathbf{W} are fixed, but the controller itself is an RNN trained to reduce both the average and the maximum of (10) over all possible $\boldsymbol{\theta}$.

Our approach is applicable to control of very general plants. To account for parametric uncertainties, we augment the common state-space description of the plant with the vector of plant parameters $\boldsymbol{\theta}$ [cf. (2) in Section I]

$$\begin{aligned} \mathbf{z}_{pt}(t) &= \boldsymbol{\phi}_1(\mathbf{z}_{pt}(t-1), \mathbf{y}_{O_{cr}}(t), \boldsymbol{\theta}_1(k), \boldsymbol{\alpha}(t)) \\ \mathbf{y}_{O_{pt}}(t) &= \boldsymbol{\phi}_2(\mathbf{z}_{pt}(t), \boldsymbol{\theta}_2(k), \boldsymbol{\beta}(t)) \end{aligned} \quad (11)$$

where $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are concatenated to form $\boldsymbol{\theta}$, and $\boldsymbol{\theta}(k)$ changes no earlier than after T time steps t (slowly, continuously changing $\boldsymbol{\theta}$ is also admissible). Different components of $\boldsymbol{\theta}$ may only be known to within component specific uncertainty bands. We do not assume models linear in parameters, as illustrated in Example 1. Usually (see, e.g., [8]) $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are assumed to be smooth vector functions. Such requirement is not necessary because we intend to use a derivative-free method of assessing sensitivities of plant outputs to its inputs. However, we do require that any discontinuities in $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ be bounded. Our approach is thus applicable to control of systems with common nonsmooth and discontinuous nonlinearities such as dead zones, saturations, and hystereses. In addition, many practical

systems, e.g., aerospace and automotive power trains, include lookup tables of various complexity, and these are also handled well by our approach, as shown in Example 2.

The process and/or actuator noise $\boldsymbol{\alpha}(t)$ and measurement noise $\boldsymbol{\beta}(t)$ in (11) can be modeled as random processes from normal or uniform distributions, which is the common assumption in the field. RNN controllers have been known to be significantly less sensitive to $\boldsymbol{\alpha}(t)$ and $\boldsymbol{\beta}(t)$ than other nonrecurrent neurocontrollers (see, e.g., [13] for RNN controllers and [15] for RNN in general) even without incorporating these noises into the training process. We confirm this observation, demonstrating robustness to measurement noise in Examples 1 and 2. We can also readily incorporate $\boldsymbol{\alpha}(t)$ and $\boldsymbol{\beta}(t)$ into our training for robustness, if necessary.

Though we stress the robustness to parametric uncertainty, additional uncertainties can come in a form of structural uncertainty. For example, different modes of plant operation can be described by models of different order. In each moment t , our knowledge of the operation mode (the structure of the plant model) can be uncertain

$$\begin{aligned} \mathbf{z}_{pt,k}(t) &= \boldsymbol{\phi}_{1k}(\mathbf{z}_{pt,k}(t-1), \mathbf{y}_{O_{cr}}(t)) \\ \mathbf{y}_{O_{pt}}(t) &= \boldsymbol{\phi}_{2k}(\mathbf{z}_{pt,k}(t)). \end{aligned} \quad (12)$$

In this case, the index k defines the mode, rather than the vector of parametric uncertainty $\boldsymbol{\theta}(k)$ in (11). We can train a recurrent neurocontroller by sampling different modes as if they were different instances of the parameter vector $\boldsymbol{\theta}$.

An RNN controller with outputs $\mathbf{y}_{O_{cr}}(t)$ implemented by pseudocode 1 is assumed to exist such that the rms error (10) for the plant model (11) or (12) is bounded for all $\boldsymbol{\theta}(k)$ or k , respectively. To achieve a decreased sensitivity of the recurrent neurocontroller to the uncertainties, we wish to utilize the most accurate plant models supplied to us externally (e.g., by system modelers) with specified ranges of uncertainties, regardless of the plant model differentiability.

B. Npr Method

The nprKF provides a much more accurate estimate of a Gaussian distribution evolution under a nonlinear transformation than that of the EKF [1]. This is done by subjecting the special vectors, which are derived from columns of the square root of the covariance matrix \mathbf{P} , to the same transformation. The nprKF is derived by replacing a Taylor expansion in the vicinity of the current weight vector with an expansion based on Stirling's formula for interpolating an analytic function over an interval. In one dimension, Stirling's formula may be obtained from the Taylor expansion by replacing derivatives by divided differences. Restricting attention to first and second orders, we can replace the first and second derivatives about \bar{x} , i.e., $f'(\bar{x})$ and $f''(\bar{x})$, with

$$f'_{DD}(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x}-h)}{2h} \quad (13)$$

$$f''_{DD}(\bar{x}) = \frac{f(\bar{x}+h) + f(\bar{x}-h) - 2f(\bar{x})}{h^2} \quad (14)$$

where h controls the interval $[\bar{x}-h, \bar{x}+h]$ for which the approximation is optimal. Norgaard, Poulsen, and Ravn (NPR) argue

that their approach is advantageous because of its more accurate treatment of the effects of nonlinearity on the Kalman filter recursion. An important side benefit is that nonlinearities need not be differentiable since derivatives are not employed.

We now describe application of the NPR method to neurocontroller training. The controller weights \mathbf{W} are denoted as a vector of trainable weights \mathbf{x} for consistency of notation with [1] and [2]. The vector \mathbf{x} is treated as the *state* to be estimated (the same interpretation is adopted in the EKF framework [27]). We denote the number of trainable weights as L . The L -dimensional covariance matrix \mathbf{P} at time step t is represented in the square root form

$$\mathbf{P}(t) = \mathbf{S}_x(t)\mathbf{S}_x^T(t) \quad (15)$$

where the matrix (square root factor) $\mathbf{S}_x(t)$ is also $L \times L$. The i th column of $\mathbf{S}_x(t)$ is denoted by $\mathbf{s}_{x,i}(t)$. We henceforth discard explicit notation of the time step for simplicity, wherever possible. From each such column vector, we form two variations of the current weight vector \mathbf{x} , i.e., $\mathbf{x} + h\mathbf{s}_{x,i}$ and $\mathbf{x} - h\mathbf{s}_{x,i}$, where h is such that $h^2 \geq 1$ ($h = \sqrt{3}$ is recommended for Gaussian and unbiased distributions of estimation errors)² [1]. We must compute system outputs for each of these variations.

Our system is the closed-loop system of Fig. 2 which includes a recurrent neurocontroller as its trainable part. We can describe the closed-loop system in the following pseudocode form (pseudocode 2):

```

for n = t - hd to t {
  Compute outputs  $\mathbf{y}_{O_{cr}}(n)$  of neurocontroller
  in Pseudocode 1 of Section II-A.

  Compute plant model outputs  $\mathbf{y}_{O_{pt}}(n)$  in
  (11).

  Compute reference outputs  $\mathbf{y}_{O_{rm}}(n)$  of the
  reference model (see Fig. 2).
}

```

Elsewhere [4], [27], we utilize derivatives of outputs with respect to RNN weights computed by truncated backpropagation through time, BPTT(h_d), where h_d is the truncation depth ($h_d = 0$ corresponds to static backpropagation). BPTT(h_d) estimates the change of outputs of the closed-loop system in response to a hypothetical change of RNN weights up to h_d time steps in the past. Dynamic derivatives obtained by BPTT(h_d) are used in [4] to adjust recurrent neurocontroller weights to minimize (10). In the nprKF, we can estimate effectively the same change by *repropagating* the closed-loop system, starting with step $t - h_d$, where t is the current step, for each set of RNN weight variations. That is why the state of the network (plus the state of the plant model and the state of the reference model, if applicable) at step $t - h_d$ must be saved just as in BPTT(h_d), so that the system can be initialized properly prior to each repropagation.

²The assumption about the underlying Gaussian distribution is also the standard EKF assumption [28]. We choose the constant h because the evolution of \mathbf{S}_x has sufficient influence on variations of \mathbf{x} , as it is affected by the learning parameters of the nprKF algorithm Q_0 and R_0 to be introduced in Section II-C.

We can compactly denote pseudocode 2 as the vector function $\mathbf{g}(\mathbf{x}, h_d)$, computing the outputs of the closed-loop system h_d time steps from the step $t - h_d$. We denote its j th output for the nominal weight vector as $g_j(\mathbf{x}, h_d)$, for variation $\mathbf{x} + h\mathbf{s}_{x,i}$ as $g_j(\mathbf{x} + h\mathbf{s}_{x,i}, h_d)$ and for variation $\mathbf{x} - h\mathbf{s}_{x,i}$ as $g_j(\mathbf{x} - h\mathbf{s}_{x,i}, h_d)$. The internal states of the recurrent neurocontroller, the plant model and the reference model, are provided as an additional implicit argument to $\mathbf{g}(\mathbf{x}, \cdot)$ and pseudocode 2. However, these serve only as memory to be stored temporarily, in exactly the same way it is done in the EKF training of RNN.

C. NprKF Recursion

Several working matrices must be set up at each step of the nprKF recursion. The matrix $\mathbf{S}_{xv}^{(1)}$ is $L \times L$ and diagonal, with diagonal elements equal to $Q_0^{(1/2)} \geq 0$ ($\mathbf{S}_{xv}^{(1)}\mathbf{S}_{xv}^{(1)T} = \mathbf{Q} = Q_0\mathbf{I}$, where Q_0 controls forgetting of past data). We denote the number of closed-loop system outputs as M (e.g., the dimensionality of $\mathbf{y}_{O_{pt}}$ in Fig. 2 is equal to M). The ij th element of the $M \times L$ matrix $\mathbf{S}_{yx}^{(1)}$ is given by

$$(\mathbf{S}_{yx}^{(1)})_{ij} = \frac{1}{2h}(g_j(\mathbf{x} + h\mathbf{s}_{x,i}, h_d) - g_j(\mathbf{x} - h\mathbf{s}_{x,i}, h_d)) \quad (16)$$

where $\mathbf{s}_{x,i}$ is the i th column of \mathbf{S}_x in (15).

The ij th element of the $M \times L$ matrix $\mathbf{S}_{yx}^{(2)}$ is given by

$$(\mathbf{S}_{yx}^{(2)})_{ij} = \frac{(h^2 - 1)^{\frac{1}{2}}}{2h^2}(g_j(\mathbf{x} + h\mathbf{s}_{x,i}, h_d) + g_j(\mathbf{x} - h\mathbf{s}_{x,i}, h_d) - 2g_j(\mathbf{x}, h_d)). \quad (17)$$

We mainly adopt the notation of [1] and [2] for consistency.

The computational cost of the original nprKF recursion scales with L^3 [1], which is impractical for neural networks with many weights. Much more efficient formulation ($O(L^2M)$) is proposed in [2] (typically $M \ll L$), which is also implemented for experiments in this paper. We describe this more efficient recursion as follows.

We define the $M \times (M + L + L)$ concatenated matrix

$$\mathbf{S}_y = [\mathbf{S}_{y\omega}^{(1)} \quad \mathbf{S}_{yx}^{(1)} \quad \mathbf{S}_{yx}^{(2)}] \quad (18)$$

where the component matrices are assembled according to (16) and (17), and the matrix $\mathbf{S}_{y\omega}^{(1)}$ is $M \times M$ and diagonal, with diagonal elements equal to $R_0^{(1/2)} > 0$ ($\mathbf{S}_{y\omega}^{(1)}\mathbf{S}_{y\omega}^{(1)T} = \mathbf{R} = R_0\mathbf{I}$, where R_0 is the inverse learning rate). Weighted average outputs of the closed-loop system to be used in the RNN weight update are calculated from

$$\bar{\mathbf{y}}_j = \frac{h^2 - L}{h^2}g_j(\mathbf{x}, h_d) + \frac{1}{2h^2} \sum_{i=1}^L [g_j(\mathbf{x} + h\mathbf{s}_{x,i}, h_d) + g_j(\mathbf{x} - h\mathbf{s}_{x,i}, h_d)]. \quad (19)$$

Then, the weight update is given by

$$\mathbf{x}_+ = \mathbf{x} + \mathbf{K}(\mathbf{y} - \bar{\mathbf{y}}) \quad (20)$$

where \mathbf{K} is the Kalman gain [28], \mathbf{y} is the target vector (e.g., $\mathbf{y}_{O_{rm}}$ in Fig. 2), and the “+” subscript denotes the after-update value.

The Kalman gain is determined from the following factored form:

$$\begin{bmatrix} \mathbf{S}_{y\omega}^{(1)} & \mathbf{S}_{yx}^{(1)} & \mathbf{S}_{yx}^{(2)} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_x & \mathbf{0} & \mathbf{S}_{x\nu}^{(1)} \end{bmatrix} \begin{bmatrix} \mathbf{S}_{y\omega}^{(1)T} & \mathbf{0} \\ \mathbf{S}_{yx}^{(1)T} & \mathbf{S}_x^T \\ \mathbf{S}_{yx}^{(2)T} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{x\nu}^{(1)T} \end{bmatrix} = \begin{bmatrix} \mathbf{S}_y & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{KS}_y & \mathbf{S}_{x+} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{S}_y^T & \mathbf{S}_y^T \mathbf{K}^T \\ \mathbf{0} & \mathbf{S}_{x+}^T \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (21)$$

which utilizes (15) and (18). We make use of the matrix factorization lemma [28], which implies that (21) holds if and only if there exists a unitary matrix Φ such that

$$\begin{bmatrix} \mathbf{S}_{y\omega}^{(1)} & \mathbf{S}_{yx}^{(1)} & \mathbf{S}_{yx}^{(2)} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_x & \mathbf{0} & \mathbf{S}_{x\nu}^{(1)} \end{bmatrix} \Phi = \begin{bmatrix} \mathbf{S}_y & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{KS}_y & \mathbf{S}_{x+} & \mathbf{0} & \mathbf{0} \end{bmatrix}. \quad (22)$$

The matrix Φ is assembled implicitly by carrying out Givens rotations [29] to annihilate the appropriate blocks of the matrix on the left-hand side of (22) such that it has the same structure as the right-hand side. We then identify the nonzero blocks with those on the right-hand side. We recover the updated square root \mathbf{S}_{x+} of the covariance matrix \mathbf{P} (15) and lower triangular \mathbf{S}_y . Once \mathbf{K} is obtained (through backsubstitution [29]), the weights are updated according to (20).

The dominant computational term of operations of the new recursion scales with L^2M , provided that the Givens rotations are performed in the following order. We first annihilate the $\mathbf{S}_{yx}^{(2)}$ block, then the $\mathbf{S}_{yx}^{(1)}$ block. Within blocks, we proceed from left to right and top to bottom. In each case, the Givens rotations involve elements of the leftmost blocks. The block that began as \mathbf{S}_x should end up as upper triangular. However, rigorous annihilation of $\mathbf{S}_{x\nu}^{(1)} > 0$ still requires operations that scale with L^3 . To avoid such prohibitively expensive scaling, we approximate the full annihilation of the $\mathbf{S}_{x\nu}^{(1)}$ block by performing L quadrature additions to the diagonal of \mathbf{P} : $(\mathbf{S}_{x+})_{jj}^2 + Q_0$, $j = 1, 2, \dots, L$.

D. Summary of nprKF Algorithm for Controller Training

We summarize the nprKF application to training our recurrent neurocontroller for robustness in the following algorithmic form.

- 1) Initialize diagonal $\mathbf{S}_x(0)$, $\mathbf{S}_{x\nu}^{(1)}$, and $\mathbf{S}_{y\omega}^{(1)}$ to $P_0^{(1/2)}\mathbf{I}$, $Q_0^{(1/2)}\mathbf{I}$, and $R_0^{(1/2)}\mathbf{I}$, respectively, where P_0 , Q_0 , and R_0 are problem-dependent values (see Section III and [2] and [3] for example values).
- 2) Choose an instance of the plant model (e.g., initialize parameters of the model $\theta(j)$ from a distribution).
- 3) Choose a segment of the reference trajectory provided by the reference model starting at time $t - h_d$.
- 4) Initialize (or restore from step $t - h_d$) states of the plant model, the reference model, and the neurocontroller; set $n = t - h_d$.

- 5) For the neurocontroller weights \mathbf{x} and each of their $2L$ variations $\mathbf{x} \pm h\mathbf{s}_{x,i}$ and $h = \sqrt{3}$, $i = 1, 2, \dots, L$, do as follows.
 - a) Perform repropagation through the closed-loop system (forward propagation of signals through the controller, the plant model, and the reference model, as in pseudocode 2) from step n to step $n + h_d$ to obtain $\mathbf{g}(\mathbf{x}, h_d)$ and $\mathbf{g}(\mathbf{x} \pm h\mathbf{s}_{x,i}, h_d)$ and populate matrices (16) and (17).
 - b) Restore states of the closed-loop system from step n to prepare for the next repropagation.

- 6) Assemble the vector $\bar{\mathbf{y}}$ according to (19).
- 7) Carry out updates (22) and (20).
- 8) Move ahead by one time step: $t = t + 1$.
- 9) Continue from step 4) until the end of the reference trajectory segment.
- 10) Choose a new segment [go to step 3)] and/or a new instance of the plant model [go to step 2)] and continue training until a required level of performance, e.g., a sufficiently low rms error (10), is attained. During our training for robustness it is worth monitoring the mean of the rms errors taken over a representative selection of θ over time and stop the training when its progress becomes sufficiently slow.

We employ a *multistream* version of the aforementioned algorithm. A concept of multistream was proposed in [22] for improved training of RNN via EKF. It amounts to training N_s copies (N_s streams) of the same RNN with N_{out} outputs. Each copy has the same weights but different, separately maintained states. With each stream contributing its own set of outputs, every update (20) is based on information from all streams, with the total effective number of outputs increasing to $M = N_s N_{\text{out}}$. The multistream training is especially effective for heterogeneous data sequences because it counters the tendency to improve local performance at the expense of performance in other regions.

The multistream is naturally suitable for the nprKF algorithm, as shown in [2], and here we resort to the multistream nprKF for training RNN controllers for robustness. We assign a separate stream to each instance of the plant model (possibly, with its own segment of the reference trajectory) and carry out steps 2)–6) for all N_s streams. (All streams share the weights of the neurocontroller, while maintaining their differences of states of the closed-loop system components as well as parameters of the plant models.) We then execute steps 7) and 8) and repeat the multistream training process from step 9). Thus, our neurocontroller is trained simultaneously on N_s plant models.

There are two straightforward ways of training on multiple plant models. In the first approach, several sets of plant model parameters may be chosen purposefully and kept constant during training. In fact, different plant models may reflect distinct operation modes of the plant, as mentioned at the end of Section II-A. Alternatively, one vector of plant parameters $\theta(1)$ might describe mode 1 of the plant operation, another vector $\theta(2)$ might apply to mode 2, etc. When the number of modes is small (on the order of ten), we can assign an individual training stream to each mode, e.g., the plant model with $\theta(j)$ is assigned to the j th stream, $j = 1, 2, \dots, N_s$.

The second approach is applicable when the total number of possible plant variations is too large or even infinite. As-

suming no prior knowledge of parameter uncertainties except their ranges, we generate parameters of plant models at random. That is, we draw N_s samples $\theta(j)$ from a distribution Θ and keep these instances of the plant model assigned to their respective streams for training during some number of training epochs. Periodically, we draw new N_s samples from Θ and continue training on them.

Such training on randomly chosen plant models is akin to training with a continuous flow of data. Examples are never repeated. Making the most of each example is likely to be inappropriate. A recurrent neurocontroller trained for too long on the same small set of N_s plant models gets too specialized to this particular set. As discussed in Section I, the neurocontroller overfits and performs inadequately when tested on other examples not used in training. This phenomenon called *recency effect* has been observed previously for both RNN trained on data sequences and recurrent neurocontrollers (see, e.g., [22] and [27]). As our goal is to synthesize a robust neurocontroller, i.e., a neurocontroller which can deliver an acceptable performance for any example, we attempt to balance the duration and intensity of training on each selection of plant models with the duration of the entire training session and the total number of plant model selections utilized in training. We recognize that the optimal balance may always remain problem dependent, but we are encouraged by our results, thus far suggesting that an adequate balance can be achieved with a modest amount of experimentation, as demonstrated in Section III.

III. EXPERIMENTS

A. Example 1

Our first example is taken from [4] with the purpose to demonstrate that the nprKF is a viable alternative to the BPTT–EKF-based technique capable of attaining an improved level of performance, even when a plant model is differentiable. We train a neurocontroller for robustness to parametric disturbances of the multiple-input–multiple-output (MIMO) plant, which is a third-order system with two inputs and two outputs described by the state equations

$$\begin{aligned} y_{pt1}(t) &= \theta_1 y_{pt1}(t-1) \sin(\theta_2 y_{pt2}(t-1)) \\ &+ \left(\theta_3 + \theta_4 \frac{y_{pt1}(t-1) y_{pt4}(t)}{1 + y_{pt1}^2(t-1) y_{pt4}^2(t)} \right) y_{pt4}(t) \\ &+ \left(\theta_5 y_{pt1}(t-1) + \frac{\theta_6 y_{pt1}(t-1)}{1 + y_{pt1}^2(t-1)} \right) y_{pt5}(t) \end{aligned} \quad (23)$$

$$\begin{aligned} y_{pt2}(t) &= y_{pt3}(t-1) (\theta_7 + \theta_8 \sin(\theta_9 y_{pt3}(t-1))) \\ &+ \frac{\theta_{10} y_{pt3}(t-1)}{1 + y_{pt3}^2(t-1)} \end{aligned} \quad (24)$$

$$y_{pt3}(t) = (\theta_{11} + \theta_{12} \sin(\theta_{13} y_{pt1}(t-1))) y_{pt5}(t) \quad (25)$$

where y_{pt1} , y_{pt2} , and y_{pt3} are components of the plant state vector, and the plant inputs y_{pt4} and y_{pt5} are set equal to the control signals. The first two components of the state vector y_{pt1} and y_{pt2} are the plant outputs $y_{O_{pt1}}$ and $y_{O_{pt2}}$, respectively. The goal is to develop a neurocontroller such that the plant outputs follow

TABLE I
NOMINAL VALUES OF PARAMETERS θ FOR THE MIMO PLANT OF [30]

Param.	Value	Param.	Value	Param.	Value	Param.	Value
θ_1	0.9	θ_2	1.0	θ_3	2.0	θ_4	1.5
θ_5	1.0	θ_6	2.0	θ_7	1.0	θ_8	1.0
θ_9	4.0	θ_{10}	1.0	θ_{11}	3.0	θ_{12}	1.0
θ_{13}	2.0						

two independent reference model outputs $y_{O_{rm1}}$ and $y_{O_{rm2}}$ as closely as possible.

This control problem was originally proposed by [30]. The authors studied five cases of the problem formulation with constant values of the parameters θ_i , $i = 1, 2, \dots, 13$ assembled in the vector θ . These cases differed by the amount of information about the plant available to the control designer. For example, in one case all state variables were accessible, and the plant equations were known. In [4], we reformulated this control problem for the case of uncertain θ (parametric disturbances), developed several controllers based on RNN and compared them with neurocontrollers based on feedforward networks.

Table I contains nominal values of the plant parameters θ . We allow each components of θ to be a random variable uniformly distributed around its nominal value in the range $\pm 20\%$. We first consider the case in which all state variables are accessible and plant equations are known (the full state feedback case); then, we provide the results for the partial state feedback case.

Our training setup (i.e., the RNN architecture, the type of reference trajectories, and the total number of training epochs) is the same as the one employed in [4] except that we use the nprKF algorithm with five streams ($h_d = 10$). We begin with $P_0 = 10^{-5}$ (value on the diagonal of \mathbf{P}), $R_0 = P_0$, and $Q_0 = 10^{-7} P_0$. After 300 epochs (about 30 000 weight updates), we switch to $R_0 = 0.05 P_0$ for another 300 epochs, followed by another 300 epochs with $R_0 = 0.005 P_0$. We complete the training process with another set of 500 epochs with $R_0 = 0.0005 P_0$.

Upon completion of training, we test the resulting neurocontroller on various reference signals and on many plant models with different parameters θ independently generated from the same distribution as that of the training set of plant models. Fig. 3 shows tracking results for the nominal plant model on the same set of reference signals as those in [30]. Fig. 4 compares our performance with performance of the BPTT–EKF-trained neurocontroller of [4] on 10 000 plant models for the reference signals of Fig. 3. Similar comparative results in favor of the nprKF-trained neurocontroller are obtained for 1 000 000 plant models.

Our performance (the average tracking rms error) is significantly better than performance of the neurocontroller trained on the nominal system. In fact, for several instances of the plant model (about 0.03%) the nominal neurocontroller fails to stabilize the closed-loop system altogether. (This system is known to be notoriously unstable to inadequate controls, and its state variables can reach unbounded growth in just a few time steps.) It is worthwhile to note that another derivative-free optimization approach [simultaneous perturbation stochastic approximation (SPSA)], [31], apparently encounters serious difficulties when applied to training a neurocontroller for this problem because

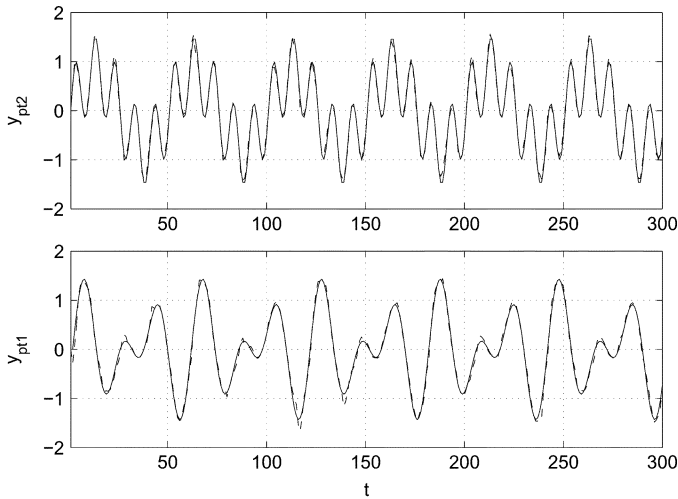


Fig. 3. Tracking of the reference signals on the nominal plant model. The reference signals are solid and the plant outputs are dashed. The rms error (10) is 0.109.

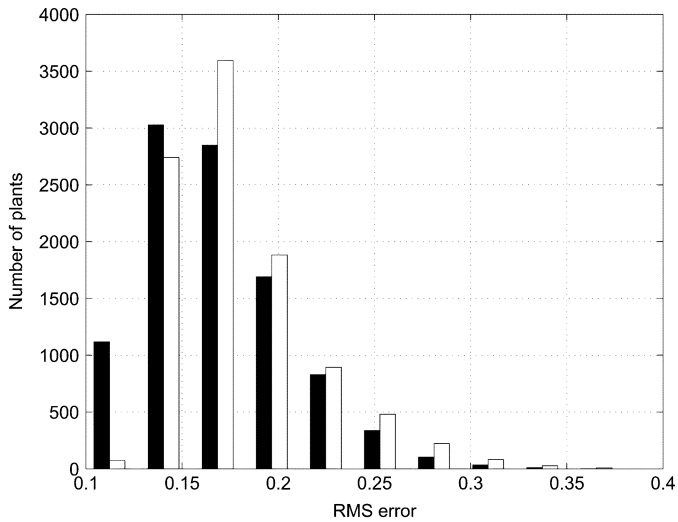


Fig. 4. Comparison of tracking rms errors (10) over 10 000 plant models for the same reference trajectory as in Fig. 3. The black bars are for the nprKF-trained neurocontroller (the mean rms error is 0.168; the maximum rms error is 0.357), the white bars are for the BPTT-EKF-trained neurocontroller of [4] (the mean rms error is 0.179; the maximum rms error is 0.379).

the plant is easily destabilizable, especially early in the training process.

Interestingly, subjecting our trained recurrent neurocontroller to the measurement noise (a uniform random distribution in the range ± 0.025), in addition to the perturbations of θ , changes the histogram of Fig. 4 very little. The mean rms error becomes 0.174 and the maximum rms error is 0.364 versus the mean rms error 0.184 and the maximum rms error 0.393 for the performance of the BPTT-EKF-trained neurocontroller of [4] in the same setting.

Though we considered so far the full state feedback control, our conclusions regarding the improved performance of the nprKF-trained neurocontroller also hold in the case of the partial state feedback control. Specifically, for the recurrent neurocontroller of the same architecture but with four inputs (y_{pt1} , y_{pt2} , $y_{O_{rm1}}$, and $y_{O_{rm2}}$) the mean rms error is 0.191 and the maximum rms error is 0.361, which is better than the performance

of the the BPTT-EKF-trained neurocontroller of [4] (the mean rms error is 0.196 and the maximum rms error is 0.405).

Our experimental conclusions are also confirmed by repeating the nprKF training process for different initial weights of the neurocontroller. It helps to test several choices when dealing with processes with random components. A useful variation of the described process of training for robustness is to test some intermediate neurocontrollers (e.g., those with reasonably good training rms errors attained before the specified number of epochs is completed) on a large set of plant models and decide whether to stop or continue the training process further.

B. Example 2

Our second example deals with a problem of electronic throttle control (ETC). The ETC is gaining popularity in the automotive industry due to its capabilities for achieving improvements in fuel economy, drivability, and other crucial performance factors. In conventional vehicles, the driver pedal is linked to the engine throttle mechanically. The ETC vehicles are “drive-by-wire” vehicles, meaning that the throttle is driven by an electric motor controlled electronically through an appropriate interpretation of the driver pedal position.

The ETC is an electromechanical system consisting of a direct current (dc) motor, a gear mechanism, and a throttle valve with a dual spring system. In the neutral position, both springs are relaxed, and throttle valve is slightly open. This is called the “limp-home” position, and it is critical in case of the power failure allowing the engine to operate in a low power mode. The two springs have substantially different stiffness. The first spring affects the throttle valve motion at angles exceeding the “limp-home” angle, whereas the second spring counteracts the motor torque for angles smaller than the “limp-home.” The second spring’s stiffness must be higher to provide higher angular resolution at small angles.

The throttle position is measured by a potentiometer. A sufficiently accurate angular control of the throttle valve’s plate between 0.14–1.57 rad is required (between 8° and 90°, respectively).

The ETC model consists of several components (Fig. 5). It includes the dc motor dynamics, with the armature time constant T_a , the armature gain K_a , the emf constant K_v , and the torque constant K_t . Our control signal is the motor voltage u , and it is limited between u_{min} and u_{max} . Further, there is a complicated model of friction due to ball bearings and the gear mechanism (one of the most accurate is the LuGre friction model [32] also used in this paper). The gear ratio is K_l , and the overall inertia is J . Finally, the dual spring system is modeled in the form of a lookup table.

The ETC model has been validated experimentally and demonstrated to be a very accurate description of the real hardware, provided that the model parameters included as components in the uncertainty vector θ (15 components including parameters of the friction and spring models) could be estimated with high accuracy (to within a few percent from their true values). It is projected that the massive use of the ETC in automotive industry in the near future will expedite the utilization of relatively cheap components with substantial

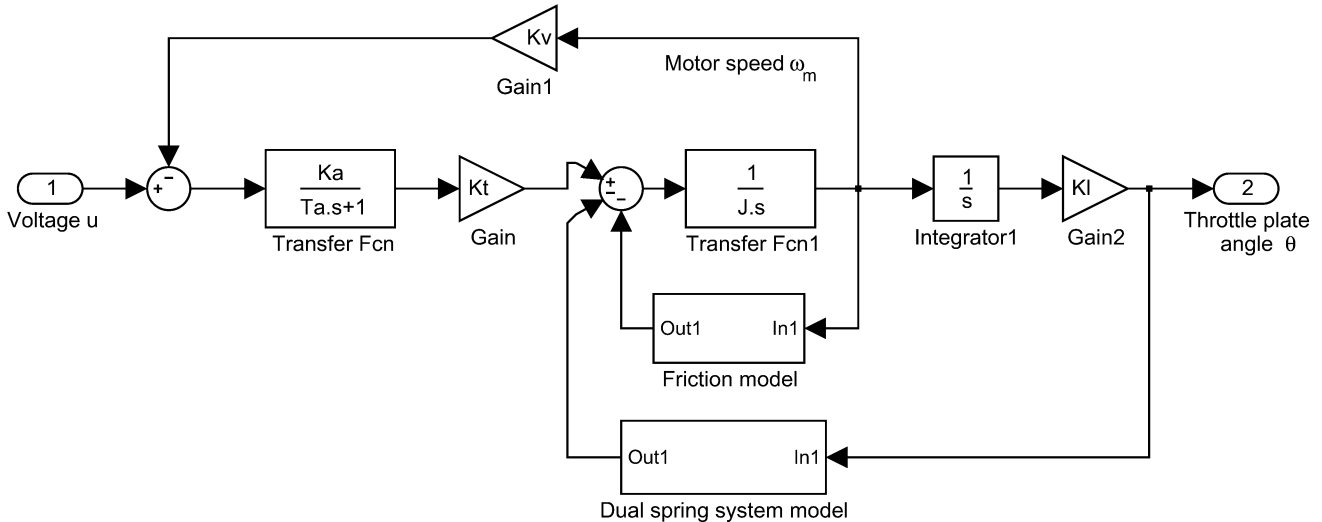


Fig. 5. Block diagram of electronic throttle. The controller (not shown) senses the throttle valve position θ , (and, possibly, the motor speed ω_m) and puts out the voltage u .

spread of parameters around their nominal values. For example, the friction model parameters or the spring stiffness may have their true values significantly different from nominal, or they may deviate significantly during the throttle service time. The ETC pervasiveness will also mean that it is too costly to calibrate any model-based control algorithm with fixed parameters. On the other hand, an adaptive control algorithm may be very difficult to design because it has to cope with too many uncertainties, entering the control equations nonlinearly. We illustrate how to employ a robust neurocontroller trained via the nprKF algorithm for the ETC problem.

The ETC closed-loop system is modeled with the fixed time step of 0.1 ms. For our ETC experiments in this paper, we specify the uncertainty ranges for all 15 ETC parameters as $\pm 20\%$ around their nominal values, although the uncertainties might eventually be set to parameter specific (but currently unavailable) values. When the control sampling rate $T_s = 1$ ms, the motor speed is measured or inferred (e.g., from the first difference of angles), and no angular measurement errors are present, a very accurate position control is achieved easily with a small RNN using the nprKF training algorithm for any set of plant model parameters kept constant throughout the neurocontroller training (see Fig. 6 for typical results; the network 3-2R-1, which stands for the architecture with three inputs, a fully recurrent hidden layer of two nodes with the one-step feedback delays, and one output, has 15 weights).

Even with the small neurocontroller, the control sampling rate $T_s = 1$ ms might be too demanding for the hardware implementation. When T_s increases and, in addition, the motor speed is not measured, the tracking accuracy gradually decreases, which is not surprising considering the growing control delay, but controllability is still maintained up to $T_s = 10$ ms with a satisfactory level of accuracy. For this most challenging sampling rate, we carry out the nprKF training for robustness. We choose a larger two-hidden-layer RNN, 2-5R-3-1 (62 weights), as a reasonable tradeoff between its capabilities and complexity of on-board implementation. Sim-

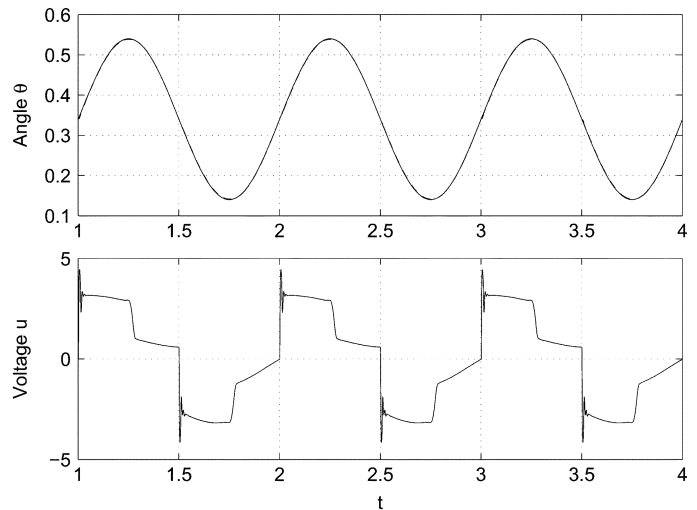


Fig. 6. Tracking of a 4-s segment of the sin function around the “limp-home” angle. The reference signal is solid, and the plant output is dashed (top panel). The bottom panel shows the corresponding output of the neurocontroller. The rms error (10) is 0.002 rad, and instantaneous errors are negligible. The sampling for control values T_s is 1 ms. The controller is a 3-2R-1 RNN [the three inputs are the current angle, its desired (reference) value, and the motor speed]. The rms error deteriorates slightly (to 0.003 rad) when no speed measurements (or inference based on the first difference of angles) are provided to the controller.

ilar to the example of Section III-A, we train according to the six-stream nprKF algorithm ($h_d = 10$) in which each stream is assigned to a particular instantiation of the ETC model, with parameters drawn from a uniform random distribution around their nominal values. Similar to [4], our training reference trajectory is chosen as random levels between 0.14–1.57 rad maintained for a random duration between 0.25–1.0 s. Changes between levels are commanded as ramps up or down, with the rise or fall limits consistent with physical limitations of the real ETC. Each stream is assigned to its own 100-point segment of the reference trajectory, with the starting point chosen at random. All parameters of every ETC model are redrawn every

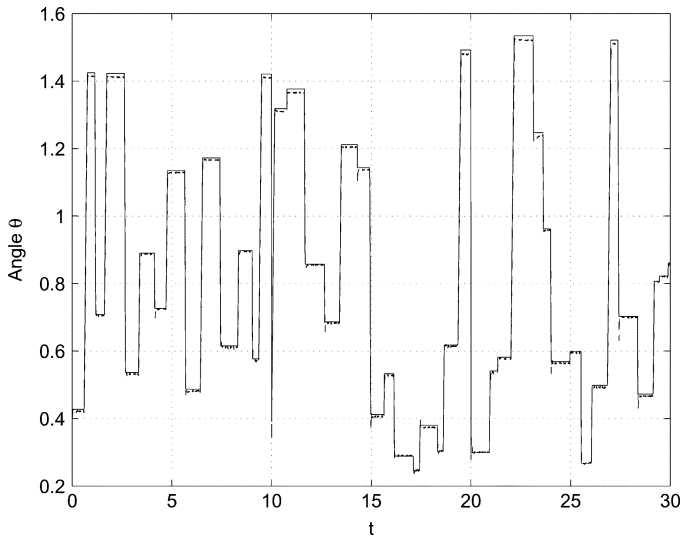


Fig. 7. Our typical tracking performance of the 2-5R-3-1 neurocontroller trained for robustness with $T_s = 10$ ms (its inputs are the current angle and the desired angle). A 30-s segment of the reference signal is solid, and the plant output is dashed. The measured angle is corrupted by noise due to potentiometer errors of no larger than 0.0009 rad. The rms tracking error (10) is 0.022 rad (within 1% of the noise-free tracking rms error). The neurocontroller seems to be very insensitive to the measurement noise, which is crucial for practical implementation.

five training epochs, each epoch consisting of processing all 100 points for all streams. We train for 200 epochs total (about 20 000 weight updates) with $P_0 = 10^{-4}$, $R_0 = 0.001P_0$, and $Q_0 = 10^{-4}P_0$. The initial value P_0 on the diagonal of the \mathbf{P} matrix is larger than that in Example 1 due to the presence of the friction model discontinuity (even larger P_0 values may be useful for successful training with wider discontinuities).

Typical tracking results after training are shown in Fig. 7 when tested with an angular measurement noise (the amplitude of 0.05°) not present during training. The maximum steady state angular error is less than 1° , which is favorable.

We compare our results of training for robustness with those of the same recurrent neurocontroller trained on the nominal ETC model via the nprKF algorithm. We test both controllers on the same reference trajectory and 10 000 ETC models. On average, our tracking rms error (0.024 rad) is 50% smaller than the rms error of the nominal neurocontroller (0.036). In addition, the maximum rms error of the recurrent neurocontroller trained for robustness is significantly smaller (0.083) than that of the nominal recurrent neurocontroller (0.131), as shown in Fig. 8. Further improvement to robustness may also be achieved, but only at the expense of a significantly increased size of the RNN controller not practical for on-board implementation. The performance advantages of our recurrent neurocontrollers trained for robustness also hold for other reference trajectories.

IV. CONCLUDING REMARKS

This paper introduces a new method for practical controller synthesis. We choose a sufficiently accurate model of the plant (e.g., reflecting all major physical phenomena) and make a practically sound assumption that the plant parameters are known only in some ranges around their nominal values. We train a

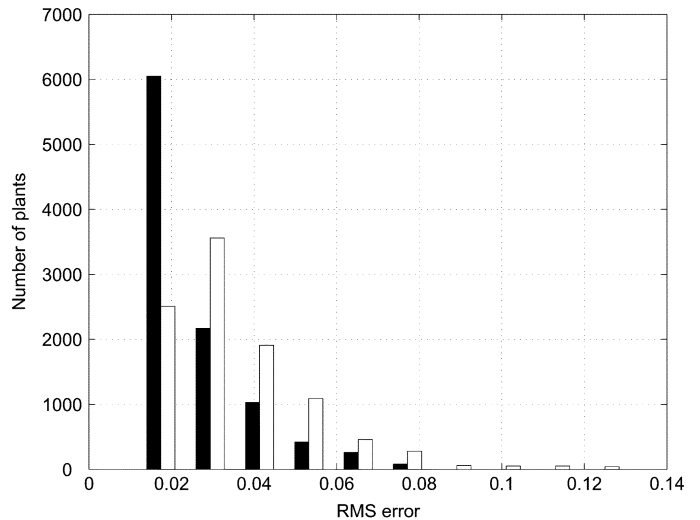


Fig. 8. Comparison of tracking rms errors (10) over 10 000 ETC plant models for the same reference trajectory as in Fig. 7. The black bars are for the neurocontroller trained for robustness by the nprKF algorithm (the mean rms error is 0.024 and the maximum rms error is 0.083), the white bars are for the nominal neurocontroller (the mean rms error is 0.036 and the maximum rms error is 0.131).

neurocontroller on many instances of plant models with different parameters. We apply a computationally efficient form of the derivative-free Kalman filter algorithm (nprKF) to training neurocontrollers for robustness in the model reference control setting. Our neurocontrollers are discrete-time RNNs to be deployed with fixed weights. They are known to be very flexible computational structures capable of exhibiting a rich spectrum of behaviors. We rely on the power of RNN and effectiveness of the nprKF algorithm to synthesize (train) controllers which are able to handle gracefully modeling uncertainties in nontrivial control problems.

In Examples 1 and 2, one can probably show that knowing the exact values of plant model parameters θ is sufficient to develop a nonlinear (nonneural) controller with accurate tracking of any reasonable reference trajectory. However, such a controller assumes too much prior knowledge which is not available in the robust problem setting of this paper. Any theoretical analysis with RNN in the control loop stumbles upon the fact that, even by itself, RNN is a dynamical system highly nonlinear in parameters. Such systems are notoriously difficult to analyze (see, e.g., [33] and [34]). Meaningful performance guarantees applicable to RNN training for robustness are yet to be developed.

A likely limitation to the classes of plant models to be controlled with neural networks can come from the following consideration. Neural networks of various architectures are proven to be universal approximators of mappings with bounded first absolute moments of their Fourier transforms (see, e.g., [35]). In practice this means that the higher the dimensionality of ϕ_1 and ϕ_2 , the smoother these functions should be for the same accuracy of the approximation. Thus, high-dimensional neurocontrollers may run into difficulties when dealing with plants with a large number of nonsmooth and discontinuous nonlinearities, but this is the limitation of the chosen controller parameterization, rather than the training method proposed in this paper.

As compared to the EKF algorithm, the nprKF recursion described here has an extra complexity due to multiple repropagations through the closed-loop system \mathbf{g} for RNN weight variations, resulting in the cost $O(h_d L^2)$. It may be possible to offset this by increasing the cost of the nprKF recursion, as discussed in [3], but further research to reduce the complexity of the nprKF algorithms is warranted.

ACKNOWLEDGMENT

The author would like to thank the anonymous reviewers and M. Samples for their comments, his former colleagues L. Feldkamp and T. Feldkamp for helpful discussions and past contributions, as well as M. Baotic, I. Petrovic, and M. Jansz for their input regarding the ETC problem.

REFERENCES

- [1] M. Norgaard, N. K. Poulsen, and O. Ravn, "New developments in state estimation for nonlinear systems," *Automatica*, vol. 36, pp. 1627–1638, 2000.
- [2] L. A. Feldkamp, T. M. Feldkamp, and D. V. Prokhorov, "Neural network training with the nprKF," in *Proc. Int. Joint Conf. Neural Netw.*, Washington, D.C., 2001, pp. 109–114.
- [3] L. A. Feldkamp, T. M. Feldkamp, and D. V. Prokhorov, "Recurrent neural network training by nprKF joint estimation," in *Proc. Int. Joint Conf. Neural Netw.*, Hawaii, 2002, pp. 2086–2091.
- [4] D. V. Prokhorov, G. V. Puskorius, and L. A. Feldkamp, J. Kolen and S. Kremer, Eds., "Dynamical Neural Networks for Control," in *A Field Guide to Dynamical Recurrent Networks*. Piscataway, NJ: IEEE Press, 2001, pp. 257–289.
- [5] T. Hayakawa, W. Haddad, J. Bailey, and N. Hovakimyan, "Passivity-based neural network adaptive output feedback control for nonlinear nonnegative dynamical systems," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 387–398, Mar. 2005.
- [6] D. Wang and J. Huang, "Neural network-based adaptive dynamic surface control for a class of uncertain nonlinear systems in strict-feedback form," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 195–202, Jan. 2005.
- [7] D. White and D. Sofge, Eds., *Handbook of Intelligent Control*. New York: Van Nostrand, 1992.
- [8] K. S. Narendra, "Neural networks for control: Theory and practice," *Proc. IEEE*, vol. 84, no. 10, pp. 1385–1406, Oct. 1996.
- [9] T. Hrycej, *Neurocontrol: Towards an Industrial Control Methodology*. New York: Wiley, 1997.
- [10] M. Agarwal, "A systematic classification of neural-network-based control," *IEEE Control Syst. Mag.*, vol. 17, no. 2, pp. 75–93, Apr. 1997.
- [11] O. Adetona, E. Garcia, and L. H. Keel, "A new method for control of discrete nonlinear dynamic systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 11, no. 1, pp. 102–112, Jan. 2000.
- [12] Q. Zhu and L. Guo, "Stable adaptive neurocontrol for nonlinear discrete-time systems," *IEEE Trans. Neural Netw.*, vol. 15, no. 3, pp. 653–662, May 2004.
- [13] J. Suykens, J. Vandewalle, and B. De Moor, *Artificial Neural Networks for Modeling and Control of Non-Linear Systems*. Norwell, MA: Kluwer, 1996.
- [14] I. Arsie, C. Pianese, and M. Sorrentino, "Nonlinear recurrent neural networks for air fuel ratio control in SI engines Soc. Automot. Eng. (SAE) Tech. Paper 2004-01-1364, 2004.
- [15] D. Mandic and J. Chambers, *Recurrent Neural Networks for Prediction*. New York: Wiley, 2001.
- [16] H.-G. Zimmermann, R. Neuneier, and R. Grothmann, A. Soofi and L. Cao, Eds., "Modeling of Dynamical Systems by Error Correction Neural Networks," in *Modeling and Forecasting Financial Data: Techniques of Nonlinear Dynamics*. Norwell, MA: Kluwer, 2002, pp. 237–263.
- [17] E. Wan and A. Nelson, "Dual Kalman Filtering Methods for Nonlinear Prediction, Estimation, and Smoothing," in *Advances in Neural Information Processing Systems 9*. Cambridge, MA: MIT Press, 1997.
- [18] E. N. Sanchez, A. Y. Alanis, and J. Rico, "Electric load demand prediction using neural networks trained by Kalman filtering," in *Proc. Int. Joint Conf. Neural Netw.*, Budapest, Hungary, 2004, pp. 2771–2776.
- [19] J. A. Perez-Ortiz, J. Schmidhuber, F. A. Gers, and D. Eck, "Improving long-term online prediction with decoupled extended Kalman filters," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, Berlin, Germany, 2002, pp. 1055–1060.
- [20] S. Younger, P. Conwell, and N. Cotter, "Fixed-weight on-line learning," *IEEE Trans. Neural Netw.*, vol. 10, no. 2, pp. 272–283, Mar. 1999.
- [21] D. Prokhorov, L. Feldkamp, and I. Tyukin, "Adaptive behavior with fixed weights in RNN: Overview," in *Proc. Int. Joint Conf. Neural Netw.*, Hawaii, 2002, pp. 2018–2022.
- [22] L. A. Feldkamp and G. V. Puskorius, "Training controllers for robustness: Multi-stream DEKF," in *Proc. Int. Joint Conf. Neural Netw.*, Orlando, FL, 1994, pp. 2377–2382.
- [23] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
- [24] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems containing neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Jan. 1990.
- [25] —, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 252–262, Mar. 1991.
- [26] D. Prokhorov, "Backpropagation through time and derivative adaptive critics: A common framework for comparison," in *Learning and Approximate Dynamic Programming*. Piscataway, NJ: IEEE Press, 2004.
- [27] L. A. Feldkamp and G. V. Puskorius, "A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering and classification," *Proc. IEEE*, vol. 86, no. 11, pp. 2259–2277, Nov. 1998.
- [28] S. Haykin, Ed., *Kalman Filtering and Neural Networks*. New York: Wiley, 2002.
- [29] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. New York: Cambridge Univ. Press, 1992.
- [30] K. S. Narendra and S. Mukhopadhyay, "Adaptive control of nonlinear multivariable systems using neural networks," *Neural Netw.*, vol. 7, no. 5, pp. 737–752, 1994.
- [31] J. C. Spall and J. A. Cristion, "A neural network controller for systems with unmodeled dynamics with applications to wastewater treatment," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 27, no. 3, pp. 369–375, Jun. 1997.
- [32] P. Dupont, V. Hayward, B. Armstrong, and F. Altpeter, "Single state elastoplastic friction models," *IEEE Trans. Autom. Control*, vol. 47, no. 5, pp. 787–792, May 2002.
- [33] N. Barabanov and D. Prokhorov, "A new method for stability analysis of nonlinear discrete-time systems," *IEEE Trans. Autom. Control*, vol. 48, no. 12, pp. 2250–2255, Dec. 2003.
- [34] I. Tyukin, D. Prokhorov, and V. Terekhov, "Adaptive control with non-convex parameterization," *IEEE Trans. Autom. Control*, vol. 48, no. 4, pp. 554–567, Apr. 2003.
- [35] A. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 930–945, May 1993.



Danil V. Prokhorov (SM'02) received the M.S. degree (with honors) in robotics from the State Academy of Aerospace Engineering (former LIAP), St. Petersburg, Russia, in 1992 and the Ph.D. degree in electrical engineering from Texas Tech University, Lubbock, in 1997.

He was with Ford Research Laboratory, Dearborn, MI, from 1997 until 2005. He has been engaged in application-driven studies of neural networks and their training algorithms. He is currently a Research Manager at Toyota Technical Center, Ann Arbor, MI. He

has authored 80 technical publications including several patents. His research interests are in machine learning algorithms and their applications to decision making under uncertainty.

Dr. Prokhorov was awarded the International Neural Networks Society (INNS) Young Investigator in 1999. He was the International Joint Conference on Neural Networks 2005 General Chair and International Joint Conference on Neural Networks 2001 Program Chair. He has been a reviewer for numerous journals and conferences, a program committee member of many conferences, and a panel expert for the National Science Foundation (NSF) every year since 1995.