2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

Training Neurocontrollers for Robustness via nprKF

Danil Prokhorov

Abstract— **We are interested in training neurocontrollers for robustness on discrete-time models of physical systems. Our neurocontrollers are implemented as recurrent neural networks. 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 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 this process yields a robust neurocontroller, which is ready for deployment with fixed weights.**

We employ a derivative-free Kalman filter algorithm proposed in [1] and extended in [2] and [3] 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 is similar in this sense to the approach in [4]. We illustrate it with two examples.

I. INTRODUCTION

Growing power of computers permits implementation of ever more sophisticated models and algorithms for control of various technological processes and mobile systems. Increasingly more elaborate and accurate dynamical models of physical systems to be controlled termed plants are being developed, often at a great cost, 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, deadzones, saturation 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 uncertainties. Adaptive controllers may not always be applicable, and additional, often costly model calibration efforts seem unavoidable, especially if high quality control is desired.

We would like to demonstrate that precise calibration of plant models is not necessary. Our approach is to train a neurocontroller on instances of plant models for robustness to uncertainties or changes of plant parameters. It has been shown in [5] that it is possible to train a neurocontroller in the form of a time-lagged recurrent neural network (RNN) 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 both [5] and this paper is to create a robust neurocontroller, i.e., a 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 instances of the plant models, the neurocontroller is supposed to be deployed with fixed weights, with no post-deployment adaptation, thereby bypassing a delicate issue of stable training neurocontrollers *on-line*[1].

In the framework of [4], plant models consist of differentiable components to enable training robust neurocontrollers via the extended Kalman filter (EKF) algorithm. In [4] backpropagation through time (BPTT) [7] is used 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 is designed to work with *any* plant models, differentiable or not, and it does not use BPTT. Our approach employs the derivative-free Kalman filter algorithm [1]. Our earlier work extended this algorithm to training RNN in [2] and [3], while improving its efficiency (we also proposed to call the algorithm of [1] the *nprKF*). In this paper we discuss the use of the nprKF to controller training.

This paper consists of four sections. In Section II we describe our method. In Section III we illustrate its application to a simple deadzone problem and an electronic throttle control problem, followed by conclusion in Section IV.

II. METHOD

A. Controller training with reference model

We follow the framework for neurocontrol established in [8], [9], and we adopt the viewpoint of model reference control, with the controller being an RNN [4].

Ford Research and Advanced Engineering, Dearborn, MI 48124, dprokhor@ford.com

The author is very grateful to his colleagues Lee and Tim Feldkamp for helpful discussions and past contributions.

[1]Stability analysis of closed-loop systems including RNN with fixed weights is beyond the scope of this paper. We assume that the closed-loop stability can always be verified in practice, similar to the viewpoint expressed in [6]. We agree with [6] that the ultimate verification of the control system performance including the closed-loop stability is done by passing field tests, irrespective of the availability of theoretical guarantees.

0-7803-9098-9/05/$25.00 ©2005 AACC

It is convenient to treat the entire closed-loop system as a heterogeneous RNN including its trainable part, the (neuro)controller. 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) or derivative-free methods. As mentioned in Section I, we wish to utilize the most accurate plant model available to us, regardless of its differentiability, for training a neurocontroller directly. We assume that the plant model is specified with some parametric uncertainties, and that the neurocontroller is to be trained to achieve a decreased sensitivity to such uncertainties (in addition to parametric, structural uncertainties and other disturbances can be handled as well). We choose the second-order derivative-free method to remain competitive with such a powerful training method as the EKF [10].

B. The NPR method

The nprKF (Kalman filter by Norgaard, Poulsen and Ravn (NPR)) 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 a 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

$$f'_{DD}(\bar{x}) = \frac{f(\bar{x}+h)-f(\bar{x}-h)}{2h} \quad (1)$$

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

where h controls the interval $[\bar{x}-h,\ \bar{x}+h]$ for which the approximation is optimal. 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 do not have to be differentiable since derivatives are not employed.

We now describe application of the NPR method to neurocontroller training. Our application, parameter estimation under the assumption of additive measurement and process noise, is a special case of NPR's more general formulation. We mainly adopt the notation of [1] and [2] for consistency.

The weight vector denoted by $\mathbf{x}$ is treated as the *state* to be estimated (the same interpretation is adopted in the EKF framework [10]). We denote the number of trainable weights as L and the number of closed-loop system outputs as M. The square root of the L-dimensional covariance matrix $\mathbf{P}$ is also $L \times L$ and is denoted by $\bar{\mathbf{S}}_x$. The ith column of $\bar{\mathbf{S}}_x$ is denoted by $\mathbf{s}_{x,i}$. From each such column vector we form two variations of the current weight vector $\mathbf{x}$, viz., $\mathbf{x}+h\mathbf{s}_{x,i}$ and $\mathbf{x}-h\mathbf{s}_{x,i}$, where $h=\sqrt{3}$ [1]. We must compute system outputs for each of these variations. We denote as $\mathbf{g}(\mathbf{x},0)$ and $\mathbf{g}(\mathbf{x},h_d)$ the closed-loop system output at the current time step and h_d time steps from the current step in the future, respectively. We denote the jth 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 $g_j(\mathbf{x}-h\mathbf{s}_{x,i},h_d)$. Additional (implicit) argument of $\mathbf{g}(\mathbf{x},\cdot)$ is the internal states of the neurocontroller (assuming an RNN as controller), the plant model and the reference model. However, these serve only as memory to be stored temporarily, similar to the way it is done in the EKF training of RNN. Elsewhere [4], [10] we utilize derivatives of outputs with respect to weights computed by truncated backpropagation through time, BPTT(h_d), where h_d is the truncation depth, and $h_d=0$ corresponds to static backpropagation. BPTT(h_d) estimates the change of the current network outputs from a hypothetical change of weights h_d time steps in the past. In the nprKF, we achieve effectively the same result by repropagating the network, starting with step $t-h_d$, where t is the current step, for each set of weight variations. That is why the state of the network (and the state of the plant model when dealing with control problems of dynamical systems, 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 network can be initialized properly prior to each repropagation.

Various working matrices must be set up at each step of the nprKF recursion. The matrix $\mathbf{S}^{(1)}_{x\nu}$ is $L\times L$ and diagonal, with diagonal elements equal to $Q_0^{\frac{1}{2}}\geq 0$. The ijth element of the $M\times L$ matrix $\mathbf{S}^{(1)}_{yx}$ is given by

$$(\mathbf{S}^{(1)}_{yx})_{ij}=\frac{1}{2h}(g_j(\mathbf{x}+h\mathbf{s}_{x,i},h_d) - g_j(\mathbf{x}-h\mathbf{s}_{x,i},h_d)) \quad (3)$$

The matrix $\mathbf{S}^{(1)}_{y\omega}$ is $M\times M$ and diagonal, with diagonal elements equal to $R_0^{\frac{1}{2}}>0$. Finally, the ijth element of the $M\times L$ matrix $\mathbf{S}^{(2)}_{yx}$ is given by

$$(\mathbf{S}^{(2)}_{yx})_{ij}=\frac{(h^2-1)^{\frac{1}{2}}}{2h^2}(g_j(\mathbf{x}+h\mathbf{s}_{x,i},h_d) + g_j(\mathbf{x}-h\mathbf{s}_{x,i},h_d)-2g_j(\mathbf{x},h_d)) \quad (4)$$

The superscripts (1) and (2) refer to first and second orders of approximation, as discussed in [1].

C. Two nprKF recursions

It is convenient to implement the nprKF algorithm[2] in the square root form [1]. We define the $M\times(M+L+L)$ concatenated matrix

$$\mathbf{S}_y = \left[\begin{array}{ccc}\mathbf{S}^{(1)}_{y\omega} & \mathbf{S}^{(1)}_{yx} & \mathbf{S}^{(2)}_{yx}\end{array}\right] \quad (5)$$

[2]This section contains a summary of [2].

where the component matrices are assembled according to (3) and (4). We reduce $\mathbf{S}_y$ to the lower triangular form, e.g., using the Householder method [12]. The outer product matrix $\mathbf{P}_y = \mathbf{S}_y\mathbf{S}_y^T$ is analogous to the matrix $\mathbf{R}+\mathbf{H^T P H}$ of the EKF [11], where $\mathbf{R} = \mathbf{S}_{y\omega}^{(1)}\mathbf{S}_{y\omega}^{(1)T}$ (the measurement noise covariance), and $\mathbf{H}$ is a matrix of derivatives of outputs with respect to adjustable parameters (as in BPTT).

The matrix

$$\mathbf{P}_{xy} \quad = \quad \bar{\mathbf{S}}_x\mathbf{S}_{yx}^{(1)T}. \tag{6}$$

is analogous to the matrix product $\mathbf{PH}$ of the EKF.

The Kalman gain $\mathbf{K}$ is obtained by solving

$$\mathbf{K}\mathbf{S}_y\mathbf{S}_y^T \quad = \quad \mathbf{P}_{xy} \tag{7}$$

As $\mathbf{S}_y$ is triangular, this is easily done by a succession of forward and back substitutions.

Weighted average outputs of the closed-loop system to be used in the weight update are calculated from

$$\bar{y}_j = \frac{h^2-L}{h^2}g_j(\mathbf{x},h_d) + \frac{1}{2h^2}\sum_{i=1}^{L}[g_j(\mathbf{x}+h\mathbf{s}_{x,i},h_d)+g_j(\mathbf{x}-h\mathbf{s}_{x,i},h_d)] \tag{8}$$

Then the weight update is given by

$$\mathbf{x}_+ \quad = \quad \mathbf{x}+\mathbf{K}(\mathbf{y}-\bar{\mathbf{y}}) \tag{9}$$

where $\mathbf{y}$ is the target vector (e.g., the desired plant output), and the plus subscript denotes the after-update value.

In the usual (not square root) implementation of the Kalman filter, the covariance matrix is updated as

$$\mathbf{P}_+ \quad = \quad \mathbf{P}-\mathbf{K}\mathbf{P}_y\mathbf{K}^T+\mathbf{Q} \tag{10}$$

where $\mathbf{Q} = \mathbf{S}_{x\nu}^{(1)}\mathbf{S}_{x\nu}^{(1)T}$ (the process noise covariance). In the square root formulation [1], we first assemble and triangularize the $L\times(M+L+L)$ matrix

$$\hat{\mathbf{S}}_x = \begin{bmatrix} \mathbf{K}\mathbf{S}_{y\omega}^{(1)} & \bar{\mathbf{S}}_x-\mathbf{K}\mathbf{S}_{yx}^{(1)} & \mathbf{K}\mathbf{S}_{yx}^{(2)} \end{bmatrix} \tag{11}$$

Then we assemble the $L\times(L+L)$ matrix

$$\bar{\mathbf{S}}_{x+} = \begin{bmatrix} \hat{\mathbf{S}}_x & \mathbf{S}_{x\nu}^{(1)} \end{bmatrix} \tag{12}$$

and triangularize it to complete the update of $\mathbf{P}_+ = \bar{\mathbf{S}}_{x+}\bar{\mathbf{S}}_{x+}^T$.

Unfortunately, the computational cost of the recursion above scales with L^3. It may be impractical for neural networks with many weights. Much more efficient formulation ($O(L^2M)$) is proposed in [2] (note that typically we have the number of outputs $M << L$), which is also used for experiments in this paper. We describe this, more efficient recursion below.

We consider the following factored form

$$\begin{bmatrix} \mathbf{S}_{y\omega}^{(1)} & \mathbf{S}_{yx}^{(1)} & \mathbf{S}_{yx}^{(2)} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{S}}_x & \mathbf{0} & \mathbf{S}_{x\nu}^{(1)} \end{bmatrix} \begin{bmatrix} \mathbf{S}_{y\omega}^{(1)T} & \mathbf{0} \\ \mathbf{S}_{yx}^{(1)T} & \bar{\mathbf{S}}_x^T \\ \mathbf{S}_{yx}^{(2)T} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{x\nu}^{(1)T} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{S}_y & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{K}\mathbf{S}_y & \bar{\mathbf{S}}_{x+} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{S}_y^T & \mathbf{S}_y^T\mathbf{K}^T \\ \mathbf{0} & \bar{\mathbf{S}}_{x+}^T \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{13}$$

which, when expanded, may be seen to reproduce expressions (5), (6), and (7). We make use of the matrix factorization lemma [11], which implies that (13) holds if and only if there exists a unitary matrix $\mathbf{\Theta}$ such that

$$\begin{bmatrix} \mathbf{S}_{y\omega}^{(1)} & \mathbf{S}_{yx}^{(1)} & \mathbf{S}_{yx}^{(2)} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{S}}_x & \mathbf{0} & \mathbf{S}_{x\nu}^{(1)} \end{bmatrix} \mathbf{\Theta} = \begin{bmatrix} \mathbf{S}_y & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{K}\mathbf{S}_y & \bar{\mathbf{S}}_{x+} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{14}$$

The matrix $\mathbf{\Theta}$ is assembled implicitly by carrying out Givens rotations [12] to annihilate the appropriate blocks of the matrix on the left hand side of (14) 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 of the covariance matrix, $\bar{\mathbf{S}}_{x+}$, and lower triangular $\mathbf{S}_y$. Once $\mathbf{K}$ is obtained (through backsubstitution), the weights are updated according to (9).

The dominant computational term of operations of the new recursion scales with L^2M, provided that the Givens rotations are performed carefully in a certain order [2]. However, rigorous annihilation of $\mathbf{S}_{x\nu}^{(1)} > 0$ still requires operations that scale with L^3. Instead of carrying out the full annihilation of the $\mathbf{S}_{x\nu}^{(1)}$ block, we may zero the diagonal elements only, ignoring the nonzero off-diagonal elements that arise as side effects of the Givens rotations (we only need to perform L quadrature additions). Thus, we avoid the unacceptable scaling with L^3 [2].

D. Summary of nprKF algorithm for controller training

We summarize the nprKF application to controller training in the algorithmic form:

0. Initialize diagonal $\bar{\mathbf{S}}_x$, $\mathbf{S}_{x\nu}^{(1)}$ and $\mathbf{S}_{y\omega}^{(1)}$ to $P_0^{\frac{1}{2}}\mathbf{I}$, $Q_0^{\frac{1}{2}}\mathbf{I}$ and $R_0^{\frac{1}{2}}\mathbf{I}$, respectively, where P_0, Q_0 and R_0 are problem dependent values (see Section III and [2] for example values).

1. Choose an instance of the plant model (e.g., randomly initialize parameters of the model).

2. Choose a segment of the reference trajectory provided by the reference model starting at time $t-h_d$.

3. Initialize (or restore from step $t-h_d$) states of the plant model, the reference model and the neurocontroller; set $k = t-h_d$.

4. For neurocontroller weights $\mathbf{x}$ and each of their variations $\mathbf{x}\pm h\mathbf{s}_{x,i}$, $h=\sqrt{3}$, $i=1,2,...,L$:

4a. Perform repropagation through the closed-loop system (forward propagation of signals through the controller, the plant model and the reference model) from step k to step $k+h_d$ to obtain $\mathbf{g}(\mathbf{x},h_d)$ and $\mathbf{g}(\mathbf{x}\pm h\mathbf{s}_{x,i},h_d)$ and populate matrices (3) and (4).

4b. Restore states of the closed-loop system from step k to prepare for the next repropagation.

5. Assemble the vector $\bar{\mathbf{y}}$ according to (8).
6. Carry out updates (14) and (9).
7. Move ahead by one time step; $t = t + 1$.
8. Continue from step 3 until the end of the reference trajectory segment.
9. Choose a new segment (go to step 2) and/or a new instance of the plant model (go to step 1) and continue training until a required level of performance, e.g., a sufficiently low root-mean-square (RMS) error, is attained.

We employ a *multi-stream* version of the algorithm above. A concept of multi-stream was proposed in [5] 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 (9) is based on information from all streams, with the total effective number of outputs increased to $M = N_s N_{out}$. The multi-stream training is especially effective for heterogeneous data sequences, and it counters the tendency to improve local performance at the expense of performance in other regions.

The multi-stream is naturally suitable for the nprKF algorithm, as shown in [2], and here we resort to the multi-stream nprKF for training 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 1 though 5 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 6 and 7 and repeat the multi-stream training process from step 8. Thus, our neurocontroller is trained simultaneously on N_s plant models.

There are two straightforward ways of training on multiple plant models. First, 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. They may differ not only in parameters but also structurally. Second, we can generate parameters of plant models at random, assuming no prior knowledge of parameter uncertainties except their ranges.

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 (e.g., minimizing the RMS error for a random selection of N_s plant models) may not be appropriate because it may well be detrimental to training on other examples (a manifestation of the well-known stability-plasticity dilemma). We intend 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 (see also [4]), 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 the next section.

III. EXAMPLES

Our first example is a simple control problem with the deadzone:

$$x_1(t+1) = x_2(t) \tag{15}$$
$$x_2(t+1) = x_2(t) - \alpha x_2(t)/(1 + x_1^2(t)) + u(t) \tag{16}$$

where

$$u(t) = \begin{cases} u(t) - \beta, & \text{if } u(t) - \beta > 0 \\ u(t) + \beta, & \text{if } u(t) + \beta < 0 \\ 0, & \text{otherwise} \end{cases}$$

In contrast with the example in [13], the parameters α and β are assumed to be uncertain, and they are drawn from the uniform random distributions in $[0.0938, 0.2813]$ and $[0.3, 0.9]$, respectively. Thus we have the case of infinitely many plants with parametric uncertainties. Our goal is to train a neurocontroller for robustness to uncertainties in α and β when tracking slowly-varying references. Our controller is the 3-5R-1 RNN which stands for the architecture with three inputs (two state variables and the reference x_{2d}), a fully recurrent hidden layer of five nodes, and one output. Applying our method to training our network with $N_s = 5$, $h_d = 10$, $P_0 = 0.05$, $R_0 = P_0$ and $Q_0 = 10^{-5} P_0$ for 300 epochs (one epoch is based on 250-point trajectory segments starting from $x_1(0)$ and $x_2(0)$ drawn from the uniform random distribution in $(-1, +1)$) results in the maximum RMS of 0.045 and the mean RMS error of 0.033 (averaged over 1000 plants). Our typical performance is shown in Figure 1. Switching between plants occurs at $t = 4000$. The differences in tracking two different plants are slight. Similar results can be obtained for other references and larger selections of plants.

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 DC motor, a gear mechanism and a throttle valve with a dual spring system. In the neutral position, both springs are relaxed, and the throttle valve is slightly open. This is called the "limp-home" position, and it is critical in the case of 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 moment for angles

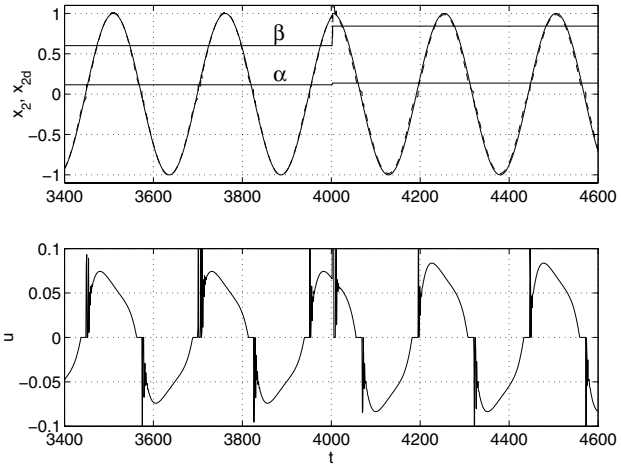

Fig. 1. Tracking of the sine reference by the trained robust neurocontroller of the first example (a fragment of much longer file). Switching from one pair of the plant parameters α and β to another occurs at $t = 4000$. After a brief transient the controller continues very good tracking, with somewhat larger deviations and spikier controls in the deadzone region (the reference output x_{2d} is solid, and the actual output x_2 is dashed).

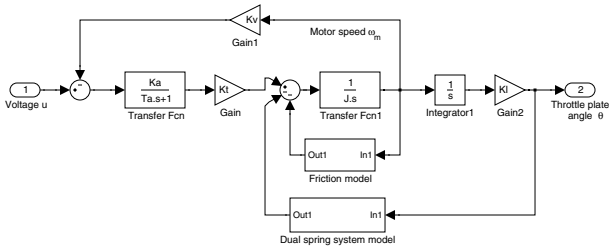

Fig. 2. Block diagram of electronic throttle. The controller (not shown) senses the throttle valve position θ (and, possibly, the motor speed ω_m) and puts out the voltage u.

smaller than the "limp-home". The second spring's stiffness must be higher to provide higher angular resolution at small angles.

The ETC model consists of several components (Figure 2). 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. The gear ratio is K_l, and the overall inertia is J. Our control signal is the motor voltage u, and it is limited between u_{min} and u_{max}. The throttle valve plate position is measured by a potentiometer, which is the only sensor available in practice for position control. Finally, a nondifferentiable model of friction due to ball bearings, the gear mechanism and the dual spring system is based on [14] and [15].

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 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

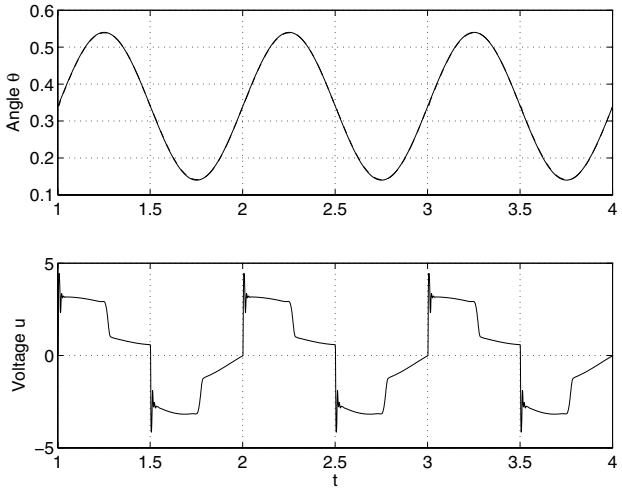

Fig. 3. Tracking of a four-second segment of the sin function around the "limp-home" angle. The reference signal is solid, and the plant output is dashed (top panel). The bottom panel shows the corresponding output of the neurocontroller. The RMS error is 0.002 rad, and instantaneous errors are almost indiscernible. The sampling for control values T_s is 1 ms. The controller is a 3-2R-1 RNN (inputting the current angle, its desired (reference) value, and the motor speed), and it is implemented easily in a vehicle on-board computer. 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.

costly to calibrate any model-based control algorithm with fixed parameters. On the other hand, an adaptive control algorithm may be very difficult to apply because it will have 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 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 Figure 3 for typical results; the network 3-2R-1 has only 15 weights); cf. [15].

Even with a small-size neurocontroller, the control sampling rate $T_s = 1$ ms might be too demanding for the hardware implementation. For the more challenging sampling rate $T_s = 10$ ms, we carry out the nprKF training for robustness. We choose a larger two-hidden-layer RNN, 2-5R-3-1 (62 weights) (its inputs are the current angle and the desired angle). 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

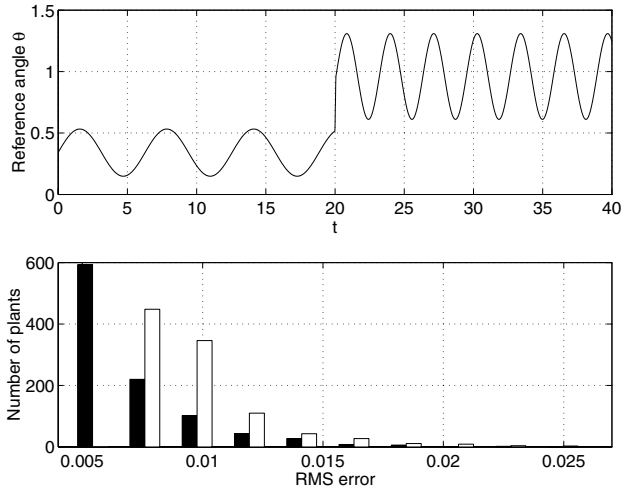

Fig. 4. Performance comparison of the 2-5R-3-1 neurocontroller trained for robustness with $T_s = 10$ ms (black bars) with the same neurocontroller trained on the nominal ETC model only (white bars). The top panel shows the 40-second reference signal. The mean RMS error is 0.007 vs. 0.010 for the nominal neurocontroller. The maximum RMS error is 0.023 vs. 0.026 for the nominal neurocontroller (see the bottom panel).

around their nominal values. Similar to [4], our training reference trajectory is chosen as random levels between 0.14 and 1.57 rad maintained for a random duration between 0.25 s and 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 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 five training epochs, each epoch consisting of processing all 100 points for all streams. We train for 200 epochs total (about 20000 weight updates) with $P_0 = 10^{-4}$, $R_0 = 0.001P_0$ and $Q_0 = 10^{-4}P_0$.

We compare our results of training for robustness with those of the same neurocontroller trained on the nominal ETC model via the nprKF algorithm. We test both controllers on the same reference trajectory and 1000 ETC models (Figure 4). On average, our tracking RMS error (0.007 rad) is almost 30% smaller than that of the nominal neurocontroller, and our individual RMS errors are always smaller than those of the nominal neurocontroller, sometimes by as much as 60%. Similar results indicating advantages of our robust neurocontroller are obtained for other reference trajectories.

IV. CONCLUSION

This paper introduces a promising 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 reasonable assumption that the plant parameters are known only in some ranges around their nominal values. We train a neurocontroller on many instances of plant models with different parameters. We propose the application of a derivative-free Kalman filter algorithm (nprKF) to training neurocontrollers for robustness in the model reference control setting. Our neurocontrollers are discrete-time recurrent neural networks to be deployed with fixed weights. They have been known as 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.

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 *Proceedings of International Joint Conference on Neural Networks '01*, 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 *Proceedings of International Joint Conference on Neural Networks '02*, Hawaii, 2002.

[4] D. V. Prokhorov, G. V. Puskorius, and L. A. Feldkamp, "Dynamical neural networks for control," chapter 16 in *A Field Guide to Dynamical Recurrent Networks*, J. Kolen and S. Kremer (eds), IEEE Press, 2001, pp. 257–289.

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

[6] T. Hrycej. *Neurocontrol: Towards an Industrial Control Methodology*, Wiley, 1997.

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

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

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

[10] 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," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2259–2277, 1998.

[11] S. Haykin (ed). *Kalman Filtering and Neural Networks*, Wiley, 2002.

[12] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* 2nd ed., New York: Cambridge University Press, 1992.

[13] P. He, S. Jagannathan, and S.N. Balakrishnan, "Adaptive critic-based neural network controller for uncertain nonlinear systems with unknown deadzones," in *Proceedings of the Conference on Decision and Control,* Las Vegas, 2002, pp. 955–960.

[14] P. Dupont, V. Hayward, B. Armstrong, and F. Altpeter, "Single state elastoplastic friction models," *IEEE Trans. Automatic Control*, vol. 47, no. 5, pp. 787–792, 2002.

[15] M. Barić, I. Petrović, and N. Perić, "Neural network based sliding mode controller for a class of linear systems with unmatched uncertainties," in *Proceedings of the Conference on Decision and Control,* Las Vegas, 2002, pp. 967–972.