Robust Reinforcement Learning Control Using Integral Quadratic Constraints for Recurrent Neural Networks

Charles W. Anderson, Peter Michael Young, *Member, IEEE*, Michael R. Buehner, *Member, IEEE*, James N. Knight, Keith A. Bush, and Douglas C. Hittle

***Abstract*—The applicability of machine learning techniques for feedback control systems is limited by a lack of stability guarantees. Robust control theory offers a framework for analyzing the stability of feedback control loops, but for the integral quadratic constraint (IQC) framework used here, all components are required to be represented as linear, time-invariant systems plus uncertainties with, for IQCs used here, bounded gain. In this paper, the stability of a control loop including a recurrent neural network (NN) is analyzed by replacing the nonlinear and time-varying components of the NN with IQCs on their gain. As a result, a range of the NN's weights is found within which stability is guaranteed. An algorithm is demonstrated for training the recurrent NN using reinforcement learning and guaranteeing stability while learning.**

***Index Terms*—Integral quadratic constraints (IQCs), recurrent neural networks (NNs), reinforcement learning, robust control.**

I. Introduction

An adaptive or learning component can be added to a feedback control loop to fine-tune the performance of the controlled system. Reinforcement learning algorithms are a class of approximate dynamic programming techniques for training such a learning component to optimize a sum over time of a performance measure.

However, the addition of a learning component to a feedback loop greatly increases concerns about the range of possible behavior of the controlled system. A learning agent might even cause a stable control loop to become unstable. Reinforcement learning control would be more generally applicable and acceptable to control designers if stability could be guaranteed while learning.

Robust control theory provides a framework for analyzing the stability of control loops [1]. A computational approach discussed in this paper, developed in the framework of integral quadratic constraints (IQCs) [2], involves the transformation of a control loop into linear time-invariant (LTI) components plus uncertainties that represent the nonlinear, time-varying, or unknown components in terms of constraints on their gain. Robust control synthesis results in controllers that perform well as long as the actual system being controlled is a member of the class of systems encompassed by the LTI model with uncertainties.

When a learning component is introduced into a feedback control loop, robust stability analysis cannot be performed unless the learning component can also be reduced to an LTI model plus uncertainties. Kretchmar *et al.* [3]–[5] showed how this could be accomplished with a learning component consisting of a feedforward artificial neural network (NN) placed in parallel to a fixed controller. The nonlinear and time-varying aspects of the NN were replaced, for analysis, with IQCs on their gains. Feedback systems consisting of LTI models and IQCs can be reduced to linear matrix inequalities (LMIs) that have a solution if the feedback system is stable. Systems of LMIs are formulated as convex optimization problems for which there exist fast polynomial time algorithms and implementations [6]–[8]. As long as stability is proved, the NN's weights are updated using an adaptive critic that is trained to predict a sum of future values of a performance measure. This approach is also used in this paper and explained in Section II.

If measurements comprising the complete state of the plant are not available, a feedforward NN may not be capable of learning to improve the control system performance. Instead, a dynamic, recurrent NN is needed to learn a mapping from current and past measurements to improved control actions.

This paper demonstrates that Kretchmar *et al.*'s robust stability analysis of feedforward NNs can be extended to recurrent NNs. The main contribution of this paper is to show that the nonlinear and time-varying aspects of the recurrent NN can be represented by constraints on their gains and that the question of the stability of the whole feedback loop including the modifiable actor can be reduced to a convex optimization problem.

Morimoto and Doya [9] have also considered the combination of robust control theory with reinforcement learning. While the work we describe here is focused on guarantees of stability while learning, Morimoto and Doya apply robust control concepts to improve the performance of a reinforcement learning system under certain disturbances. A disturbance function is learned to maximally disrupt control performance within predefined constraints in tandem with the learning agent that is trying to optimize performance. Thus, the reinforcement learning agent develops a conservative control policy that optimizes performance under a range of disturbances.

Perkins and Barto [10] take another approach to robust performance in a reinforcement learning system. In their approach,

Manuscript received January 15, 2006; revised November 3, 2006; accepted February 5, 2007. This work was supported in part by the National Science Foundation under Grant 0245291.

The authors are with the Department of Computer Science, Colorado State University, Fort Collins, CO 80523-1873 USA (e-mail: anderson@cs.colostate.edu).

Digital Object Identifier 10.1109/TNN.2007.899520

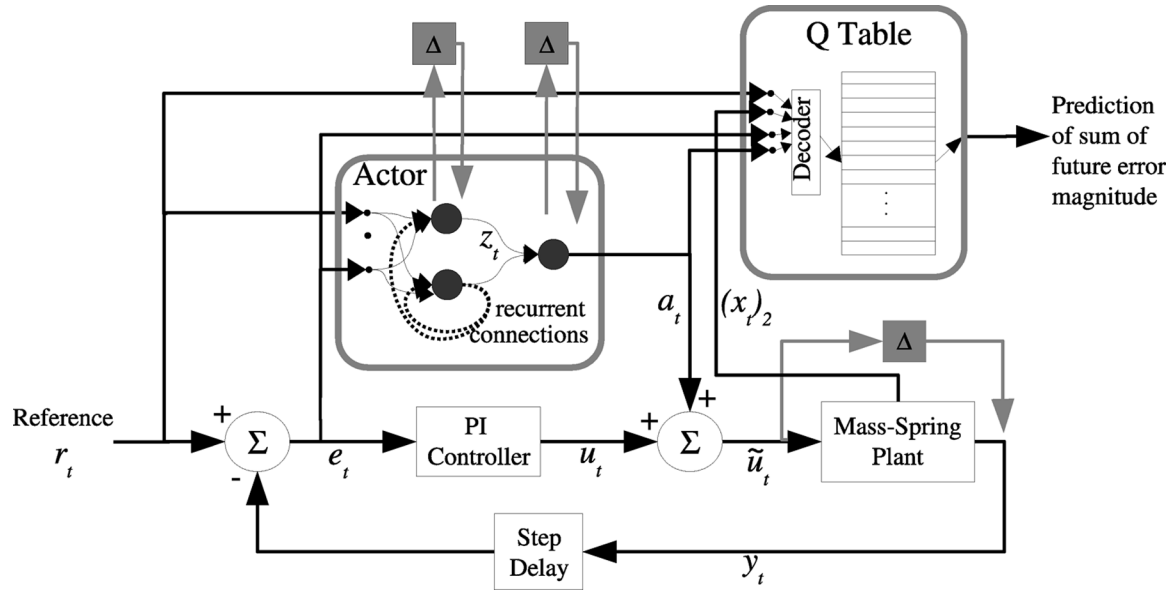

Fig. 1. Conventional feedback loop consisting of a fixed PI controller and the mass-spring plant. A recurrent NN as the actor injects a signal that is added to the output of the PI controller. The actor is trained to produce the optimal action as determined by the critic, which in turn is trained to predict the sum of future error magnitudes. The gray uncertainties are used only during analysis as explained in Section II-E.

Lyapunov principles are used to design controllers that are stable and that achieve a level of desired performance even as a strategy for switching between the controllers is being learned. Sufficient knowledge must be available for designing the required Lyapunov functions, though they suggest that robust control concepts can be combined with Lyapunov methods when complete knowledge of a system to be controlled is not available. The IQC framework is more generally useful, in that the search for a Lyapunov function is replaced by a more sophisticated search [2].

In Section II, the incorporation of IQCs in recurrent NNs is described. The combination of a recurrent network with a proportional–integral (PI) controller and plant, all within the IQC framework is then explained and the operation of the whole system is summarized. Section III presents the results of experiments in which this approach is applied to a simulated system. Section IV summarizes the contributions of this paper and suggests future directions.

II. Method

A. Introduction

The robust analysis methods discussed here are used to determine the stability of feedback loops of LTI systems, so systems with nonlinear or time-varying components cannot be directly analyzed. However, they can be analyzed by replacing nonlinear parts with linearized approximations plus constraints on the gain due to the nonlinearities of the true system. Similarly, time-varying aspects are replaced during analysis by current values plus constraints on how much they are allowed to vary. The robust control framework provides the mechanisms for reducing such a system into a single feedback loop containing a known LTI part and a block-structured uncertainty incorporating all of the constraints. Two examples of such frameworks are μ-analysis [1] and IQC [2], [11].

In this paper, we describe the use of IQCs to analyze the stability of a feedback control loop composed of a plant to be controlled, a fixed PI controller, and a recurrent NN trained via reinforcement learning to augment the controller's action. The recurrent NN is trained to produce actions that, when added to the output of the PI controller, minimize a sum over time of error magnitudes between a varying reference signal and the observable plant variable that is to track the reference signal.

The overall system is shown in Fig. 1. The critic, described later, is used to train the actor and is not part of the feedback loop. The sum of the output a_t of the actor and the output u_t of the fixed PI controller form the control input to the plant. The input to the PI controller is just the error e_t between an observed variable y_t from the plant and its desired value given by the reference signal r_t. Input to the actor is this error plus any other available measurements that might be useful in learning a dynamic, nonlinear function to generate additions to the PI control output that produce better tracking of the reference signal.

B. Nonlinear Mass-Spring-Damper Model

The controlled plant is a nonlinear second-order mass-spring damper with linear viscous friction and a hardened spring (i.e., a spring that pulls harder toward the equilibrium point as a function of distance cubed) [12]. The nonlinear differential equation describing this is

$$m\ddot{y}(t) + (c + c_f)\dot{y}(t) + ky(t) + kh^2y(t)^3 = u(t) \quad (1)$$

where m is the mass, k is the spring constant, c is the damper coefficient, c_f is the coefficient of linear viscous friction, and h is the spring hardening constant. Two different settings of these parameters are used to define two experiments. For experiment 1, $m = 1$ kg, $k = 1$ N/m, $c + c_f = 0.1$ Ns/m, and $h = 0$. For the second experiment, $m = 1$ kg, $k = 0.5$ N/m, and $c = 1.6$ Ns/m, and the parameters h and c_f are unmodeled (nonlinear)

parameters, assumed to be in the ranges $0 \leq h \leq 0.8$ and $0 \leq c_f \leq 0.4$. For simulating the physical plant, the values were set to $h = 0.1$ and $c_f = 0.05$. A larger value for c_f represents more (linear) viscous friction between the mass and the platform, and a larger value for h represents a harder spring. The value of c_f is changed during the second experiment.

The following state–space equations (with $x_1(t) = y(t)$ and $x_2(t) = \dot{y}(t)$) describe this system:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-k}{m} & \frac{-(c+c_f)}{m} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t) + \begin{bmatrix} 0 \\ \frac{-k}{m} h^2 x_1^3(t) \end{bmatrix}$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}.$$

The equivalent discrete-time state–space equations, for sampling period T_s, are

$$\begin{bmatrix} x_1[n+1] \\ x_2[n+1] \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ \frac{-k}{m} T_s & 1 - \frac{(c+c_f)}{m} \end{bmatrix} \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{T_s}{m} \end{bmatrix} u[n] + \begin{bmatrix} 0 \\ \frac{-k}{m} T_s h^2 x_1[n]^3 \end{bmatrix}$$

$$y[n] = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix}.$$

The value of T_s for experiment 1 was $T_s = 0.01$ and for experiment 2 $T_s = 0.1$.

C. PI Controller

For each experiment, the parameters of the PI controller were chosen to obtain good, but conservative, performance. For experiment 1, the PI terms were set to $K_p = 0.01$ and $K_i = 0.1$, and for experiment 2 $K_p = 1.0$ and $K_i = 0.2$. Given samples e_t of the error input to the controller, the output u_t of the PI controller is given by

$$u_t = u_{t-1} + (K_p + K_i T_s) e_t - K_p e_{t-1}.$$

D. Output of Actor

The output of the recurrent NN serving as the actor is calculated as follows. Let z_t be the vector of outputs of all hidden units in the first layer. $z_t^{(r)}$ is the subset of these values that are fed back to these units as inputs through the recurrent connections. The input vector to the recurrent network is defined as $i_t = [r_t,\ e_t,\ z_{t-1}^{(r)}]$. The input is multiplied by the hidden-layer weight matrix W_t to form the weighted sums s_t. The $\tanh$ function is applied to the weighted sums to obtain the outputs z_t of the hidden units. These are multiplied by the output layer's weight matrix V_t to obtain the network's output a_t. These steps are summarized by the following:

$$s_t = [i_t,\ 1] W_t$$
$$z_t = \tanh(s_t)$$
$$a_t = [z_t,\ 1] V_t.$$

E. Stability Analysis With IQCs

The previous equations define the operation of the feedback loop. All components are LTI, except for the recurrent network. To analyze the stability of the loop, the unknown parts of the mass-spring plant and the nonlinear and time-varying aspects of the recurrent NN are replaced by IQCs. A library of IQCs for common uncertainties implemented in Matlab[1] is available [13], and more complex IQCs can be built by combining the basic IQCs. This library employs the LMI solver from the Matlab robust control toolbox [14] and the LTI system representation from the Matlab control system toolbox [15] to provide a comprehensive tool for IQC analysis.

As mentioned previously, in modeling this system, the values of a and c_f are assumed not known, only that their actual values fall within a range. To analyze the stability of the feedback loop, an additive IQC is placed around the system to represent the range of gains the actual system may have. During experiment 2, the value of c_f is modified partway through the experiment.

For stability analysis, the nonlinear function $\tanh$ is replaced with an IQC. The $\tanh$ nonlinearity meets the conditions of the odd slope-restricted nonlinearity IQC [13]—it is an odd function $\tanh(-x) = -\tanh(x)$ and it has restricted slope

$$0 \leq (\tanh(x_1) - \tanh(x_2))(x_1 - x_2) \leq g(x_1 - x_2)^2$$

where g defines the maximum slope. To bound the slope of the $\tanh$ function, $g = 1$.

The time-varying weights W_t and V_t in the actor network are replaced by time-varying IQCs [13]. We define δ_t to be an amount that is added or subtracted from the weights W_t and V_t to identify the region in weight space within which we want to prove that the feedback loop remains stable. The IQC involving δ_t is defined by constant β in

$$|\delta_t| \leq \beta.$$

The value of β is determined by a search procedure defined in the following.

Once the nonlinearity and the time-varying parts of the recurrent network are replaced by these IQCs, the analysis of stability is performed by converting the resulting LTI system with IQCs into an LMI problem whose solution is attempted using convex optimization routines. If a solution is found, the system is known to be stable. If a solution is not found, the system is not guaranteed to be stable. The IQCβ toolbox [13] automates this process.

The search for a value of β in the definition of the slowly time-varying IQC is performed as follows. The objective is to identify a region in the space of W_t and V_t values for which stability can be determined. For experiment 1, two hidden units are used so W_t is 4×2 and V_t is 3×1 and the combined weight is 11-D. For experiment 2, three hidden units are used so W_t is 4×3 and V_t is 4×1 making a 16-D space. A rectangular region is defined in the combined weight space centered at the current values of W_t and V_t with edges of length one. The scale factor β is applied to each dimension to scale the size of the allowable region of W_t and V_t values. Setting $\beta = 1$, stability analysis is performed. If found to be stable, the size of the region is doubled by doubling the value of β and repeating the stability analysis. This continues until analysis determines that stability cannot be guaranteed inside the region. Similarly, if the initial analysis

[1]A commercial software product by The MathWorks described at http://www.mathworks.com

with $\beta = 1$ determines the region is not guaranteed to be stable, the value of β is halved and stability analysis is repeated until a value for β is found for which stability is guaranteed. At this point, a binary search is conducted to find the largest value of β for which stability can be guaranteed.

F. Learning Algorithms for Critic and Actor

As long as the actor's weights W_t and V_t remain within their regions of known stability, the following procedure is used to update the critic and the actor.

The purpose of the critic is to approximate a sum of future tracking error magnitudes as a function of current observed measurements and the actor's output. With such an approximation, possible values of the actor's output, from now on called an action, can be evaluated by the critic. The best action from a set of possible actions can be picked as the one for which the critic predicts the best future. The reinforcement learning literature often calls this action value function a Q function [16]. For our experiment, if o_t is a vector of observable variable values at time t, the critic learns a function $Q(o_t, a_t)$ that approximates $\sum_{k=0}^{T} \gamma^k |e_{t+k+1}|$ where k indicates a range of discrete time samples during which the reference signal is constant, γ is a discount factor between 0 and 1, and $T+t+1$ defines the time step at which reference r changes . In general, the critic can be defined to learn a sum of any performance measure over time.

Here, the Q function in the critic is represented as a table. The Q table values are updated using the state-action-reward-state-action (SARSA) temporal-difference algorithm, an on-policy update method with convergence properties discussed by Sutton and Barto [16]. For the experiment described here, the reinforcement on each step is the absolute value of the tracking error e_t, which is to be minimized. The Q table is a 4-D table constructed as a quantization of the cross product of the spaces of possible reference values r_t, errors e_t, the velocity of the mass which is the second component of the plant state x_t, and the output of the actor network a_t. These four dimensions were quantized into 5, 14, 14, and 6 intervals, respectively. These quantization levels were chosen experimentally, though experimentation was not sufficient to determine optimal quantization levels.

Using a table to implement the Q function is only practical when the dimensionality of the input is small. For larger problems, a continuous function approximator must be used. For the experiments described here, only the actor appears in the feedback loop, so the Q table can be replaced by other function approximators without modifying the stability analysis.

A prediction of the sum of future error magnitudes is found by simply mapping the current values of r_t, e_t, x_t, and a_t to a particular entry in the 4-D Q table and returning the value Q_t that is currently stored there. It is updated using the following temporal-difference error:

$$Q_t = Q_t + \rho_q(|e_{t+1}| + \gamma Q_{t+1} - Q_t).$$

When the reference signal changes, i.e., $r_{t+1} \neq r_t$, Q_t is updated by

$$Q_t = Q_t + \rho_q(0 - Q_t)$$

driving the value of Q towards zero. This is the desired result because the action sequence just prior to this reference change cannot affect the large error that occurs immediately following the reference change.

Given the current Q table and plant state, reference signal, and tracking error, the best actor action that minimizes the predicted sum of future error magnitudes can be determined from a set of possible action values by setting a_t to each of the values, leaving the other inputs to the Q table constant, and comparing the Q table outputs. The action $\tilde{a}_t$ resulting in the lowest Q table value is the best action. $\tilde{a}_t$ serves as a target value for one update step of the actor network's weights.

The actor network's weights are updated using error backpropagation, a gradient-descent procedure in the squared error of the network's output. Williams and Zipser's [17] real-time recurrent learning (RTRL) algorithm is used to approximate the gradient for the recurrent hidden units. This procedure is summarized by the following update equations for the components of V_t and W_t. Time subscripts are dropped for clarity. The network's input i is as defined previously, $i = [r_t,\ e_t,\ z_{t-1}^{(r)}]$

$$\begin{aligned} V_k &= V_k + \rho_v(\tilde{a} - a)z_k \\ p_{ij}^k &= (1 - z_k^2)\left(\sum_{l=1}^{h} W_{lk} p_{ij}^l + \delta_{ik}[i_j,\ 1]\right) \\ W_{ij} &= W_{ij} + \rho_w \sum_{k=1}^{h} (\tilde{a} - a) V_k p_{ij}^k \end{aligned}$$

where δ_{ik} is the Kronecker delta, h is the number of hidden units, ρ_v and ρ_w are constant learning rates, and p_{ij}^k is the accumulated partial derivative of the output of each hidden unit k with respect to its inputs. Fig. 1 shows that for our experiment the outputs of some hidden units are not recurrently connected. This specialization can be implemented simply by setting corresponding values of W to zero.

To force exploration of possible actions, an action is randomly selected from a set of possible actions with probability ϵ_t, which is a decreasing function of time

$$\epsilon_{t+1} = \lambda \epsilon_t$$

where $0 < \lambda < 1$. The set of possible actions used in the experiment has a mean of the current output of the actor a_t. The set of actions used are $\{a_t - 2d, a_t - d, a_t, a_t + d, a_t + 2d\}$, where d is an experimentally determined parameter. This set of possible actions is also the set evaluated by the current Q table to find a target value for training the actor. This set of actions is chosen independently of the quantization intervals of the action range in constructing the Q table. The motivation for limiting the search for the best action to this set of actions is to assume the current output of the actor is close to optimal and to explore a small range of actions centered on the actor's output. During the early stages of training, this assumption is not correct, and perhaps choosing the set of actions as values centered in each action interval in the Q table would work just as well. This was not tested.

G. Experiments

The Q table entries were initialized to zero. The actor's weights W and V were initialized to uniformly distributed random values in the range -0.1 to 0.1 and -0.01 to 0.01, respectively. The probability of exploration ϵ was initialized to

0.5. Before learning begins, a stability analysis of the system with these weight values was performed, which identifies a region of known stability in the space of W_t and V_t values, centered on their current values.

For experiment 1, the following steps were performed. The interaction of the PI controller, actor, and mass-spring plant is simulated for a given number of steps. For each time step, the output of the PI controller u_t is calculated first. Then, the output of the actor is calculated to be a_t, or, with probability ϵ_t, one of the five possible actions is randomly chosen. The sum of u_t and a_t is provided to the mass-spring plant whose state is then updated. The observable y_t is compared with the reference r_{t+1} to obtain the error e_{t+1}. At this point, the Q table and actor network are updated as described previously. Constants used in this experiment are $\rho_q = 0.5$, $\rho_v = 0.001$, $\rho_w = 0.05$, $\lambda = 0.9999$, $\gamma = 0.9$, and $d = 1/16$.

This process continues for 60 000 steps, which is equivalent to 600 s of simulated time with a time step of 0.01 s. (All uses of the variable t in this paper refer to a discrete time step, and not the simulated time in seconds.) Now, the stability analysis is repeated to determine a new region of known stability centered on the current W and V values and training continues for another 60 000 steps. Training is similarly interrupted for additional analyses of stability three more times to obtain the results described in Section III. Anytime the training algorithm results in new weights that fall outside the stability region, stability analysis is also performed. For the simulations reported in Section III, this did not occur.

For experiment 2, the mass-spring plant parameter values resulted in a more difficult control problem. To obtain the results described here, the following steps were followed. First, for 100 000 steps, which is equivalent to 10 000 s, the critic is trained and is used to select the best action to add to the output of the PI controller. Parameter values used were $\rho_q = 0.1$, $\lambda = 0.99999$, $\gamma = 0.9$, and $d = 5$. From then on, the output of the actor selects the best action to add to the PI controller and the actor is trained by the critic to output the action the critic judges to be optimal, as described previously for experiment 1. In this phase, $\rho_q = 0$, $\rho_w = 0.002$, and $\rho_v = 0.001$. The actor is trained for 30 000 steps. At this point, the friction coefficient c_f is changed from 0.05 to 0.1, to simulate an increase in the amount of friction on the mass.

In experiment 2, many of the updates to the actor's weights, as determined by the RTRL reinforcement learning algorithm, caused the weights to exceed their guaranteed stability bounds, causing additional stability analyses to be performed. As described previously, once new bounds are determined, the learning algorithm continues.

III. Results and Discussion

The behavior of the mass-spring plant in experiment 1 when controlled by just the PI controller is shown in Fig. 2. Fig. 2(a) shows the desired mass position as the reference signal r_t and the actual mass position y_t. The reference changes every 2 s. The PI controller forces the mass to track the desired position, getting very close to the reference within 1 s. The output u_t of the PI controller is shown in Fig. 2(b). The average absolute value of the tracking error is about 0.38 over approximately 600 simulated seconds.

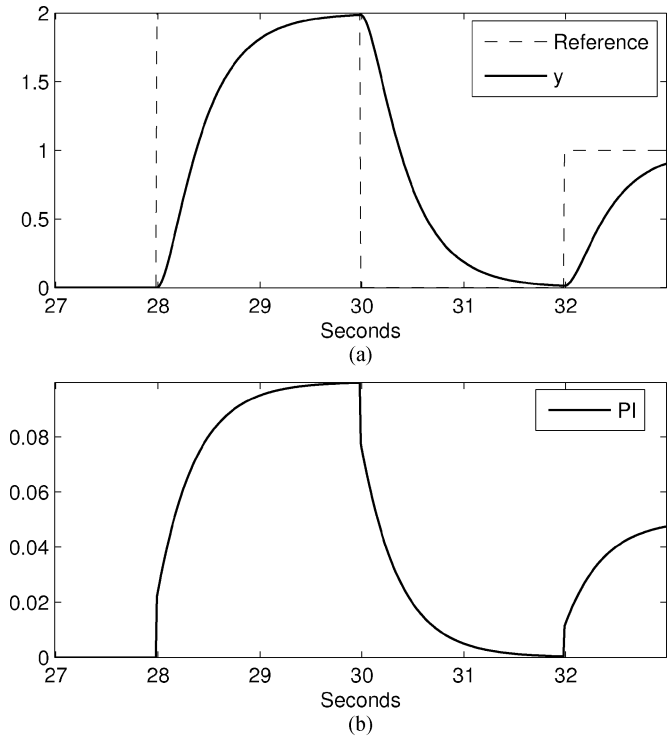

Fig. 2. Experiment 1: Behavior of PI controller without the actor. (a) y is gradually brought to the reference signal level. (b) Output of the PI controller, which is the control input to the plant.

When the training of the complete system with actor and critic is initiated, ϵ is close to 0.5, so for half of the steps the actor generates a randomly chosen action from a set of possible actions close to zero. As expected, the combination of the untrained actor and the PI controller does not track the desired position as well as the PI controller alone. However, as the number of steps increases, the actor learns to choose actions that considerably reduce the tracking error below that of the PI controller. Fig. 3 shows the average magnitude of the tracking error while learning for 3000 s. After about 1000 s, tracking performance has decreased below the level achieved by the PI controller alone. Error magnitude is continuing to decrease when the simulation is stopped at 3000 s.

The change due to learning in the behavior of the plant and the PI controller and actor are illustrated in Figs. 4 and 5. Fig. 4 shows the behavior before learning has occurred. Fig. 4(a) shows poor tracking performance with many small deviations in y due to the random actions of the actor. The random actions of the actor are shown in Fig. 4(b) and the output of the PI controller can be observed as it responds to the error.

Fig. 5 shows the behavior at the end of the 3000-s simulation. By this stage, the actor has learned a policy for selecting actions that track the reference signal very well, as shown in Fig. 5(a). The mass position y rises and falls as set point changes much more rapidly than it did with just the PI controller, as shown in Fig. 2. Fig. 5(b) shows that most of the control signal, which is the sum of the actor and PI controller outputs, is determined by the actor.

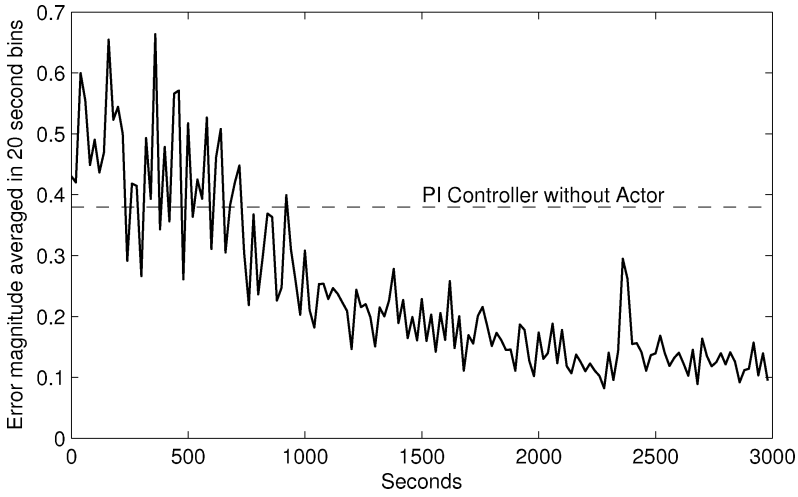

Fig. 3. Experiment 1: Magnitude of error, averaged into bins of 20 s, during entire training run of 3000 s. Error with just PI controller is about 0.38, as shown by the horizontal dotted line.

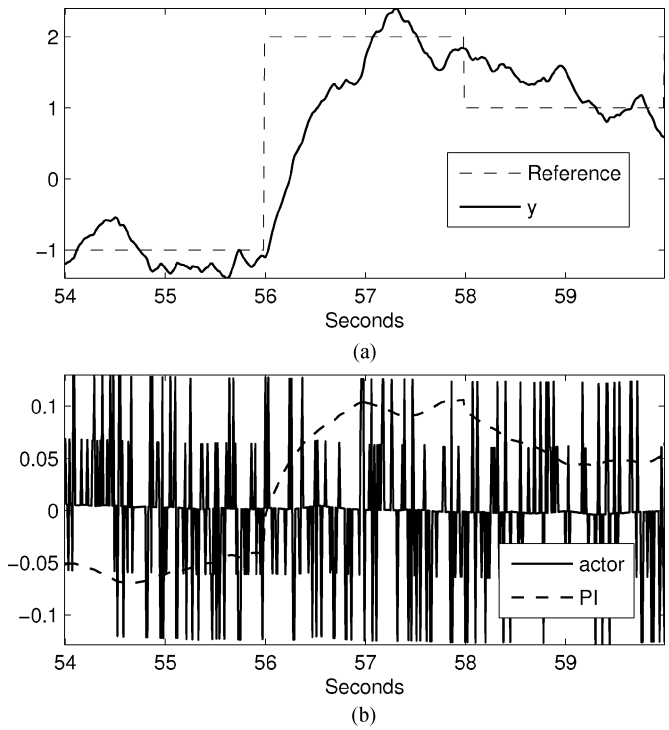

Fig. 4. Experiment 1. (a) Behavior of observed plant variable y and the reference signal value that it must track. (b) Output action of the actor as solid lines and the output of the PI controller as a dotted line. Before learning has occurred, the actor is choosing actions randomly, resulting in worse tracking performance than the PI controller alone, shown in Fig. 2.

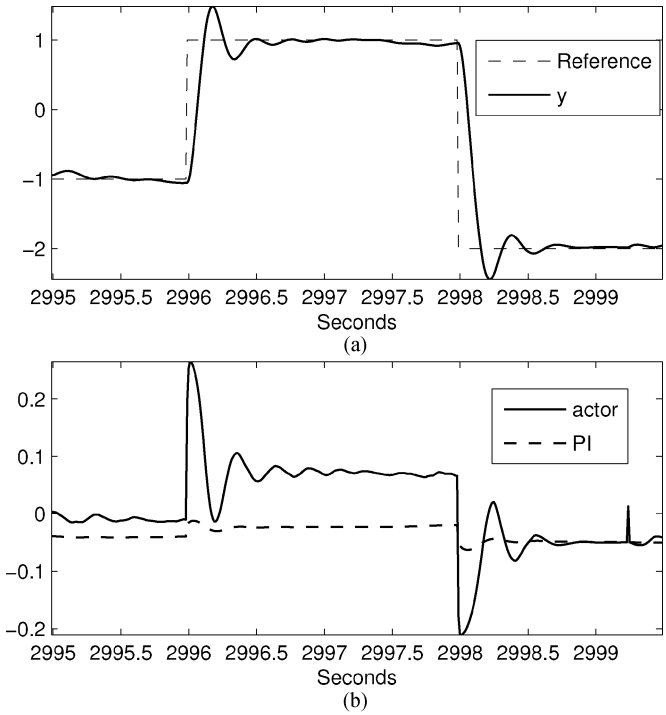

Fig. 5. Experiment 1: After learning for 3000 s, the actor has learned to produce the actions shown in the bottom graph. The actor adds large positive values to the output of the PI controller on positive reference changes and large negative values on negative reference changes. Responsibility for control has largely been assumed by the actor; the PI controller output is very small.

The operation of the search procedure used to find the largest scale factor β that determines the W and V regions of known stability is illustrated in Fig. 6. Recall that stability analysis is repeated five times during the simulation. Each time β is initialized to 1. If the region for $\beta = 1$ is found to be stable, then β is doubled, and it continues to be doubled until stability cannot be guaranteed within the larger region. At this point, a binary search is conducted to find the largest β for which stability is guaranteed. If the analysis for the initial value of $\beta = 1$ does not result in stability, then β is halved until stability is achieved. Then, a similar binary search is conducted to find largest stable β value.

Fig. 6 shows that at later stages, when the values of W and V have increased in magnitude, the largest β values for which regions are proved stable decrease. This would occur if the weights are getting close to regions for which performance of the feedback loop becomes unstable. It might also be due to the fact that, for each analysis phase, the size of the initial region is proportional to the magnitudes of the weights. Further analysis

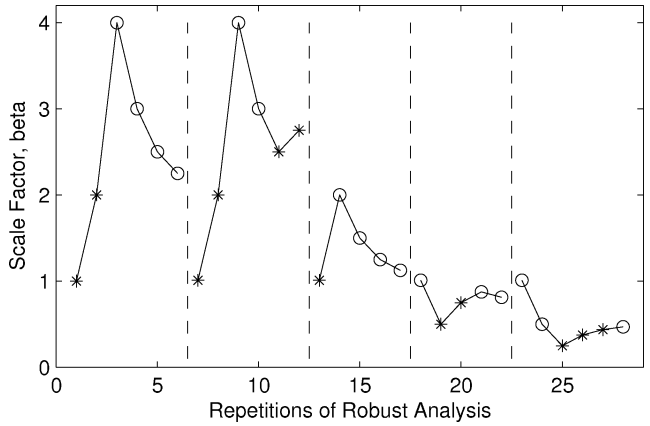

Fig. 6. Experiment 1: Values of the scale factor β during the search for its largest value for which the regions in the space of actor weights can be guaranteed to result in stable behavior. Asterisks mark values for which stability is proved and circles mark values for which stability cannot be proved.

of the results must be conducted to determine which of these possibilities is the true cause.

The performance of the PI controller for experiment 2 is illustrated in Fig. 7 for three changes in the reference signal. Fig. 7(a) shows that the PI controller drives the state variable y towards the reference signal but does not reach the reference value before the reference changes again. Fig. 7(b) shows the corresponding output of the PI controller.

As described in Section II, for experiment 2, the critic is first trained without the actor for 100 000 s. Fig. 8 shows the magnitude of the error decrease as the critic learns and as the exploration probability ϵ decreases. After about 20 000 simulated seconds, the combination of the critic and the PI controller outperforms the PI controller alone, ultimately reaching an average error magnitude of about 0.2. The average error for the PI controller by itself is about 0.45.

In the second phase, the actor is trained to output the optimal action on each step as selected by the critic. During the training of the actor, stability analysis is performed to guarantee that the actor's weights do not exceed the bounds within which stability of the feedback loop can be guaranteed.

The actor learns to duplicate the action choices of the critic rather quickly, after about 1000 simulated sections, as shown by the plot of error magnitude in Fig. 9. The coefficient of friction c_f is doubled after 3000 s, but the effect on the error is not noticeable in this error curve which consists of error magnitudes averaged in bins of 100 s.

The combination of the PI controller and the trained actor is shown in Fig. 10. Fig. 10(a) shows that y reaches the reference signal level within about 1 s and oscillates slightly about the reference. Performance is better than what was obtained with the PI controller alone, shown in Fig. 7. Fig. 10(b) shows that the actor has learned to generate considerably larger actions than those of the PI controller.

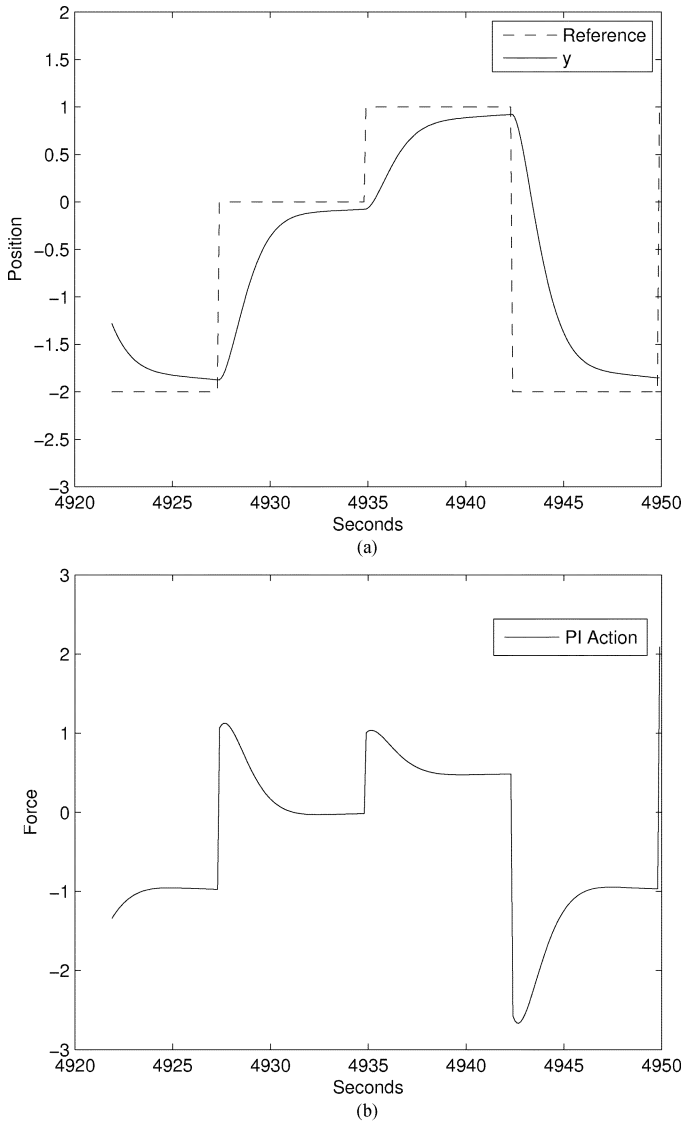

Fig. 7. Experiment 2: Behavior of the PI controller on the plant for experiment 2. (a) State variable y approaches the reference signal level but does not quite reach it before the reference signal is changed. (b) Corresponding output u of the PI controller.

IV. Conclusion

By representing the nonlinear and time-varying aspects of a recurrent NN with IQCs on their gain, a robust analysis is performed on a feedback control loop consisting of a PI controller, a mass-spring system, and the recurrent network trained via reinforcement learning. The result is a control learning algorithm with dynamic learning agent that can fine-tune performance with guarantees of stability even while learning. This paper is a direct extension of Kretchmar *et al.* [3]–[5], who developed a similar analysis but only for feedforward NNs.

Limitations of this paper that are currently being addressed include the following. In this paper, the recurrent NN was provided with more input information than required. With the dynamics afforded by the recurrent network architecture, complex functions over sequences of past inputs can be represented; in the limit, a sequence of values of a single variable, such as the error e_t, may be sufficient to learn an optimal mapping from input information to the optimal action, though the network would face a much more difficult learning problem and may not

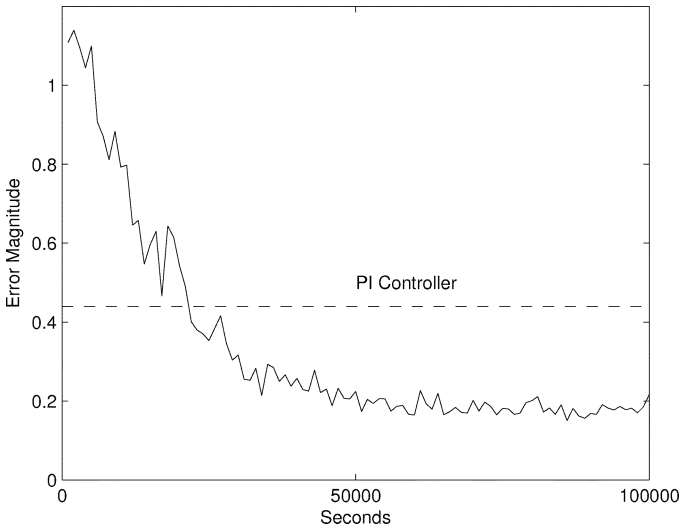

Fig. 8. Experiment 2: Magnitude of the error, averaged into bins of 100 s. Error with just the PI controller is about 0.45, while the combination of the critic and the PI controller achieves an average error of about 0.2.

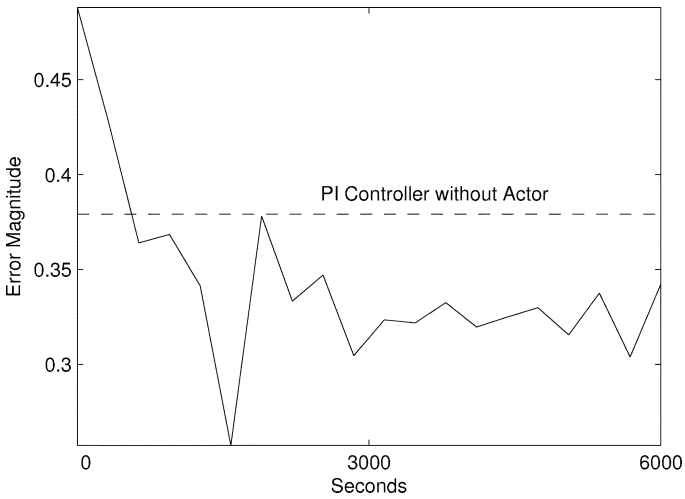

Fig. 9. Experiment 2: Error, averaged into bins of 300 s, while the actor is learning using the critic resulting from the first training phase whose errors are shown in Fig. 8. The coefficient of friction c_f is doubled at 3000 s.

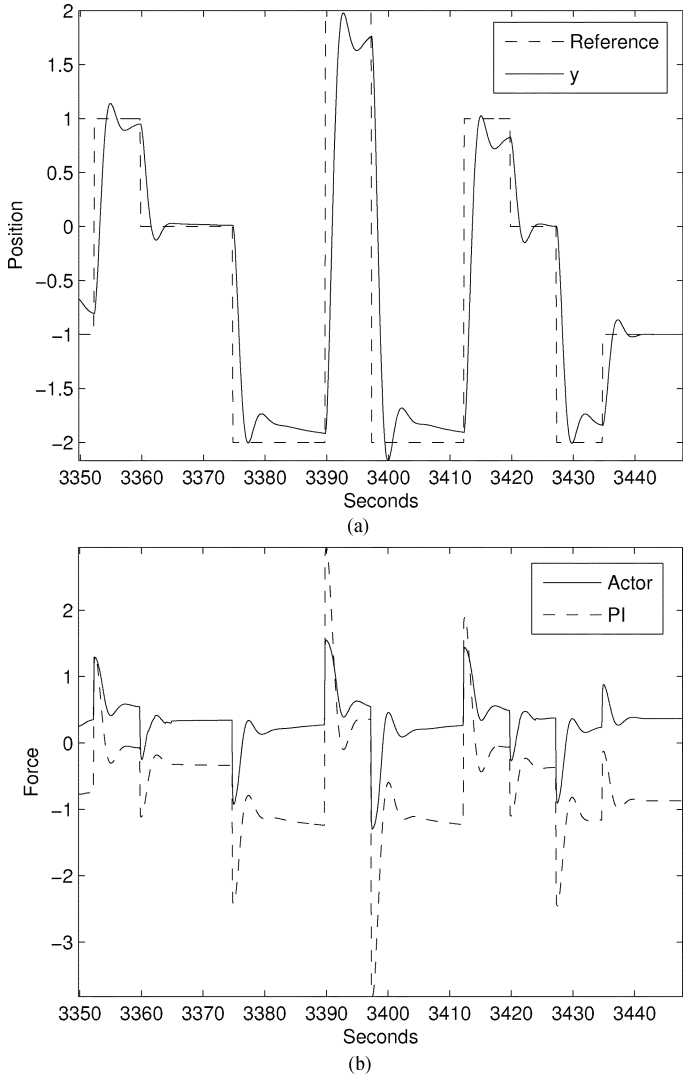

Fig. 10. Experiment 2: Behavior of the trained actor and PI controller. (a) y being driven quickly to the reference signal level followed by minor oscillations. (b) PI output and the actor output. The sum of these two values are the control input to the plant.

be able to learn the desired function. Another limitation concerns the method used to train the recurrent network. The RTRL algorithm has known difficulties when long-time dependencies must be learned and when a larger number of inputs or hidden units are used. One alternative approach is the use of a fixed recurrent network consisting of randomly generated sparse connections. Echo state networks (ESNs) [18] are one way to generate and use such recurrent structures; we are investigating the use of ESNs to learn critic functions [19].

Another limitation is that the Q table used in the critic does not scale well to larger problems that would require additional inputs to the critic. Future work will involve replacing the Q table with a second recurrent NN. Recall that stability analysis does not involve the critic at all, so this limitation does not affect the scalability of the demonstrated approach to robust reinforcement learning.

The use of the odd slope-restricted nonlinearity IQC to characterize the gain of the $\tanh$ function is much too conservative when the $\tanh$ function is biased away from zero. Barabanov and Prokhorov [20] have shown how the bias can be used to define less conservative bounds for the gain of $\tanh$ for a static stability analysis. It may be possible to extend their approach for the dynamic stability analysis illustrated in this paper.

A limitation critical to real-time application is that, while convex optimization algorithms based on interior point methods are polynomial in time, the execution time required to prove stability may be high. The demonstration shown here involved LMI problems consisting of around 80 variables. It is important to realize that most of these variables arise from the IQCs constraining the NN, and not the plant; the plant model only included uncertainties for a and c_f. Additional uncertainties may be added to the plant model without greatly affecting the size of the LMI problem. It is also important to realize that the analysis is only required when the reinforcement learning update to the actor's weights drives the weights outside of the known stability

region, but this could occur often if the actor function that generates optimal actions requires weight values that are close to regions of instability. The experiments reported here were solved in a practical amount of time using the MathWorks LMI toolbox to solve the LMI problems. Faster solvers, such as SeDuMi [7] and SDPT3 [8], would result in more efficient implementations.

ACKNOWLEDGMENT

C. W. Anderson, J. N. Knight, and K. A. Bush would like to thank the Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP), Martigny, Switzerland, and the Hamilton Institute, National University of Ireland, Maynooth, Ireland, for hosting them during the completion of this paper.

REFERENCES

[1] K. Zhou and J. C. Doyle, *Essentials of Robust Control*. Englewood Cliffs, NJ: Prentice-Hall, 1998.
[2] A. Megretski and A. Rantzer, "System analysis via integral quadratic constraints," *IEEE Trans. Autom. Control*, vol. 42, no. 6, pp. 819–830, Jun. 1997.
[3] C. W. Anderson, R. M. Kretchmar, P. M. Young, and D. C. Hittle, "Robust reinforcement learning using integral-quadratic constraints," in *Learning and Approximate Dynamic Programming*, J. Si, A. Barto, W. Powell, and D. Wunsch, Eds. New York: Wiley, 2004, pp. 337–358.
[4] R. Kretchmar, P. Young, C. Anderson, D. Hittle, M. Anderson, C. Delnero, and J. Tu, "Robust reinforcement learning control with static and dynamic stability," *Int. J. Robust Nonlinear Control*, vol. 11, pp. 1469–1500, 2001.
[5] R. M. Kretchmar, "A synthesis of reinforcement learning and robust control theory," Ph.D. dissertation, Dept. Comput. Sci., Colorado State Univ., Fort Collins, CO, 2000.
[6] P. Gahihet, A. Nemirovski, A. J. Laub, and M. Chilali, LMI Control Toolbox. MathWorks Inc., 1995.
[7] J. F. Sturm, "Using SEDUMI 1.02, a Matlab toolbox for optimization over symmetric cones," *Optim. Methods Software* vol. 11–12, pp. 625–653, 1999 [Online]. Available: http://sedumi.mcmaster.ca
[8] K. C. Toh, R. H. Tutuncu, and M. J. Todd, "SDPT3—A Matlab software for semidefinite-quadratic-linear programming," 2006 [Online]. Available: http://www.math.nus.edu.sg/~mattohkc/sdpt3.html
[9] J. Morimoto and K. Doya, "Robust reinforcement learning," *Neural Comput.*, vol. 17, pp. 335–359, 2005.
[10] T. J. Perkins and A. G. Barto, "Lyapunov design for safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 3, pp. 803–832, 2002.
[11] A. Megretski and A. Rantzer, "System analysis via integral quadratic constraints: Part II," Lund Inst. Technol., Lund, Sweden, 1997, Tech. Rep. ISRN LUTFD2/TFRT–7559–SE.
[12] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 2002.
[13] U. Jönsson, C.-Y. Kao, A. Megretski, and A. Rantzer, "A guide to IQCβ: A Matlab toolbox for robust stability and performance analysis," [Online]. Available: http://www.ee.unimelb.edu.au/staff/cykao/
[14] Robust Control Toolbox. MathWorks, Inc. [Online]. Available: http://www.mathworks.com/products/robust
[15] Control System Toolbox. MathWorks, Inc. [Online]. Available: http://www.mathworks.com/products/control
[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[17] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, pp. 270–280, 1989.
[18] H. Jaeger, "The echo state approach to analyzing and training recurrent neural networks," Fraunhofen, Germany, Tech. Rep., 2001.
[19] K. Bush and C. Anderson, "Modeling reward functions for incomplete state representations via echo state networks," in *Proc. Int. Joint Conf. Neural Netw.*, 2005, vol. 5, pp. 2995–3000.
[20] N. E. Barabanov and D. V. Prokhorov, "Stability analysis of discrete-time recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 292–303, Mar. 2002.

Charles W. Anderson received the B.S. degree from the University of Nebraska, Lincoln, in 1978, and the M.S. and Ph.D. degrees from the University of Massachusetts, Amherst, in 1982 and 1986, respectively, all in computer science.

From 1986 to 1990, he was a Senior Member of Technical Staff at GTE Labs, Waltham, MA. Currently, he is an Associate Professor at the Department of Computer Science, Colorado State University, Fort Collins. His research interests are in NNs and reinforcement learning for signal processing and control, with applications to brain-computer interfaces and robustly stable control of dynamic systems.

Peter Michael Young (S'90–M'94) received the B.A. degree in engineering science from Oxford University, Oxford, U.K., in 1985, the M.S. degree in electrical engineering from the University of Florida, Gainesville, in 1988, and the Ph.D. in electrical engineering from California Institute of Technology, Pasadena, in 1993.

From 1985 to 1986, he was an Executive Engineer at British Telecom Research Laboratories, and from 1993 to 1995, he was a Postdoctoral Associate at Massachusetts Institute of Technology, Cambridge. Currently, he is an Associate Professor at Colorado State University, Fort Collins. His recent research interests include the development of analysis and design techniques for large scale uncertain systems, and robust learning controllers, as well as a number of specific application areas. These include control of heating, ventilation, and air-conditioning (HVAC) systems, power system distribution grids and sustainable energy, and control of biological systems.

Michael R. Buehner (S'05–M'07) received the B.S. and M.S. degrees in electrical engineering from Colorado State University, Fort Collins, in 2002 and 2004, respectively, where currently, he is working towards the Ph.D. degree in electrical and computer engineering.

He is an Intern at LSI Corporation, Fort Collins, CO. His research interests are in nonlinear controls, robust controls, communication systems, and digital signal processing.

James N. Knight received the B.S. degree in math and computer science from Oklahoma State University, Stillwater, in 2001 and the M.S. degree in computer science from Colorado State University, Fort Collins, in 2003. Currently, he is working towards the Ph.D. degree in computer science at the Oklahoma State University.

His research focuses on combining ideas from reinforcement learning and robust control theory.

Keith A. Bush received the B.S.E. degree in chemical engineering from the University of Pennsylvania, Philadelphia, in 1998 and the M.S. degree in computer science from Colorado State University, Fort Collins, in 2003, where currently, he is working towards the Ph.D. degree at the Department of Computer Science.

His research interests are in recurrent NNs and reinforcement learning with application to control in partially observable state spaces.

Douglas C. Hittle received the B.S. degree in mechanical engineering, the M.S. degree in environmental engineering, and the Ph.D. degree in mechanical engineering from the University of Illinois at Urbana-Champaign, Urbana, in 1969, 1975, and 1981, respectively.

From 1969 to 1983, he was a Federal employee, first at Chanute Air Force where he designed energy systems and then at the Construction Engineering Research Laboratory where he was a Research Team Leader carrying out work in building systems simulation, solar energy, and control. From 1983 to 1989, he was a Professor at the Purdue University, West Lafayette, IN, carrying out research in air-conditioning system at the Ray W. Herrick. In 1989, he became a Professor at the Colorado State University (CSU), Fort Collins in the Mechanical Engineering Department and became the Director of the Solar Energy Applications Laboratory. He is also the Director of CSU's Industry Assessment Center.

Dr. Hittle received the Army R&D Achievement Award and the College of Engineering Distinguished Service Award. He is a Fellow of the American Society of Heating, Refrigerating, and Air-Conditioning Engineers.