Robust Reinforcement Learning Control

R. Matthew Kretchmar[a], Peter M. Young[b], Charles W. Anderson[c], Douglas C. Hittle[d], Michael L. Anderson[b], Jilin Tu[c], Christopher C. Delnero[d]

[a] Department of Mathematics and Computer Science, Denison University, Granville, OH 43023

[b] Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523

[c] Department of Computer Science, Colorado State University, Fort Collins, CO 80523

[d] Department of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523

Abstract

Robust control theory is used to design stable controllers in the presence of uncertainties. By replacing nonlinear and time-varying aspects of a neural network with uncertainties, a robust reinforcement learning procedure results that is guaranteed to remain stable even as the neural network is being trained. The behavior of this procedure is demonstrated and analyzed on a simple control task. Reinforcement learning with and without robust constraints results in the same control performance, but at intermediate stages the system without robust constraints may go through a period of unstable behavior that is avoided when the robust constraints are included.

1 Introduction

The design of a controller is based on a mathematical model that captures as much as possible all that is known about the plant to be controlled and that is representable in the chosen mathematical framework. The objective is not to design the best controller for the plant model, but for the real plant. Robust control theory achieves this goal by specifying the model in a Linear-Time-Invariant (LTI) framework and including "uncertainties" with gains that are guaranteed to bound the true gains of unknown or known, nonlinear parts of the plant. Robust control techniques are applied to the plant model augmented with uncertainties and candidate controllers to analyze the stability of the true system. This is a significant advance in practical control, but designing a controller that remains stable in the presence of uncertainties limits the aggressiveness of the resulting controller, resulting in suboptimal control performance. We refer the interested reader to [12, 16, 3] for examples.

Here we describe an approach for combining robust control techniques with a reinforcement learning algorithm to improve the performance of a robust controller while maintaining the guarantee of stability. Reinforcement learning is a class of algorithms for solving multi-step, sequential decision problems by finding a policy for choosing sequences of actions that optimize the sum of some performance criterion over time [13]. They avoid the unrealistic assumption of known state-transition probabilities that limits the practicality of dynamic programming techniques. Instead, reinforcement learning algorithms adapt by interacting with the plant itself, taking each state, action, and new state observation as a sample from the unknown state transition probability distribution.

Neural networks as controllers, or neuro-controllers, constitute much of the recent non-LTI control research. Because neural networks are both nonlinear and adaptive, they can realize superior control compared to LTI, especially when trained via reinforcement learning. The stability issue for systems with neuro-controllers encompasses two aspects. *Static stability* is achieved when the system is proven stable provided that the neural network weights are constant. *Dynamic stability* implies that the system is stable even while the network weights are changing.

To solve the static stability problem, we must ensure that the neural network with a fixed set of weights implements a stable control scheme. Since exact stability analysis of the nonlinear neural network is intractable, we need to extract the LTI components from the neural network and represent the remaining parts as uncertainties. To accomplish this, we treat the nonlinear hidden units of the neural network as sector-bounded, nonlinear uncertainties. We use Integral Quadratic Constraint (IQC) analysis [8] to determine the stability of the system consisting of the plant, the nominal controller, and the neural network with given weight values. Others have analyzed the stability of neuro-controllers using other approaches. The most significant of these other static stability solutions is the NLq research of Suykens and DeMoor [14]. Our approach is similar in the treatment of the nonlinearity of the neural network, but we differ in how we arrive at the stability guarantees. To solve the dynamic stability problem, we add uncertainty in the form of a slowly time-varying scalar to cover weight changes during learning. Again, we apply IQC-analysis to determine whether the network (with the weight uncertainty) forms a stable controller.

The remainder of this paper describes our approach and demonstrates its use on a simple control problem.

2 Robust Reinforcement Learning

In this section we describe our robust reinforcement learning approach. First, reinforcement learning algorithms are introduced, then IQC stability analysis of neural networks is described. This is followed by an outline of our dynamic stability procedure.

2.1 Q-Learning and SARSA

A key concept in reinforcement learning is the formation of the value function. The value function is the expected sum of future reinforcement signals that the agent receives and is associated with each observation of the environment. In Watkins' Q-learning algorithm [15], a function $Q_\pi(s_t, a_t)$ of the observed state, s_t, and action, a_t, at time t for a policy π is learned whose value is the expected value of the sum of future reinforcement. Let the reinforcement resulting from applying action a_t while the system is in state s_t be $R(s_t, a_t)$. Thus, the desired value of $Q_\pi(s_t, a_t)$ is

$$Q_\pi(s_t, a_t) = \mathrm{E}_\pi \left\{ \sum_{k=0}^{T} \gamma^k R(s_{t+k}, a_{t+k}) \right\},$$

where γ is a discount factor between 0 and 1 that weights reinforcement received sooner more heavily than reinforcement received later. The Q function implicitly defines the policy, π, defined as

$$\pi(s_t) = \operatorname*{argmin}_{a \in A} Q(s_t, a).$$

Rummery and Niranjan [11, 10] defined the SARSA algorithm for learning this Q function. SARSA updates the Q function according to

$$\begin{aligned} \Delta Q_\pi(s_t, a_t) =& \alpha_t \Big[R(s_t, a_t) + \gamma Q_\pi(s_{t+1}, a_{t+1}) \\ & - Q_\pi(s_t, a_t) \Big], \end{aligned} \tag{1}$$

which is a Monte-Carlo approach to the value iteration algorithm in dynamic programming. SARSA is used as the reinforcement learning component of our experiments described later.

Though not necessary, the policy implicitly represented by a Q-value function can be explicitly represented by a second function approximator, called the actor. This was the strategy followed by Jordan and Jacobs [5] and is very closely related to the actor-critic architecture of Barto, et al., [2]. In the work reported here, we place a reinforcement learning algorithm within the robust stability framework by choosing the actor-critic architecture, shown in Figure 1. The actor implements a

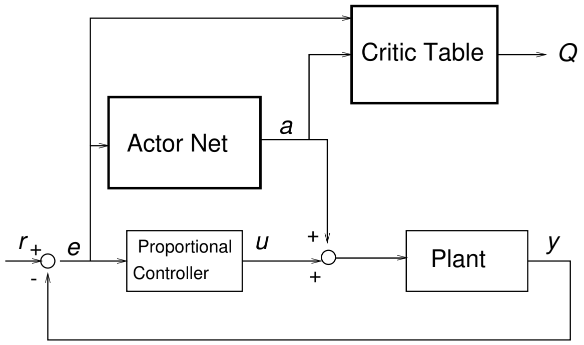


Figure 1: Actor-Critic architecture added to conventional, nominal controller and plant.

policy as a mapping from input to control signal, just as a regular feedback controller would. Thus, a system with a fixed, feedback controller and an actor can be analyzed if the actor can be represented in a robust framework. The critic guides the learning of the actor, but the critic is not part of the feedback path of the system. For the actor, we select a two-layer, feedforward neural network with hidden units having hyperbolic tangent activation functions and linear output units. This feedforward network explicitly implements a policy as a mathematical function and is thus amenable to the stability analysis detailed in the next section.

2.2 IQC Stability Analysis of Neural Network Control

Integral quadratic constraints (IQC) [8, 9, 7] are a tool for verifying the stability of systems with uncertainty. In this section, we present a very brief summary of the IQC theory relevant to our problem. Consider the feedback interconnection shown in Figure 2. The upper block, M, is a known Linear-Time-Invariant (LTI) system, and the lower block, Δ is a (block-diagonal) structured uncertainty. An *Integral Quadratic Constraint* (IQC) is an inequality describing the relationship between two signals, w and v, characterized by a Hermitian matrix function Π as:

$$\int_{-\infty}^{\infty} \begin{vmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{vmatrix}^* \Pi(j\omega) \begin{vmatrix} \hat{v}(j\omega) \\ \hat{w}(j\omega) \end{vmatrix} d\omega \geq 0 \tag{2}$$

where $\hat{v}$ and $\hat{w}$ are the Fourier Transforms of $v(t)$ and $w(t)$. The basic IQC stability theorem can be stated as follows.

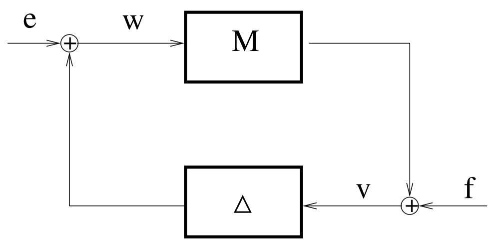


Figure 2: Feedback System

Theorem 1 *Consider the interconnection system represented in Figure 2 and given by the equations*

$$v = Mw + f \tag{3}$$

$$w = \Delta(v) + e \tag{4}$$

Assume that:

- *$M(s)$ is a stable, proper, real-rational transfer matrix, and Δ is a bounded, causal operator.*
- *The interconnection of M and $\tau\Delta$ is well-posed for all $\tau \in [0,1]$. (i.e., the map from $(v,w) \rightarrow (e,f)$ has a causal inverse)*
- *The IQC defined by Π is satisfied by $\tau\Delta$ for all $\tau \in [0,1]$.*
- *There exists an $\epsilon > 0$ such that for all ω:*

$$\left| \begin{array}{c} M(j\omega) \\ I \end{array} \right|^{*} \Pi(j\omega) \left| \begin{array}{c} M(j\omega) \\ I \end{array} \right| \leq -\epsilon I \tag{5}$$

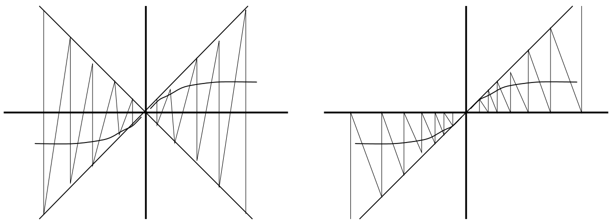
Then the feedback interconnection of M and Δ is stable.

The computation involved to meet the requirements of the theorem is tractable, since the theorem requirements can be transformed into a Linear Matrix Inequality (LMI). As is well known, LMIs are convex optimization problems for which there exist fast, commercially available, polynomial time algorithms [4]. In fact there is now a beta-version of a Matlab IQC toolbox available at http://web.mit.edu/˜cykao/home.html. This toolbox provides an implementation of an IQC library in Simulink, facilitating an easy-to-use graphical interface for setting up IQC problems. Moreover, the toolbox integrates an efficient LMI solver to provide a powerful comprehensive tool for IQC analysis. This toolbox was used for the calculations throughout this article.

Now we develop an LTI formulation with IQCs for the actor neural network. We begin with the conversion of the nonlinear dynamics of the network's hidden layer into an uncertainty function. Consider a neural network with input vector $x = (x_1, ..., x_n)$ and output vector $a = (a_1, ..., a_m)$. For the experiments described in the next section, the input vector has two components, the error $e = r - y$ and a constant value of 1 to provide a bias weight. The network has h hidden units, input weight matrix W_{hxn}, and output weight matrix V_{mxh}, where the bias terms are included as fixed inputs. The hidden unit activation function is the commonly used hyperbolic tangent function, which produces the hidden unit outputs as vector $\Phi = (\phi_1, \phi_2, \ldots, \phi_h)$. The neural network computes its output by

$$\begin{aligned}
\Phi &= Wx, \\
\gamma_j &= \begin{cases} \frac{\tanh(\phi_j)}{\phi_j}, & \text{if } \phi_j \neq 0; \\ 1, & \text{if } \phi_j = 0, \end{cases} \\
\Gamma &= \mathsf{diag}\{\gamma_j\}, \\
a &= V\Gamma\Phi.
\end{aligned} \tag{6}$$

The function, γ, computes the output of the hidden unit divided by the input of the hidden unit; this is the *gain* of the hyperbolic tangent hidden unit. Note that tanh is a sector bounded function (belonging to the sector [0,1]), as illustrated in Figure 3.



a. tanh in [0,1] b. improved sector

Figure 3: Sector bounds on tanh

Equation 6 offers two critical insights. First, we have not changed the functionality of the neural network by restating the computation in this equation form; this is still the applied version of the neuro-controller. Second, Equation 6 cleanly separates the nonlinearity of the neural network hidden layer from the remaining linear operations of the network. This equation is a multiplication of linear matrices (weights) and one nonlinear matrix, Γ. Our goal, then, is to replace the matrix Γ with an uncertainty function to arrive at a "testable" version of the neuro-controller (i.e., in a form suitable for IQC analysis).

First, we must find an appropriate IQC to cover the nonlinearity in the neural network hidden layer. From Equation 6, we see that all the nonlinearity is captured in a diagonal matrix, Γ. This matrix is composed of individual hidden unit gains, γ, distributed along the diagonal. These act as nonlinear gains via

$$y(t) \quad = \quad \gamma x(t) \quad = \left(\frac{\tanh(x(t))}{x(t)} \right) x(t) \quad = \quad \tanh(x(t)) \tag{7}$$

(for input signal $x(t)$ and output signal $y(t)$). In IQC terms, this nonlinearity is referred to as a *bounded odd slope nonlinearity*. There is an Integral Quadratic Constraint already configured to handle such a condition. The IQC nonlinearity, ψ, is characterized by an odd condition and a bounded slope, i.e., the input-output relationship of the block is $y(t) = \psi(x(t))$ where ψ is a static nonlinearity satisfying (see [7]):

$$\psi(-x) = -\psi(x), \tag{8}$$

$$\begin{aligned} \alpha(x_1 - x_2)^2 &\leq (\psi(x_1) - \psi(x_2))(x_1 - x_2) \\ &\leq \beta(x_1 - x_2)^2. \end{aligned} \tag{9}$$

For our specific network, we choose $\alpha = 0$ and $\beta = 1$. Note that each nonlinear hidden unit function ($\tanh(x)$) satisfies the odd condition, namely:

$$\tanh(-x) = -\tanh(x) \tag{10}$$

and furthermore the bounded slope condition

$$0 \leq (\tanh(x_1) - \tanh(x_2))(x_1 - x_2) \leq (x_1 - x_2)^2 \tag{11}$$

is equivalent to (assuming without loss of generality that $x_1 > x_2$)

$$0 \leq (\tanh(x_1) - \tanh(x_2)) \leq (x_1 - x_2) \tag{12}$$

which is clearly satisfied by the tanh function since it has bounded slope between 0 and 1 (see Figure 3). Hence the hidden unit function is covered by the IQCs describing the bounded odd slope nonlinearity (8,9). We now need only construct an appropriately dimensioned diagonal matrix of these bounded odd slope nonlinearity IQCs and incorporate them into the system in place of the Γ matrix.

In addition to the nonlinear hidden units, we must also cover the time-varying weights that are adjusted during training. The *slowly time-varying real scalar IQC* allows for a linear gain block which is (slowly) time-varying, i.e., a block with input-output relationship $y(t) = \psi(t)x(t)$, where the gain $\psi(t)$ satisfies (see [8]):

$$|\psi(t)| \leq \beta, \tag{13}$$

$$|\dot{\psi}(t)| \leq \alpha, \tag{14}$$

where ψ is the non-LTI function. In our case ψ is used to cover a time varying weight update in our neuro-controller, which accounts for the change in the weight as the network learns. The key features are that ψ is bounded, time-varying, and the rate of change of ψ is bounded by some constant, α. We use the neural network learning rate to determine the bounding constant, α, and the algorithm checks for the largest allowable β for which we can still prove stability. This determines a safe neighborhood in which the network is allowed to learn.

Dynamic Stability Procedure: We are now ready to state the dynamic stability procedure. Details of this algorithm may be found in Kretchmar [6].

1. Design the nominal, robust LTI controller for the given plant model so that this nominal system is stable.

2. Add a feedforward, nonlinear neural network in parallel to the nominal controller. We refer to this as the applied version of the neuro-controller.

3. Recast the neural network into an LTI block plus the odd-slope IQC function described above to cover the nonlinear part of the neural network. We refer to this as the testable version of the neuro-controller.

4. Introduce an additional IQC block, the slowly time-varying IQC, to the testable version, to cover the time-varying weights in the neural network.

5. Perform a search procedure and IQC analysis to find bounds on the perturbations of the current neural network weight values within which the system is stable. This defines a known "stable region" of weight values.

6. Train the neural network in the applied version of the system using reinforcement learning while bounding the rate of change of the neuro-controller's vector function by a constant. Continue training until any of the weights approach its bounds of the stable region, at which point repeat the previous step, then continue with this step.

3 Results

We now demonstrate our robust reinforcement learning algorithm on a simple control task, a first-order positioning control system. A single reference signal, r, moves on the interval $[-1, 1]$ at random points in time. The plant is a first order system and thus has one internal state variable, x. A control signal, u, is provided by the controller to position the plant output, y, closer to r. The dynamics of the discrete-time system are given by:

$$x_{t+\Delta t} = x_t + u_t \tag{15}$$

$$y_t = x_t \tag{16}$$

where Δt is the discrete time step for which we used 0.01 seconds. We implement a simple proportional controller (the control output is proportional to the size of the current error) with $K_p = 0.1$.

$$e_t = r_t - y_t \tag{17}$$

$$u_t = 0.1\, e_t \tag{18}$$

The critic stores the Q function as a table with input vector $[e, a]$ and the single value function output, $Q(e, a)$. The table has 25 partitions separating each input forming a 25x25 matrix. The reinforcement signal is the absolute value of the error, e, so the optimal action is the one that minimizes $Q(e, a)$. The actor network is a two-layer, feed forward neural network that is trained to output the optimal action. The input is e. There are three *tanh* hidden units, and one network output a. The control signal to the plant is u_t plus a_t plus random noise with a magnitude that decreases to zero during training. For training, the reference input r is changed to a new value on the interval $[-1, 1]$ stochastically with an average period of 20 time steps (every half second of simulated time). We trained for 4,000 time steps.

The Simulink diagram for the plant, proportional controller, and actor network including the IQCs is shown in Figure 4. The matrices dW and dV are the perturbation matrices. An increase or decrease in dW implies a corresponding increase or decrease in the uncertainty associated with W. Similarly we can increase or decrease dV to enact uncertainty changes to V. If analysis of the system in Figure 4 shows it is stable, we are guaranteed that our control system is stable for the current neural network weight values. Furthermore, the system will remain stable if we change the neural network weight values as long as the new weight values do not exceed the range specified by the perturbation matrices, dW and dV. In the learning phase, we apply the reinforcement learning algorithm until one of the network weights approaches the range specified by the additives.

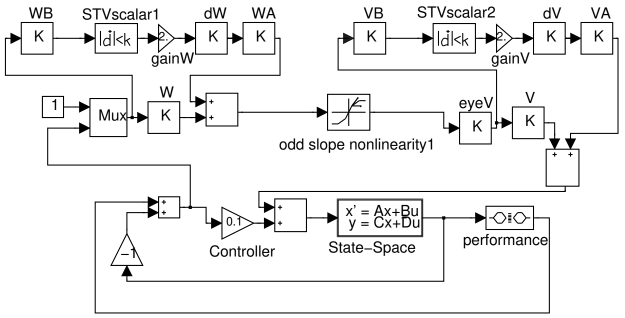


Figure 4: Simulink Diagram for Dynamic IQC-analysis

We now present the results of applying the robust reinforcement learning algorithm. A time-series plot of the simulated system is shown in Figure 5. The top diagram shows the system with only the proportional controller. The bottom diagram shows the same system with both the proportional controller and the neuro-controller after 4,000 steps of training. The reference input takes six step changes. The small-magnitude line is the combined output of the neural network and prortional controller. Clearly, the reinforcement learning neuro-controller is able to improve the tracking performance dramatically. Note, however, with this simple first-order system it is not difficult to construct a better performing proportional controller. In fact, setting the constant of proportionality to 1 ($K_p = 1$) achieves minimal control error. We have purposely chosen a suboptimal controller so that the neuro-controller has room to learn to improve control performance.

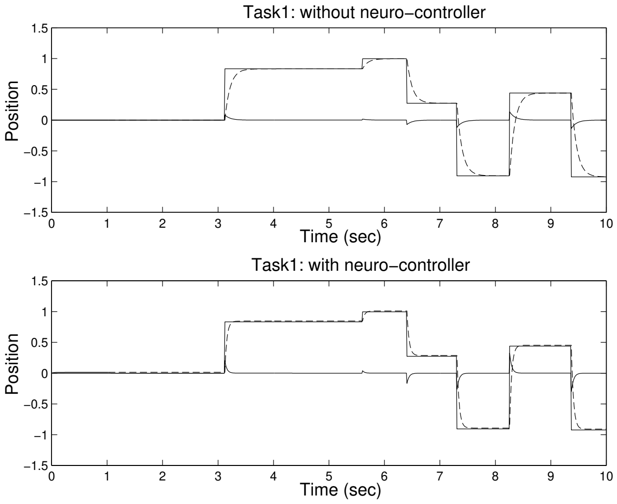


Figure 5: Simulation Run

4 Conclusions

We developed a *static stability* test to determine whether a neural network controller, with a specific fixed set of weights, implements a stable control system, and a *dynamic stability* test in which the neuro-controller is stable even while the neural network weights are changing during the learning process. Our algorithm is essentially a repetition of two phases. In the stability phase, we use IQC-analysis to compute the largest amount of weight uncertainty the neuro-controller can tolerate without being unstable. We then use the weight uncertainty in the reinforcement learning phase as a restricted region in which to change the neural network weights.

Reinforcement learning is well suited to the type of information available in the control environment. It performs the trial-and-error approach to discovering better controllers, and it naturally optimizes our performance criteria over time. The actor-critic design allows the control agent to operate both like a reinforcement learner and also a controller.

The simplicity of the control task demonstrated here does not permit conclusions regarding the generality or scalability of our approach. We are currently extending our robust reinforcement learning procedure to more difficult control problems, including a simulated distillation column for which Kretchmar [6] demonstrates that the robust constraints on the reinforcement learning procedure do indeed maintain stability during a learning process that is otherwise unstable. We are also developing a robust controller for a heating and cooling system, both in simulation and on a physical air duct. Known nonlinearities in the real system are being modeled as uncertainties [1]. Once the robust controller is operating on the air duct, we will add our reinforcement learning algorithm. We expect an improvement in performance due to unknown quantities in the real air duct and also to the fact that nonlinear relationships among the measured variables are known to exist.

Acknowledgments
This work is from a thesis submitted to the Academic Faculty of Colorado State University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and was partially supported by the National Science Foundation through grant CMS-9804757 and 9732986.

References

[1] Michael L. Anderson. Mimo robust control for heating, ventilation and air conditioning (HVAC) systems. Master's thesis, Department of Electrical and Computer Engineering, Colorado State University, 2001.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983. Reprinted in J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, 1988.

[3] John C. Doyle, Bruce A. Francis, and Allen R. Tannenbaum. *Feedback Control Theory*. Macmillan Publishing Company, 1992.

[4] Pascal Gahihet, Arkadi Nemirovski, Alan J. Laub, and Mahmoud Chilali. *LMI Control Toolbox*. MathWorks Inc., 1995.

[5] M. I. Jordan and R. A. Jacobs. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 324–331. Morgan Kaufmann, San Mateo, CA, 1990.

[6] R. Matthew Kretchmar. *A Synthesis of Reinforcement Learning and Robust Control Theory*. PhD thesis, Department of Computer Science, Colorado State University, 2000.

[7] Alexandre Megretski, Chung-Yao KAO, Ulf Jonsson, and Anders Rantzer. *A Guide to IQCβ: Software for Robustness Analysis*. MIT / Lund Institute of Technology, http://www.mit.edu/people/ameg/home.html, 1999.

[8] Alexandre Megretski and Anders Rantzer. System analysis via integral quadratic constraints. *IEEE Transactions on Automatic Control*, 42(6):819–830, June 1997.

[9] Alexandre Megretski and Anders Rantzer. System analysis via integral quadratic constraints: Part ii. Technical Report ISRN LUTFD2/TFRT–7559–SE, Lund Institute of Technology, September 1997.

[10] Gavin A. Rummery. *Problem Solving with Reinforcement Learning*. PhD thesis, Cambridge University Engineering Department, 1995.

[11] Gavin A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, Engineering Department, Cambridge University, 1994.

[12] Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control*. John Wiley and Sons, 1996.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[14] Johan Suykens and Bart De Moor. Nlq theory: a neural control framework with global assymptotic stability criteria. *Neural Networks*, 10(4), 1997.

[15] C.J.C.H. Watkins. *Learning with Delayed Rewards*. PhD thesis, Cambridge University Psychology Department, Cambridge, England, 1989.

[16] Kemin Zhou and John C. Doyle. *Essentials of Robust Control*. Prentice Hall, 1998.