Dynamic Programming and Time-Varying Delay Systems

Lincoln, Bo

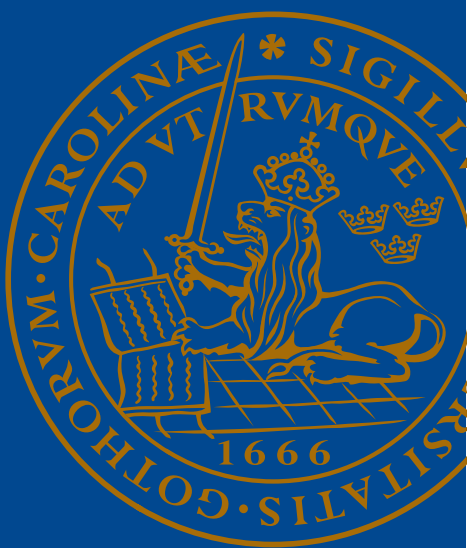
2003

Document Version:
Publisher's PDF, also known as Version of record

Dynamic Programming and Time-Varying Delay Systems

Bo Lincoln

Automatic Control



Dynamic Programming and Time-Varying Delay Systems

Dynamic Programming and Time-Varying Delay Systems

Bo Lincoln

Department of Automatic Control
Lund Institute of Technology
Lund, 2003

Till Hanna

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 LUND
Sweden

ISSN 0280–5316
ISRN LUTFD2/TFRT--1067--SE

Printed in Sweden by Bloms i Lund Tryckeri AB.
Lund 2003

Abstract

This thesis is divided into two separate parts. The first part is about Dynamic Programming for non-trivial optimal control problems. The second part introduces some useful tools for analysis of stability and performance of systems with time-varying delays.

The two papers presented in the first part attacks optimal control problems with finite but rapidly increasing search space. In the first paper we try it reduce the complexity of the optimization by exploiting the structure of a certain problem. The result, if found, is an optimal solution.

The second paper introduces a new general approach of relaxing the optimality constraint. The main contribution of the paper is an extension of the Bellman equality to a double inequality. This inequality is a sufficient condition for a suboptimal solution to be within a certain distance to the optimal solution. The main approach of solving the inequality in the paper is value iteration, which is shown to work well in many different applications.

In the second part of the thesis, two analysis methods for systems with time-varying delays are presented in two papers. The first paper presents a set of simple graphical stability (and performance) criteria when the delays are bounded but otherwise unknown. All that is needed to verify stability is a Bode diagram of the closed loop system.

For more exact computations, the last paper presents a toolbox for MATLAB called JITTERBUG. It calculates quadratic costs and power spectral densities of interconnected continuous-time and discrete-time linear systems. The main contribution of the toolbox is to make well known theory easily applicable for analysis of real-time systems.

Acknowledgments

When I came to the department in early 1999, I had been a Lund resident for six months. I had just finished my Masters thesis, and realized that it was impossible to be 24 years old and live in Lund *without* studying. I got in touch with my supervisor-to-be, Bo Bernhardsson at the department, who tried to recruit me although I was not really a control person. Everyone in Lund could tell me that the Department of Control was the place to be, and after receiving a phone call from Karl-Johan Åström (a trick they still use), I could hardly refuse.

In my first year as a PhD student, I had trouble with the new way of working. "What, define the problem *myself*? Which problem?" I am very grateful to my first advisor Bo Bernhardsson for leading me through the maze of research with patience and a great deal of humor. When Bo was not teaching me we actually learned some things together, as well. Starting our own one-man consultant businesses was very rewarding in many ways. Still, though, I have not succeeded with my plan of getting course credit, teaching credit, a paper, and finally paid for the same piece of work. But I am working on it.

After Bo Bernhardsson left for a Swedish phone-making company, Anders Rantzer was kind enough to be my advisor. Anders was an excellent choice on my side. He is always enthusiastic and full of great ideas, which he never hesitates to share. Thanks to him, writing this thesis was almost easy. I am very grateful to both Anders and Bo.

At the department, I have had the pleasure to work with Anton Cervin, who has always had new ideas in the perspective of real-time systems. Without him, many ideas such as the starting points of Paper III and Paper IV would have ended up somewhere under all the other stuff on my desk.

Chung-Yao Kao ("Isaac") came to the department as a post-doc in the fall of 2002, and we all got to like this witty Taiwanese. We merged our ideas on time-delay analysis and produced Paper III in this thesis. It was

a pleasure to work with Isaac, and I am sure he will soon be a professor somewhere.

The European Commission (IST project Control an Computation), the Swedish Research Council and Vinnova (LUCAS) have provided me with generous financial support. Thank you!

I wish to thank my second advisor Björn Wittenmark for reading my thesis in spite of his great work load as a vice rector. I also send my greatest thanks to Leif Andersson, Rolf Braun and Anders Blomdell, whose work load I have constantly increased by coming up with new projects to build and maintain. In my world, everything would be made out of tejp. Eva, Britt-Marie and Agneta still talk to me in spite of my not-always-so-humble requests. Thank you all for your help!

Me and my next-door friend Rasmus Olsson have helped each other not to finish too soon by dragging the other person to Lomma for windsurfing every time DMI promised more than 5 m/s. This afternoon?

My roommate Johan Bengtsson have always had time to listen to my complaints, and we have had a lot of fun planning new things to buy. Ari Ingimundarsson, Stéphane Velut and Anton Cervin and I had 52 (most often) delicious lunches together in matlaget. And I would like to thank all the people at the department who I have not yet mentioned, my friends in Lund, and my family for making my PhD studies a great time!

Finally, I would like to express my greatest gratitude and love to my sambo Hanna who has supported me through all this. I love you!

Bo

Contents

1

Introduction

This thesis consists of two major parts, and a total of four papers. The first part describes research in *optimal control design* using *dynamic programming*, and will be introduced in Chapter 2. The second part focuses on *systems with time-varying delays*, and problems in real-time control. It is introduced in Chapter 3.

1.1 Background and Motivation

In 1999, when I came to the department, the telecom industry was booming. The control community was, as always, curious to follow (and contribute to) the development. Therefore, my work together with my advisor at the time, Bo Bernhardsson, started around control and wireless networks; specifically the new Bluetooth network technology. We studied the network technology in the perspective of using it to transfer real-time control data. This inspired us to formulate several theoretical problems, like "How should a control network share the communication resources?", and "How are the individual control loops affected by this sharing?" One problem lead to another, and the two main topics of this thesis were formed: Dynamic Programming of hard optimal control problems (such as resource sharing) and analysis of systems with time-varying delays. Much of the work has been carried out with my current advisor Anders Rantzer.

The individual motivations for those two subjects can be found in Chapters 2 and 3. The rest of this chapter will be devoted to describing the main contributions of the thesis.

1.2 Contributions of this Thesis

This section is the abstract of the abstract; for a slightly longer introduction and motivation, read Chapters 2 and 3, and, for all the details,

continue to Papers I–IV.

Paper I

Lincoln, B. and B. Bernhardsson (2002): **"LQR optimization of linear system switching."** *IEEE Transactions on Automatic Control*, vol 47 pp. 1701–1705, October 2002.

This paper presents a switched optimal control problem, and attempts to solve it through Dynamic Programming and by exploiting the structure of the problem. The main contributions are:

- Presentation of a clean and relevant optimal control problem which is provably hard to solve.
- Development of a tree pruning method which often reduces optimization search space drastically. A "complexity parameter" is chosen by the user. It is shown that if this is large enough, the optimal solution will always be found.
- An algorithm to prove that a resulting switching sequence is optimal.

Paper II

An extended version of

Lincoln, B. and A. Rantzer (2003): **"Relaxing Dynamic Programming."** Submitted to *IEEE Transactions on Automatic Control*.

In this paper, similar types of problems as in Paper I are studied, but another method is used to beat the complexity of the optimization. Here, optimality is relaxed so that the algorithm tries to find a solution within a user-specified distance from the optimal solution. This makes it possible to find close-to-optimal solutions to problems which are usually considered hopeless. The contributions are:

- An extension of the Bellman equality of Dynamic Programming to a double inequality. This inequality is a sufficient condition for a suboptimal solution to be within a certain distance from the optimal solution.
- A value-iteration method to find solutions to the above inequality. The solution error bound is specified by the user in advance, and enables a time-versus-accuracy trade-off.
- Examples showing the feasibility of the above method in a number of problem domains:
  - Optimal control of switched linear systems

- Optimal control of piecewise linear systems
- Partially Observable Markov Decision Processes (POMDPs)
- Network routing tables

Paper III

Kao, C-Y. and B. Lincoln (2003): **"Simple Stability Criteria for Systems with Time-Varying Delays"** Submitted to *Automatica*.

This is the first paper in the second part of this thesis, where the topic is "Systems with Time-Varying Delays". This paper presents a set of simple, graphical, stability criteria for control systems with bounded but otherwise freely time-varying delays. The contributions are:

- Simple Bode-diagram-verifiable stability criteria for continuous-time, discrete-time, and mixed continuous-discrete-time systems are presented. The theorems are simply based on the Small Gain Theorem.
- It is indicated that the criteria are not very conservative.
- A performance degradation bound is proven.

Paper IV

Lincoln, B. and A. Cervin (2002): **"Jitterbug: A tool for analysis of real-time control performance."** In *Proceedings of the 41st IEEE Conference on Decision and Control*, pp. 1319–1324, December 2002.

Finally, on the practical side, a MATLAB toolbox named JITTERBUG is presented. It can be used to do the theoretically well-known but tedious performance calculations for linear continuous-time and discrete-time interconnected systems with random time-delays. The contribution is:

- A MATLAB toolbox which can calculate the steady-state performance of interconnected continuous- and discrete-time linear systems driven by Gaussian noise. The toolbox includes support for adding random events such as time-varying delays or data drop-outs.

Other Publications

A number of other papers which I choose not to include in this thesis have been published as well. Several of these papers have "paved the way" to the papers in this thesis, and can therefore be seen as intermediate steps.

- Cervin, A., D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén (2003): "How does control timing affect performance?" *IEEE Control Systems Magazine*. To appear.

- Lincoln, B. and A. Rantzer (2003): "Relaxed Optimal Control of Piecewise Linear Systems." To be presented at the IFAC Conference on Analysis and Design of Hybrid Systems, Saint-Malo, France, June 2003.
- Lincoln, B. and A. Rantzer (2002): "Suboptimal dynamic programming with error bounds." In *Proceedings of the 41st Conference on Decision and Control*, pp. 2354–2359, December 2002.
- Lincoln, B. (2002): "A simple stability criterion for control systems with varying delays." In *Proceedings of the 15th IFAC World Congress*, paper T-Th-A21, July 2002.
- Lincoln, B. (2002): "Jitter compensation in digital control systems." In *Proceedings of the 2002 American Control Conference*, pp. 2985–2990, May 2002.
- Lincoln, B. and A. Rantzer (2001): "Optimizing linear system switching." In *Proceedings of the 40th Conference on Decision and Control*, pp. 2063–2068, December 2001.
- Lincoln, B. and B. Bernhardsson (2000): "Efficient pruning of search trees in LQR control of switched linear systems." In *Proceedings of the Conference on Decision and Control* pp. 1828–1833, December 2000.
- Lincoln, B. and B. Bernhardsson (2000): "Optimal control over networks with long random delays." In *Proceedings of the International Symposium on Mathematical Theory of Networks and Systems*.

1.3 Future Work

Both research topics presented in this thesis are interesting, and many open questions remain. Especially Papers II and III point out new directions of research.

Much remains to be done for the Relaxed Dynamic Programming in Paper II. The method to find a steady-state solution presented in this thesis, value iteration, is only one way to go. Other methods, such as direct solution of the steady-state inequality, policy iteration, etcetera, should definitely be investigated. New types of value function parameterizations should be developed, and maybe most importantly, new areas of application remain to be found. Some promising applications should also be investigated in-depth. One such example is the switched power controller in Section 3.3, Paper II.

The stability analysis for systems with time-varying delays presented in Paper III could be extended. Robustness and uncertainty issues should be added to the criterion. Also, robust design of time-varying delay compensators based on the stability criterion is certainly a possibility. Work in this direction has already been started in [Lincoln, 2002].

Finally, it would definitely be interesting to apply some of the theory to real world systems, such as for example a switched power controller.

2

Optimal Control and Dynamic Programming

2.1 Introduction

Optimal control is the art of controlling a *system* using a set of *control actions* from an initial state x_0 in the best possible way with respect to some *cost function*. The control may stop after a finite time or continue for infinite time, and there may or may not be a fixed final state x_f which should be reached.

Some optimal control problems, such as the *shortest-path* problem in graphs, or the *LQG* (Linear-Quadratic-Gaussian) problem can be solved with simple, explicit, solutions. Other optimal control problems, with a controllable *uncertainty* in the system (so called *Dual Control* problems) are extremely hard to solve. In this thesis, problems in the gray zone slightly outside of the simple problems are studied. These problems can often be shown to be hard in general. For many realistic problem instances, though, it is possible to solve them in reasonable time by exploiting their structure. This is the approach of Paper I. In Paper II, the approach is to relax the optimality condition to *close to optimality*, and thereby obtain less complex solutions. The basic methodology used in both papers is *Dynamic Programming*.

2.2 Dynamic Programming

The term Dynamic Programming was introduced in the 1950's with Bellman's *Principle of Optimality* (see [Bellman, 1957]):

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

The Principle of Optimality simply states that if one step is taken along an optimal path from **A** to **O**, then the remaining path is also optimal for the new point **B**. See Figure 2.1.

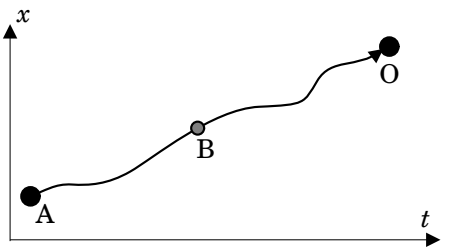


Figure 2.1 *Principle of Optimality:* As **B** is on the optimal path from **A** to **O**, the optimal path from **B** to **O** is what remains of the same path. t is time and x is the state of the system.

The main use of the Principle of Optimality is that if we know all optimal paths for any initial state x at time $n+1$, then every optimal path starting at time n must use one of these optimal paths from time $n+1$ and onwards. See Figure 2.2 for an illustration.

Mathematical Formulation

Let us introduce some mathematical terms to be used next. We let the state of our system be denoted x. This is usually a finite-dimensional vector of real numbers, and in the illustrations we let it have dimension 1 for simplicity (i.e. x is a number). We let time evolve discretely, in steps. At time n with a state of $x(n)$, the controller has to choose a control action $u(n)$ that brings the system to state $x(n+1)$ at time $n+1$ according to the system dynamics $f(x,u)$:

$$x(n+1) = f\big(x(n), u(n)\big). \tag{2.1}$$

To be able to discuss optimality of certain paths and control actions, there must be a notion of cost (or, equivalently, reward). An optimal path is a path from an initial state $x(n)$ which minimizes some cost J. For Dynamic Programming, we often use a time-additive cost, which consists

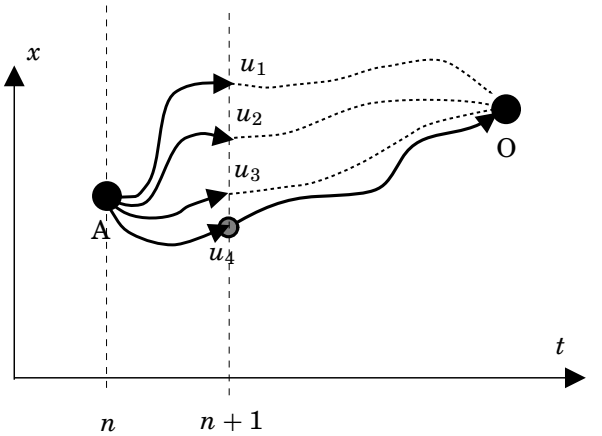


Figure 2.2 Optimal paths for all initial states x from time $n+1$ to O are known. Which is of the actions u_i is optimal to go from A to O?

of a sum of step costs $l(x,u)$:

$$J\Big(x(\cdot),u(\cdot)\Big)=\sum_{n=0}^{N} l\big(x(n),u(n)\big). \tag{2.2}$$

A feedback control law

$$u(n)=\mu(x,n), \tag{2.3}$$

assigns an action for any time n and state x. When this control law is applied to the system, it evolves as

$$x(n+1)=f\big(x(n),\mu(x,n)\big). \tag{2.4}$$

An optimal control law $\mu^*(x,n)$ is one that minimizes the cost J for any initial state.

Optimal Control Action

Having defined what an optimal control law should do, all that remains is to find it. As stated before, if we know the optimal feedback law for any state x at time $n+1$, then all the controller has to decide at time n is which action to take at the first step – the remaining steps are already known. See Figure 2.2 for an illustration.

Let the total cost using the optimal control law from state x, at time $n+1$ be $V^*(x,n+1)$. We call this function the optimal *value function*, and

it will be referred to a lot throughout this thesis (see Figure 2.3). As we know the optimal control law from time $n+1$, we also know $V^*(x, n+1)$.

The task of the controller at time n is to choose an action u such that the sum of the step cost from n to $n+1$ and the remaining cost from $n+1$ is minimized. This is the essence of Bellman's famous equation:

$$V^*(x,n) = \min_{u(n)} \left\{ \underbrace{V^*\Big(f\big(x(n),u(n)\big),n+1\Big)}_{\text{Remaining cost}} + \underbrace{l\big(x(n),u(n)\big)}_{\text{Step cost}} \right\}. \tag{2.5}$$

Given the value function $V^*(x, n+1)$ we can thus calculate both the value function $V^*(x,n)$ and control law $u(n) = \mu(x,n)$ at time n. Therefore, the optimal feedback control law can be found by iteration, starting at the final step and solving backwards in time. This procedure is called *value iteration*. We note that the only data that needs to be kept between each iteration is the value function.

Another way to solve for a steady-state solution of Bellman's equation is *policy iteration*. This means the following two-step method is iterated:

- Given a steady-state control law $\mu_k(x)$, the corresponding infinite-time value function $V_k(x)$ is calculated.
- From a value function $V_k(x)$, an improved control law $\mu_{k+1}(x)$ is calculated.

This method can give very fast convergence when we have a good initial value function (or control law).

Success Stories

In this section we will describe a classical optimization/optimal control problem where Dynamic Programming provides a beautiful solution.

Shortest Path in Graphs. The problem of finding the shortest path in a graph (see Figure 2.4) is very old and well known, and its most famous solution is called Dijkstra's algorithm (see e.g. [Cormen *et al.*, 1989]). This algorithm is an application of Dynamic Programming.

The problem is to find the shortest path to a final state O, given an initial state $x \in X$, where X is the set of all nodes in the graph. This can be viewed as finding an optimal control law which tells us which way to take in each node. To find the *shortest* path, we define the cost function to be

$$J = \sum_{n=0}^{N} l(x,u), \quad l(x,u) = \begin{cases} 1, & \text{if } x \neq \text{O}, \\ 0, & \text{if } x = \text{O}. \end{cases} \tag{2.6}$$

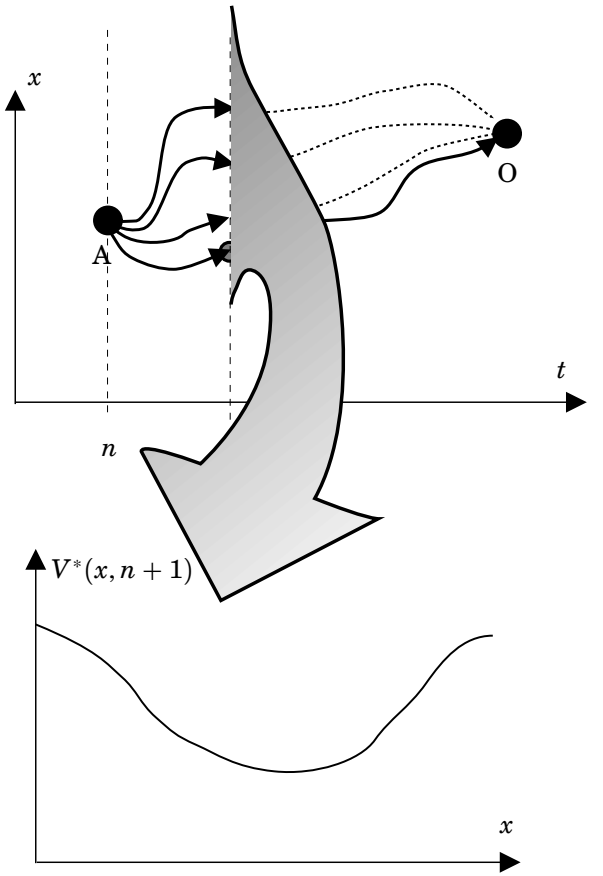


Figure 2.3 The optimal value function $V^*(x, n+1)$ corresponds to the best obtainable cost starting at state x at time $n+1$.

where N is a number larger than the longest distance in the graph. This cost is then equivalent to the length of the path.

The solution of the problem is the following: Let

$$V^*(x, N) = \begin{cases} \infty, & \text{if } x \neq \text{O}, \\ 0, & \text{if } x = \text{O}. \end{cases}$$

Now calculate $V^*(x, N-1)$ from $V^*(x, N)$ using (2.5). The interpretation of (2.5) in this case is "for each node, check all neighboring nodes' distance to O, and let my distance to O be the shortest distance plus one." The

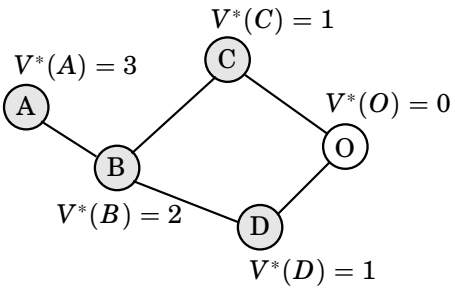


Figure 2.4 A simple shortest path problem. O is the target node. The steady state value function $V^*(x)$ (i.e. the distance to O) is indicated at each node.

possible control action u for each node x is one of the outgoing edges from that node.

After iterating for N steps or less, a steady-state solution $V^*(x)$ is found. $V^*(x)$ is then the distance between any state x and the final state O. The optimal control action (i.e., which outgoing edge to follow) for each node x is obtained from the last iteration of (2.5).

Problems of huge size can easily be solved, thanks to the fact that the value function can be represented by a vector of length $|X|$, where $|X|$ is the number of nodes. The complexity stays the same for each iteration.

Limitations

The shortest path problem in the previous section, as well as the LQG optimal controller, and the Viterbi algorithm in coding theory are famous examples of problems where Dynamic Programming gives beautiful solutions. Unfortunately, these are exceptions.

As mentioned before, the most important entity of Dynamic Programming is the value function $V^*(x, n)$. The value function stores all information about the problem at the current iteration, and must therefore be parameterizable in some way so that it can be stored. For most problems, though, V^* is not easily described; it can be highly irregular, non-convex, discontinuous and so on. Also, the *curse of dimensionality* makes the problem exponentially harder with the number of state variables.

The problems studied in this thesis have the property that the value function $V^*(x, n)$ can be exactly parameterized by a finite but often rapidly increasing number of functions. One common such parameterization is the piecewise linear

$$V^*(x, n) = \max_{\pi \in P(n)} \pi^T x, \tag{2.7}$$

where $P(n)$ is a set of vectors. $V^*(x, n)$ is thus the maximum of a number of hyperplanes, see Figure 2.5. For problems with value functions of this

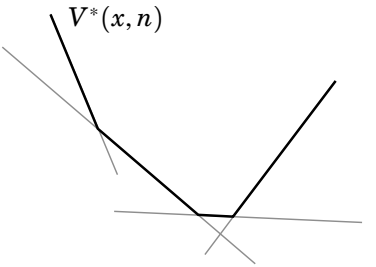


Figure 2.5 A piecewise linear value function.

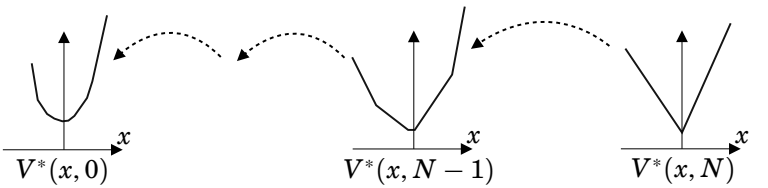


Figure 2.6 The complexity of a value function, growing over iterations.

kind, the difficulty is most often that the complexity (i.e number of functions which the value function is made up of) grows with the number of iterations; see Figure 2.6. For most practical problems, it grows rapidly, and the complexity of the value function becomes prohibitive. One such example can be seen in Figure 2.7, where a two-state double tank control problem with a piecewise linear step cost is being solved. Already after three iterations, the parameterization of $V^*(x,n)$ has become very complex, and the fourth iteration is almost unsolvable.

2.3 Beating Complexity 1: Exploiting Structure

The success of Dynamic Programming in Dijkstra's algorithm as well as the Viterbi algorithm stems from the fact that the problem they solve have a finite state space, and thus the value function can be parameterized by a vector of the same dimension.

For problems with continuous state space, which are common in control, this approach cannot be used. In some problems, though, the *decision space*, i.e. the possible control actions the controller can take is finite. In this way, the value function can be parameterized by a growing set of control *sequences*. For example, if the controller can choose from only two different control actions, 1 and 2, at each time step, the number of possible

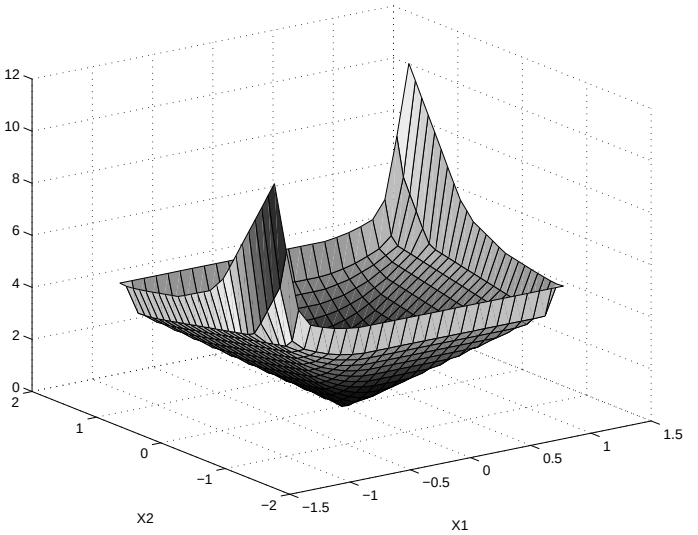


Figure 2.7 The value iteration for the double tank example starts with $V^*(x, N)$ consisting of 8 hyperplanes. Shown in the image is $V^*(x, N-3)$ which is parameterized by 157 hyperplanes. $V^*(x, N-4)$ consists of more than 1000 hyperplanes, and so on. *Note:* The gridding is for plotting purposes only.

control sequences is 2^N for a time horizon of N time steps. Let $K(n, N)$ denote one such sequence from time step n to N, and let the set of all possible sequences be denoted $\kappa(n, N)$. One example of a sequence is

$$K(0,5) = [\,1 \quad 1 \quad 2 \quad 1 \quad 2\,],$$

and $\kappa(0, 5)$ consists of 32 sequences. As the number of possible sequences grows exponentially with the time horizon, some sequences must be removed (pruned) to keep the search complexity at a reasonable level.

For a given initial state $x(0)$ (or probability distribution $P_x(0)$), some of the possible sequences in $\kappa(n, N)$ may be obviously suboptimal. If these are removed carefully, it may still be possible to prove optimality of the best kept sequence. This is the topic of Paper I.

A Switching Control Problem

A specific optimal control problem which is inspired by *Networked Control Systems* is described in this section. It is the main problem studied in Paper I.

Networked Control Systems is a relatively new area of control where the interconnection of control systems and computer networks are studied

(or, at least, the interconnection of network theory and control theory). In some cases it may be useful to have a wireless network to transmit control data, for example if the different sensors and actuators are separated and moving independently. The problem of getting a time-slot to send a packet on the network may be more significant than e.g. classical CPU scheduling, as many networks are rather slow. Therefore, we study a simple problem where one controller controls two plants by communicating over a network, see Figure 2.8. The network is assumed to be time-slotted (such as e.g. Bluetooth [Haartsen, 1998]), and the controller can only communicate with one plant each time slot. We would like to find

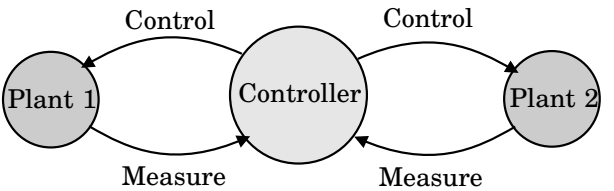


Figure 2.8 A switching control problem: Which is the best sequence to control the two plants if only one can be controlled each time step?

a static *schedule*, i.e. an access sequence, which is the best on average (i.e. in presence of noise).

Assuming linear time-invariant models of the plants

$$x_1(n+1) = \varphi_1 x_1(n) + \gamma_1 u(n) + g_1 v_1(n), \tag{2.8}$$
$$x_2(n+1) = \varphi_2 x_2(n) + \gamma_2 u(n) + g_2 v_2(n), \tag{2.9}$$

where $v_i(n)$ are white, Gaussian noise sources. Note that the φ_i etc. denote matrices of arbitrary dimension in spite of the lower-case notation. The whole system can then be written as

$$x(n+1) = \Phi_{k(n)} x(n) + \Gamma_{k(n)} u(n) + G_{k(n)} v(n), \quad k(n) \in \{1, 2\} \tag{2.10}$$

where

$$\Phi_1 = \begin{bmatrix} \varphi_1 & 0 \\ 0 & \varphi_2 \end{bmatrix}, \qquad \Gamma_1 = \begin{bmatrix} \gamma_1 \\ 0 \end{bmatrix}, \qquad G_1 = \begin{bmatrix} g_1 & 0 \\ 0 & g_2 \end{bmatrix}, \tag{2.11}$$
$$\Phi_2 = \begin{bmatrix} \varphi_1 & 0 \\ 0 & \varphi_2 \end{bmatrix}, \qquad \Gamma_2 = \begin{bmatrix} 0 \\ \gamma_2 \end{bmatrix}, \qquad G_2 = \begin{bmatrix} g_1 & 0 \\ 0 & g_2 \end{bmatrix}. \tag{2.12}$$

The problem is to find a linear feedback law

$$u(n) = -Lx(n),$$

for the continuous control action $u(n)$, and a sequence

$$K(0,N) = [\, k(0) \quad k(1) \quad \dots \quad k(N-1) \,],$$

such that a cost function

$$J = \mathbf{E}_v \left\{ \sum_{n=0}^{N-1} \begin{bmatrix} x(n) \\ u(n) \end{bmatrix}^T Q \begin{bmatrix} x(n) \\ u(n) \end{bmatrix} + x(N)^T Q_N x(N) \right\}$$

is minimized. Q and Q_N are positive symmetric step cost matrices.

If the switching sequence K is fixed, it is easily solved as a standard (time-varying) LQR problem. The problem is, as mentioned before, to prune out unwanted candidate sequences from the exponentially growing set of possible sequences.

The Pruning Method of Paper I

It is rather straightforward to see that the optimal value function in the switching problem is on the form

$$V^*(x,n) = \min_{K \in \kappa(n,N)} \left\{ x^T S_K x + c_K \right\}, \tag{2.13}$$

where the set of S_K matrices represent the state-dependent cost, and the constants c_K represent the expected costs due to noise.

First, we note that for two sequences K_1 and K_2, if

$$S_{K_1} \geq S_{K_2} \quad \text{and} \quad c_{K_1} \geq c_{K_2},$$

then sequence K_1 is worse than sequence K_2 for *all* initial states, and therefore it can be pruned without loss of optimality. Unfortunately, for most problems, this case rarely occurs. Secondly, if

$$S_{K_1} \not\geq S_{K_2} \quad \text{but} \quad c_{K_1} \gg c_{K_2},$$

then for small initial states $x(0)$ it is not very likely that K_1 will ever be optimal due to the very high average noise cost (usually because only one plant was given attention for a long time). The main idea promoted in Paper I is to prune the tree while expanding, and to save information on pruned sequences. The pruning rule in this paper allows a proof of optimality of the best found sequence by using the information on pruned sequences. The proof is based on calculating a lower bound on the obtainable cost from a pruned sequence, and comparing this to the actual best cost obtained. If the lower bound is worse, we know that the pruned

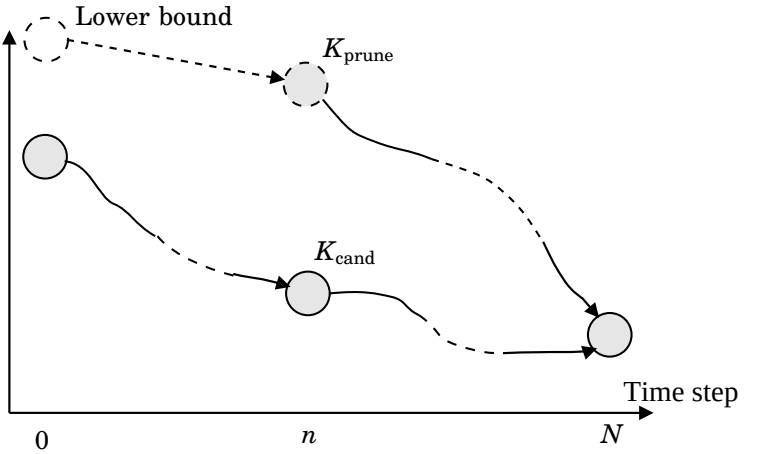


Figure 2.9 At time step n, the sequence K_{prune} has a significantly worse noise cost c_K than K_{cand} and is therefore pruned. By combining its cost by a lower bound on the cost from time 0 to time n, a lower bound on all sequences ending in K_{prune} is obtained. In the illustration the lower bound cost was proven to be higher than the lowest found cost, and therefore optimality is preserved.

sequence could not be optimal. The idea is illustrated graphically in Figure 2.9.

The method in Paper I works fine in many cases, but it may still run in exponential time as the problem is actually NP-hard. The method of beating complexity in the next section has better chances of solving hard Dynamic Programming problems.

2.4 Beating Complexity 2: Relaxing Optimality

The contribution of Paper II in this thesis is a method to relax the optimality requirement of our problems. By allowing us to search for a solution which is *close* to the optimal solution, in many cases a much less complex value function can be found. Throughout the thesis, we will denote such a non-optimal value function $V(x, n)$. For a lack of a better name, we will use general term *Relaxed Dynamic Programming* or *RDP* for the specific relaxation method in Paper II.

The approach of suboptimal Dynamic Programming has been tried before in many ways, most of which imply that not only optimality is lost, but also that the solution has no guaranteed close-to-optimality properties what so ever. The RDP method in this thesis allows the user to pre-specify

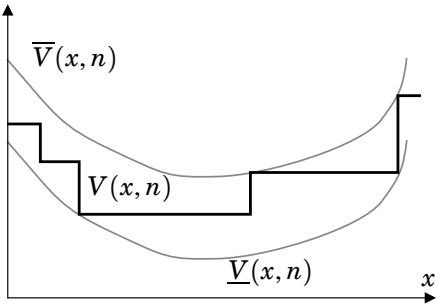


Figure 2.10 A simplified value function $V(x,n)$ between the upper and lower bounds.

an error bound, and the algorithm will search for a solution within that bound.

Approximating the Value Function

In all methods described in this section, including our method, the relaxation of the Dynamic Programming is done by approximating the value function. In the RDP method, this is done by fitting a new function in between upper some and lower bounds, $\overline{V}(x,n)$ and $\underline{V}(x,n)$, respectively. See Figure 2.10. For many of the other methods the idea is simply to get as good fit as possible without guarantees. In this section, some of the most common value function approximations are explained.

Gridding of the State Space. The most immediate way to approximate a general value function is to sample it in a uniform or non-uniform grid of the state space, see Figure 2.11. This works fine for low-dimensional problems (up to two or three), but the complexity grows exponentially with problem dimension. For example, for a simple two-dimensional system with, say, 20 grid points along each dimension, the value function consists of $20^2 = 400$ points. Increasing the state space to dimension four gives 160,000 points, and so on.

Solving the Bellman equation (2.5) in the value iteration is conceptually trivial (loop through all grid points), but computationally expensive. In some cases, it is also possible to solve for the steady-state solution immediately, without value iteration. This is done in [Hedlund and Rantzer, 2002].

Linear Combination of Basis Functions. *Neuro-Dynamic Programming* is a Dynamic Programming method, where the approximation of

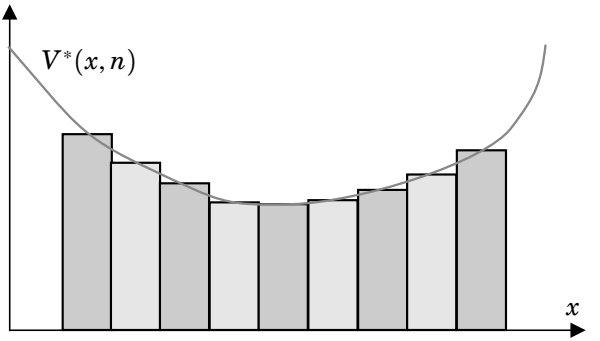


Figure 2.11 Approximation of the value function $V^*(x,n)$ by gridding the state space x.

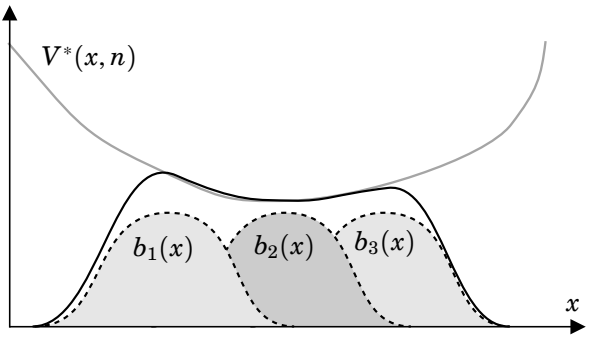


Figure 2.12 Approximation of the value function $V^*(x,n)$ by using basis functions $b_i(x)$.

the value function is based on a linear combination of a set B of basis functions $b_i(x)$.

$$V(x,n) = \sum_{i \in B} \lambda_i(n) b_i(x) \tag{2.14}$$

These basis functions usually have to be chosen in advance by the user, and must be based on a priori knowledge of the the nature of the value function. The gridding method in the previous section is one specific basis function, but generally more smooth basis functions are used. See Figure 2.12 for an illustration. For references on Neuro-Dynamic Programming, see e.g. [Roy, 2001; de Farias and Roy, 2002; Bertsekas and Tsitsiklis, 1996].

The Main Idea

The RDP method presented in Paper II applies to any value function approximation method. Its main contribution is to give upper and lower bounds within which the approximation must stay to guarantee the prespecified error bound.

Briefly, the idea is the following: We define upper and lower step costs, $\overline{l}(x,u)$ and $\underline{l}(x,u)$ respectively, as

$$\overline{l}(x,u) = \overline{\alpha} l(x,u) \quad \overline{\alpha} \geq 1 \tag{2.15}$$

$$\underline{l}(x,u) = \underline{\alpha} l(x,u) \quad \underline{\alpha} \leq 1 \tag{2.16}$$

where $\overline{\alpha}$ and $\underline{\alpha}$ are relaxation bounds, chosen by the user. Note that

$$\overline{\alpha} V^*(x,n) = \overline{\alpha} \min_u \sum_{i=n}^{N} l\big(x(i),u(i)\big) = \min_u \sum_{i=n}^{N} \overline{l}\big(x(i),u(i)\big) \tag{2.17}$$

i.e. the step cost $\overline{l}(x,u)$ gives a value function which is scaled by $\overline{\alpha}$, and similarly for $\underline{l}(x,u)$ and $\underline{\alpha}$. These $\overline{\alpha}$ and $\underline{\alpha}$ will be used as our upper and lower relative error bounds, respectively.

Now assume we have a $V(x,n+1)$ which satisfies

$$\underline{\alpha} V^*(x,n+1) \leq V(x,n+1) \leq \overline{\alpha} V^*(x,n+1), \tag{2.18}$$

i.e. is within the relative error of $\overline{\alpha}$ and $\underline{\alpha}$ from the optimal value function. Then any function $V(x,n)$ that satisfies

$$\begin{aligned} \min_u \Big\{ V\Big(f(x,u),n+1\Big) + \underline{l}(x,u) \Big\} \\ \leq V(x,n) \leq \\ \min_u \Big\{ V\Big(f(x,u),n+1\Big) + \overline{l}(x,u) \Big\} \end{aligned} \tag{2.19}$$

also satisfies

$$\underline{\alpha} V^*(x,n) \leq V(x,n) \leq \overline{\alpha} V^*(x,n). \tag{2.20}$$

Thus, if (2.19) is satisfied, no matter which specific parameterization, the close-to-optimality property (2.20) is also satisfied. This procedure can then be iterated just like the common value iteration, and thanks to the fact that the "slack" is introduced in the step cost and not in the previous value function, the relative error bound will stay the same for each iteration.

In Paper II the method is presented more thoroughly and with many applications from different domains.

2.5 Conclusions

This chapter has introduced Paper I and Paper II. The papers both try to solve optimal control problems using Dynamic Programming, and use different methods to reduce complexity. Paper I exploits the structure of a certain switched linear system problem, whereas Paper II uses a novel general technique to relax the optimality constraint. The latter method has proven to work well for many different problems.

3

Time-Varying Delays

3.1 Introduction

The second part of this thesis consists of two papers on analysis of control systems with time-varying delays. Paper III, "Simple Stability Criteria for Systems with Time-Varying Delays" presents simple, graphical criteria for stability of control systems with time-varying delays. The paper also presents a performance degradation bound based on the same analysis. Paper IV presents a Matlab toolbox used to analyze interconnected continuous-time and discrete-time linear systems with random time-delays (or other random events).

In this chapter, the research on time-varying delays will be motivated by pointing out some reasons for time-varying delays, after which some earlier work will be described, and finally the main ideas from the two papers will be explained.

A Control Loop with Delays

First, let us introduce the type of delay problems we are studying. A typical control loop with delays can be seen in Figure 3.1. Data both from process to controller and from controller to process can be delayed, and we assume that this delay is time-varying. In Paper III, these delays are assumed to be bounded but otherwise freely changing (randomly or worst-case). In Paper IV, more structure is imposed on the delays; it is assumed the delays are random and that the probability distributions are known.

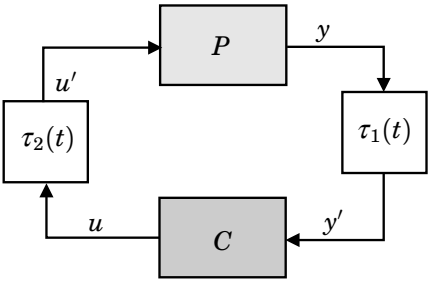


Figure 3.1 A control loop with a process P and a controller C with time delays $\tau(t)$. The delayed outputs are $y'(t) = y(t - \tau_1(t))$ and $u'(t) = u(t - \tau_2(t))$.

3.2 Where Do Time-Varying Delays Come From?

There exists a lot of theory for time-delay systems, but often the papers in the field only cover systems with a fixed but possibly unknown delay. As we will show in this section, there are many cases where the delays are actually time-varying. This motivates the analysis methods presented in this thesis.

Networked Control Systems

Thanks to the development of computer networks after the break-through of the Internet, there are many cheap and reliable network technologies available. At the same time, controllers are most often implemented in computers, and cheap micro-controllers make it possible to connect almost every sensor to a network. A shared network means flexibility and lower costs, as well as better logging and maintenance features. Therefore, it is increasingly popular to use networks to transfer real-time control data.

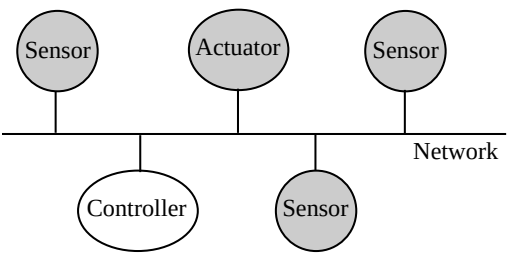


Figure 3.2 A control network: The sensors and actuators may be at different locations, but share a common bus (network). For example, a modern car often uses this technology.

Most network technologies are not designed to do a good job on real-time data. Rather, they are most often designed to maximize the average throughput. As we will see, the result is that real-time data will often exhibit random (or at least time-varying) delays.

Ethernet. This section will briefly describe *Ethernet*, as it includes most of the common delays in todays network technologies. Ethernet is an extremely popular LAN (Local Area Network) technology, thanks to its low price tag and simplicity. The original idea is that all devices share a common bus or network, and that CSMA/CD (Carrier Sense Multiple Access/Collision Detect) is used for bus arbitration. This means that if a host (computer, sensor etc.) wants to send a packet, it listens for activity on the network. If there seems to be none, it starts sending a packet (carrier sensing). At the same time, it listens to the resulting traffic on the network, and if it detects some other transmission it will immediately stop (collision detection). Before it will try again, it will wait for a random amount of time τ in the range

$$0 \leq \tau \leq 2^N \times 51.2\ \mu\text{s} \tag{3.1}$$

where N is the number of retries. This counter will increase for every failed attempt to send the same packet, and thanks to this, probability of a collision will decrease as the random interval increases. This procedure is called *exponential back-off*.

The shortest allowed packet is 512 bits, and for a normal Ethernet transmission rate of 10 Mbit/s it implies a minimum transmission time of 51.2 μs. This packet holds 46 bytes of data, which is more than enough for most control applications. The total delay for sending some data is thus a sum of delays from

- waiting for free time on the network (resource sharing time). This depends on how many hosts are connected to the network and the amount of data generated from each host. Because of other random traffic, but also because the hosts have different sending periods, this delay may be time-varying.
- random waiting after a collision (see above). This delay is by design randomly time-varying.
- data transmission time (see above). This delay is more or less fixed.

In the Ethernet version 10Base-T, all communication takes place over twisted pair copper wires. This means that physically, all wires are point-to-point as opposed to a common bus. Logically, though, the network acts as one bus if the different hosts are connected by *hubs*. A hub simply

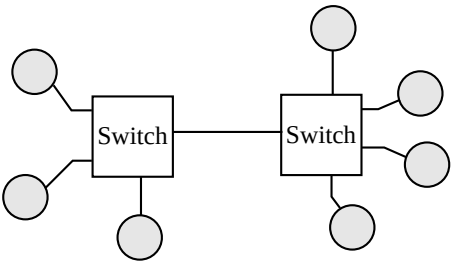


Figure 3.3 An Ethernet network with switches. As opposed to hubs, which are simply mirroring data, switches act as real CSMA/CD hosts with buffers. Therefore much fewer collisions can be expected.

mirrors data from any port to all others without actually reading the data, implying that collisions can take place although hosts are on different physical wires.

One way of removing the risk of exponential back-off delays because of collisions is to use full-duplex *switches* instead of hubs, see [Martinsson, 2002]. A switch acts as a real CSMA/CD host in the network, and it contains data buffers. It waits until the network is free before sending data from one port to the others, thus avoiding collisions. The twisted-pair wires in 10Base-T contains one pair for sending data and one for receiving, and therefore the communication will always be collision-free. Also, a switch sends data only on the wire where the destination host is located (if the packet is not a broadcast packet). This further decreases the amount of traffic on each wire. See Figure 3.3 for an illustration.

CPU Scheduling

Just like Networked Control Systems introduce delays due to resource sharing, the same thing happens when several computer tasks share a single processor. All modern operating systems contain some kind of scheduling mechanism to share the available processing time between active tasks. General-purpose operating systems such as Linux or Windows may introduce a lot of varying delays as they are designed without focus on hard deadlines, whereas specific real-time operating systems such as QNX, VxWorks, or RT-Linux may have more deterministic delay behavior.

Some scheduling frameworks such as TTA [Kopetz, 1997] are designed to give as deterministic scheduling behavior as possible. This comes at a cost, though, as it is more efficient with respect to resource utilization to use a dynamic scheduling such as e.g. Earliest Deadline First (EDF) (see e.g. [Liu, 2000]). For a control process with highly varying execution time, it may be favorable to let it actuate the control signal immediately when

done. This does of course lead to time-varying delays, but if the worst-case delay is long it can still be beneficial for the average process performance.

3.3 Previous Work

This section will present a small subset of the previous research on analysis of systems with time-varying delays. Most often, stability analysis is considered, but in some cases it is also extended to performance analysis.

Analysis — Stability and Performance

The classical Nyquist theorem [Nyquist, 1932] gives a necessary and sufficient condition for stability for a SISO linear time-invariant system in either discrete or continuous time. As *constant* time delays are linear and time-invariant, they can also be incorporated in the system and therefore analyzed by the Nyquist theorem. Thus, it is straightforward to find the delays for which the closed loop system is stable. Note that this theorem is *not* usable for time-varying delays (except for some special cases such as known periodic delay variations).

One common approach to show stability is to assume *bounded* and sometimes *derivative-bounded* delays $\tau(t)$, and to use LMIs to find Lyapunov-like stability proofs. Examples of this technique can be found in e.g. [Kharitonov and Niculescu, 2002; Mehdi *et al.*, 2002; Bugong, 2002; Su *et al.*, 2000; Phoojaruenchanachai and Furuta., 1990]. In [Fridman and Shaked, 2002], a nice overview on conservatism of such approaches is presented followed by a new method to find a stabilizing feedback for time-varying delay systems.

[Liu and Su, 1999; Goubet-Bartholomeus *et al.*, 1997; Goubet *et al.*, 1995] among others use matrix norm criteria for robust stability of time-varying delay systems. Usually, these criteria are relatively simple and require little or no numerical optimization. Unfortunately, when the uncertainty of the system description is removed (not the varying time-delay, though), the criteria become conservative.

In many papers, such as [Ekanayake *et al.*, 2001; Kolmanovsky and Maizenberg, 2000; Nilsson, 1998], the time-varying delays are modeled by a Markov chain. In the linear case, this forms a Jump Linear System, for which a lot of analysis theory exist. This method is also used in Paper IV in this thesis.

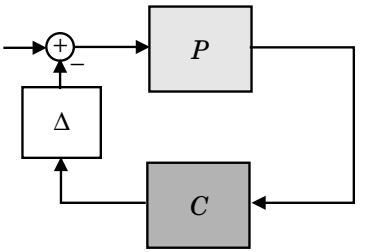


Figure 3.4 The delay set-up in Paper III, where Δ is the time-varying delay operator. Note that this is equivalent to the loop in Figure 3.1 as long as the controller does not use any information on the delay τ_1.

3.4 Simple Stability Criteria — Paper III

As mentioned in the previous section, a lot of research has been done in analysis of systems with time-varying delays. Many of the recent results are formulated as Linear Matrix Inequalities (LMI), which are often nicely solvable. The stability criteria often give yes/no answers, though, and cannot tell the control designer how to change the design to improve. The main contribution of Paper III in this thesis is a set of *simple* graphical stability criteria for continuous-time, discrete-time and mixed systems. The stability can be checked in a Bode diagram of the closed loop system $G_{\text{cl}} = \frac{CP}{1+CP}$, making it very easy to adjust the controller to improve the stability margin. The paper also indicates that the criteria are not very conservative, and finally gives a simple performance degradation bound in presence of delays.

Problem Formulation

In Paper III, the delay loop in Figure 3.4 is studied for three cases: continuous-time process and controller, discrete-time process and controller and finally a continuous-time process and a discrete-time controller (which is the most realistic case). The delay operator Δ simply delays a signal $u(t)$ to $u'(t)$ as

$$u'(t) = u(t - \tau(t)), \quad 0 \leq \tau(t) \leq \tau_{\max}. \tag{3.2}$$

There are no restrictions on how $\tau(t)$ may vary, except that it has to be bounded by $\tau_{\max}$.

Stability Through Small Gain Theorem

The main idea of the paper is to use the Small Gain Theorem to prove stability. The delay is transformed to a direct feedthrough path and an

error path; see Figure 3.5.

In all three cases, it turns out that the delay error path can be written as an operator Λ in series with a differentiator. For the continuous-time case, the operator Λ is simply

$$\Lambda = (\Delta - 1) \circ \frac{1}{s} \tag{3.3}$$

with a slight abuse of operator notation for the integrator. It is assumed that the closed loop system without delay is stable. The Small Gain Theorem can then be used to prove stability of the whole system.

The gain of the subsystem from point **A** in Figure 3.5 to point **B** is easily calculated, as the system is linear. In the continuous-time case, it is also time-invariant, and thus the gain is

$$\gamma_{AB} = \sup_{\omega} \left| \frac{P(i\omega)C(i\omega)}{1 + P(i\omega)C(i\omega)} \omega \right| \tag{3.4}$$

In all three cases, the gain of the operator Λ is relatively straight-forward to calculate. In the continuous-time case the $\mathbf{L}_2$-gain is

$$\gamma_{BA} = \tau_{\max}. \tag{3.5}$$

Thus, the Small Gain Theorem guarantees stability if

$$\begin{aligned} \gamma_{\text{loop}} = \gamma_{AB} \cdot \gamma_{BA} < 1 \quad &\Rightarrow \\ \left| \frac{P(i\omega)C(i\omega)}{1 + P(i\omega)C(i\omega)} \right| < \frac{1}{\tau_{\max}\omega} \quad &\forall \omega \end{aligned} \tag{3.6}$$

This criterion is easily checked in a Bode diagram of the closed loop system without delays. For the designer of the controller, it is also immediately clear how to alter the design if the criterion does not hold. Finally, in the paper, it is indicated that this criterion is not very conservative.

Performance

Thanks to the fact that an upper bound on the total loop gain is calculated in the stability analysis, a simple bound on performance deterioration due to time-varying delays can be obtained.

The performance measure that is considered in this thesis is the $\mathbf{L}_2$-gain γ_{xd} from a disturbance d to the output x in Figure 3.6. For the system without delays, the gain is simply

$$\gamma_{xd} = \sup_{\omega} \left| \frac{W(i\omega)}{1 + P(i\omega)C(i\omega)} \right|, \tag{3.7}$$

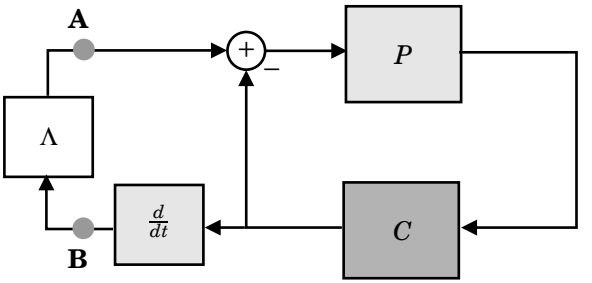


Figure 3.5 The system in Figure 3.4 with the delay Δ expanded to a direct feedthrough path and a delay difference operator. It turns out that this delay difference is best written as an operator Λ in series connection with a differentiator.

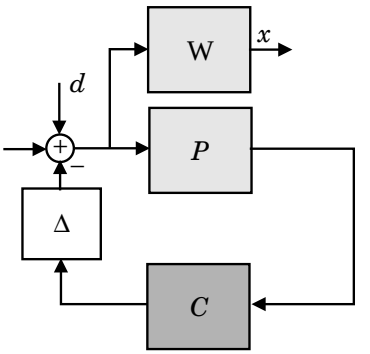


Figure 3.6 Performance measured as gain from a disturbance d to the filtered output x.

i.e. the maximum gain of the sensitivity function filtered by the weighting function W. Using small gain arguments, it is easy to show that the performance gain γ_{xd} with time-varying delays is at most

$$\gamma_{xd} \leq \sup_{\omega} \left| \frac{W(i\omega)}{1 + P(i\omega)C(i\omega)} \right| \frac{1}{1 - \gamma_{\text{loop}}}. \tag{3.8}$$

Thus, performance is at most degraded by a factor of $\frac{1}{1-\gamma_{\text{loop}}}$. For example, if the maximum delay is half of what is allowed for stability, the gain will at most increase by a factor of $\frac{1}{1-0.5} = 2$.

3.5 The JITTERBUG Toolbox — Paper IV

When more details on the system, and in particular the delays, are known, it may be interesting to do exact computations of the system behavior and performance. For linear systems, Gaussian noise, and independent random events (such as delays), things like state variance and quadratic costs are in theory easily calculated. The problem is that solving a specific problem often requires a lot of special program code. Writing and debugging this program is usually very tedious. Therefore, the MATLAB toolbox JITTERBUG was developed and presented in Paper IV. The theory used in the paper is old and well known, and the contribution of the toolbox is mainly to make the theory easily applicable.

A Typical JITTERBUG Problem

Typically, when analyzing the effects of delays, there is a mix of continuous-time components and discrete-time components in the system. If there were no continuous-time (real-world) components, the delays would typically not be meaningful. On the other hand, if there were no discrete-time components, there would in many cases be no jitter. Therefore, one of the main features of JITTERBUG is the ability to interconnect any combination of discrete-time and continuous-time components.

A typical JITTERBUG problem can be seen in Figure 3.7. The system consists of a continuous-time plant $P(s)$, a periodic sampler (period h seconds), a delay τ from sampler to controller, a discrete-time controller $C(z)$ and finally an actuator. This set-up could model a Networked Control System, where the delays are introduced by sending data over a network, or simply a controller implemented in a shared processor which introduces computation delays due to other competing processing tasks.

In JITTERBUG, time is discretized to a grain of δ seconds, which is chosen by the user. This means that the delay probabilities are discrete, and can be given as vectors P_τ:

$$\text{Prob}(\tau = k\delta) = P_\tau(k), \quad P_\tau = [\, p(0) \quad p(1) \quad \dots \,]. \tag{3.9}$$

Note that this assumption of time-discrete delay probabilities does not change the fact that all calculations in JITTERBUG are exact, including continuous-time systems and noise.

Signal Model. As JITTERBUG deals with both signals and timings, two different model concepts are needed. The first is the usual signal model, which tells JITTERBUG how signals and systems are connected – see Figure 3.8a. All inputs to the system are described by white noise (which can of course be colorized by adding filters).

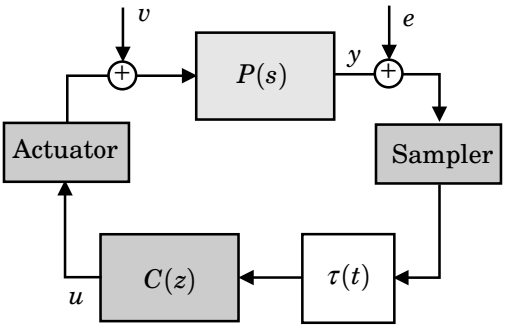


Figure 3.7 A typical problem for JITTERBUG. The system consists of a continuous-time plant $P(s)$ and a discrete-time controller $C(z)$, with a time-varying delay in between. Typical problems are: What is the quadratic performance of the system? What if the controller is "gain-scheduled" on the delay τ?

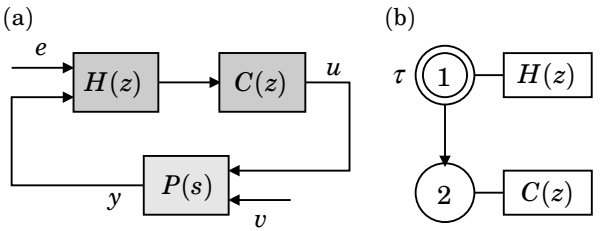


Figure 3.8 The JITTERBUG models for the system in Figure 3.7, where the sampler is introduced as the block $H(z)$. (a) is the signal model, and (b) is the timing model. The double-circled timing node is a periodic node, which means that it is restarted once every sample period.

Timing Model. The second model is the timing model. It consists of a number of *timing nodes* (or execution nodes), to which the execution of the discrete-time components can be bound. The timing model can be seen as a Markov graph with delays and transition probabilities. See Figure 3.8b. In the toolbox, also more advanced Markov graphs can be constructed.

Performance Measures. The toolbox calculates two performance measures:

- A quadratic cost function

$$J = \lim_{T\to\infty} \frac{1}{T} \int_0^T x^T(t) Q x(t)\, dt$$

where Q is a positive semidefinite matrix chosen by the user, and

x are all states in the system. Both continuous-time and discrete-time states use this continuous-time cost. This kind of performance measure is very common in control.

- The Power Spectral Density (PSD) for all output signals. This can be used to get a hint on what changes in the system when, for example, delays are introduced.

As the calculations in JITTERBUG are usually very fast, it is easy to sweep over one or several parameters in the system and plot performance graphs. One such graph is shown in Figure 3.9, where the mean delay in the system from sampler to actuator is varied, as well as the "variance" of the delay around the mean time.

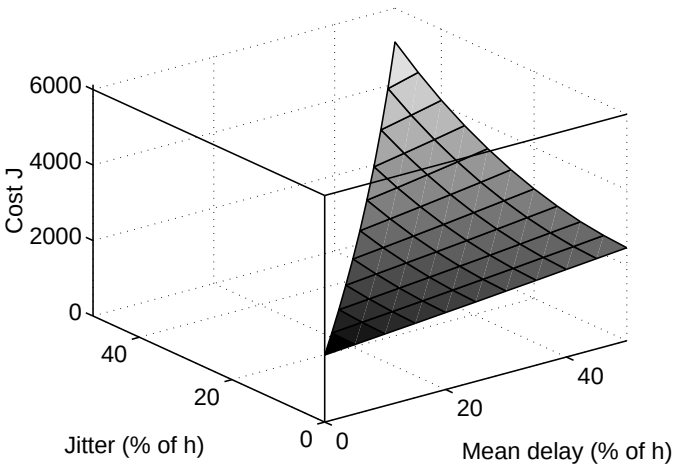


Figure 3.9 The cost of delay and jitter a control example. The controller is designed assuming a constant delay equal to the mean delay.

Internal Workings of JITTERBUG

As mentioned before, JITTERBUG requires that all delays are discretized to a user-definable grain of δ seconds. Thanks to this, all continuous-time states, as well as costs and noises can be discretized to this grain (without approximation), and the system can be written as a Jump Linear System [Krasovskii and Lidskii, 1961]. If the system is periodic, i.e. has some timing node which is executed periodically, the variances, and thus costs, can be calculated algebraically. In the case of aperiodic systems, JITTERBUG also has an iterative solver.

3.6 Conclusions

This chapter has introduced Papers III-IV, which deal with analysis of systems with time-varying delays. The former paper presents a set of simple graphical stability criteria for systems with bounded but otherwise unknown time-delays. It is indicated that the criteria are not very conservative. Paper IV presents JITTERBUG, a MATLAB toolbox which does exact performance calculations on systems with random events. The time-varying delays must be independent and have known probability distributions. The main contribution of the toolbox is to make known theory easily applicable to common real-time problems.

References

Bellman, R. E. (1957): *Dynamic Programming*. Princeton Univ. Press, Princeton, N.J.

Bertsekas, D. P. and J. Tsitsiklis (1996): *Neuro-Dynamic Programming*. Athena Scientific.

Bugong, X. (2002): "Further results on the stability of linear systems with multiple delays." *Journal of Mathematical Analysis and Applications*, **267**, pp. 20–28.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest (1989): *Introduction to algorithms*. MIT Press, Cambridge, MA.

de Farias, D. and B. V. Roy (2002): "Approximate dynamic programming via linear programming." *Advances in Neural Information Processing Systems 14, MIT Press*.

Ekanayake, M., K. Premaratne, and C. Douligeris (2001): "Stability of discrete-time systems with time-varying delays." In *Proceedings of the 2001 American Control Conference*, vol. 5, pp. 3914–391. IEEE.

Fridman, E. and U. Shaked (2002): "An improved stabilization method for linear time-delay systems." *IEEE Transaction on Automatic Control*, **47:11**, pp. 1931–1937.

Goubet, A., M. Dambrine, and J. Richard (1995): "An extension of stability criteria for linear and nonlinear time delay systems." In *Proceedings of the IFAC Conference on System Structure and Control*, pp. 278–283. Nantes, France.

Goubet-Bartholomeus, A., M. Dambrine, and J. Richard (1997): "Stability of perturbed systems with time-varying delays." *Systems and Control Letters*, **31**, pp. 155–163.

Haartsen, J. (1998): "Bluetooth–the universal radio interface for ad hoc, wireless connectivity." *Ericsson Review*, **3**, pp. 110–117.

Hedlund, S. and A. Rantzer (2002): "Convex dynamic programming for hybrid systems." *IEEE Transactions on Automatic Control*, **47:9**, pp. 1536–1540.

Kharitonov, V. and S.-I. Niculescu (2002): "On the stability of linear systems with uncertain delay." In *Proceedings of the 2002 American Control Conference*, vol. 3, pp. 2216–2220.

Kolmanovsky, I. and T. Maizenberg (2000): "Stochastic stability of a class of nonlinear systems with randomly varying time-delay." In *Proceedings of the 2000 American Control Conference*, vol. 6, pp. 4304–4308.

Kopetz, H. (1997): *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Pub., Boston, MA.

Krasovskii, N. and E. Lidskii (1961): "Analytic design of controllers in systems with random attributes, I, II, III." *Automation and Remote Control*, **22:9–11**, pp. 1021–1025, 1141–1146, 1289–1294.

Lincoln, B. (2002): "Jitter compensation in digital control systems." In *Proceedings of the 2002 American Control Conference*.

Liu, J. (2000): *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ.

Liu, P.-L. and T.-J. Su (1999): "Stability for single and large-scale uncertain systems with time-varying delays." *IEE Proceedings on Control Theory and Applications*, **146**, pp. 591–597.

Martinsson, A. (2002): "Scheduling of real-time traffic in a switched ethernet network." Technical Report Masters thesis ISRN LUTFD2/TFRT--5683--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Mehdi, D., E. K. Boukas, and Z. K. Liu (2002): "Dynamical systems with multiple time-varying delays: Stability and stabilizability." *Journal of Optimization Theory and Applications*, **113:3**, pp. 537–565.

Nilsson, J. (1998): *Real-Time Control Systems with Delays*. PhD thesis ISRN LUTFD2/TFRT--1049--SE, Department of Automatic Control, Lund Institute of Technology, Sweden.

Nyquist, H. (1932): "Regeneration theory." *Bell Syst. Tech. J.*, **11**, pp. 126–147.

Phoojaruenchanachai, S. and K. Furuta. (1990): "Stability and stabilization of uncertain linear systems with time-varying state delays." In *Proceedings of the 29th IEEE Conference on Decision and Control, 1990*, vol. 3, pp. 1622–1623.

Roy, B. V. (2001): *Handbook of Markov Decision Processes: Methods and Applications*, chapter Neuro-Dynamic Programming: Overview and Recent Trends. Kluwer.

Su, T., C. Lu, and G. Jong (2000): "An LMI approach for robust stability of linear uncertain systems with time-varying multiple state delays." In *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 2, pp. 1507–1508. IEEE.

Paper I

LQR Optimization of Linear System Switching

Bo Lincoln, Bo Bernhardsson

Abstract

This paper considers off-line optimization of a switching sequence for a given finite set of linear control systems, together with joint optimization of control laws. A linear quadratic full information criterion is optimized and dynamic programming is used to find an optimal switching sequence and control law. The main result is a method for efficient pruning of the search tree to avoid combinatoric explosion. A method to prove optimality of a found candidate switch sequence and corresponding control laws is presented.

1. Introduction and Motivation

Optimal hybrid control problems arise in many applications, see e.g. [Branicky and Mitter, 1995; Branicky *et al.*, 1998]. An interesting subclass of hybrid systems consists of piecewise linear systems where either controlled or uncontrolled switches between linear systems are used, see [DeCarlo *et al.*,]. The main question in most optimal hybrid control problem formulations is how to avoid the combinatorical explosion associated with exploring all possible switching alternatives. The problem area combines the traditionally separate research areas of search over graphs and control theory. In most papers on optimal hybrid control, this issue is generally not dealt with. The problem is sometimes solved by exhaustive search. In this paper we propose and evaluate a promising pruning method for efficient tree search, obtained by using information about the search objective.

The motivation for our work has mainly come from real-time control systems, where there often are restrictions on common resources such as communication bandwidth or CPU power. The different control loops have to share some media. This is often done by time-division-multiplexing, i.e. using some time slots for one loop and some other for another loop. One example where this problem is found is control over a wireless network environment such as Bluetooth [Haartsen, 1998]. The data packets are long and the maximum sample rate is restricted. In Bluetooth only one network device can be accessed every 1.25 ms, so the controller has to choose which device to control (or sample). See Figure 1.

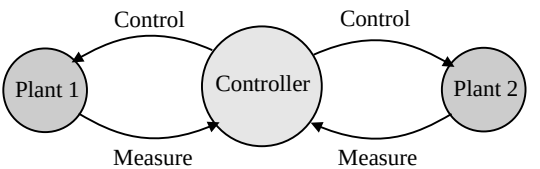


Figure 1. A simple problem. The controller can only access one plant each time slot. Which sequence gives the best expected cost?

The scheduling, i.e the choice of control and measurement sequences, is normally optimized off-line. The possibility to use on-line information in the scheduling algorithms, such as local information about signal values, has also been suggested recently, see [Årzén *et al.*, 2000]. Such on-line scheduling will not be studied here.

Off-line scheduling of linear control systems under quadratic criteria has been treated recently in [Skafidas and Nerode, 1998; Skafidas *et al.*, 1997; Rehbinder and Sanfridson, 2000]. These references, however, do not

present any efficient solving methods and lead to search problems over large trees. When the control horizon increases, the size of the trees grows exponentially. The purpose of the present paper is to present a pruning method which often decreases this complexity drastically.

2. Problem Formulation

The problems we are interested in can be formulated as finding the best switching sequence of system matrices for a discrete-time linear system

$$z(n+1) = \Phi(n)z(n) + \Gamma(n)u(n) + G(n)v(n). \tag{1}$$

Here z is the (extended) state space vector, u the control signals, and v standard stochastic, independent, disturbances with zero mean and unit covariance. The system matrices $\Phi(n)$, $\Gamma(n)$ and $G(n)$ are chosen by the controller in each step from a small set of M alternative systems $\{(\Phi_k, \Gamma_k, G_k, Q_k)\}, k \in \{1, \ldots, M\}$ (where Q_k has to do with the cost of the system). Note that the system is time varying *only* since the controller can choose system matrices from a set at every control instant. The set of possible matrices does *not* change over time, so the *problem* is time-invariant.

The problem is to find a linear feedback law

$$u(n) = -L(n)z(n),$$

and a sequence

$$K(0,N) = [\, k(0) \quad k(1) \quad \ldots \quad k(N-1) \,]$$

corresponding to choosing $\Phi(n) = \Phi_{k(n)}$, $\Gamma(n) = \Gamma_{k(n)}$, $G(n) = G_{k(n)}$, and $Q(n) = Q_{k(n)}$ that minimize the cost

$$V\Big(P_z(0), 0, N, L(\cdot), K(0,N)\Big) = \underset{v}{\mathbf{E}}\left\{ \sum_{n=0}^{N-1} \begin{bmatrix} z(n) \\ u(n) \end{bmatrix}^T Q(n) \begin{bmatrix} z(n) \\ u(n) \end{bmatrix} + z(N)^T Q_N z(N) \right\}, \tag{2}$$

where $\mathbf{E}\{z(0)\} = 0$, $\mathbf{E}\{z(0)z(0)^T\} = P_z(0)$, $Q(n) \geq Q_N \geq 0$.

3. Finding an Optimal Sequence

We will find an optimal scheduling sequence and control law by doing backwards recursion of the cost (dynamic programming), evaluating all possible choices of $K(n,N)$. See Figure 2 for an illustration. If this is done without care, the tree will of course grow exponentially. Therefore, we present a pruning algorithm which aims at keeping the tree size down to a reasonable level. The whole optimization of the sequence is done off-line, so no feedback information is used in the scheduling.

We will use the notation "optimal sequence" for a sequence which achieves the optimal cost. There may be more than one sequence which does this, and we will aim at finding at least one.

Notation: Throughout the rest of the paper, the cost function V will be written as

$$V\big(start, end, sequence\big),$$

where *start* and *end* denote the first and last time-step of the cost. *Sequence* is either a sequence of choices (such as e.g. $K(0,N)$) from the start to the end time-step, a set of such sequences, or omitted. If *sequence* is a set, V denotes the minimum over all sequences in the set, and if omitted V denotes the minimum over all possible sequences. The initial variance $P_z(0)$ is omitted for notational convenience.

3.1 Cost Representation and Feedback Gain

For a fixed choice of $K(0,N)$, the problem is a standard time-varying linear-quadratic control problem. Under standard assumptions, it is well known that the best achievable cost can be written as

$$V\big(0,N,K(0,N)\big) = z(0)^T S_{K(0,N)} z(0) + c_{K(0,N)}, \tag{3}$$

where $S_{K(0,N)}$ is a positive symmetric matrix and $c_{K(0,N)}$ is a constant term due to the noise. The optimal feedback law is

$$u(n) = -L_{K(n,N)} z(n) = {F^{uu}_{K(n,N)}}^{-1} {F^{zu}_{K(n,N)}}^T z(n), \tag{4}$$

where

$$F_{K(n,N)} = \left[\begin{array}{c|c} F^{zz}_{K(n,N)} & F^{zu}_{K(n,N)} \\ \hline F^{uz}_{K(n,N)} & F^{uu}_{K(n,N)} \end{array}\right] = Q + [\,\Phi_k \quad \Gamma_k\,]^T\, S_{K(n+1,N)}\, [\,\Phi_k \quad \Gamma_k\,]\,. \tag{5}$$

As the minimization of L is straightforward, it will be left out in the remaining sections.

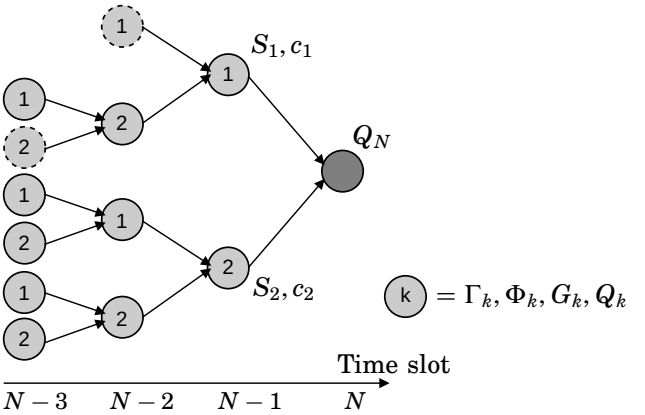


Figure 2. The control sequence tree for $M = 2$ when expanding all possibilities from $N - 3$ to N (except for two nodes, dashed, where the tree was pruned).

3.2 Finding a Candidate Sequence

Finding a candidate sequence is, as mentioned before, done by backwards iteration. One "step" in the iteration means expanding the set of candidate sequences one step in time. This is done by first expanding the search tree with new possible sequence choices (see Figure 2), and then pruning (removing branches) using the algorithm described below. The set $\kappa_{\text{cand}}(n+1, N)$ of possible control sequences from time $n+1$ to N is expanded by

$$\kappa_{\text{expand}}(n, N) = \{1, 2, \ldots, M\} \times \kappa_{\text{cand}}(n+1, N). \tag{6}$$

Let $K_{\text{prune}} \in \kappa_{\text{expand}}(n, N)$ be a sequence in the set. The corresponding cost is parameterized by $S_{\text{cand}} = S_{K_{\text{prune}}}(n, N)$ and $c_{\text{cand}} = c_{K_{\text{prune}}}(n, N)$. The idea of the algorithm is to remove K_{prune} if another sequence $K_{\text{cand}} \in \kappa_{\text{expand}}(n, N)$ will perform better for all states. The algorithm calculates an $\alpha \in [0, 1]$ showing how close the quadratic cost S_{prune} is to be worse than S_{cand}, with $\alpha = 1$ meaning that K_{prune} is worse for all states. If α is close enough to 1, and the noise cost c_{prune} is sufficiently much larger than c_{cand}, then K_{prune} is removed (tested by the inequality in step 3 of the algorithm in Table 1).

"Sufficiently much larger" is here represented by one parameter, $R > 0$, which must be chosen by hand. The exact meaning of R is shown in Theorem 1.

The algorithm creates a set $M(n, N)$, called *motivation set*, which contains data on every pruned sequence and on the candidate sequences

which were judged better. This data is used in Section 3.3 to prove optimality of the found sequence. The algorithm is described in Table 1.

1. Start with $\kappa_{\text{prune}}(n,N)$, $\kappa_{\text{cand}}(n,N)$, and $M(n,N)$ being empty sets and calculate $c_{\min} = \min_{K\in\kappa_{\text{expand}}(n,N)} c_K$.
2. If $\kappa_{\text{expand}}(n,N)$ is empty then, quit else choose $K_{\text{prune}} \in \kappa_{\text{expand}}(n,N)$ for possible pruning.
3. If $\exists K_{\text{cand}} \in \kappa_{\text{expand}}(n,N) \cup \kappa_{\text{cand}}(n,N)$ such that for
$$\alpha = \max\{0 \le \lambda \le 1 \mid (S_{K_{\text{prune}}} - Q_N) \ge \lambda(S_{K_{\text{cand}}} - Q_N)\}$$
it holds that
$$c_{\text{prune}} - c_{\text{cand}} - (1-\alpha)(c_{\min} + R - c_{\text{cand}}) \ge 0$$
then K_{prune} is pruned, i.e.
   - Move K_{prune} from the set $\kappa_{\text{expand}}(n,N)$ to the set $\kappa_{\text{prune}}(n,N)$.
   - If $K_{cand} \notin \kappa_{\text{cand}}(n,N)$, move it there from $\kappa_{\text{expand}}(n,N)$.
   - Let $M(n,N)$=$M(n,N) \cup \{(c_{\text{prune}}, \alpha, c_{\text{cand}})\}$.
   - Go to step 2.
4. Else move K_{prune} from the set $\kappa_{\text{expand}}(n,N)$ to $\kappa_{\text{cand}}(n,N)$. Go to step 2.

Table 1. The branch-and-bound algorithm

By using this tree pruning, the number of sequences in $\kappa_{\text{cand}}(n,N)$ can be kept reasonably low when recursing backwards if R is chosen small enough. After N iteration stages, the final proposed sequence is

$$K_{\text{cand}}(0,N) = \underset{K\in\kappa_{\text{cand}}(0,N)}{\operatorname{argmin}} \left(\mathbf{tr}(P_z(0)S_K) + c_K\right), \tag{7}$$

with cost $V_{\text{cand}} = V(0,N,K_{\text{cand}}(0,N))$.

3.3 Optimality of the Candidate Sequence

The candidate sequence found by the algorithm above may or may not be optimal, depending on the choice of R. A method will now be presented

which can prove optimality of the proposed sequence if R is large enough. The idea of the proof is to show that a lower bound on the obtainable cost by using one of the pruned sequences is still higher than the cost of the found sequence. If this holds for every pruned sequence, the found candidate is optimal. Throughout the rest of the section, $M(n,N)$ is the motivation set from the tree pruning, and consists of

$$M(n,N) = \Big\{(c_{\text{prune}},\alpha,c_{\text{cand}})_1,(c_{\text{prune}},\alpha,c_{\text{cand}})_2,\dots\Big\}.$$

We will also use $V_{\text{lowb}}\big(0,N,\kappa_{\text{cand}}(n,N)\big)$, which fulfills

$$\min_{\substack{K(0,n)\\ K(n,N)\in\kappa_{\text{cand}}(n,N)}} V\Big(0,N,[\,K(0,n)\quad K(n,N)\,]\Big) \geq V_{\text{lowb}}\Big(0,N,\kappa_{\text{cand}}(n,N)\Big). \tag{8}$$

Thus $V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(n,N))$ is a lower bound on the optimal cost using one of the sequences in $\kappa_{\text{cand}}(n,N)$ for steps from n to N (from now on *passing* $\kappa_{\text{cand}}(n,N)$).

LEMMA 1—BEST COST INCLUDING PRUNED SEQUENCES
It holds that

$$\begin{aligned} V\Big(0,N,\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)\Big) \geq \\ \min_{(c_{\text{prune}},\alpha,c_{\text{cand}})\in M'(n,N)} \Big\{(1-\alpha)V_{\text{lowb}}(0,n)+ \\ \alpha\Big(V_{\text{lowb}}\big(0,N,\kappa_{\text{cand}}(n,N)\big)-c_{\text{cand}}\Big)+c_{\text{prune}}\Big\} =: \\ V_{\text{lowb}}\Big(0,N,\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)\Big), \end{aligned} \tag{9}$$

where $V_{\text{lowb}}(0,n)$ is a lower bound on the optimal cost in the length-n-problem, and $M'(n,N) = M(n,N)\cup\big\{(0,1,0)\big\}$ to include the current lower bound.

Thus $V_{\text{lowb}}\big(0,N,\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)\big)$ is a lower bound for the cost achieved if no sequences were pruned at step n. If

$$V_{\text{cand}} = V_{\text{lowb}}\Big(0,N,\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)\Big)$$

then V_{cand} is also the optimal cost passing $\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)$. □

For notational convenience we put

$$\sigma_0^{n-1} = \sum_{i=0}^{n-1}\begin{bmatrix} z(i)\\ u(i)\end{bmatrix}^T Q_{k(i)}\begin{bmatrix} z(i)\\ u(i)\end{bmatrix}$$

in what follows:

PROOF OF LEMMA 1
First, we notice that (8) gives

$$\min_{\substack{K(0,n)\\ K(n,N)\in\kappa_{\text{cand}}(n,N)}} V\Big(0,N,[\,K(0,n)\quad K(n,N)\,]\Big) =$$

$$\min_{\dots}\mathop{\mathbf{E}}_{v}\Big\{\sigma_0^{n-1}+z(n)^T S_{K(n,N)}z(n)+c_{K(n,N)}\Big\}\geq V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(n,N))$$

$$\Rightarrow$$

$$\min_{K(0,n)}\mathop{\mathbf{E}}_{v}\Big\{\sigma_0^{n-1}+z(n)^T S_{K(n,N)}z(n)\Big\}\geq$$
$$V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(n,N))-c_{K(n,N)},\quad \forall K(n,N)\in\kappa_{\text{cand}}(n,N).$$

Second, for two sequences from n to N, $K_{\text{prune}}(n,N)$ and $K_{\text{cand}}(n,N)$, for which

$$S_{K_{\text{prune}}(n,N)}-Q_N\geq\alpha\Big(S_{K_{\text{cand}}(n,N)}-Q_N\Big),\tag{10}$$

and $K_{\text{cand}}\in\kappa_{\text{cand}}(n,N)$ it holds that

$$\min_{K(0,n)}\mathop{\mathbf{E}}_{v}\Big\{\sigma_0^{n-1}+z(n)^T S_{K_{\text{prune}}(n,N)}z(n)\Big\}\geq$$
$$\min_{K(0,n)}\mathop{\mathbf{E}}_{v}\Big\{\sigma_0^{n-1}+z(n)^T Q_N z(n)+\alpha z(n)^T(S_{K_{\text{cand}}(n,N)}-Q_N)z(n)\Big\}\geq$$
$$\alpha\min_{K(0,n)}\mathop{\mathbf{E}}_{v}\Big\{\sigma_0^{n-1}+z(n)^T S_{K_{\text{cand}}(n,N)}z(n)\Big\}+$$
$$(1-\alpha)\min_{K(0,n)}\mathop{\mathbf{E}}_{v}\Big\{\sigma_0^{n-1}+z(n)^T Q_N z(n)\Big\}\ \geq$$
$$\alpha\Big(V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(n,N))-c_{K_{\text{cand}}(n,N)}\Big)+(1-\alpha)V_{\text{lowb}}(0,n).\tag{11}$$

Thus, we can put a lower bound on the optimal cost passing a pruned sequence $K_{\text{prune}}(n,N)\in\kappa_{\text{prune}}(n,N)$ using lower bounds for the cost of the length-n-problem and for the cost of sequences passing $\kappa_{\text{cand}}(n,N)$:

$$\min_{K(0,n),}V\Big(0,N,[\,K(0,n)\quad K_{\text{prune}}(n,N)\,]\Big)=$$
$$\min_{K(0,n)}\mathop{\mathbf{E}}_{v}\Big\{\sigma_0^{n-1}+z(n)^T S_{K_{\text{prune}}(n,N)}z(n)\Big\}+c_{K_{\text{prune}}(n,N)}\geq$$
$$\alpha\Big(V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(n,N))-c_{K_{\text{cand}}(n,N)}\Big)+(1-\alpha)V_{\text{lowb}}(0,n)+c_{K_{\text{prune}}(n,N)}.\tag{12}$$

A lower bound of all sequences passing $\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)$ is obtained by taking the minimum of lower bounds for all pruned sequences and the remaining sequences, yielding Equation (9). □

LEMMA 2—LOWER BOUND ON FULL COST
Given

- the candidate sequence $K_{\text{cand}}(0,N)$ from equation (7)
- lower bounds $V_{\text{lowb}}(0,i)$ for the length-i-problems, $i \in \{0,\ldots,N-1\}$
- the pruned sequence motivations $M(i,N)$ $i \in \{0,\ldots,N\}$

a lower bound $V_{\text{lowb}}(0,N)$ on the optimal cost can be found by iterating equation (9) from $n=1$ to $n=N$. The iteration is started from

$$V_{\text{lowb}}\Big(0,N,\kappa_{\text{cand}}(0,N)\cup\kappa_{\text{prune}}(0,N)\Big)=V_{\text{cand}},$$

and after the Nth iteration we define

$$V_{\text{lowb}}(0,N)=V_{\text{lowb}}\Big(0,N,\kappa_{\text{cand}}(N,N)\cup\kappa_{\text{prune}}(N,N)\Big).$$

If $V_{\text{cand}}=V_{\text{lowb}}(0,N)$, then $K_{\text{cand}}(0,N)$ is a sequence that gives the optimal cost. □

PROOF OF LEMMA 2
Since all sequences $K(n,N)\in\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)$ pass $\kappa_{\text{cand}}(n+1,N)$, the iteration can use that

$$V_{\text{lowb}}\Big(0,N,\kappa_{\text{cand}}(n,N)\cup\kappa_{\text{prune}}(n,N)\Big)=V_{\text{lowb}}\Big(0,N,\kappa_{\text{cand}}(n+1,N)\Big). \quad (13)$$

Let $V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(0,N))=V_{\text{cand}}$. Equation (13) and Lemma 1 gives $V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(i,N))$, $i\in\{0,\ldots,N\}$. A lower bound on the cost for the length-N-problem is then $V_{\text{lowb}}(0,N)=V_{\text{lowb}}(0,N,\varnothing)$. □

Using Lemma 2 iteratively, lower bounds (or optimal costs) for the length-N-problem can be found by starting with a length-1-problem and iterating. The lower bound on the solution for each problem is found and used in the calculation of lower bounds for larger problems. The sequence tree can be kept from the previous problem length and expanded by one step for each iteration, keeping complexity low.

THEOREM 1—OPTIMAL SEQUENCE FOR FINITE R
If

$$R\geq V(0,N)-\Big(V(0,n)+c_{\min}(n,N)\Big),\quad \forall n\in\{0...N\}, \quad (14)$$

then an optimal sequence will be found. Also, if the optimal costs for problem lengths $1,\ldots,N-1$ have been found before s.t. $V_{\text{lowb}}(0,n)=$

$V_{\text{cand}}(0,n) = V(0,n),\ \forall n \in \{0,\dots,N-1\}$, then the found cost for problem length N will also be proven optimal by Lemma 2. □

Note: This theorem cannot be used to choose R directly, as $V(0,N)$ is not known before the optimization. It only states that for large enough R, we will find and prove the optimal cost. The second assumption in the theorem, that optimal costs must be found for all shorter problems, is automatically satisfied if the tree is expanded iteratively and (14) holds in each step.

PROOF OF THEOREM 1

To show that all optimal sequences cannot be pruned, we note that the lower bound on costs of pruned sequences in Eq. (9) holds also when all lower bounds on candidate sequence costs are replaced by optimal costs. We want to show that these lower bounds are all worse than the optimal cost:

$$\forall (c_{\text{prune}}, \alpha, c_{\text{cand}}) \in M,$$
$$\Big((1-\alpha)V(0,n) + \alpha\big(V(0,N) - c_{\text{cand}}\big) + c_{\text{prune}}\Big) - V(0,N) \geq 0, \quad (15)$$

and expanding the left hand side

$$\begin{aligned}
(1-\alpha)\Big(V(0,n) - V(0,N)\Big) - \alpha c_{\text{cand}} + c_{\text{prune}} \geq \\
(1-\alpha)\Big(V(0,n) - V(0,N)\Big) - \alpha c_{\text{cand}} + c_{\text{cand}} + \\
+ \underbrace{(1-\alpha)(c_{\min}(n,N) + R - c_{\text{cand}})}_{\text{From pruning rule}} = \\
(1-\alpha)\Big(\Big\{V(0,n) + c_{\min}(n,N) + R\Big\} - V(0,N)\Big) \geq 0 \Leftarrow \\
R \geq V(0,N) - \Big(V(0,n) + c_{\min}(n,N)\Big). \quad (16)
\end{aligned}$$

This shows the role of R: $V(0,n)$ is the cost for the problem from 0 to n and $c_{\min}(n,N)$ is the cost for the problem from n to N without initial variance. If we add the costs of these two shorter problems together it will be lower than the cost from 0 to N ($V(0,N)$), and R has to be larger than the difference for the inequality to hold.

Thus, if (14) holds, the optimal sequence is found, i.e. $V_{\text{cand}}(0,N) = V(0,N)$. If also $V_{\text{lowb}}(0,n) = V(0,n)$, then the lower bound on cost from pruned sequences in Eq. (15) equals the one in (9) and from the latter we obtain

$$V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(n,N) \cup \kappa_{\text{prune}}(n,N)) = V_{\text{lowb}}(0,N,\kappa_{\text{cand}}(n,N)).$$

Thus, Lemma 2 will prove that our candidate sequence is optimal. □

By keeping R small, the number of branches in the tree can be kept down to a reasonable level. If R is chosen too small, the optimal solution might however not be found, or at least not be proven optimal using Lemma 2. A lower bound on the optimal case is always found, though.

4. Time Complexity of the Algorithm

The described algorithm does not run in polynomial time (unless possibly if P=NP). In fact, one special version of the problem (which was not considered in the design of the algorithm) is easily shown to be NP-hard. The proof is based on solving an *N-mortality problem* which is NP-complete (see [Blondel and Tsitsiklis, 1997]).

5. Examples

To show the feasibility of the method, two examples have been constructed. They are based on the select-which-system-to-control problem, with different properties. The optimization times range from seconds to minutes in Matlab code on a standard computer.

5.1 Example 1

Consider the following two simple linear systems:

$$\begin{aligned} x_1(n+1) &= \begin{bmatrix} 1 & 0.4 \\ 0.3 & 0.8 \end{bmatrix} x_1(n) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(n) + \begin{bmatrix} 0 \\ 0.8 \end{bmatrix} v_1(n), \\ x_2(n+1) &= 0.9 x_2(n) + 1 u(n) + 1 v_2(n), \\ Q &= \mathrm{diag}\Big(\begin{bmatrix} 1 & 1 & 10 & 1 \end{bmatrix} \Big). \end{aligned}$$

The problem is to find the control access sequence which yields the lowest cost. In this problem, the control signal will only be held at the actuator for one sample period. If the system is not being controlled during the next sample period, the control signal is zero. Running the described tree optimization algorithm with $R = 23$ and $N = 300$ produce candidate sequences for all length-i-problems, $i \in \{0, \ldots, 300\}$, which all prove to be optimal. For illustration, the algorithm has been run with $R = 15$, and $R = 10$ as well. Optimality could not be shown for these choices of R, but for $R = 15$ the same cost as for $R = 23$ was achieved. See Figure 3 for the optimal sequence, and Figure 4 for the number of sequences left in each

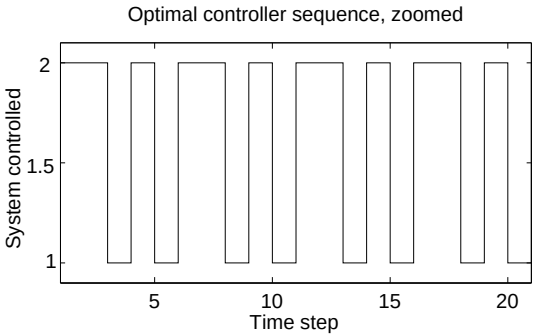


Figure 3. The optimal controller sequence for Example 1, which is "periodic" with period 5. Generally, the algorithm often finds sequences with short period, but not always.

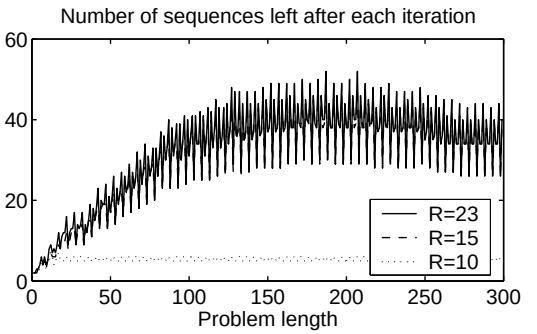


Figure 4. The number of sequences left after pruning in each iteration for Example 1. The $R = 15$ case is almost the same as for $R = 23$ and therefore not visible.

iteration of the algorithm. Figure 5 compares found costs and guaranteed lower bounds for the different choices of R.

5.2 Example 2

We now show a larger example consisting of the two systems in Example 1 plus another unstable second order system. The control signal is held at the actuator if the system is not controlled, so extra "control-signal" states have been added, making the original order 5 system grow to order 8. The optimal controller sequence for the length-100-problem can be seen in Figure 6, and the tree size can be seen in Figure 7.

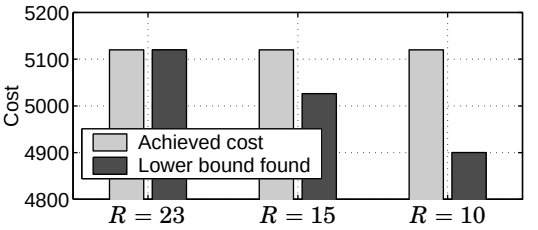


Figure 5. Found sequence costs and lower bounds when running the tree pruning algorithm with different R's.

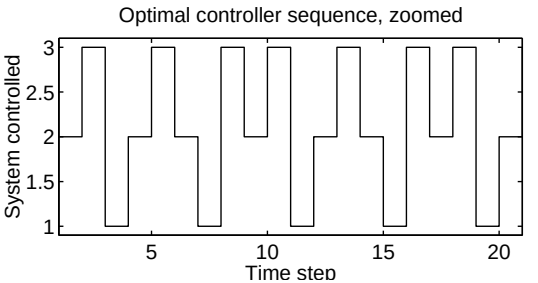


Figure 6. The optimal controller sequence for Example 2, with three choices in each time step (left). The control signal is held when the system is not controlled, so the augmented system is of order 8. The sequence is periodic with period 8.

6. Conclusions

A method to find the optimal switching sequence in a linear-quadratic full-information problem has been presented, together with a method to prove

Figure 7. The number of sequences left after pruning in each iteration of Example 2. Note that although the tree becomes rather large, it is still solvable, and a length-100-problem should give us a good insight in the steady-state behavior.

optimality in each case. Empirically, the method works well in that it finds the solution in reasonable time. Future work could include formulating other problems in the same framework, such as for example choosing among distributed sensors. The infinite horizon problem and the problem of joint actuator scheduling and sensor scheduling are open.

References

Årzén, K.-E., A. Cervin, J. Eker, and L. Sha (2000): "An introduction to control and scheduling co-design." In *Proceedings of the 39th IEEE Conference on Decision and Control*. Sydney, Australia.

Blondel, V. D. and J. N. Tsitsiklis (1997): "When is a pair of matrices mortal?" *Information Processing Letters*, **No 63**, pp. 283–286.

Branicky, M. and S. Mitter (1995): "Algorithms for optimal hybrid control." In *Proceedings of the 34th IEEE Conference on Decision and Control*, pp. 2661–2666.

Branicky, M. S., V. S. Borkar, and S. M. Mitter (1998): "A unified framework for hybrid control: Model and optimal control theory." *IEEE Transactions on Automatic Control*, **43:1**, pp. 31–45.

DeCarlo, R., M. Branicky, S. Pettersson, and B. Lennartsson "Perspectives and results on the stability and stabilizability of hybrid systems." *To appear in IEEE Transactions on Automatic Control*.

Haartsen, J. (1998): "Bluetooth–the universal radio interface for ad hoc, wireless connectivity." *Ericsson Review*, **3**, pp. 110–117.

Rehbinder, H. and M. Sanfridson (2000): "Scheduling of a limited communication channel for optimal control." In *Proceedings of the 39th Conference on Decision & Control*.

Skafidas, E., R. Evans, and I. Mareels (1997): "Optimal controller switching for stochastic systems." In *Proceedings of the 36th Conf. on Decision and Control*, pp. 3950–3955.

Skafidas, E. and A. Nerode (1998): "Optimal measurement scheduling in linear quadratic gaussian control problems." In *Proceedings of the 1998 IEEE Int. Conf. on Control Applications*, pp. 1225–1229.

Paper II

Relaxing Dynamic Programming

Bo Lincoln, Anders Rantzer

Abstract

The idea of dynamic programming is general and very simple, but the "curse of dimensionality" is often prohibitive and restricts the fields of application. This paper introduces a method to reduce the complexity by relaxing the demand for optimality. The distance from optimality is kept within prespecified bounds and the size of the bounds determines the computational complexity.

Several computational examples are considered. The first is optimal switching between linear systems, with application to CPU scheduling as well as design of a DC/DC voltage converter. This example is followed by several others such as control of piecewise linear systems, and Partially Observable Markov Decision Processes (POMDPs).

This paper is based on "Relaxed Dynamic Programming" by the same authors, submitted to *IEEE Transactions on Automatic Control*, 2003.

1. Introduction

1.1 Motivation

Since the 1950's, the idea of dynamic programming [Bellman, 1957; Bertsekas, 2000] has propagated into a vast variety of applications. This includes as diverse problems as portfolio theory and inventory control in economics, shortest path problems in network routing and speech recognition, task scheduling in real time programming, and receding horizon optimization in process control.

For many optimal control problems, dynamic programming could have been used to solve the problem if there had been an efficient way to parameterize the optimal value function $V^*(x)$. Unfortunately, this is most often not the case. This paper presents a way of relaxing Dynamic Programming (DP) to make it possible to find a more easily parameterized suboptimal value function $V(x)$ within a strict, pre-specified, distance from the optimal $V^*(x)$.

The paper first introduces some background, followed by the main DP relaxation method in Section 2. Then five different applications are presented. Section 8, finally, contains some theoretical convergence results.

1.2 Dynamic Programming

Let $x(n) \in X$ be the state of a given system at time n. $u(n) \in U$ is the value of the control signal. Given these, the state of a system evolves as

$$x(n+1) = f(x(n), u(n)). \tag{1}$$

Also, given a cost function

$$J = \sum_{n=0}^{\infty} l(x(n), u(n)), \tag{2}$$

we would like to find an optimal control policy $u = \mu(x)$, such that the cost J is minimized from any initial state. A function which returns the the resulting cost from any state x for a specific control law μ is called a *value function* and is denoted $V^\mu(x)$. The best value function describing the optimal cost for each x is denoted $V^*(x)$.

A common method to solve for the optimal value function is *value iteration*, i.e. to start at an initial value function $V_0(x)$ and update iteratively using Bellman's equation:

$$V_{k+1}(x) = \min_u \Big\{ V_k\big(f(x,u)\big) + l(x,u) \Big\}. \tag{3}$$

This can be seen as solving a finite-time-horizon dynamic programming problem of length $k+1$ from the solution of length k. Under general conditions, it can be shown that the iteration converges to $V^*(x)$, which satisfies

$$V^*(x) = \min_u \Big\{ V^*\big(f(x,u)\big) + l(x,u) \Big\}. \tag{4}$$

For a more extensive treatment of dynamic programming, see e.g. [Bertsekas, 2000].

1.3 Relaxed Value Function

The main contribution of this paper is to give a method to approximate the optimal value function $V^*(x)$, which *guarantees* that the suboptimal solution is within a *user-specified distance* from the optimal solution.

Approximating value functions has been done before in a variety of ways. One popular approach is *neuro-dynamic programming*; see e.g. [Roy, 2001] or [Bertsekas and Tsitsiklis, 1996] for an introduction. Specifically, [de Farias and Roy, 2002; Schweitzer and Seidmann, 1985] present methods where the approximate value function is parameterized as a linear combination of a set of basis functions. The problems studied are often Partially Observable Markov Decision Processes (see Section 6) of very high dimension, and the methods give no error bounds on the obtained solutions.

It should also be mentioned that many formal verification techniques of nonlinear or hybrid systems are related to this work. For example, the toolbox HyTech, presented in [Henzinger *et al.*, 1995], calculates over- and under-approximations of a reachable set. This can be viewed as calculating upper and lower bounds of a value function.

In this paper we focus on problems where $V_k(x)$ has a finite description whose complexity grows rapidly with iteration number k. Again, the contribution of the paper is a simple algorithm to find a sub-optimal value function $V(x)$ which is within a pre-specified distance from the true $V^*(x)$. Using this method, some problems for which dynamic programming has been considered hopeless can now be practically solved.

1.4 An Example

To illustrate the main idea, consider the following: Let A_1 and A_2 be two system matrices and let let a discrete-time dynamical system evolve as

$$x(n+1) = A_i x(n), \quad i \in \{1,2\}.$$

The goal is to find a control law, that given the current state $x(n)$ assigns

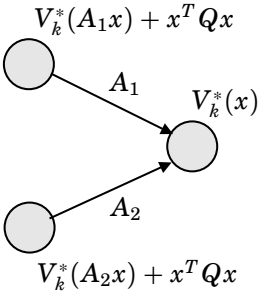


Figure 1. Iterating the Bellman equation one step, which can be viewed as going backwards in time.

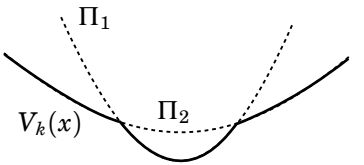


Figure 2. The piecewise quadratic value function $V_k(x)$.

an i (i.e. system matrix) for this time step so as to minimize

$$J = \sum_{k=0}^{\infty} x(k)^T Q x(k).$$

Let us consider value iteration for this problem, starting with $V_0(x) = 0$. At each time step, there are two choices; use A_1 or A_2 (see Figure 1). This leads to a sequence of functions $V_k(x)$ of the form

$$V_k(x) = \min_{\Pi \in P_k} x^T \Pi x$$

where P_k is a set of positive matrices (see Figure 2 for an illustration). Typically, the size of P_k grows exponentially with k due to the two choices in each time step. In the following section we present a method of finding an approximate value function for this problem.

2. Relaxed Dynamic Programming

This section will describe a method to find a $V(x)$ which fulfills

$$\min_u \Big\{ V\big(f(x,u)\big) + \underline{l}(x,u) \Big\} \le V(x) \le \min_u \Big\{ V\big(f(x,u)\big) + \overline{l}(x,u) \Big\}. \qquad (5)$$

The inequalities imply that

$$\min_u \sum_{n=0}^{\infty} \underline{l}(x(n),u(n)) \le V(x) \le \min_u \sum_{n=0}^{\infty} \overline{l}(x(n),u(n)). \tag{6}$$

Here $\overline{l}$ and $\underline{l}$ are chosen to satisfy $\underline{l}(x,u) \le l(x,u) \le \overline{l}(x,u)$, for example

$$\overline{l}(x,u) = \overline{\alpha} l(x,u), \qquad \overline{\alpha} \ge 1 \tag{7}$$

$$\underline{l}(x,u) = \underline{\alpha} l(x,u), \qquad \underline{\alpha} \le 1. \tag{8}$$

With this relaxation of Bellman's equation, we can search for a solution $V(x)$ that is more easily parameterized than $V^*(x)$.

2.1 Relaxed Value Iteration

Given $V_{k-1}(x)$ satisfying

$$\min_u \sum_{n=0}^{k-1} \underline{l}(x(n),u(n)) \le V_{k-1}(x) \le \min_u \sum_{n=0}^{k-1} \overline{l}(x(n),u(n)), \tag{9}$$

define $\overline{V}_k(x)$ and $\underline{V}_k(x)$ according to

$$\overline{V}_k(x) = \min_u \Big\{ V_{k-1}\big(f(x,u)\big) + \overline{l}(x,u) \Big\}. \tag{10}$$

and

$$\underline{V}_k(x) = \min_u \Big\{ V_{k-1}\big(f(x,u)\big) + \underline{l}(x,u) \Big\}. \tag{11}$$

The expressions for $\overline{V}_k(x)$ and $\underline{V}_k(x)$ are generally more complicated than for $V_{k-1}(x)$. From this, a simplified $V_k(x)$ which satisfies

$$\underline{V}_k(x) \le V_k(x) \le \overline{V}_k(x) \tag{12}$$

is calculated. This $V_k(x)$ satisfies

$$\min_u \sum_{n=0}^{k} \underline{l}(x(n),u(n)) \le V_k(x) \le \min_u \sum_{n=0}^{k} \overline{l}(x(n),u(n)), \tag{13}$$

and the procedure can be iterated.

The iteration of (10–12), which we call *relaxed value iteration*, can often be used to find a solution of (5). A strict criterion for convergence is given in Section 8.

$\overline{\alpha}$ and $\underline{\alpha}$ (as well as $\overline{l}$ and $\underline{l}$) are chosen as a trade-off between complexity (time and memory) and accuracy. Note in particular that if l is chosen as in (7)–(8), then the relative error in the value function defined by $\overline{\alpha}$ and $\underline{\alpha}$ is independent of the number of iterations.

2.2 Stopping Criterion

If the value function $V_k(x)$ at iteration k satisfies (5) for $V(x) = V_k(x)$, then it also satisfies (6), and a solution to the problem has been found. If the iteration is stopped before (5) holds, it is possible to calculate $\overline{l}(x,u)$ and $\underline{l}(x,u)$ for which it does hold and thus test for which relaxation the current $V_k(x)$ holds as a solution. This is done in Section 5.5.

Remark: This method of calculating the slack to optimality can be used *no matter how* $V(x)$ was obtained. For example, a $V(x)$ obtained using a receding-horizon method for some optimal control problem of length N could be used.

2.3 Parameterization of V

As mentioned before, the aim of this paper is problems where the value function $V(x)$ can be described by a finite parameterization, typically growing fast with time. The problems which will be presented in this paper all have value functions which can be written as

$$V_k(x) = \operatorname*{select}_{p \in P_k} p(x), \tag{14}$$

where P_k is a set of (simple) functions on x, and the "select" operator selects one of the functions according to some criterion (e.g. "maximum", "minimum", or "feasible region"). In this paper, we define the complexity of the representation as $|P_k|$, i.e. the number of elements in the set P_k. If

$$V_k(x) = \min_{p \in P_k} p(x),$$

we will denote the value function "minimum-type". Note that adding a new function p to P_k *decreases* (or leaves unchanged) $V(x)$ for all x if V is minimum-type. This will be used in the next section.

2.4 A Simple Algorithm to Calculate V

The value iteration in Section 2.1 does not tell us how $V_k(x)$ is calculated to satisfy (12). Doing this in an optimal way with respect to complexity of the optimization may be a very hard problem. In this section we present a simple and efficient (albeit not optimal) algorithm to obtain $V_k(x)$ for the minimum-type parameterization defined in the previous section. The algorithm is described in Procedure 3.1.
The algorithm simply adds elements from the lower bound $\underline{V}_k(x)$ until the resulting value function $V_k(x)$ satisfies the upper bound $V_k(x) \leq \overline{V}_k(x)$. The resulting value function satisfies (12) by construction. Naturally, the algorithm can be made more efficient by changing details such as removing functions in $\overline{P}_k(x)$ once they have been tested as non-active. For

PROCEDURE 3.1—GENERATING V_k FROM V_{k-1} (MINIMUM-TYPE)

1. Calculate $\overline{V}_k$ and $\underline{V}_k$ from V_{k-1}:

$$\overline{V}_k(x) = \min_u \Big\{ V_{k-1}\big(f(x,u)\big) + \overline{l}(x,u) \Big\}$$
$$\underline{V}_k(x) = \min_u \Big\{ V_{k-1}\big(f(x,u)\big) + \underline{l}(x,u) \Big\}$$

   Define $\overline{P}_k$ and $\underline{P}_k$ such that

$$\overline{V}_k(x) = \min_{p \in \overline{P}_k} p(x) \quad \text{and} \quad \underline{V}_k(x) = \min_{p \in \underline{P}_k} p(x)$$

2. Let $P_k = \emptyset$.

3. Define

$$V_k(x) = \begin{cases} \min\limits_{p \in P_k} p(x) & \text{if } P_k \neq \emptyset \\ \infty & \text{if } P_k = \emptyset \end{cases}.$$

   If possible, find $x_0 \in X$ such that

$$V_k(x_0) > \overline{V}_k(x_0).$$

   If not, $V_k(x)$ satisfies (12) $\Rightarrow$ Done.

4. Let

$$\underline{p} = \operatorname*{argmin}_{p \in \underline{P}_k} p(x_0)$$

   and add $\underline{p}$ to the set P_k. Go to step 3.

□

maximum-type value functions the procedure is simply changed to add from $\overline{V}_k(x)$ until the lower bound holds.

Procedure 3.1 can now be iterated until (5) or some other stopping criterion is satisfied.

3. Application: Switched Linear Systems with Quadratic Costs

In this section, a linear system switching problem will be described. Given a set of alternative system matrices for a linear system, the problem is to find a control law both for the continuous inputs and for switching between system matrices at each time step. All switches are initiated by the control law; there is no autonomous switching. The example in Section 1.4 is a special case of this problem.

For the N triples of system matrices

$$\Big\{(\Phi_1, \Gamma_1, Q_1), \ldots, (\Phi_N, \Gamma_N, Q_N)\Big\},$$

consider the linear system

$$x(n+1) = \Phi_{w(n)} x(n) + \Gamma_{w(n)} u(n), \tag{15}$$

and cost function

$$J = \sum_{n=0}^{\infty} \underbrace{\begin{bmatrix} x(n) \\ u(n) \end{bmatrix}^T Q_{w(n)} \begin{bmatrix} x(n) \\ u(n) \end{bmatrix}}_{l(x(n),u(n))}, \tag{16}$$

for $w = 1, \ldots, N$. The problem is to find a feedback control law

$$\begin{aligned} u(n) &= \mu(x(n)), \\ w(n) &= \nu(x(n)), \quad w(n) \in \{1, 2, \ldots, N\}, \end{aligned} \tag{17}$$

such that the closed loop system system

$$x(n+1) = \Phi_{\nu(x(n))} x(n) + \Gamma_{\nu(x(n))} \mu(x(n)), \tag{18}$$

minimizes J for every initial state.

3.1 Value Function

We assume $V_{k-1}(x)$ at iteration $k-1$ to be on the form

$$V_{k-1}(x) = \min_{\Pi \in P_{k-1}} x^T \Pi x, \tag{19}$$

where P is a set of non-negative, symmetric matrices (see Figure 2 for an illustration). To obtain a relaxed value function, the procedure in Section 2

is used with $\underline{\alpha} = 1$ and $\overline{\alpha} = \alpha > 1$ as the relaxation parameter. The lower bound $\underline{V}_k(x)$ is calculated as in (11):

$$\underline{V}_k(x) = \min_{u,w}\left\{ \min_{\Pi\in\underline{P}_{k-1}} \begin{bmatrix} x \\ u \end{bmatrix}^T [\,\Phi_w \;\; \Gamma_w\,]^T \,\Pi\, [\,\Phi_w \;\; \Gamma_w\,] \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} x \\ u \end{bmatrix}^T Q_w \begin{bmatrix} x \\ u \end{bmatrix} \right\} =$$

$$\min_{w,\Pi\in\underline{P}_{k-1}} \min_{u} \begin{bmatrix} x \\ u \end{bmatrix}^T \Big\{ \underbrace{[\,\Phi_w \;\; \Gamma_w\,]^T \,\Pi\, [\,\Phi_w \;\; \Gamma_w\,] + Q_w}_{F_{w,\Pi}} \Big\} \begin{bmatrix} x \\ u \end{bmatrix} =$$

$$\min_{w,\Pi\in\underline{P}_{k-1}} \begin{bmatrix} x \\ -L_{w,\Pi}\cdot x \end{bmatrix}^T F_{w,\Pi} \begin{bmatrix} x \\ -L_{w,\Pi}\cdot x \end{bmatrix} = \min_{\Pi\in\underline{P}_k} x^T \Pi x. \quad (20)$$

This is on the same form as $V_{k-1}(x)$, and thus we can theoretically continue the value iteration. The size $|\underline{P}_k|$ can be up to $N\cdot|P_{k-1}|$, though, so worst-case complexity of $V_k(x)$ grows exponentially with k. This is why the proposed relaxed dynamic programming is needed.

The upper bound $\overline{V}_k(x)$ is calculated in the same way as $\underline{V}(x)$. The two sets of matrices $\overline{P}_k$ and $\underline{P}_k$ are stored as *ordered sets* and $\overline{P}_k$ is sorted so that

$$\operatorname{tr}\overline{\Pi}_i \le \operatorname{tr}\overline{\Pi}_j \quad \forall i<j.$$

$\underline{P}_k$ is ordered like $\overline{P}_k$, i.e. such that $\overline{\Pi}_i$ and $\underline{\Pi}_i$ correspond to the same discrete choice $w(n)$. This implies that $\overline{\Pi}_i \ge \underline{\Pi}_i,\ \forall i$. After that, Procedure 3.2 (which is a special case of Procedure 3.1) is used to calculate V_k.

PROCEDURE 3.2—RELAXED $V_k(x)$, SWITCHED SYSTEM

1. Let $P_k = \emptyset$ and define
$$V_k(x) = \min_{\Pi \in P_k} x^T \Pi x.$$
2. Pick the first $\overline{\Pi} \in \overline{P}_k$ and remove it from $\overline{P}_k$.
3. If there exists x s.t.
$$x^T \overline{\Pi} x < x^T \Pi x \quad \forall \Pi \in P_k$$
(tested using S-procedure [Yakubovich, 1971]) then
   - Pick the first $\underline{\Pi} \in \underline{P}_k$.
   - Add this $\underline{\Pi}$ to P_k and remove $\underline{\Pi}$ from $\underline{P}_k$.
   - Go to step 2.
4. Remove the first $\underline{\Pi}$ from $\underline{P}_k$.
If $\overline{P}_k \neq \emptyset$, go to step 2.

□

Remark: The sorting of $\overline{P}_k$ on trace ensures that small Π's are added first to P_k. In practice it means that more $\overline{\Pi}$'s can be discarded and thus a smaller set P_k is found.

3.2 Example — Scheduling of Inverted Pendulums

Consider the classical inverted pendulum. As the system is unstable, the controller needs to give the system constant attention to stabilize it. Assume there are several pendulums to be controlled by one controller simultaneously. If the controller has limited computing or communication resources and can only make one control decision per time unit, one obvious problem is to choose which pendulum to control each time.

We can pose such a problem based on a linearized model of a rotating inverted pendulum ("the Furuta pendulum") from a student lab i Lund.

$$\frac{dx}{dt}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 31.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.588 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ -71.2 \\ 0 \\ 191 \end{bmatrix} u(t) \tag{21}$$

The step cost matrix is set to

$$Q = \mathrm{diag}\Big(\begin{bmatrix} 1 & 0.1 & 1 & 0.1 & 1 \end{bmatrix}\Big).$$

A reasonable sample period for this pendulum is around 20–50 ms. As several pendulums are to be controlled, the controller "time-slot" is set to $h = 20$ ms. The sampled system matrices are denoted Φ and Γ. We assume that a pendulum which does not get attention in the current time slot holds its previous control signal. This increases the system order.

For the two-pendulum-problem, the system matrices become

$$\begin{aligned} \Phi_1 &= \begin{bmatrix} \Phi & \Gamma & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \Phi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \Gamma_1 = \begin{bmatrix} 0 \\ 0 \\ \Gamma \\ 1 \end{bmatrix}, \\ \Phi_2 &= \begin{bmatrix} \Phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \Phi & \Gamma \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \Gamma_2 = \begin{bmatrix} \Gamma \\ 1 \\ 0 \\ 0 \end{bmatrix}, \end{aligned} \tag{22}$$

which is essentially the augmented system consisting of two pendulums plus states for holding the control signal.

For the two pendulum problem, a reasonable α is $\alpha = 1.1$, meaning we have at most a 10% slack to the optimal solution. The resulting complexity-over-iterations graph can be seen in Figure 3. As the system is unstable and also sampled quite fast, the value iteration takes about 100 steps to converge (the steady state $V(x)$ is on the magnitude of 1000 times $l(x, 0)$). As can be seen, the complexity stays under 10 for all iterations, though. The result of the optimization is a feedback control law for choosing which of the two pendulums to control, as well as linear feedback laws for the continuous control signal u.

Extending the problem to three pendulums, the dynamic programming gets harder. Setting $\alpha = 1.5$, the problem is still solvable, and the resulting complexity graph can be seen in Figure 4. Note that the state-space for this problem is 15-dimensional (or, effectively, 14-dimensional)!

3.3 Example — A Switched Power Controller

A naturally switched control problem is the design of a switched power controller for DC to DC conversion. The idea is to use a set of semiconductor switches to effectively change polarity of a voltage source. The controller has to decide which polarity to use each time slot (at a high

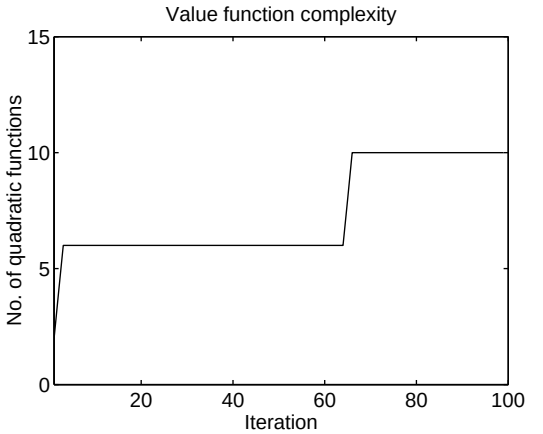


Figure 3. The complexity of the value function for the example of controlling two Furuta pendulums with $\alpha = 1.1$.

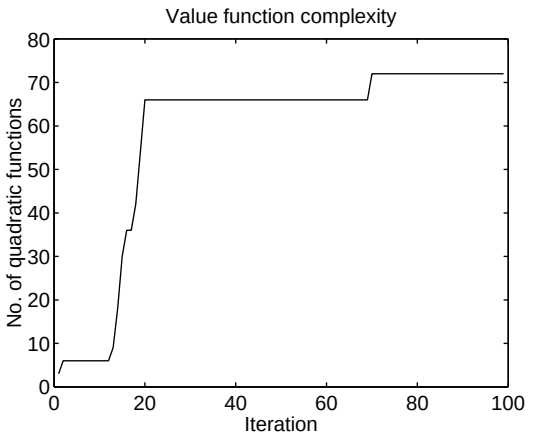


Figure 4. The complexity of the value function for the example of controlling three Furuta pendulums with $\alpha = 1.5$.

frequency) so that the load voltage and current are kept as constant as possible. See Figure 5 for the setup. This kind of DC/DC-converter is used in practice, often with a PWM control for the switch.

Modeling. Except for the switch, all components in the power system can be viewed as linear. For the purpose of control optimization, the load

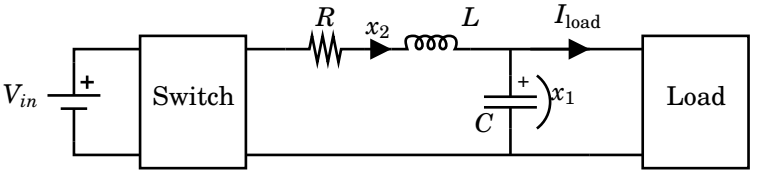


Figure 5. The setup for the switched DC/DC-converter.

is modeled as a constant current sink. The model becomes

$$\dot{x}_1 = \frac{1}{C}\Big(x_2 - I_{\text{load}}\Big), \tag{23}$$

$$\dot{x}_2 = -\frac{1}{L}x_1 - \frac{R}{L}x_2 + \frac{1}{L}s(t)V_{\text{in}}, \tag{24}$$

where $s(t)$ is the sign of the switch as set by the controller. To obtain integral action in the controller, a third state is added as the integral of the voltage error

$$\dot{x}_3 = V_{\text{ref}} - x_1. \tag{25}$$

Using the affine extension

$$x_e(n) = \begin{bmatrix} x(n) \\ 1 \end{bmatrix},$$

and a sample period of h seconds, the model can be described in discrete time as

$$x_e(n+1) = \Phi_i x_e(n),\ i \in \{1, 2\}, \tag{26}$$

where Φ_i is a 4x4 matrix due to the integral state and the affine extension of the state vector.

Cost Function. The objective of the controller is to keep the voltage x_1 as constant as possible around V_{ref}. To avoid constant errors, but also strong harmonics, a combination of average, current and derivative deviations are punished. This is done by using the step cost

$$l(x) = q_P(x_1 - V_{\text{ref}})^2 + q_I x_3^2 + q_D(x_2 - I_{\text{load}})^2. \tag{27}$$

With the extended state vector this can also be written on standard form

$$l(x) = x_e^T Q x_e. \tag{28}$$

The switching controller will never be able to bring the system to a steady state at the set-point. Therefore, the standard cost function will grow

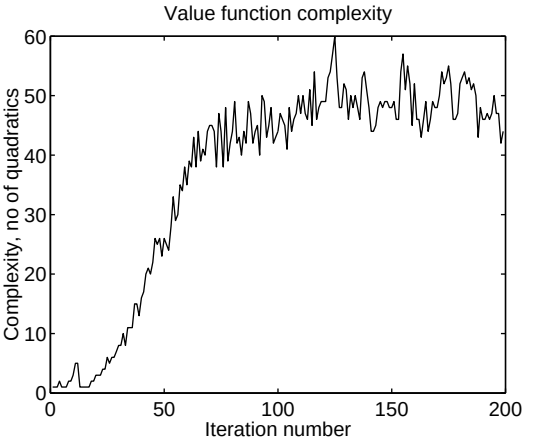


Figure 6. Complexity of the value function in the power controller example.

indefinitely. In this example, a "forgetting factor" $\lambda \leq 1$ is introduced in the cost function to cope with this problem:

$$J = \sum_{n=0}^{\infty} \lambda^n l(x(n)). \tag{29}$$

Finding a Controller. The above example is now on standard form, and the algorithm in this section can be used to find a controller. Generally, a forgetting factor λ simplifies the problem solution a lot, but also disqualifies $V(x)$ as a Lyapunov function. The example has been tried with λ in the range 0.95 to 1. For λ's less than 1, the value function complexity stabilizes on a reasonable level. We set the parameters to

$$\begin{array}{ccc} R = 1 & C = 4 & L = 0.1 \\ \lambda = 0.95 & \alpha = 2 & h = 0.1 \\ V_{\text{in}} = 1 & V_{\text{ref}} = 0.5 & I_{\text{load}} = 0.3 \\ q_P = 1 & q_I = 1 & q_D = 0.3 \end{array}$$

and again note that the I_{load} is only nominal and will change in the simulation later. The value iteration is done for 200 steps. For all steps, the complexity of the value function, i.e. number of quadratic functions, stays below 60, as can be seen in Figure 6.

The controller defined implicitly by $V_{200}(x)$ is used in the simulation shown in Figure 8. The explicit controller, evaluated each time step, is

$$s(n) = R\Big(\operatorname*{argmin}_{\Pi \in P_{200}} \Big\{ x(n)^T \Pi x(n) \Big\}\Big), \tag{30}$$

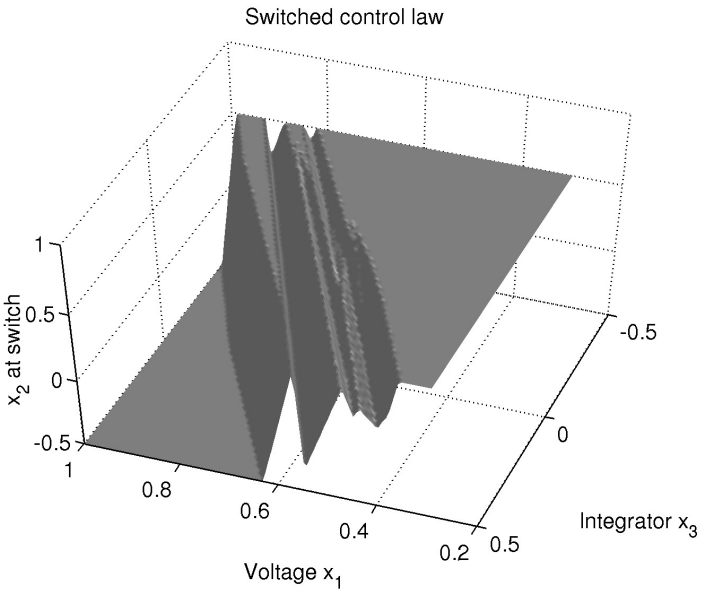


Figure 7. The resulting switching feedback law is monotonous in the current x_2 (by observation), and therefore it can be plotted in 3D. The plot shows at which current x_2 the switch from $s(n) = +1$ to $s(n) = -1$ takes place for varying voltages x_1 and integral states x_3. Note that the gridding is only for plotting purposes.

where $R(\Pi)$ is simply a table lookup mapping one specific member of P_{200} to a switch position ($+1$ or -1). Note that calculating this control action on-line is not very computationally intensive, compared to solving the off-line DP. A plot of the explicit control law is shown in Figure 7.

In the simulation in Figure 8, the load current I_{load} is changed at several points in time, but the controller keeps the voltage well thanks to the integral action.

4. Application: Piecewise Linear System

In this section a different version of the previous application is shown. Here, the switchings between different systems are not controlled by the controller, but are state-dependent. This may seem like a small change, but makes the problem quite different. The problem formulation is the following:

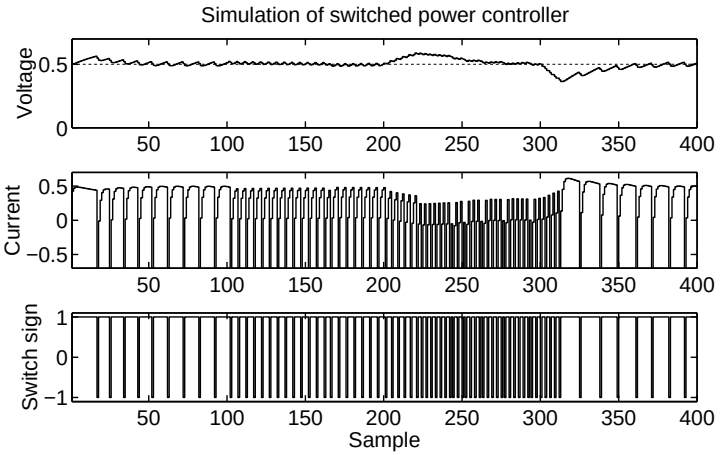


Figure 8. A simulation of the power system example with the obtained controller. At $n = 100$, the load current I_{load} is changed from its nominal 0.3A to 0.1A, at $n = 200$, to -0.2A and at $n = 300$ back to the nominal 0.3A.

Given a piecewise linear, discrete-time system, defined as

$$x(n+1) = f(x(n), u(n)) = \begin{cases} \Phi_1 x(n) + \Gamma_1 u(n), & \text{if } x(n) \in \Omega_1, \\ \Phi_2 x(n) + \Gamma_2 u(n), & \text{if } x(n) \in \Omega_2, \end{cases} \tag{31}$$

where $\Omega_1 = \{x | x^T G x \geq 0\}$ and $\Omega_2 = \{x | x^T G x < 0\}$. Throughout the paper, we will assume this special case of only two different linear dynamics for simplicity. In general, it is easy to extend to an arbitrary number of regions. Note that this system is linear, but can be generalized to affine by using an extended state vector

$$x_e(n) = \begin{bmatrix} x(n) \\ 1 \end{bmatrix}. \tag{32}$$

This is done in some of the examples.

The objective is to find a feedback controller $u = \mu(x)$ which minimizes the piecewise quadratic cost function

$$J = \sum_{n=0}^{\infty} l(x(n), u(n)) = \sum_{n=0}^{\infty} \left\{ \begin{bmatrix} x(n) \\ u(n) \end{bmatrix}^T Q(n) \begin{bmatrix} x(n) \\ u(n) \end{bmatrix} \right\}, \tag{33}$$

where

$$Q(n) = \begin{cases} Q_1, & \text{if } x(n) \in \Omega_1, \\ Q_2, & \text{if } x(n) \in \Omega_2. \end{cases} \tag{34}$$

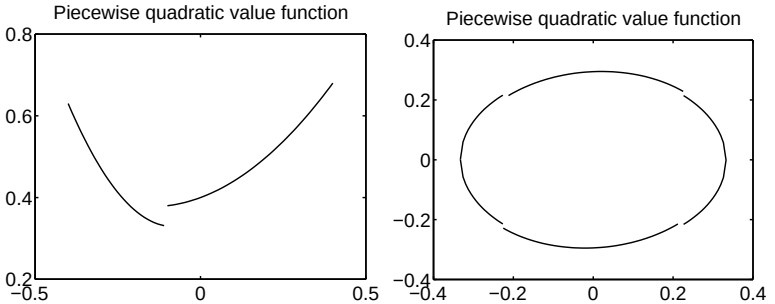


Figure 9. Examples of piecewise quadratic cost functions. The left plot shows a 1-D function, and the right plot shows a level set of a 2-D function. Note the discontinuities at the switching surfaces.

4.1 Finding a Relaxed Value Function

To get some intuition on the structure of the value function in the piecewise linear system, we start by a quadratic function $V_0(x) = x^T \Pi x$ and calculate the next iteration $V_1(x)$ exactly. Bellman's equation gives

$$V_1(x) = \begin{cases} \min_u \left(V_0(\Phi_1 x + \Gamma_1 u) + \begin{bmatrix} x \\ u \end{bmatrix}^T Q_1 \begin{bmatrix} x \\ u \end{bmatrix} \right) = x^T \Pi_1 x, & x \in \Omega_1, \\ \min_u \left(V_0(\Phi_2 x + \Gamma_2 u) + \begin{bmatrix} x \\ u \end{bmatrix}^T Q_2 \begin{bmatrix} x \\ u \end{bmatrix} \right) = x^T \Pi_2 x, & x \in \Omega_2. \end{cases} \tag{35}$$

This is not on the same form as $V_0(x)$, as it contains the condition $x \in \Omega_i$ (see Figure 9 for an illustration). Using exact value iteration, the number of regions may grow exponentially.

4.2 The Relaxed Value Function

We have two requirements for a suitable approximating value function parameterization:

1. The Bellman iteration (i.e. (10) and (11)) must be computationally feasible.
2. There must be a computationally feasible algorithm to find a *simple* $V_k(x)$ that satisfies (12).

For the rest of this section we will focus on one specific such value function parameterization, namely

$$V(x) = \min_{\Pi \in P} x^T \Pi x. \tag{36}$$

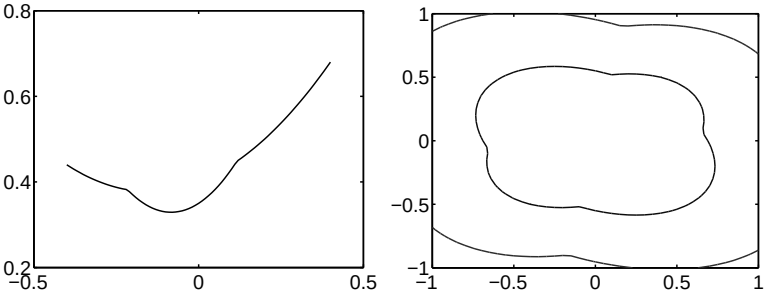


Figure 10. Examples of value functions $V(x) = \min_{\Pi \in P} x^T \Pi x$. The left plot shows a 1-D function, and the right plot shows the level sets of a 2-D function. In this example, $|P| = 2$.

See Figure 10 for an illustration of level sets of this $V(x)$. This function satisfies requirement 1 above as the solution to the Bellman iteration step is simply a two-region min-of-quadratics as will be shown in Section 4.3. It also satisfies requirement 2, which, again, will be shown in the Section 4.4. Note that this function *cannot*, in general, describe the exact value function.

In the following two sections, the two steps leading to a full value iteration are presented.

4.3 Upper and Lower Bounds

We assume that after iteration $k-1$,

$$V_{k-1}(x) = \min_{\Pi \in P_{k-1}} x^T \Pi x.$$

Using Bellman's equation the upper and lower bounds

$$\overline{V}_k(x) = \begin{cases} \min_{\Pi \in \overline{P}_k^1} x^T \Pi x, & x \in \Omega_1, \\ \min_{\Pi \in \overline{P}_k^2} x^T \Pi x, & x \in \Omega_2, \end{cases} \tag{37}$$

and

$$\underline{V}_k(x) = \begin{cases} \min_{\Pi \in \underline{P}_k^1} x^T \Pi x, & x \in \Omega_1, \\ \min_{\Pi \in \underline{P}_k^2} x^T \Pi x, & x \in \Omega_2, \end{cases} \tag{38}$$

are calculated similarly to (35).

4.4 Finding a Simple V_{k+1}

The final, and most important step in the iteration is to find a simple-to-represent $V_k(x)$ in between the upper and lower bounds. Doing this in an optimal way with respect to parameterization size is a very hard problem. In this section, an ad-hoc algorithm to find $V_k(x)$ is presented.

The algorithm is based on the following reasoning:

1. Since $V_k(x)$ is a minimum of quadratic functions, each additional quadratic function will decrease the value of $V_k(x)$.
2. The upper bound consists of several quadratic functions. If the upper bound holds for some of those quadratic functions, then due to point 1, this will not change when adding more functions to $V_k(x)$.
3. Due to points 1 and 2, it is possible to check one $\overline{\Pi} \in \overline{P}_k^j$ at a time, and if the upper bound inequality does not hold, add one quadratic function to $V_k(x)$ as far away from $\overline{\Pi}$ as the lower bound permits.

The following algorithm starts with $\overline{V_k}(x)$ and $\underline{V_k}(x)$ and will try to find a $V_k(x)$ such that (12) is satisfied.

PROCEDURE 3.3—RELAXED REPRESENTATION

1. Let $P_k = \emptyset$.
2. Define $V_k(x) = \min_{\Pi \in P_k} x^T \Pi x$.

   If there exists a $j \in \{1, 2\}$, a $\overline{\Pi} \in \overline{P}_k^j$, and $x \in \Omega_j$ such that $x^T \overline{\Pi} x < V_k(x)$ (i.e. the upper bound does not hold) then solve

$$\begin{aligned} \min_{\Pi,\gamma} \gamma \text{ s.t.} \\ x^T \Pi x &\le \gamma x^T \overline{\Pi} x \quad \forall x \in \Omega_j \\ x^T \Pi x &\ge x^T \underline{\Pi} x \quad \forall x \in \Omega_1,\ \underline{\Pi} \in \underline{P}_k^1 \\ x^T \Pi x &\ge x^T \underline{\Pi} x \quad \forall x \in \Omega_2,\ \underline{\Pi} \in \underline{P}_k^2 \end{aligned}$$

   for Π and γ. If $\gamma \le 1$, add this Π to P_k, otherwise the procedure fails (which is discussed later in this section).
3. Repeat from 2.

□

The optimization of Π (and γ) in Procedure 3.3 can be relaxed to the following LMI, using the S-procedure [Yakubovich, 1971]. Again, the sign

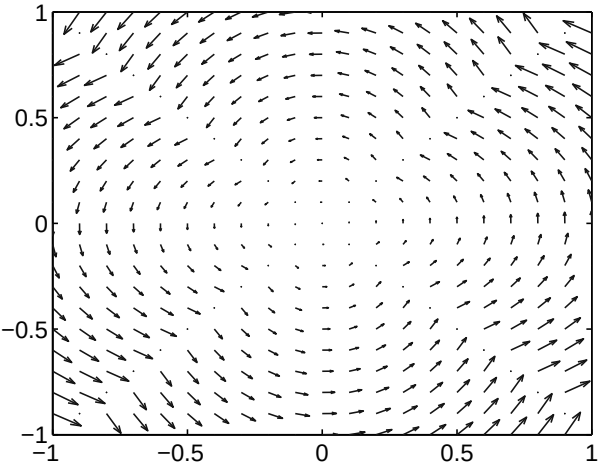


Figure 11. The phase plot of the "flower" example. The control input direction direction is x_1, i.e horizontally.

of $x^T Gx$ defines the regions Ω_1 and Ω_2.

$$\begin{aligned}\min_{\gamma,\Pi\geq 0,\lambda_i\geq 0} \gamma \text{ s.t.}\\ \Pi &\leq \gamma\overline{\Pi} \pm \lambda_1 G \ (*)\\ \Pi &\geq \underline{\Pi} + \lambda_2 G \quad \forall \underline{\Pi} \in \underline{P}_k^1\\ \Pi &\geq \underline{\Pi} - \lambda_3 G \quad \forall \underline{\Pi} \in \underline{P}_k^2,\end{aligned}$$

where the sign at $(*)$ depends on j, $-$ for $j = 1$ and $+$ for $j = 2$.

4.5 Examples

This section will show two optimal control examples where the method has been used to find close-to-optimal solutions. The algorithm has been implemented in Matlab with SeDuMi (see [Sturm, 1999]) to solve the LMIs.

Example 1: The Flower System. This example is a version of the "flower" example from [Johansson, 1999]. In this case, the piecewise linear, oscillative system has been extended with a control input. The name of the example stems from the phase plot, see Figure 11.

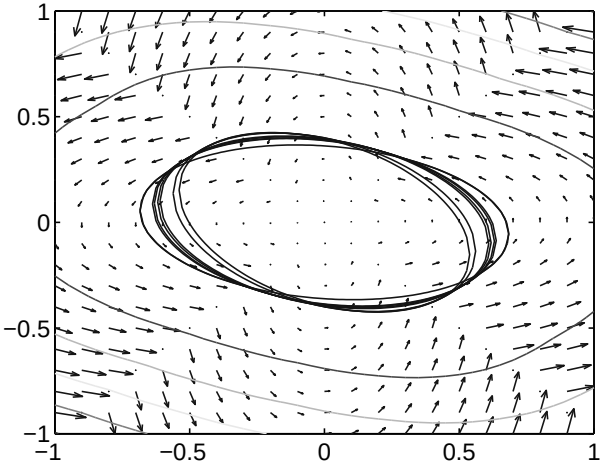


Figure 12. The level functions of the resulting $V(x)$ in the "flower" example, as well as level sets of the seven quadratic functions.

The system matrices are

$$\Phi_1 = \begin{bmatrix} 1 & -0.001 \\ 0.005 & 0.9999 \end{bmatrix}, \quad \Gamma_1 = \begin{bmatrix} 0.01 \\ 0 \end{bmatrix},$$
$$\Phi_2 = \begin{bmatrix} 1 & -0.005 \\ 0.001 & 0.9999 \end{bmatrix}, \quad \Gamma_2 = \begin{bmatrix} 0.01 \\ 0 \end{bmatrix},$$
$$Q_1 = Q_2 = I, \qquad G = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Using the relaxation $\overline{\alpha} = 2$ and $\underline{\alpha} = \frac{1}{2}$, a value function $V(x)$ satisfying (5) is found in less than 100 iterations. The resulting $V(x)$ consists of *seven* quadratic functions. See Figure 12 for a cost plot.

One interesting property of the min-of-quadratics value function parameterization is that the switching surfaces for the relaxed optimal control law are found implicitly. Thus, they do not have to coincide with the switching surfaces for the system dynamics nor be chosen in advance, as opposed to e.g. the method in [Johansson, 1999].

To illustrate the controller in action, some trajectories of the controlled system are shown in Figure 13.

Example 2: A Piecewise Affine System. As long as we stick to piecewise linear systems (as in the previous example), modeling of nonlinear behavior from the real world is quite limited. Extending to piecewise affine

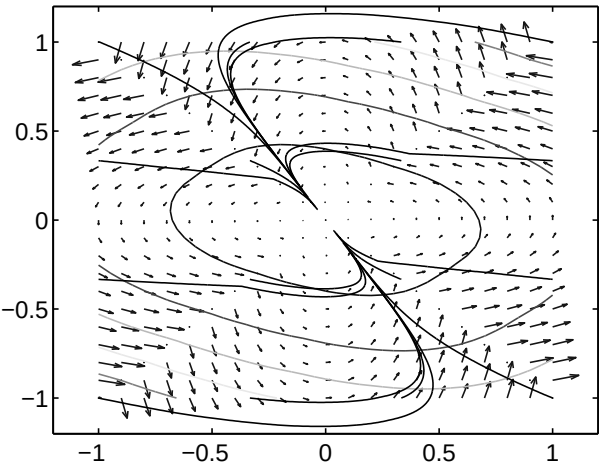


Figure 13. Some trajectories of the "flower" system with the obtained controller. Note that the arrows show the direction of the uncontrolled system.

systems, though, increases the chance of being able to model something useful. In this approach, affine terms are added by extending the state vector to

$$x_e = \begin{bmatrix} x \\ 1 \end{bmatrix}.$$

This enables us to use the same machinery at the cost of one extra state variable. As the problem now contains constant, linear, and quadratic terms, it is most often necessary to limit the state space where $V(x)$ is to be valid. Otherwise, at "large" states, the quadratic terms totally dominate the linear and constant terms, and the problem may be very different from what was intented. For example, when optimizing a control law for a linear system with bounded input, large initial states are meaningless as the bounded control signal can do almost nothing.

In this example the dynamics consists of stable oscillating dynamics in $x_1 < 0.2$ and unstable dynamics in $x_1 \geq 0.2$ (0.2 chosen only because it is not zero). Using the extended state vector above, the state space becomes

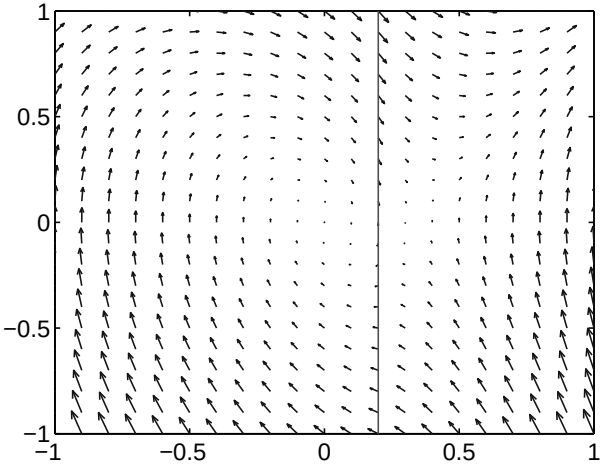


Figure 14. The phase plot of the affine example system. The control input is in x_1 (horizontal) direction.

3-dimensional. The system matrices are

$$A_1 = \begin{bmatrix} 0 & 0.1 & 0 \\ -0.2 & -0.1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} 0 & 0.1 & 0 \\ 0.3 & -0.1 & -0.1 \\ 0 & 0 & 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

$$Q_1 = Q_2 = \operatorname{diag}[\,1 \quad 1 \quad 0 \quad 1\,],$$

$$x_e^T G x_e = (0.2 + 3)^2 - (x_1 + 3)^2,$$

with Φ_i and Γ_i as discrete-time versions of A_1 and B_1 with sample period $h = 0.1$ s and a zero-order-hold. Note that the switched system is defined in discrete-time, and switches will thus occur outside of the region boundaries. A phase-plane plot is shown in Figure 14. As the region-dividing function $x_e^T G x_e$ is quadratic and thus symmmetric (in this case around -3), it is necessary to restrict the state-space for which $V(x)$ is valid. In this example state-space is restricted to the ball $\|x\| \leq 3$.

Using the proposed algorithm with $\overline{\alpha} = 2$ and $\underline{\alpha} = \frac{1}{2}$, the value function after 100 iterations, $V_{100}(x)$, consists of 9 quadratic functions. The level sets of $V_{100}(x)$ are shown in Figure 15.

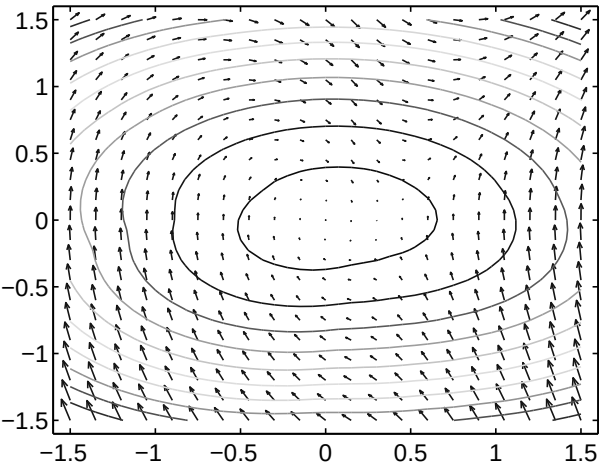


Figure 15. Level sets of $V_{100}(x)$ in the affine example.

By visual inspection, not much changes in the cost function after 100 iterations, but the formal stopping criterion (5) is not satisfied. As value-iteration can be interpreted as solving finite-time-horizon problems, $V_{100}(x)$ can be seen as the value function for a 100-time-step problem, and this might be enough for practical purposes.

To satisfy (5), we can either continue the iteration, or increase the slack until $V_{100}(x)$ does in fact satsify (5). For this example, the latter method requires $\overline{\alpha} = 2.8$ and $\underline{\alpha} = \frac{1}{2.8}$.

4.6 Problems and Discussion

The optimal control problem focused on in this section does not yet have a simple explicit solution. The approach taken here is to use an approximating value function and dynamic programming. This approach does simplify the problem a lot, but it is not without problems. In this section, some of the most apparent issues are discussed.

Algorithm Fails to Return a $V_k(x)$**.** There are two main problems that can make Procedure 3.3 fail; the first is due to the problem formulation itself, and the second is due to the S-procedure used in the LMI.

The first problem is that the value function is often discontinuous. This is due to the discrete-time nature of the problem. The $V_k(x)$ representation is always continuous, but thanks to the introduced slack it is still possible to satisfy the inequalities. If the slack is too small, though, the non-overlapping situation illustrated in Figure 17 may occur. This makes

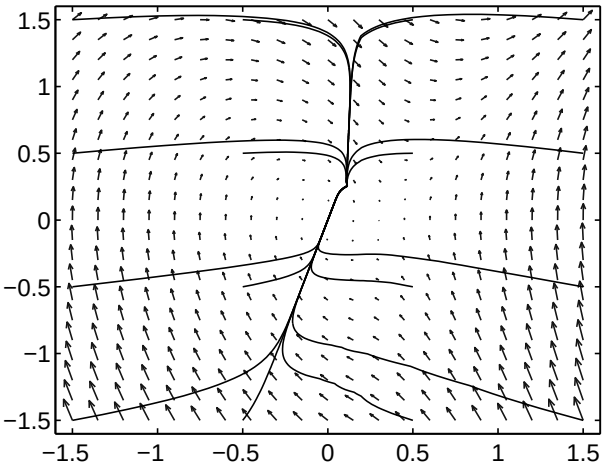


Figure 16. Trajectories of the controller obtained for the affine example. Note that the arrows show the uncontrolled system dynamics.

it impossible to find an approximating $V_k(x)$, and the only way out is to increase the slack.

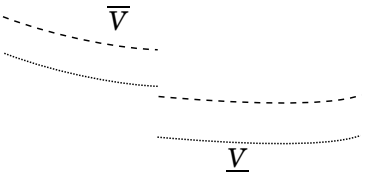


Figure 17. A case where an approximating value function is impossible to find as the upper and lower bounds are non-overlapping.

The second problem is the S-procedure that transfers the region-dependent LMI in the algorithm to a global LMI. This LMI may not have a solution with $\gamma \leq 1$ even if the original problem does.

In the Matlab implementation for the examples, the slack is automatically increased upon detection of any of the above problems.

Is Minimum of Quadratics a Good Choice? In Section 4.2 two properties for a good value function are listed, and the minimum-of-quadratics representation has these properties. There is one obvious question that we have not yet not asked: Is a minimim-of-quadratics function a good approximator of the kind of value functions generally obtained in optimal control of piecewise linear systems? Some insight is given in Figure 18.

In general, disregarding the desired properties from Section 4.2, the min-of-quadratics function is probably not the best approximator choice, but it works as a compromise.

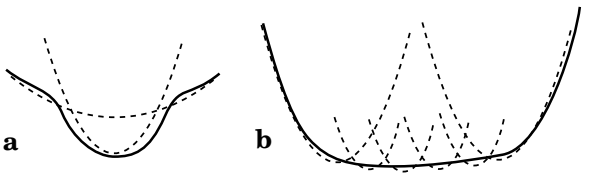


Figure 18. In **a**, the true function (solid line) is well represented by a small number of quadratic functions. In **b**, on the other hand, the true function requires a large number of quadratic functions to be approximated well.

5. Application: Linear System with Piecewise Linear Cost

This section describes yet another version of the optimal control problem. The plant to be controlled is an LTI system, and the cost to be minimized is piecewise linear. This makes it possible to punish states in more elaborate ways than the usual quadratic cost. For example, it is possible to make the cost asymmetric such that negative states are more costly than positive.

5.1 Problem Formulation

The controlled system is LTI

$$x(n+1) = \Phi x(n) + \Gamma u(n), \tag{39}$$

where $x \in X$, $u \in U$ and X and U are polyhedra. The cost function is on the form (2), with

$$l(x,u) = \max_{\pi \in Q} \pi^T \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}, \tag{40}$$

where π is a vector and Q is a finite set of vectors. $l(x,u)$ is thus piecewise linear, convex, and of "maximum-type" (see Section 2.3). The goal of the controller is to minimize the cost.

5.2 Value Function

Assume the value function $V_{k-1}(x)$ at some time $k-1$ is on the form

$$V_{k-1}(x) = \max_{\Pi \in P_{k-1}} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix}. \tag{41}$$

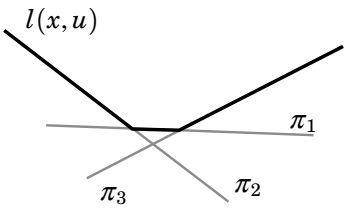


Figure 19. The piecewise linear cost function $l(x,u)$.

Bellman's equation is used to calculate $V_k(x)$

$$V_k(x) = \min_{u\in U}\left\{ \max_{\Pi\in P_{k-1}} \Pi^T \begin{bmatrix} \Phi x + \Gamma u \\ 1 \end{bmatrix} + \max_{\pi\in Q} \pi^T \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} \right\} = \min_{u\in U} \max_{r\in R}\Big\{ F_r^T x + G_r^T u + H_r \Big\}. \quad (42)$$

For the iteration to work, we need to rewrite this on the form (41). This can be done by writing $V_k(x)$ as the linear program

$$\begin{aligned} V_k(x) = \min_{u,f}\ & f \quad \text{s.t} \\ & F_r^T x + G_r^T u + H_r \le f \quad \forall r \in R, \end{aligned} \quad (43)$$

where the $x \in X$ and $u \in U$ constraints have been removed for clarity (they can easily be added). Solving this for any initial position x yields a value function that can again be written on the form (41). The solution can either be found by solving a *Multi-Parametric Linear Programming* (MPLP) problem (see e.g. [Borrelli *et al.*, 2002]), or by doing explicit enumeration of the dual extreme points as is shown below.

Introducing Lagrange multipliers λ, we rewrite (43) as

$$\begin{aligned} V_k(x) &= \min_{u,f}\max_{\lambda_i\ge 0}\Big\{ f + \sum_{r\in R} \lambda_k \big(F_r^T x + G_r^T u + H_r - f\big) \Big\} \\ &= \max_{\lambda_i\ge 0}\min_{u,f}\Big\{ \big(1 - \sum_{r\in R}\lambda_r\big) f + \sum_{r\in R}\lambda_r\big(F_r^T x + H_r\big) + \sum_{r\in R}\lambda_r G_r^T u \Big\}, \end{aligned} \quad (44)$$

where the last equality is due to strong duality for linear problems. From

this we see that

$$\sum_{r \in R} \lambda_r = 1, \tag{45}$$

$$\sum_{r \in R} \lambda_r G_r^T = 0, \tag{46}$$

and

$$V_k(x) = \max_{\lambda \in L} \sum_{r \in R} \lambda_r \left(F_r^T x + H_r \right), \tag{47}$$

where the set L is defined by $\lambda_r \geq 0$ and (45) and (46).

As $V_k(x)$ is linear in λ, the maximum can be found in one of the extreme points of L. L is a $|R| - 2$ dimensional set bounded by $|R|$ hyperplanes, and thus we can find at most $|R|(|R| - 2)/2$ extreme points. This can be done by selecting all pairs of $\left\{ (i,j) \in \mathbf{R}^2 \;\middle|\; i \neq j \right\}$, setting all other $\lambda_r = 0,\ r \neq i,\ r \neq j$. If λ_i and λ_j have positive solutions in (45) and (46), the resulting λ is an extreme point in L. Note that the extreme points do not depend on the state x.

The extreme points of L form the new set of vectors P_k, and again the value function is on the form

$$V_k(x) = \max_{\Pi \in P_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix}. \tag{48}$$

5.3 Parsimonious Representation

$V_k(x)$ can usually be represented by a much smaller set than the P_k obtained by the above extreme point enumeration. A set P_k is called a *parsimonious* representation, if only the members of P_k which are ever active (maximum for some state) are included. Such a set can be obtained from Procedure 3.4 (which is well known [Cassandra, 1998a], and, again, a special case of Procedure 3.1).

PROCEDURE 3.4—PARSIMONIOUS $V_k(x)$

1. Let $P_k^{\text{pars}} = \emptyset$.
2. Pick one $\Pi \in P_k$. Find an x where
$$\Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} > \pi^T \begin{bmatrix} x \\ 1 \end{bmatrix}, \ \forall \pi \in P_k^{\text{pars}}.$$
3. If such an x exists, find the $\pi \in P_k$ with the highest cost $\pi^T \begin{bmatrix} x \\ 1 \end{bmatrix}$.
Add π to P_k^{pars}.
Remove π from P_k.
4. If no such x exists, remove Π from P_k.
5. Repeat from 2 until P_k is empty.

□

Note that after this procedure, naturally

$$\max_{\Pi \in P_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} = \max_{\Pi \in P_k^{\text{pars}}} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} \quad \forall x(n). \tag{49}$$

5.4 Relaxed Value Function

To be able to use the proposed relaxed DP, we define the relaxed cost

$$\underline{l}(x,u) = l(x,u) = \alpha_1 l(x,u) - \alpha_2, \tag{50}$$

and

$$\overline{l}(x,u) = l(x,u), \tag{51}$$

where $\alpha_1 \leq 1$ and $\alpha_2 \geq 0$. Choosing $\alpha_1 < 1$ is of course only meaningful if $l(x,u) \geq 0, \forall x,u$. The procedure in Section 2 is used to find a relaxed $V_k(x)$. Assume $V_{k-1}(x)$ satisfies (9) for time step $k-1$. From this $V_{k-1}(x)$, Bellman's equation gives the upper and lower bounds

$$\overline{V}_k(x) = \max_{\Pi \in \overline{P}_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} = \min_u \Big\{ V_k\big(f(x,u)\big) + \overline{l}(x,u) \Big\}. \tag{52}$$

and

$$\underline{V}_k(x) = \max_{\Pi \in \underline{P}_k} \Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} = \min_u \Big\{ V_k\big(f(x,u)\big) + \underline{l}(x,u) \Big\}. \tag{53}$$

Now, finding a V_k in between $\underline{V}_k(x)$ and $\overline{V}_k(x)$ is done by adding one element from $\overline{P}_k$ at a time according to Procedure 3.5 (which is again a special case of Procedure 3.1, but for maximum-type).

PROCEDURE 3.5—RELAXED $V_k(x)$

1. Let $P_k = \emptyset$.
2. Pick one $\Pi \in \underline{P}_k$. Find a state x where $\Pi^T \begin{bmatrix} x \\ 1 \end{bmatrix} > \pi_i^T \begin{bmatrix} x \\ 1 \end{bmatrix}$, $\forall \pi \in P_k$.
3. If such an x exists, find the $\pi \in \overline{P}_k$ with the highest cost $\pi^T \begin{bmatrix} x \\ 1 \end{bmatrix}$. Add π to P_k.
4. If no such x exists, remove Π from $\underline{P}_k$.
5. Repeat from 2 until $\underline{P}_k$ is empty.

□

5.5 Example — Stock Order Control

To illustrate the capability to have non-symmetric cost functions, this section presents an example of a stock of some product. The control problem is to meet customer demand while not storing too many products nor running out of products when there is customer demand.

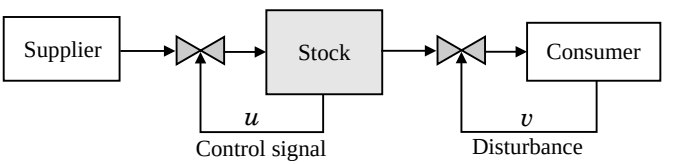


Figure 20. The stock controlling the orders u to the manufacturer and the consumer controlling orders v from the stock in the example. As seen from the stock, u is the control signal and v is a disturbance.

The system is modeled in discrete time, where the sample period h is one day. In one sample period, the stock controller can order from 0 to 0.5 units of the product (where one unit is a batch of products), and anything in between. The order control signal is denoted $u(n)$ at day k. It takes three days for the order to arrive at the stock.

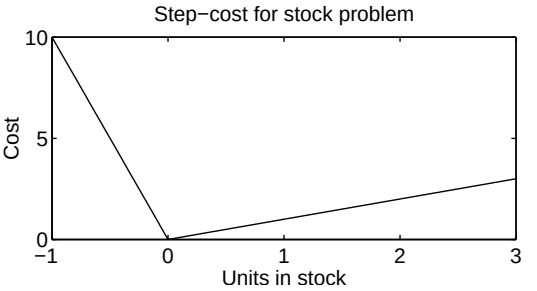


Figure 21. The step cost for the stock example. Negative stock (backlog) is more expensive than storing.

After the stock order has been placed, the consumers buy v units of the product from the stock (and the products are removed immediately, without delay). The amount $v(n)$ at day n is random and independent with the following probabilities:

$$v(n) = \begin{cases} 1 & \text{with probability } 0.1 \\ 0.3 & \text{with probability } 0.2 \\ 0 & \text{with probability } 0.7. \end{cases} \tag{54}$$

The cost of the system is a sum of the backlog cost when the stock is negative and the storing cost when the stock is positive. For each day with negative stock y, the cost is $l = -10y$. For each day with positive stock, the cost is $l = y$ (see Figure 21).

Problem formulation. This problem can be written as

$$\begin{aligned} x(n+1) &= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} x(n) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(n) - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} v(n), \\ y(n) &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x(n), \\ 0 &\le u(n) \le \frac{1}{2}. \end{aligned} \tag{55}$$

The step cost is

$$l(x,u) = \max(\begin{bmatrix} -10 & 0 & 0 \end{bmatrix} x, \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x), \tag{56}$$

and the objective is to minimize the cost

$$J = \lim_{N\to\infty} \sum_{n=0}^{N} \lambda^n l(x(n), u(n)), \tag{57}$$

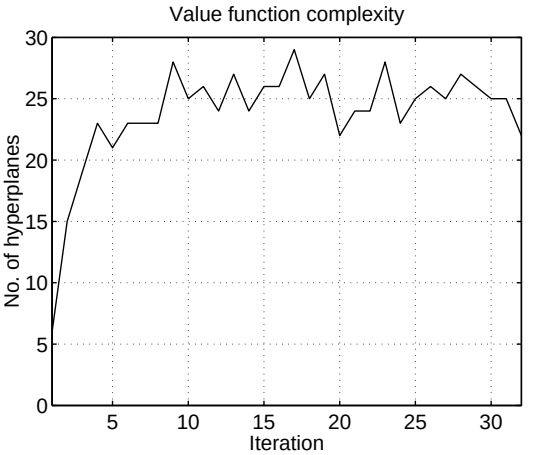


Figure 22. The complexity of the value function for the stock example over iterations.

where the "forgetting factor" $\lambda = 0.9$. This factor ensures a finite value function, and, loosely speaking, weighs future versus immediate costs.

Solution. The method in Section 5.4 is used to solve for a steady-state value function (and thus a control law). The relaxation parameters $\alpha_1 = 0.5$ and $\alpha_2 = 0.1$ turn out to give a good trade-off between solution complexity and accuracy for this problem. With these parameters, our solution is guaranteed to have

$$V_k^*(x) \geq V_k(x) \geq 0.5V_k^*(x) - \frac{0.1}{1-\lambda} = 0.5V_k^*(x) - 1 \tag{58}$$

for each iteration k. The value function has been iterated 32 times and the resulting complexity plot is shown in Figure 22. As can be seen, at this level of accuracy, the value function can be described by less than 30 hyperplanes for all iterations. Using $V(x) = V_{32}(x)$ as the infinite-time horizon solution, it can be shown to obey

$$V^*(x) \geq V(x) \geq 0.5V^*(x) - 1.51, \tag{59}$$

as $V(x)$ satisfies (5) when α_2 is increased to $\alpha_2 = 0.151$.

As the problem is three-dimensional, the resulting value function is somewhat hard to visualize. In Figure 23, the two delay-states have been set to zero, resulting in a one-dimensional value function in "units-in-stock" (x_1). Apparently, with no units in the delay pipeline, a stock of about 1 unit is good.

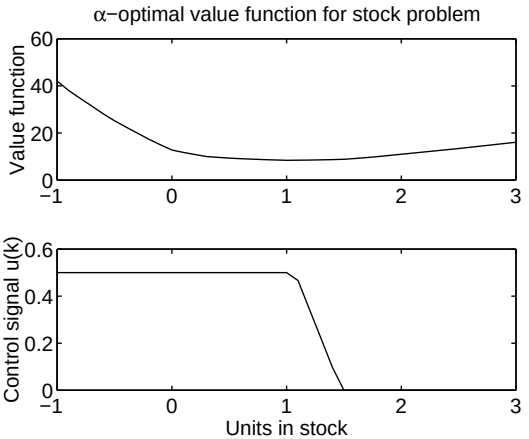


Figure 23. The relaxed value function and corresponding control law for the stock control problem. This curve corresponds to $x_2 = x_3 = 0$, i.e. there are no units in the "pipeline".

6. Application: POMDPs

Partially Observable Markov Decision Processes (POMDPs) have been around for a long time (see e.g. [Åström, 1965; Cassandra, 1998a]). Lately, it has mostly been investigated in the AI/robotics fields, e.g. robot navigation problems where limited sensor information is available. A POMDP is basically a Dual Control problem [Feldbaum, 1960 1961] where the state-space X is finite, as is the control signal (or action) space U and observation space Y. As for dynamic programming, the problem is very similar to the piecewise linear cost application in Section 5.

The state $x(n)$ is a Markov state and, and the dynamics is specified by transition matrix $\pi(u)$, where element m, l denotes the probability to move to state m if the system is currently in state l. This probability can be controlled by u. See Figure 24 for an illustration.

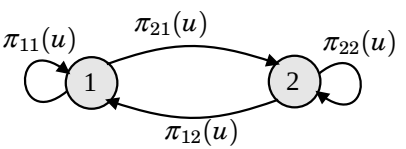


Figure 24. An example POMDP Markov graph with a set of transition matrices $\pi(u)$, one for each control action u.

A specific observation $y(n) \in Y$ will be obtained with probability

$$P\big(y(n) \mid x(n) = m\big) = \Omega_m(y(n), u(n)), \tag{60}$$

where Ω is the observation probability vector (note that it may depend on the action $u(n)$). Thus, the controller never really knows exactly in which state the process is (if it would, the problem would be a Markov Decision Process and easily solved). To be able to use DP, the state is changed to the belief state $z(n) : \; z_m(n) \; = \; P\big(x(n) \; = \; m\big)$. Note that state space is closed as $z_m(n) \geq 0, \; \forall m$ and $\sum_m z_m(n) = 1$. The dynamics of the belief state is linear,

$$z(n+1) = \pi(u) z(n). \tag{61}$$

and for each observation $y \in Y$, our belief state is changed according to Bayes' rule:

$$\begin{aligned} z_m(n+1 \mid y(n)) = P\big(x(n) = m \mid y(n)\big) = \\ \frac{P\big(x(n) = m \cap y(n)\big)}{P\big(y(n)\big)} = \\ \frac{P\big(y(n) \mid x(n) = m\big) \cdot P\big(x(n) = m\big)}{P\big(y(n)\big)} = \\ \frac{\Omega_m(y(n), u(n)) \cdot z_m(n)}{P\big(y(n)\big)}. \end{aligned} \tag{62}$$

The expected state over all possible observations is then

$$\mathop{\mathbf{E}}_{y(n) \in Y} \Big\{ z(n+1) \Big\} = \sum_{y(n)} P(y(n)) \frac{\Omega(y(n), u(n)) z(n)}{P\big(y(n)\big)} = \sum_{y(n)} \Omega(y(n), u(n)) z(n). \tag{63}$$

Thus, the dynamics of the belief state z is linear.

The cost in a POMDP problem is usually replaced by a reward, so we will stick to that. The reward J is defined as

$$J = \mathbf{E} \sum_{n=0}^{\infty} \lambda^n l(x(n), u(n)) = \sum_{n=0}^{\infty} \lambda^n R(u(n))^T z(n), \tag{64}$$

where $R(u(n))$ is a vector of rewards of using control signal $u(n)$ for each Markov state $x(n)$, and $\lambda \leq 1$.

For each time step, the controller has to make a control decision $u(n)$ based on the current belief state $z(n)$. After the control decision, an observation $y(n)$ based on $z(n)$ and $u(n)$ is obtained. We would like to find

an optimal control policy $u = \mu(z)$ which maximizes the reward J for any initial state. As it turns out, the value function $V_k(z)$ is again of the form

$$V_k(z) = \max_{\Pi \in P_k} \Pi^T z, \tag{65}$$

where P_k is a finite index set and Π_k is a vector. Thus the value function is piecewise linear in the state z.

If the value function $V_{k-1}(z)$ is known and in the form (65), we can calculate the value $V_k(z)$ from Bellman's equation:

$$\begin{aligned} V_k(z) = \max_u \mathbf{E}\Big\{ V_{k-1}\big(\pi(u)z\big) + l\big(x,u\big) \Big\} = \\ \max_u \mathbf{E}\Big\{ \max_{\Pi \in P_{k-1}} \big(\Pi^T \pi(u) + R(u)^T\big) z(k \mid y(n)) \Big\} = \\ \max_u \sum_{y \in Y} \max_{\Pi \in P_{k-1}} \big(\Pi^T \pi(u) + R(u)^T\big) \Omega(y,u) z = \max_{\Pi \in P_k} \Pi^T z(n). \end{aligned} \tag{66}$$

Note that the "raw" size of P_k is significantly larger than P_{k-1} (actually $|P_k| = |P_{k-1}|^{|Y|}|U|$, where $|U|$ denotes the number of elements in U).

6.1 Parsimonious Representation

Just like for the control problem problem in Section 5.3, the set P_k is often unnecessarily large and may be pruned without changing the value of $V_k(x)$. Procedure 3.4 can be used to obtain a parsimonious representation.

6.2 Relaxed Value Function

Analogous to Section 5.4, a modified pruning procedure can be used to obtain an α-optimal value function. The relaxed step cost is the same as in (50) and (51).

6.3 Example

There is a wide variety of reference POMDP problems defined in literature. In this section we focus on the 4x3 maze problem found in [Cassandra, 1998b], which is a modified version from [Russell and Norvig, 1994]; see Figure 25. The state x is a position in a square 4x3 maze where one state is inaccessible, and therefore the state space X has size 11 (z is 11-dimensional). Y consists of six observations, and there are four actions in U. The immediate reward is

$$l(x) = \begin{cases} +1 & \text{if } x = \text{good} \\ -1 & \text{if } x = \text{bad} \\ -0.04 & \text{otherwise.} \end{cases} \tag{67}$$

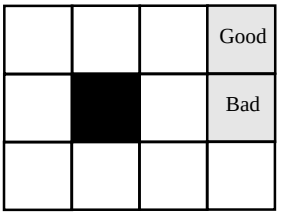


Figure 25. The 4x3 maze with one good and one bad state. The black state is inaccessible, and the initial position is randomly selected from the 9 neutral states.

After reaching the "good" or "bad" state, the state is reset to a random initial position. The problem is solved over an infinite horizon using value iteration with a discount factor $\lambda = 0.95$.

Running POMDP-SOLVE from [Cassandra, 1998b] with incremental pruning and searching for the *optimal solution* fails to return within a reasonable time as the set P grows too fast (after 10 iterations and 36 CPU-minutes on a fast PC the set size $|P_{10}|$ is 3393).

Setting $\alpha_1 = 1$ and $\alpha_2 = 0.01$, the algorithm keeps a value function V of complexity (set size) of about 150 after reaching steady state. The algorithm was run with a finite horizon of 50 time steps, and the resulting average value (for random initial states) is ≈ 1.7. Using our α and the discount factor λ, we can bound the optimal value $V^*(x)$ by

$$V(x) \leq V^*(x) \leq V(x) + \sum_{i=0}^{\infty} \lambda^i \alpha = V(x) + 0.2. \tag{68}$$

A smaller α_2 produces a larger search and a tighter bound, and vice versa.

7. Application: Network Routing Tables

In this final example, a different view on DP is shown. Here, the iteration is done over links in a network instead of over time, and it is also decentralized with one value function at each node in the network.

In a huge computer network like the Internet, there are hundreds of millions of computers with unique addresses. We assume that every computer want to be able to send data to any other computer on the network. Doing optimal routing (using some cost) forces each computer to keep a vector of N entries, where N is the number of computers, with the cost to every other computer on the net. This is of course not realistic,

and maybe also not desirable – routing does not have to be optimal, just good enough.

This example applies the relaxed dynamic programming method to decrease the number of entries in the cost vector. As usual in path-finding, this can be done distributively so that each computer iteratively asks its neighbors about their knowledge of costs to other computers.

The idea is to use the fact that in practice, network addresses (or at least IP addresses) are often semi-hierarchically selected. Two neighboring nodes x and y have addresses that with large probability only differ in the least significant bits. A node z, far away from both x and y may be able to treat x and y as having the same cost, and represent this in the vector as the common bits in x's and y's addresses.

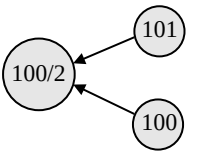


Figure 26. Two nodes, with binary addresses 101 and 100 may be treated as one node, 100/2 (indicating that the two most significant bits are used) if the costs are close enough.

7.1 Value Function

The value function $V(x)$ for a certain node corresponds to the cost to send data to another node x in the network. The relaxed DP is very dependent on having a suitable value function representation. In this case, we would like to be able to group sets of addresses together, but still have the ability to break out some addresses in a range and give them special values (as the network addresses are not completely hierarchical).

Without going into details, in this example a value function representation is used where both the address and a the number of significant bits are stored in each entry. This way, a range of addresses can be represented by one entry with less significant bits. To be able to handle "outliers", it is also possible to give some specific address ranges within a larger range of addresses special values. For example, if the cost to go to nodes 1–12, 14–16 is approximately the same then this can be represented as the range 1–16 with a special value for address 13.

7.2 Value Iteration

In this network example, the value function can be viewed as being iterated over *links*, instead of *time*. The step cost $l(x)$ is some cost between

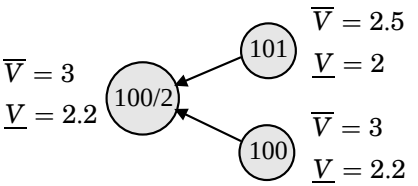


Figure 27. The two nodes, 101 and 100 have overlapping cost intervals. The nodes are therefore replaced by one node with the common interval as cost.

two links, calculated from e.g. delays, bandwidth, or some other Quality-of-Service measure. The step cost will be relaxed to

$$\overline{l}(x) = \overline{\alpha} l(x) \tag{69}$$

for $\overline{\alpha} \geq 1$ and

$$\underline{l}(x) = \underline{\alpha} l(x) \tag{70}$$

for $\underline{\alpha} \leq 1$. In this application, the iteration will be done such that both a lower bound $\underline{V}$ and an upper bound $\overline{V}$ are kept.

Let $V_k(x)$ be the value function at iteration k for a certain node in the network. Let node have neighboring nodes $n \in N$, and let the cost to send data to n be $C(n)$. $V_k^n(x)$ is the value function of neighbor n at this iteration. The aim for the relaxed DP is to find $V_{k+1}(x)$ satisfying

$$\min_{n \in N} \left\{ \underline{V}_k^n(x) + \underline{\alpha} C(n) \right\} \leq V_{k+1}(x) \leq \min_{n \in N} \left\{ \overline{V}_k^n(x) + \overline{\alpha} C(n) \right\}. \tag{71}$$

Note that node itself is a member of N as well, with zero cost added.

The algorithm to find the simplified $V_k(x)$ is conceptually similar to what has been presented in the preceding sections. The idea is, as mentioned before, to merge entries in $V_{k+1}(x)$ which are neighbors in address space *and* have overlapping upper and lower bounds on the cost. See Figure 27.

7.3 Example

To test the above procedure, random networks of various sizes have been produced. The results presented in this section do of course depend a lot on the properties of the network (and in particular network address) generator.

The Network Generator. The relaxed shortest-path algorithm described above does, as mentioned before, depend on a network with semi-hierarchical address space. This means that two neighboring nodes should

have addresses that are "close" in address space with high probability. The network graph should also be tree-like, but with some cycles to make it more interesting and realistic.

The network generator used in this example will not be described in full detail, but here is an outline:

PROCEDURE 3.6—NETWORK GENERATION FOR M NODES
Generate a first node with a random address between zero and M. Let the graph G consist of only this node. Then loop the following until N nodes are in the graph G:

1. Draw number-of-neighbors d from the probabilities 90%, 8% and 2% for 1, 2 and 3 neighbors, respectively.
2. Draw one node n from G, as a first neighbor candidate. Create a new address from n's address by altering the least l significant bits, where l is chosen randomly with a larger probability for lower numbers. If this address is not used in the network, keep n as the first neighbor. Otherwise, do step 2 again.
3. Draw the remaining neighbors and add the new node to the graph.

□

The graph designed by the algorithm does not have a simple 2D-structure, and is thus hard to visualize.

Dynamic Programming. The relaxed cost optimization algorithm is, just like the optimal algorithm, decentralized. In this simulation, the optimization has been performed by simulating all M nodes simultaneously, exchanging information at certain time ticks. If D is the largest distance (in number of links) between two nodes in the network, at most D iterations are required for the value function to converge.

To fill up the address space completely, network sizes of $M = 2^m$ have been generated, for M up to 2048 nodes. The output data from the optimization is the total number of entries E_k in all routing tables of all nodes at iteration k. A smaller number means that the average node keeps a smaller number of nodes in its routing table.

Using $\overline{\alpha} = 1.5$ and $\underline{\alpha} = \frac{1}{1.5}$, a typical E_k graph for 40 iterations of a $N = 1024$ network is shown in Figure 28. For optimal routing a total number of $N^2 \approx 10^6$ entries would be needed in the routing tables, and thus about 7-8 times fewer entries are needed with this relaxation. This generally holds for this relaxation and network generator.

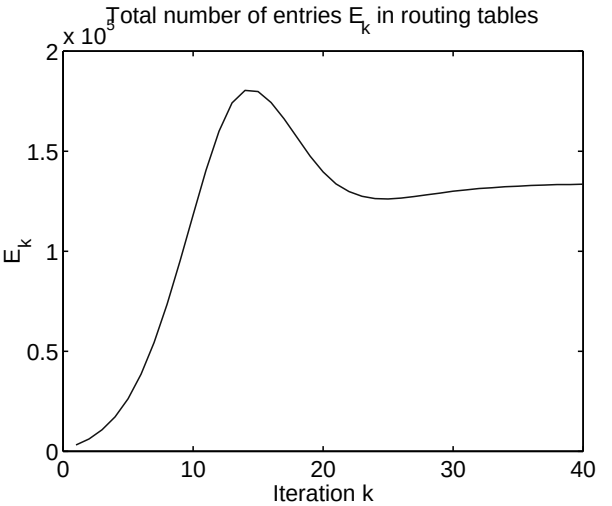


Figure 28. The total number of entries in all routing tables for a $N = 1024$ network, over iterations from 1 to 40. To do optimal control, a total number of $N^2 \approx 10^6$ entries would be needed in the routing tables.

The maximum distance of this network is 31, so the value function $V_k(x)$ satisfies (5) for all $k \geq 31$. Note that since this is an inequality, the value function may still change, but it will stay within the limits.

8. Value Function Convergence

It is well known that under mild conditions, $V_k^*(x)$ defined by

$$V_k^*(x) = \min_u \sum_{n=0}^{k} l(x(n), u(n))$$

converges to $V^*(x)$ as $k \to \infty$ in the standard value iteration

$$V_{k+1}^*(x) = \min_u \Big\{ V_k^*(f(x,u)) + l(x,u) \Big\}. \tag{72}$$

This is stated in Theorem 1 on the next page, for completeness. In Section 2 it was shown that the relaxed value iteration guarantees a value function $V_k(x)$ in the range

$$\min_u \sum_{n=0}^{k} \underline{l}(x(n), u(n)) \leq V_k(x) \leq \min_u \sum_{n=0}^{k} \overline{l}(x(n), u(n)) \tag{73}$$

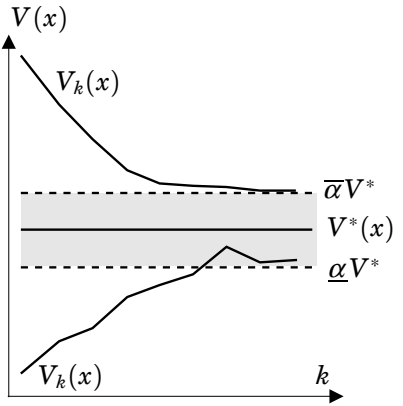


Figure 29. An illustration of two different value functions $V_k(x)$ converging to the close-to-optimal-set.

for iteration k. If, for example, $\overline{l}$ and $\underline{l}$ are defined by $\underline{\alpha}$ and $\overline{\alpha}$ as in (7)–(8), then

$$\underline{\alpha}V_k^*(x) \leq V_k(x) \leq \overline{\alpha}V_k^*(x). \tag{74}$$

Thus, the convergence of $V_k^*(x)$ implies convergence of $V_k(x)$ to the desired region close to $V^*(x)$ as well (see Figure 29).

THEOREM 1—VALUE ITERATION CONVERGENCE
Assume that there exists a constant $\gamma < \infty$ such that

$$V^*(f(x,u)) \leq \gamma l(x,u) \quad x \in X, u \in U. \tag{75}$$

Define $\beta = 1 + 1/\gamma$. If there exists an index p and a constant P such that that

$$\frac{V^*(x)}{V_p^*(x)} - 1 \leq P < \infty \quad \text{and} \quad V_0^*(x) \leq V^*(x)$$

then

$$\left| \frac{V^*(x)}{V_k^*(x)} - 1 \right| \leq \frac{P}{\beta^{k-p}} \quad k \geq p. \tag{76}$$

□

Remark 1 The constant γ gives a measure on how "stable" the optimally controlled system is, i.e. how close the optimal cost is to the step cost. The smaller γ is, the faster convergence.

PROOF OF THEOREM 1
First, we note that since $V_0^*(x) \leq V^*(x)$, then

$$V_k^*(x) \leq V^*(x) \quad \forall k \geq 0 \tag{77}$$

by induction. Then, for any x, let u_k, and u^* be (implicitly) defined by

$$V_k^*(x) = \min_u \Big\{ V_{k-1}^*(f(x,u)) + l(x,u) \Big\} = V_{k-1}^*(f(x,u_k)) + l(x,u_k)$$
$$V^*(x) = \min_u \Big\{ V^*(f(x,u)) + l(x,u) \Big\} = V^*(f(x,u^*)) + l(x,u^*).$$

Then

$$\begin{aligned} V^*(x) - V_k^*(x) &\leq V^*(f(x,u^*)) + l(x,u^*) - V_{k-1}^*(f(x,u_k)) - l(x,u_k) \\ &\leq V^*(f(x,u_k)) + l(x,u_k) - V_{k-1}^*(f(x,u_k)) - l(x,u_k) \\ &= V^*(f(x,u_k)) - V_{k-1}^*(f(x,u_k)). \end{aligned} \tag{78}$$

Thanks to (75), (77), it holds that

$$V_{k-1}^*(f(x,u_k)) \leq V^*(f(x,u_k)) \leq \gamma l(x,u_k),$$

and thus

$$\begin{aligned} V_k^*(x) = V_{k-1}^*(f(x,u_k)) + l(x,u_k) &\geq V_{k-1}^*(f(x,u_k)) + \frac{1}{\gamma} V_{k-1}^*(f(x,u_k)) \\ &= \underbrace{(1 + \frac{1}{\gamma})}_{\beta} V_{k-1}^*(f(x,u_k)). \end{aligned}$$

Both sides of (78) can be divided by $V_k^*(x)$ and we obtain

$$\begin{aligned} \frac{V^*(x) - V_k^*(x)}{V_k^*(x)} &\leq \frac{V^*(f(x,u_k)) - V_{k-1}^*(f(x,u_k))}{V_k^*(x)} \\ &\leq \frac{V^*(f(x,u_k)) - V_{k-1}^*(f(x,u_k))}{\beta V_{k-1}^*(f(x,u_k))} \Rightarrow \\ \left(\frac{V^*(x)}{V_k^*(x)} - 1 \right) &\leq \frac{1}{\beta} \left(\frac{V^*(f(x,u_k))}{V_{k-1}^*(f(x,u_k))} - 1 \right). \end{aligned} \tag{79}$$

Finally, denote

$$f^n(x,u) = f(f(\cdots f(x,u_k), u_{k-1}) \cdots, u_{k-n+1}),$$

where

$$u_j(x) = \underset{u}{\operatorname{argmin}} \Big\{ V_j^*(f(x,u)) + l(x,u) \Big\}.$$

It then follows directly that

$$\left(\frac{V^*(x)}{V_k^*(x)} - 1 \right) \leq \frac{1}{\beta} \left(\frac{V^*(f(x,u))}{V_{k-1}^*(f(x,u))} - 1 \right)$$
$$\leq \frac{1}{\beta^{k-p}} \left(\frac{V^*(f^{k-p}(x,u))}{V_p^*(f^{k-p}(x,u))} - 1 \right) \leq \frac{P}{\beta^{k-p}}.$$

□

9. Conclusions

A novel method for reduction of the computational complexity in dynamic programming has been presented. The method allows the user to specify an error bound on the solution, enabling a trade-off between complexity and accuracy. Therefore, many problems which may be very hard to solve optimally can be effectively solved.

Applications to several well known classes of optimal control problems show that the method has a potential for significant improvement compared to other approaches. Most likely, the same is true for many other application areas which still remain to be investigated.

References

Åström, K. J. (1965): "Optimal control of Markov processes with incomplete state information I." *Journal of Mathematical Analysis and Applications*, **10**, pp. 174–205.

Bellman, R. E. (1957): *Dynamic Programming*. Princeton Univ. Press.

Bertsekas, D. P. (2000): *Dynamic Programming and Optimal Control*, 2nd edition. Athena Scientific.

Bertsekas, D. P. and J. Tsitsiklis (1996): *Neuro-Dynamic Programming*. Athena Scientific.

Borrelli, F., A. Bemporad, , and M. Morari (2002): "A geometric algorithm for multi-parametric linear programming." *Journal of Optimization Theory and Applications*. To appear.

Cassandra, A. R. (1998a): *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Department of Computer Science, Providence, RI.

Cassandra, A. R. (1998b): "Tony's POMDP page." `http://www.cs.brown.edu/research/ai/pomdp/`.

de Farias, D. and B. V. Roy (2002): "Approximate dynamic programming via linear programming." *Advances in Neural Information Processing Systems 14, MIT Press*.

Feldbaum, A. (1960–1961): "Dual control theory. I-IV." *Automation Remote Control*, **21, 22**, pp. 874–880, 1033–1039,1–12,109–121.

Henzinger, T., H. Pei-Hsin, and H. Wong-Toi (1995): "HYTECH: the next generation." In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pp. 55–65. IEEE Comput. Soc. Press.

Johansson, M. (1999): *Piecewise Linear Control Systems*. PhD thesis ISRN LUTFD2/TFRT--1052--SE, Department of Automatic Control, Lund Institute of Technology, Sweden.

Roy, B. V. (2001): *Handbook of Markov Decision Processes: Methods and Applications*, chapter Neuro-Dynamic Programming: Overview and Recent Trends. Kluwer.

Russell, S. J. and P. Norvig (1994): *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Eaglewood Cliffs, NJ.

Schweitzer, P. J. and A. Seidmann (1985): "Generalized polynomial approximations in markovian decision processes." *Journal of Mathematical Analysis and Applications*, **110:2**, pp. 568–582.

Sturm, J. F. (1999): "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones." *Optimization Methods and Software*, **11–12**, pp. 625–653.

Yakubovich, V. (1971): "S-procedure in nonlinear control theory." *Vestnik Leningrad University*, pp. 62–77. (English translation in Vestnik Leningrad Univ. 4:73-93, 1977).

Paper III

Simple Stability Criteria for Systems with Time-Varying Delays

Chung-Yao Kao[1] **and Bo Lincoln**

Abstract

This paper considers the problem of stability of linear feedback systems with time-varying but bounded delays. Simple but powerful criteria of stability, as well as a simple performance bound, are presented for both continuous-time and discrete-time systems. Using these criteria, stability can be checked in a Bode plot of the closed loop system. This makes it easy to design the system for robustness.

This paper is based on "Simple Stability Criteria for Systems with Time-Varying Delays" by the same authors, submitted to *Automatica*, 2003.

[1]Authors in alphabetic order

1. Introduction

This paper presents a simple stability criterion for linear systems with time-varying but bounded delays.

Varying time delays (sometimes termed jitter) is a common problem in real-time implementations of control systems. Often several tasks share the same resource, such as a CPU or a network, and some kind of scheduling is needed. The scheduling can be of static and periodic type (see e.g. [Kopetz, 2002]), such that delays are constant and known. This simplifies control synthesis and analysis, but can give very bad resource utilization and inflexibility. Dynamic scheduling on the other hand is flexible and gives efficient resource utilization. It can lead to varying and unknown time delays, though, which may be bad for a real-time controller in a closed loop (see e.g. [Cervin *et al.*, 2003]). Using computer networks to send control data can also give rise to varying delays. For example, Ethernet's Media Access Control introduces explicit randomness after a packet collision on the network. This paper presents some simple graphical criteria for stability under such time-varying delays, which can be used to analyze time delay robustness.

Time-delay robustness is a large research topic, and many sub-problems have been extensively explored. One such problem is "stability independent of delay", where the system stability is tested for delays of arbitrary length; see [Chen and Latchman, 1995; Huang and Zhou, 2000; Verriest *et al.*, 1993]. Another problem is "delay-dependent stability" (implying restrictions on the delays), which has been explored in [Gu and Han, 2000; Verriest, 1994; Yan, 2001] among others. Most of these references study the problem of unknown but constant delays. In this paper, which is based on the results in [Kao and Rantzer, 2003; Lincoln, 2002], the delays are assumed to be bounded but otherwise freely time-varying. The main advantage of the method is that the stability criterion is a simple graphical check in a closed loop Bode plot, which makes design for robustness easy.

The paper is organized as follows: in Section 2 the main theorems are presented, followed by proofs in Section 3. Section 4 contains some extensions of the main theorems, and Section 5 contains a performance degradation bound. Finally, in Section 6 an example is shown to illustrate the theory.

Notation

We use $\mathbf{L}_2$ and $\mathbf{l}_2$ to denote spaces of square summable functions and sequences, respectively.

2. Main Result

Consider the systems in Figures 1 and 2. P is a plant and C is a controller, and the control system is perturbed by the delay Δ. The delay can be placed anywhere in the loop, but for the rest of the paper we will assume it to be after the controller as shown in the figures. We assume that the closed loop system of P and C is stable for zero delay. The following theorems give simple criteria of stability for the system with an arbitrarily time-varying but bounded delay. The proofs of the theorems are presented in Section 3.

THEOREM 1—CONTINUOUS TIME CASE
For the closed loop system in Figure 1 with continuous-time $P(s)$ and $C(s)$, the system is stable for any time-varying delays defined by

$$\Delta(v) = v(t - \delta(t)), \quad 0 \le \delta(t) \le \delta_{\max}, \tag{1}$$

if

$$\left| \frac{P(j\omega)C(j\omega)}{1 + P(j\omega)C(j\omega)} \right| < \frac{1}{\delta_{\max}\omega} \quad \forall \omega \in [0, \infty]. \tag{2}$$

□

THEOREM 2—DISCRETE TIME CASE
For the closed loop system in Figure 1, with discrete-time $P(z)$ and $C(z)$, the system is stable for any time-varying delays defined by

$$\Delta(v) = v(n - k(n)), \quad k(n) \in \{0, 1, \cdots, N\}, \tag{3}$$

if

$$\left| \frac{P(e^{j\omega})C(e^{j\omega})}{1 + P(e^{j\omega})C(e^{j\omega})} \right| < \frac{1}{N\left|e^{j\omega} - 1\right|} \quad \forall \omega \in [0, \pi]. \tag{4}$$

□

THEOREM 3—MIXED CT-DT CASE
For the closed loop system in Figure 2, with continuous-time, strictly proper and stable $P(s)$, and discrete-time $C(z)$ sampled with a period of h seconds, the system is stable for any time-varying delays defined by

$$\Delta(v) = v(t - \delta(t)), \quad 0 \le \delta(t) \le Nh \tag{5}$$

for some integer N if

$$\left| \frac{P_{\text{alias}}(\omega)C(e^{j\omega})}{1 + P_{\text{ZOH}}(e^{j\omega})C(e^{j\omega})} \right| < \frac{1}{N\left|e^{j\omega} - 1\right|} \quad \forall \omega \in [0, \pi]. \tag{6}$$

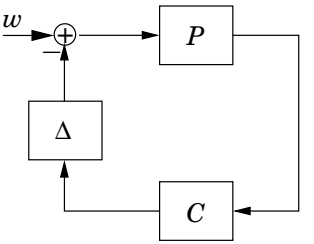


Figure 1. Linear time invariant system with an uncertain time-varying time delay in the feedback loop. P represents the plant to be controlled, C represents the controller, and Δ represents the time-delay operator. w is the input to the system (a disturbance or reference).

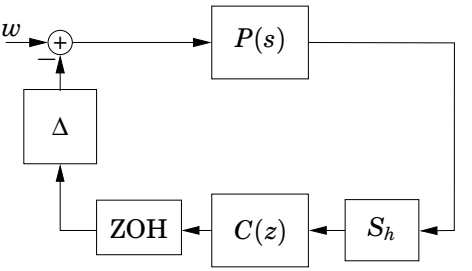


Figure 2. The control system with continuous-time plant $P(s)$, a discrete-time controller $C(z)$, the sample-and-hold S_h with a sample period of h seconds, a Zero-Order-Hold, and finally a time-varying delay Δ.

where $P_{\text{ZOH}}(z)$ is the ZOH discretization of $P(s)$, and

$$P_{\text{alias}}(\omega) = \sqrt{\sum_{k=-\infty}^{\infty} \left| P\left(j(\omega + 2\pi k)\frac{1}{h} \right) \right|^2}. \tag{7}$$

□

Remark 1 Note that the delay Δ in expressions (1) and (5) is defined on space of functions, while in (3) Δ is defined on space of sequences.

Remark 2 There are no other restrictions on the delay than (1), (3), (5); the delay could be constant or infinitely fast time-varying within the interval.

Remark 3 For a well chosen sample period h, the aliasing sum (7) converges to

$$P_{\text{alias}}(\omega) \approx P_{\text{ZOH}}(e^{j\omega}),$$

and therefore essentially (4) and (6) are the same.

Remark 4 If the closed loop system has a resonance peak, then this is usually where the inequalities (2), (4), (6) are most likely to fail. For a system with a non-complicated Nyquist curve, the resonance peak is usually around the cut-off frequency ω_c, i.e. where $|C(j\omega_c)P(j\omega_c)| = 1$. Then

$$\left|\frac{P(j\omega_c)C(j\omega_c)}{1+P(j\omega_c)C(j\omega_c)}\right| = \frac{1}{|e^{j\varphi_m}-1|} = \frac{1}{\sqrt{\varphi_m^2+O(\varphi_m^4)}},$$

where φ_m is the phase margin (which is assumed to satisfy $\varphi_m < 1$). Relaxing (2) to only one point, ω_c, it becomes

$$\left|\frac{P(j\omega_c)C(j\omega_c)}{1+P(j\omega_c)C(j\omega_c)}\right| \approx \frac{1}{\varphi_m} < \frac{1}{\delta_{\max}\omega_c} \Leftrightarrow$$
$$\varphi_m > \delta_{\max}\omega_c,$$

which is the exact stability criterion for a constant time delay $\delta_{\max}$ according to the Nyquist theorem. Note that this delay is allowed according to (1). This indicates that the stability criterion is not very conservative.

Remark 5 In light of the previous remark, a natural question is whether a time-varying delay is really a problem when $\delta(t) \equiv \delta_{\max}$ is most often the worst for stability. The answer is that the time-varyingness may not make a "non-complicated" system (see above) less stable, but it does make it *harder to compensate for the time-delays* in the controller.

Remark 6 In this paper, all subsystems we consider are signal-input-single-output and, as to be show in the next session, the stability criteria we derive are based on the small gain theorem. These criteria can be easily generalized to the multiple-input-multiple-output case. In the multiple-input-multiple-output case, using the idea presented in the next session together with IQC theory [Megretski and Rantzer, 1997] and S-procedure [Megretski and Treil, 1993; Yakubovich, 1971], one can derive stability criteria which can be posed as linear matrix inequalities and can be easily verified using computers.

3. Proofs

The proofs of Theorems 1–3 are simply based on the small gain theorem. The basic idea is to transform the delay into a direct feedthrough and an error path. The gains of the time-varying delay-error operator and the remaining linear system are calculated, and the small gain requirement of $\gamma < 1$ gives the criteria.

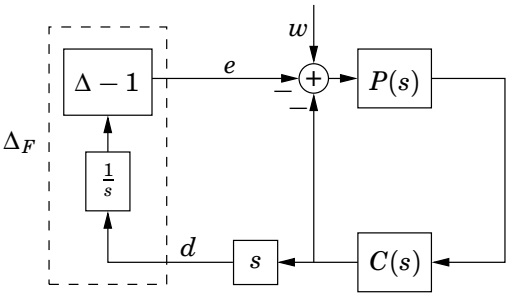


Figure 3. Loop transformation of the time-delayed system in Figure 1.

3.1 Proof of Theorem 1

The proof of Theorem 1 mainly relies on the following technical result.

LEMMA 1
Consider the operator $\Delta_F := (\Delta - 1) \circ \frac{1}{s}$, where $\frac{1}{s}$ denotes the integration operator. Then the $\mathbf{L}_2$ induced norm of Δ_F is bounded by $\delta_{\max}$. □

PROOF OF LEMMA 1
Let $w = \Delta_F(v)$, and

$$y(t) = \int_0^t v(\tau)d\tau.$$

Then

$$w(t) = y(t - \delta(t)) - y(t) = -\int_{t-\delta(t)}^{t} v(\tau)d\tau.$$

Hence, the following inequalities hold

$$\begin{aligned} w(t)^2 &= \left(-\int_{t-\delta(t)}^{t} v(\tau)d\tau\right)^2 \\ &\leq \delta(t)\left(\int_{t-\delta(t)}^{t} v(\tau)^2 d\tau\right) \leq \delta_{\max}\left(\int_{t-\delta_{\max}}^{t} v(\tau)^2 d\tau\right), \end{aligned}$$

using the Cauchy-Schwarz inequality. The $\mathbf{L}_2$ norm of $w(t)$ can be bounded

as follows

$$\begin{aligned}\|w\|_{\mathbf{L}_2}^2 &\le \int_0^\infty \delta_{\max}\left(\int_{t-\delta_{\max}}^{t} v(\tau)^2 d\tau\right) dt \\ &= \delta_{\max}\int_0^\infty \left(\int_{-\delta_{\max}}^{0} v(t+s)^2 ds\right) dt \\ &\le \delta_{\max}\int_{-\delta_{\max}}^{0}\int_0^\infty v(t)^2 dt ds \\ &= \delta_{\max}\|v\|_{\mathbf{L}_2}^2 \int_{-\delta_{\max}}^{0} 1 ds = \delta_{\max}^2 \|v\|_{\mathbf{L}_2}^2.\end{aligned}$$

Therefore the $\mathbf{L}_2$-gain of $(\Delta - 1)\circ\frac{1}{s}$ is bounded by $\delta_{\max}$. □

Remark 7 We note that the upper bound $\delta_{\max}$ is actually tight; i.e., $\|\Delta_F\|_{\mathbf{L}_2} = \delta_{\max}$. The proof is given in the appendix.

Now, consider the system in Figure 3. This transformed system is equivalent to the system in Figure 1 in the sense that the $\mathbf{L}_2$-gain from w to any other signal is the same. BIBO stability of the original system can be verified by checking whether the transformed system is stable. The stability criterion in Theorem 1 follows the small gain theorem applied to the transformed system: the transformed system is stable if

$$\sup_\omega \left|\frac{j\omega P(j\omega)C(j\omega)}{1+P(j\omega)C(j\omega)}\right| < \frac{1}{\delta_{\max}},$$

which in turn gives the condition stated in (2).

3.2 Proof of Theorem 2

Again, the delay operator Δ can be transformed to a direct feedthrough path and a delay error (see Figure 4). The gain calculations of Δ_F and the linear loop are very similar to the Proof of Theorem 1. Note that in this case Δ_F is an operator defined on the space of sequences.

LEMMA 2
The operator $\Delta_F := (\Delta - 1)\circ\frac{z}{z-1}$ has an $\mathbf{l}_2$ induced gain bounded by N. □

PROOF OF LEMMA 2
Let $w = \Delta_F(v)$. Then

$$w(n) = \sum_{i=0}^{n-k(n)} v(i) - \sum_{i=0}^{n} v(i) = -\sum_{i=n-k(n)+1}^{n} v(i),$$

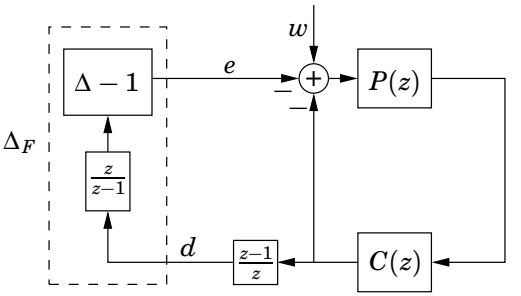


Figure 4. Loop transformation of the delayed discrete-time system.

and

$$w(n)^2 = \Big(\sum_{i=n-k(n)+1}^{n} v(i) \Big)^2 \leq N \Big(\sum_{i=n-N+1}^{n} v(i)^2 \Big),$$

using the Cauchy-Schwarz inequality and $k(n) \leq N$. The output energy is then

$$\begin{aligned} \|w\|_{\mathbf{l}_2}^2 &\leq \sum_{n=0}^{\infty} N \Big(\sum_{i=n-N+1}^{n} v(i)^2 \Big) = N \sum_{n=0}^{\infty} \Big(\sum_{i=1-N}^{0} v(i+n)^2 \Big) \\ &= N \sum_{i=1-N}^{0} \Big(\sum_{n=0}^{\infty} v(n)^2 \Big) = N^2 \|v\|_{\mathbf{l}_2}^2. \end{aligned}$$

□

The gain of the the linear part of the system from e to d in Figure 4 is simply

$$\gamma(G_{ed}) = \sup_{\omega \in [0,\pi]} \left| \frac{P(e^{j\omega})C(e^{j\omega})}{1 + P(e^{j\omega})C(e^{j\omega})} (1 - e^{-j\omega}) \right|,$$

and, again, the small gain theorem proves Theorem 2.

3.3 Proof of Theorem 3

The proof of the mixed continuous-time discrete-time Theorem 3 is very similar to the proof of Theorem 2. The main difference is that the signals d and e are now continuous-time signals (see Figure 5), and thus all gains are calculated in continuous-time.

LEMMA 3
The operator Δ_F in Figure 5 has an $\mathbf{L}_2$ induced gain bounded by N if the class of input signals is restricted to ZOH signals of period h; i.e., signals of the following form:

$$v(t) = \sum_{i=0}^{\infty} v_i \theta(t - ih), \tag{8}$$

where v_i are real numbers and

$$\theta(t) = \begin{cases} 1, & t \in [0, 1), \\ 0, & \text{otherwise.} \end{cases}$$

□

PROOF OF LEMMA 3
Let $w = \Delta_F(v)$. Then for any fixed t,

$$w(t) = \sum_{i=0}^{p_2(t)} v_i - \sum_{i=0}^{p_1(t)} v_i = \sum_{i=p_2(t)+1}^{p_1(t)} -v(ih),$$

where $p_1(t) = \lfloor \frac{t}{h} \rfloor$, $p_2(t) = \left\lfloor \frac{t-\delta(t)}{h} \right\rfloor$, and $\lfloor x \rfloor$ denotes the greatest integer $\leq x$. Since $\delta(t) \leq Nh$ and N is an integer, we have $p_2(t) = \lfloor \frac{t-\delta(t)}{h} \rfloor \geq \lfloor \frac{t-Nh}{h} \rfloor = \lfloor \frac{t}{h} \rfloor - N$ and

$$w(t)^2 = \left(\sum_{i=p_2(t)+1}^{p_1(t)} v(ih) \right)^2 \leq N \left(\sum_{i=p_1(t)-N+1}^{p_1(t)} v(ih)^2 \right). \tag{9}$$

The output energy is then

$$\begin{aligned} \|w\|_{\mathbf{L}_2}^2 &\leq \int_0^{\infty} N \left(\sum_{i=p_1(t)-N+1}^{p_1(t)} v(ih)^2 \right) dt \\ &= N \sum_{i=1-N}^{0} \left(\int_0^{\infty} v((p_1(t)+i)h)^2 dt \right) \\ &= N^2 \|v\|_{\mathbf{L}_2}^2, \end{aligned} \tag{10}$$

where the last equality follows the fact that v is in the form of (8). □

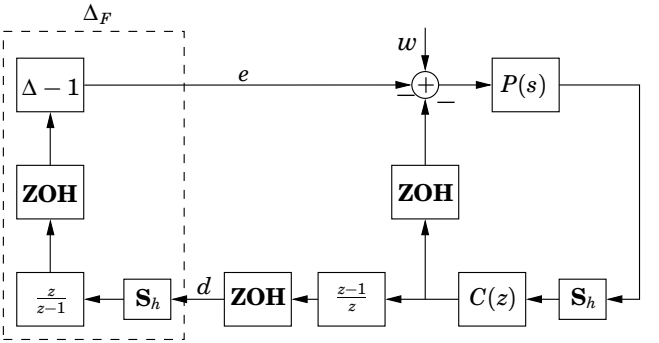


Figure 5. The continuous-discrete control system with the delay transformed to a feedthrough path and an error path.

The gain of the operator G_{ed} is more complicated than in the proof of Theorem 2, as it consists of both continuous-time and sampled systems. This gain can be calculated in a number of ways, but the most useful result for our purposes is presented in [Yamamoto and Araki, 1994]. The gain of G_{ed} is

$$\gamma(G_{ed}) = \sup_{\omega\in[0,\pi]} \left\{ \sqrt{\sum_{k=-\infty}^{\infty} \left| P\left(j(\omega+2\pi k)\frac{1}{h}\right)\right|^2} \left| \frac{C(e^{j\omega})}{1+P(e^{j\omega})C(e^{j\omega})}(1-e^{-j\omega})\right| \right\}.$$

Again, the small gain theorem proves Theorem 3.

4. Extensions

This section contains some handy extensions of Theorems 1–3.

4.1 Sub-Sampleperiod Delay Margins

Theorem 3 states that if the inequality (6) holds for a certain integer N, then the system is stable for any (time-varying) delays up to a maximum of Nh seconds. In this section we will relax this N to be any positive real number.

Let $g := N - \lfloor N \rfloor$, i.e. the fractional part of N. Consider the k^{th} sam-

pling period and let $t \in [kh, (k+1)h)$. Then it can be easily verified that

$$\left\lfloor \frac{t-\delta(t)}{h} \right\rfloor \geq \begin{cases} \lfloor \frac{t}{h} \rfloor - \lfloor N \rfloor - 1, & \text{if } t \in T_1, \\ \lfloor \frac{t}{h} \rfloor - \lfloor N \rfloor, & \text{if } t \in T_2, \end{cases}$$

where $T_1 = [kh, (k+g)h)$ and $T_2 = [(k+g)h, (k+1)h)$. Hence, inequality (9) can be replaced by

$$w(t)^2 \leq \begin{cases} (\lfloor N \rfloor + 1) \left(\sum_{i=p_1(t)-\lfloor N \rfloor}^{p_1(t)} v(ih)^2 \right), & t \in T_1, \\ \lfloor N \rfloor \left(\sum_{i=p_1(t)-\lfloor N \rfloor+1}^{p_1(t)} v(ih)^2 \right), & t \in T_2. \end{cases}$$

The above inequalities hold for any sampling period. Now, since v is restricted to be the ZOH signal of the form (8), inequality (10) can be rewritten as

$$\begin{aligned} \|w\|_{\mathbf{L}_2}^2 &\leq g \int_0^\infty (\lfloor N \rfloor + 1) \left(\sum_{i=p_1(t)-\lfloor N \rfloor}^{p_1(t)} v(ih)^2 \right) dt \\ &\quad + (1-g) \int_0^\infty \lfloor N \rfloor \left(\sum_{i=p_1(t)-\lfloor N \rfloor+1}^{p_1(t)} v(ih)^2 \right) dt \\ &= \left(\lfloor N \rfloor^2 + 2\lfloor N \rfloor g + g \right) \|v\|_{\mathbf{L}_2}^2. \end{aligned} \tag{11}$$

This leads to the following corollary:

COROLLARY 3.1
Theorem 3 holds for all delays

$$0 \leq \delta(t) \leq Nh,$$

where N is a real number if $\forall \omega \in [0, \pi]$

$$\left| \frac{P_{\text{alias}}(\omega) C(e^{j\omega})}{1 + P_{\text{ZOH}}(e^{j\omega}) C(e^{j\omega})} \right| < \frac{1/\sqrt{\lfloor N \rfloor^2 + 2\lfloor N \rfloor g + g}}{\left| e^{j\omega} - 1 \right|},$$

with $g = N - \lfloor N \rfloor$. □

The proof is given by (11) and the small gain theorem.

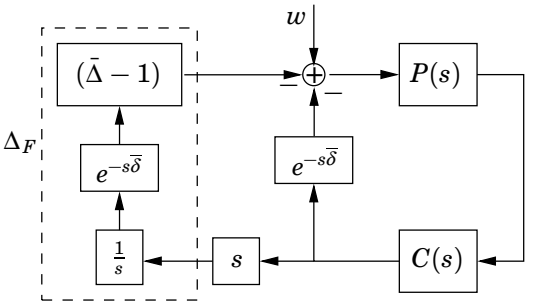


Figure 6. The transformed system in Figure 3 with a nominal delay of $\overline{\delta}$ seconds included.

4.2 Nominal Time Delay

In many cases, the controller in the loop is designed for a nominal time delay. For these cases, a more interesting stability region is a stability interval around this nominal delay. This section presents three corollaries which are extensions of Theorems 1–3 for this purpose. Throughout the section, we assume that the closed loop system including the nominal delay is stable.

COROLLARY 3.2
For the closed loop system in Figure 1, with a continuous-time, strictly proper plant $P(s)$ and a continuous-time compensator $C(s)$, the system is stable for any time-varying delays defined by

$$\Delta(v) = v(t - \overline{\delta} - \delta(t)), \quad |\delta(t)| \leq \delta_{\max} \leq \overline{\delta},$$

if

$$\left| \frac{P(j\omega)C(j\omega)}{1 + P(j\omega)C(j\omega)e^{-j\omega\overline{\delta}}} \right| < \frac{1}{\sqrt{2}\delta_{\max}\omega}, \quad \forall \omega \in [0, \infty].$$

□

Proof. Define $\bar{\Delta}$ and Δ_F to be

$$\overline{\Delta}(v) = v(t - \delta(t)), \quad |\delta(t)| \leq \delta_{\max}$$
$$\Delta_F = (\bar{\Delta} - 1) \circ e^{-s\overline{\delta}} \circ \frac{1}{s}.$$

Then the original system can be equivalently expressed as the one in Figure 6, and the corollary follows the proof of Theorem 1 as long as the

$\mathbf{L}_2$ gain of Δ_F is $\sqrt{2}\delta_{\max}$. Since

$$(\bar{\Delta}-1)\circ e^{-s\overline{\delta}}\circ\frac{1}{s}=(\bar{\Delta}-1)\circ\frac{1}{s}\circ e^{-s\overline{\delta}},$$

hence $\|\Delta_F\|_{\mathbf{L}_2}\leq\|(\bar{\Delta}-1)\circ\frac{1}{s}\|_{\mathbf{L}_2}$. Furthermore, it can be shown that $\|(\bar{\Delta}-1)\circ\frac{1}{s}\|_{\mathbf{L}_2}\leq\sqrt{2}\delta_{\max}$. For the details, see Appendix 7.

COROLLARY 3.3
For the closed loop system in Figure 1, with discrete-time $P(z)$ and $C(z)$, the system is stable for any time-varying delays defined by

$$\Delta(v)=v(n-\overline{\delta}-k(n)),\quad k(n)\in\{-N,\cdots,N\}$$

for $N\leq\overline{\delta}$, if

$$\left|\frac{P(e^{j\omega})C(e^{j\omega})}{1+P(e^{j\omega})C(e^{j\omega})e^{-j\overline{\delta}\omega}}\right|<\frac{1}{\sqrt{2}N|e^{j\omega}-1|},\quad\forall\omega\in[0,\pi].$$

□

Proof. The proof is very similar to the proof of Corollary 3.2 and of Theorem 2, and is therefore omitted.

COROLLARY 3.4
For the closed loop system in Figure 2, with continuous-time, strictly proper and stable $P(s)$, and discrete-time $C(z)$, the system is stable for any time-varying delays defined by

$$\Delta(v)=v(t-\overline{\delta}-\delta(t)),\quad|\delta(t)|\leq Nh\leq\overline{\delta},$$

if $\forall\omega\in[0,\pi]$

$$\left|\frac{P_{\text{alias}}(\omega)C(e^{j\omega})}{1+P_{\text{delay}}(e^{j\omega})C(e^{j\omega})}\right|<\frac{1/\sqrt{\lfloor N\rfloor^2+2\lfloor N\rfloor g+g}}{\sqrt{2}|e^{j\omega}-1|},$$

where $g=N-\lfloor N\rfloor$, $P_{\text{delay}}(z)$ is the ZOH discretization of $P(s)e^{-s\overline{\delta}}$, and $P_{\text{alias}}(\omega)$ is defined in (7). □

Proof. Again, the proof follows the line of that of Corollary 3.2. The gain calculation of Δ_F is given in Appendix 7.

5. Performance

There are many possible performance measures for a control system. Here, we will define the $\mathbf{L}_2$ gain (or $\mathbf{l}_2$ gain in the discrete-time case) γ_{wx} from w to x in Figure 7 as the performance measure. We say that the system has good performance if the gain is small. For the system without delays, this gain is simply

$$\gamma_{wx} = \sup_{\omega} \frac{1}{1+PC} W, \tag{12}$$

i.e., the maximum gain of the weighted sensitivity function.

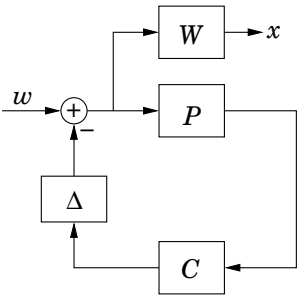


Figure 7. The performance measure is defined as the $\mathbf{L}_2/\mathbf{l}_2$ gain from w to x. W is a weighting function.

For systems with time delay, the stability criteria presented in the previous section can be used to estimate an upper bound on the performance measure. In the rest of this section, we consider only the case where both

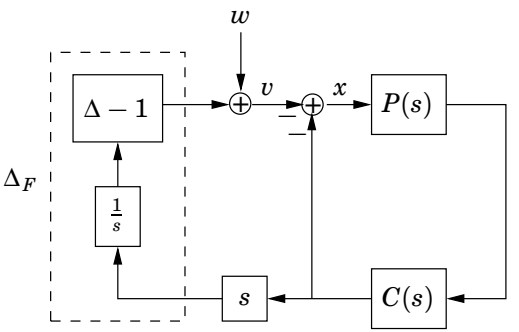


Figure 8. The feedback system in Figure 7 is transformed in the same way as for the stability criteria.

the plant and the controller are continuous-time systems. The discrete-time and the mixed continuous-time discrete-time cases can be treated similarly.

We note that the system in Figure 7 can be equivalently expressed as the one in Figure 8. The $\mathbf{L}_2$-induced gain from w to x will now be estimated in two steps. First we compute an upper bound of the $\mathbf{L}_2$ gain from w to v. Notice that signals w and v satisfy the following equation

$$v = w + (G \circ \Delta_F)v,$$

where G is the linear time-invariant system $\frac{sP(s)C(s)}{1+P(s)C(s)}$. Therefore,

$$\|v\|_{\mathbf{L}_2} \leq \|w\|_{\mathbf{L}_2} + \gamma\|v\|_{\mathbf{L}_2},$$

where

$$\gamma = \sup_{\omega} \left\{ \left| \frac{\omega P(j\omega)C(j\omega)}{1 + P(j\omega)C(j\omega)} \right| \right\} \cdot \delta_{\max}.$$

Thus, for $\gamma < 1$ (i.e. when the closed-loop system is stable), we have

$$\|v\|_{\mathbf{L}_2} \leq \frac{1}{1-\gamma}\|w\|_{\mathbf{L}_2}.$$

Now we can easily obtain an upper bound on $\|x\|_{\mathbf{L}_2}$

$$\begin{aligned} \|x\|_{\mathbf{L}_2} &\leq \sup_{\omega} \left| \frac{W(j\omega)}{1 + P(j\omega)C(j\omega)} \right| \|v\|_{\mathbf{L}_2} \\ &\leq \left(\sup_{\omega} \left| \frac{W(j\omega)}{1 + P(j\omega)C(j\omega)} \right| \right) \left(\frac{1}{1-\gamma} \right) \|w\|_{\mathbf{L}_2}, \end{aligned}$$

and hence

$$\gamma_{wx} \leq \left(\sup_{\omega} \left| \frac{W(j\omega)}{1 + P(j\omega)C(j\omega)} \right| \right) \left(\frac{1}{1-\gamma} \right).$$

The impact of the delay on performance is simply the scaling factor of $\frac{1}{1-\gamma}$ on the disturbance gain γ_{wx}. The same performance degradation bound applies for both discrete-time and mixed continuous-time discrete-time systems.

6. Example

The following example illustrates the theorems presented in this paper.

One of the most important control loops in a DVD player is the radial positioning loop. It keeps the laser pick-up-head centered over the track while the disk is spinning. In this example we will analyze the timing stability of such a control loop.

The process model of the electromechanical pick-up is given in continuous time as

$$\dot{x}(t) = \begin{bmatrix} 13.48 & -613.3 \\ 160.4 & -221.6 \end{bmatrix} x(t) + \begin{bmatrix} -9.57 \\ -1046 \end{bmatrix} u(t),$$
$$y(t) = \begin{bmatrix} 3354 & 5.395 \end{bmatrix} x(t).$$

This model is obtained the system identification of a real-world DVD servo. The continuous-time transfer function of the process is denoted $P(s)$. Using this model, a fourth order continuous-time LQG controller is designed (the two extra states model the noise). The bandwidth of the controller is about 2.1 kHz, and we denote the transfer function of the controller $C(s)$. In this example, the control loop stability has been analyzed for the three cases (corresponding to Theorems 1–3)

1. *Continuous-time plant and controller:* $P(s)$ and $C(s)$ defined above are used.
2. *Discrete-time plant and controller:* The controller is obtained by discretizing the $C(s)$ to $C(z)$ using the matched pole-zero method, and the $P(z)$ is obtained by discretizing $P(s)$ using a zero-order-hold.
3. *Continuous-time plant and discrete-time controller:* This is the most realistic case. The controller is the same as in case 2. It should be mentioned that for a real implementation, the LQG controller should of course be designed in discrete-time instead.

The discrete-time components of the system are running at a sample rate of 80 kHz, corresponding to a sample period $h = 12.5$ μs.

6.1 Stability

Applying Theorems 1–2 and Corollary 3.1 to the example, we get the following maximum delays to guarantee stability:

Case	$\delta_{\max}$	δ_{const}
1	50 μs	57 μs
2	25 μs	25 μs
3	32 μs	37 μs

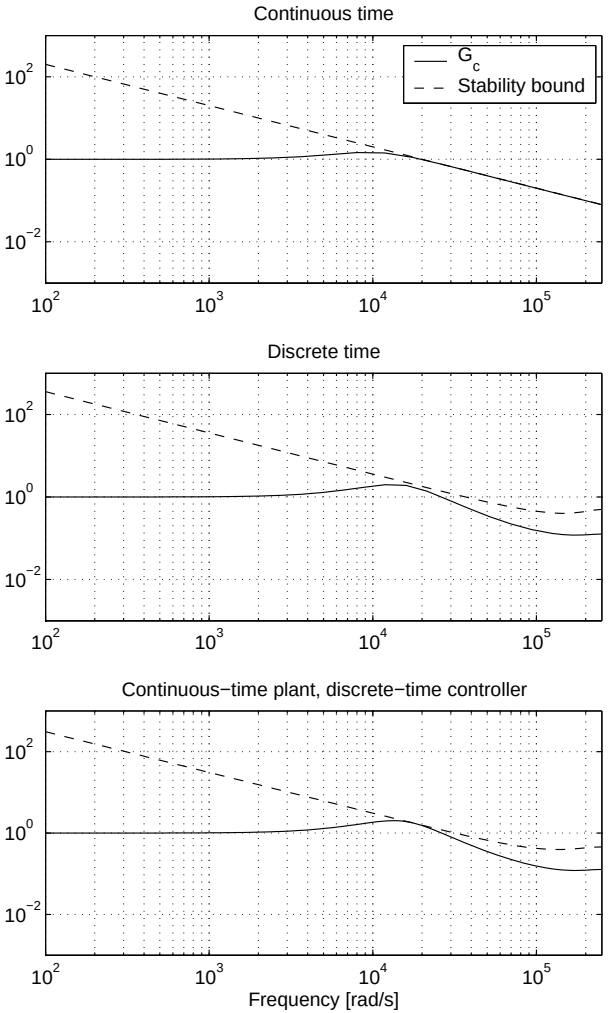


Figure 9. Proving stability for the longest possible delay of the three cases in the DVD control example.

The column $\delta_{\max}$ indicates the maximum time-varying delay allowed according to the theorems, and, for comparison, δ_{const} is the maximum *constant* delay allowed according to the Nyquist theorem. The graphical proofs of the maximum time-varying delays are shown in Figure 9.

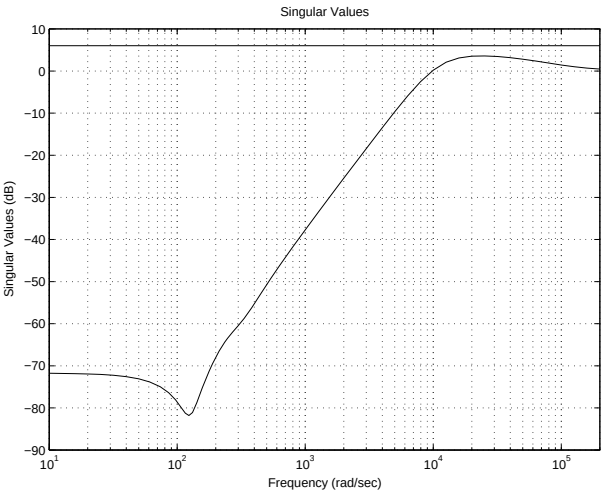


Figure 10. The sensitivity function $S(j\omega)$ for the continuous-time controller and radial servo in the example. The line corresponds to the upper bound on the gain with a maximum delay of $h = 12.5$ μs and no weighting function W.

6.2 Performance

The performance bound from Section 5 can be used to calculate the degradation of performance in the control loop due to delays. The measure of performance is the sensitivity gain, as defined in Section 5. We assume that a maximum delay of $h = 12.5$ μs can be guaranteed in the loop. The total loop gain γ can then be calculated, and from that the increase in sensitivity gain, $\frac{1}{1-\gamma}$.

For the continuous-time controller, the loop gain $\gamma = 0.25$, which implies that the sensitivity gain can be increased by up to a factor of $\frac{1}{1-0.25} \approx 1.3$. See Figure 10.

For the discrete-time controller and continuous-time plant (the realistic case), the loop gain is $\gamma = 0.39$, implying a increase in sensitivity gain by a factor of ≈ 1.6. To get a hint on the conservativeness of this factor, numerical calculations has been performed for the system using the Matlab Toolbox JITTERBUG [Lincoln and Cervin, 2002]. Modeling the input signal d as shaped noise, and fixing the delay at the maximum value, an actual sensitivity gain increase of a factor of ≈ 1.4 is obtained.

7. Conclusions

In this paper we have presented simple but powerful stability criteria for linear systems with time-varying bounded delays. Stability is checked graphically in a Bode plot, and therefore the criteria can be easily used for robust loop-shaping design. It has been indicated that the criteria are not very conservative.

A performance degradation bound which is directly calculated from the stability margin has also been presented.

Acknowledgments

The authors would like to thank professor Anders Rantzer for valuable discussion and suggestions.

Appendix A: Proof of $\|\Delta_F\| = \delta_{\max}$

To show that $\|\Delta_F\| = \delta_{\max}$, let $\delta(t) = \delta_{\max}$ for all t. Then operator Δ_F is an LTI operator with transfer function $\Delta_F(s)$ equal to

$$\frac{e^{-s\delta_{\max}} - 1}{s}.$$

The $\mathbf{L}_2$ induced norm of Δ_F satisfies

$$\|\Delta_F\|_{\mathbf{L}_2}^2 \geq \lim_{\omega \to 0} \left| \frac{e^{-j\omega\delta_{\max}} - 1}{j\omega} \right|^2 = \delta_{\max}^2.$$

Hence, $\|\Delta_F\|_{\mathbf{L}_2} \geq \delta_{\max}$ which in turn implies that $\|\Delta_F\| = \delta_{\max}$.

Appendix B: Proof of $\|(\overline{\Delta} - 1) \circ \frac{1}{s}\| \leq \sqrt{2}\delta_{\max}$

Define Δ_G to be $(\overline{\Delta} - 1) \circ \frac{1}{s}$. Let $w = \Delta_G(v)$, and

$$y(t) = \int_0^t v(\tau) d\tau.$$

Then $w(t)$ is equal to

$$y(t - \delta(t)) - y(t) = -\int_{t-\delta(t)}^{t} v(\tau) d\tau.$$

Hence,

$$w(t)^2 \leq |\delta(t)| \left(\int_{t-|\delta(t)|}^{t} v(\tau)^2 d\tau \right),$$

if $\delta(t) \leq 0$, and

$$w(t)^2 \leq |\delta(t)| \left(\int_{t}^{t+\delta(t)} v(\tau)^2 d\tau \right),$$

if $\delta(t) > 0$. In either case,

$$w(t)^2 \leq \delta_{\max} \left(\int_{t-\delta_{\max}}^{t+\delta_{\max}} v(\tau)^2 d\tau \right).$$

The $\mathbf{L}_2$ norm of $w(t)$ can be estimated as follows

$$\begin{aligned} \|w\|_{\mathbf{L}_2}^2 &\leq \delta_{\max} \int_0^\infty \left(\int_{t-\delta_{\max}}^{t+\delta_{\max}} v(\tau)^2 d\tau \right) dt \\ &= \delta_{\max} \left(\int_{-\delta_{\max}}^{\delta_{\max}} \int_0^\infty v(t+s)^2 dt \right) ds \\ &= 2\delta_{\max}^2 \|v\|_{\mathbf{L}_2}. \end{aligned}$$

Therefore the $\mathbf{L}_2$-gain of $(\overline{\Delta} - 1) \circ \frac{1}{s}$ is bounded by $\sqrt{2}\delta_{\max}$.

Appendix C: Proof of Corollary 3.4

Define Δ_G to be

$$(\overline{\Delta} - 1) \circ ZOH \circ \frac{z}{z-1} \circ S_h,$$

where ZOH and S_h denote Zero-Order-Hold operator and Sample-and-hold operator with a sampling period of h seconds. Let $w = \Delta_G(v)$. By arguments similar to those in Section 4.1, one can verify that the following

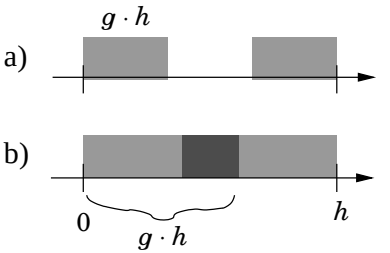


Figure 11. The two cases $g < 0.5$ (a) and $g \geq 0.5$ (b). The gray areas where the longer delay $\lfloor N \rfloor + 1$ can be active.

inequalities hold in every sampling period $[kh, (k+1)h)$

$$w(t)^2 \leq \begin{cases} (\lfloor N \rfloor + 1) \left(\sum_{i=p_1(t)-\lfloor N \rfloor}^{p_1(t)} v(ih)^2 \right), & p_1(t) \in T_1 \\ & \delta(t) \geq 0, \\ \lfloor N \rfloor \left(\sum_{i=p_1(t)-\lfloor N \rfloor+1}^{p_1(t)} v(ih)^2 \right), & p_1(t) \in T_2 \\ & \delta(t) \geq 0, \\ (\lfloor N \rfloor + 1) \left(\sum_{i=p_1(t)}^{p_1(t)+\lfloor N \rfloor} v(ih)^2 \right), & p_1(t) \in T_3 \\ & \delta(t) < 0, \\ \lfloor N \rfloor \left(\sum_{i=p_1(t)}^{p_1(t)+\lfloor N \rfloor-1} v(ih)^2 \right), & p_1(t) \in T_4 \\ & \delta(t) < 0, \end{cases}$$

where $p_1(t) = \lfloor \frac{t}{h} \rfloor$, $T_1 = [kh, (k+g)h)$, $T_2 = [(k+g)h, (k+1)h)$, $T_3 = [(k+1-g), (k+1)h)$, and $T_4 = [kh, (k+1-g)h)$.

Now consider the two cases $g < 0.5$ and $g \geq 0.5$ (see Figure 11). In the former case, the long delay $\lfloor N \rfloor + 1$ (compared to $\lfloor N \rfloor$) can only be active in the beginning and end of the sample interval, and only in one direction. In the latter case, there is an interval of length $2(1-g)$ in the center of the sample interval where the long delay can be active in both directions.

For $g < 0.5$, the energy of the output is bounded as

$$\begin{aligned}\|w\|_{\mathbf{L}_2}^2 = \int_0^\infty w(t)^2 dt \leq &\left(2g\Big((\lfloor N \rfloor + 1)^2 + \lfloor N \rfloor^2\Big)\right.\\ &\left. + (1-2g)\Big(2\lfloor N \rfloor^2\Big)\right)\|v\|_{\mathbf{L}_2}^2\\ =&2\Big(\lfloor N \rfloor^2 + 2\lfloor N \rfloor g + g\Big)\|v\|_{\mathbf{L}_2}^2,\end{aligned}$$

and for $g \geq 0.5$

$$\begin{aligned}\|w\|_{\mathbf{L}_2}^2 \leq &\left(2(1-g)\Big((\lfloor N \rfloor + 1)^2 + \lfloor N \rfloor^2\Big)\right.\\ &\left. + (2g-1)\Big(2(\lfloor N \rfloor + 1)^2\Big)\right)\|v\|_{\mathbf{L}_2}^2\\ =&2\Big(\lfloor N \rfloor^2 + 2\lfloor N \rfloor g + g\Big)\|v\|_{\mathbf{L}_2}^2.\end{aligned}$$

Thus, the $\mathbf{L}_2$ induced gain of Δ_G is

$$\gamma(\Delta_G) \leq \sqrt{2\Big(\lfloor N \rfloor^2 + 2\lfloor N \rfloor g + g\Big)}.$$

References

Cervin, A., D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén (2003): "Analysis and simulation of controller timing." *IEEE Control Systems Magazine*. To appear.

Chen, J. and H. A. Latchman (1995): "Frequency sweeping tests for stability independent of delay." *IEEE Transactions on Automatic Control*, **40:9**, pp. 1640–1645.

Gu, K. and Q. Han (2000): "Discretized Lyapunov functional for linear uncertain systems with time-varying delay." In *Proceedings of the American Control Conference*, pp. 1375–1379.

Huang, Y. and K. Zhou (2000): "Robust stability of uncertain time-delay systems." *IEEE Transactions on Automatic Control*, **45:11**, pp. 2169–2173.

Kao, C.-Y. and A. Rantzer (2003): "Stability critera for systems with bounded uncertain time-varying delays." Submitted to European Control Conference 2003, Cambridge, UK.

Kopetz, H. (2002): "Time-triggered real-time computing." *IFAC World Congress, Barcelona, July 2002, IFAC Press*, jul.

Lincoln, B. (2002): "A simple stability criterion for control systems with varying delays." In *Proceedings of the 15th IFAC World Congress*.

Lincoln, B. and A. Cervin (2002): "Jitterbug: A tool for analysis of real-time control performance." In *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Megretski, A. and A. Rantzer (1997): "System analysis via Integral Quadratic Constraints." *IEEE Transactions on Automatic Control*, **47:6**, pp. 819–830.

Megretski, A. and S. Treil (1993): "Power distribution inequalities in optimization and robustness of uncertain systems." *Journal of Mathematical Systems, Estimation, and Control*, **3:3**, pp. 301–319.

Verriest, E. I. (1994): "Robust stability of time varying systems with unknown bounded delays." In *Proceedings of the 33rd Conference on Decision and Control*, pp. 417–422.

Verriest, E. I., M. Fan, and J. Kullstam (1993): "Frequency domain robust stability criteria for linear delay systems." In *Proceedings of the 32nd Conference on Decision and Control*, pp. 3473–3478.

Yakubovich, V. (1971): "S-procedure in nonlinear control theory." *Vestnik Leningrad University*, pp. 62–77. (English translation in Vestnik Leningrad Univ. 4:73-93, 1977).

Yamamoto, Y. and M. Araki (1994): "Frequency responses for sampled-data systems – their equivalence and relationships." *Linear Algebra and its Applications*, **No 205–206**, pp. 1319–1339.

Yan, J.-J. (2001): "Robust stability analysis of uncertain time delay systems with delay-dependence." *Electronic Letters*, **37:2**, pp. 135–137.

Paper IV

Jitterbug: A Tool for Analysis of Real-Time Control Performance

Bo Lincoln and Anton Cervin

Abstract

The paper presents JITTERBUG, a MATLAB-based toolbox for real-time control performance analysis. The control system is described using a number of connected continuous-time and discrete-time linear systems driven by white noise. The control performance is measured by a continuous-time quadratic cost function. A stochastic timing model is used to describe when the different discrete-time systems are updated during the control period. Building different models, the tool makes it easy to investigate how the control performance is affected by e.g. delay, jitter, lost samples, aborted computations, and jitter compensation. Aperiodic and multi-rate controllers may also be studied. The tool is also capable of computing the spectral densities of the different signals in the system.

1. Introduction

Controllers are often designed with little regard for the real-time implementation. In the case of continuous-time design, it is typically assumed that the controller can be subsequently discretized and executed at a sufficiently high frequency. In the case of discrete-time design, it is commonly assumed that the computing platform can provide deterministic sampling and that the task execution will introduce negligible or at least constant computational delay.

In systems with limited computing resources (e.g. embedded control systems), however, a combination of slow sampling and other timing problems may lead to significant performance degradation. Often, the controller is implemented as a task in a (more or less real-time) operating system, and the task scheduling can introduce additional delays as well as sampling and actuation jitter (depending on how the I/O is implemented). In real-time operating systems which enforces hard deadlines, a control task may be aborted before it has finished its computations and produced a control signal. Networked control systems are another source of timing problems. The network can introduce delay and jitter, and messages (measurement or control signals) may be lost.

To achieve good performance in systems with limited computer resources, the constraints of the implementation must be taken into account at design time. Typically, trade-offs between different activities in the system must be made. For instance, boosting the priority of one task will improve its responsiveness but may introduce delay and jitter in other tasks. The periods of all tasks must be chosen such that the CPU is not overloaded, and so on. Having a quality-of-service measure which takes the timing effects into account can be a help when allocating system resources to the different tasks.

1.1 Contribution of This Paper

This paper presents a MATLAB-based toolbox called JITTERBUG which facilitates the computation of a quadratic performance criterion for a linear control system under various timing conditions. The tool helps to quickly assert how sensitive a control system is to delay, jitter, lost samples, etc, without resorting to simulation. The tool is quite general and can also be used to investigate for instance jitter-compensating controllers, aperiodic controllers, and multi-rate controllers. The toolbox is built upon well-known theory, and its main contribution is to make it easy to apply this type of stochastic analysis to a wide range of problems. The toolbox and a reference manual are available at

`http://www.control.lth.se/~lincoln/jitterbug/`

The analysis in this paper is based on jump linear systems, which were first studied in [Krasovskii and Lidskii, 1961]. Discrete-time jump linear systems are treated in e.g. [Ji *et al.*, 1991]. Analysis similar to this paper, but more specific, has been done in [Nilsson, 1998]. It is focused on optimal control of time-delay systems.

An alternative to analysis is simulation. For example, the MATLAB/Simulink-based tool TRUETIME [Henriksson *et al.*, 2002] can be used for detailed co-simulation of plant dynamics, control task execution, and real-time scheduling of CPU and network.

2. System Description

In JITTERBUG, a control system is described by two parallel models: a signal model and a timing model. A simple model of a computer-controlled system is shown in Figure 1. The plant is described by the continuous-time system G, and the controller is described by two discrete-time systems, H_1 and H_2. The system H_1 could for instance represent a periodic sampler, while H_2 could represent the computation and actuation of the control signal. The associated timing model says that, at the beginning of each control period, H_1 should first be executed (updated). Then there is a (possibly random) delay τ_1 until H_2 is executed. This simple model could be used to investigate for instance the impact of delay and jitter on control performance. We will return to this example in Section 4.1.

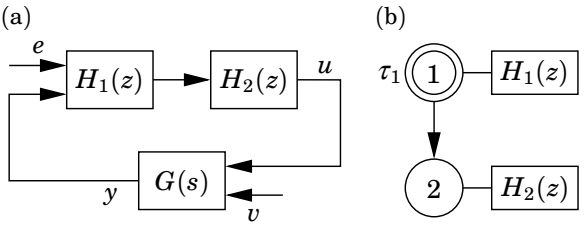


Figure 1. A simple JITTERBUG model of a computer-controlled system: (a) signal model, and (b) timing model.

2.1 Signal Model

The signal model consists of a number of inter-connected continuous-time and discrete-time linear systems driven by white noise.

A *continuous-time system* is described by

$$\dot{x}_c(t) = Ax_c(t) + Bu(t) + v_c(t),$$
$$y(t) = Cx_c(t),$$

where A, B, and C are constant matrices, and v_c is a continuous-time white noise process with covariance[2] R_{1c}. The cost of the system is defined as

$$J_c = \lim_{T\to\infty} \frac{1}{T} \int_0^T x_c^T(t) Q_c x_c(t)\, dt,$$

where Q_c is a positive semidefinite matrix.

A *discrete-time system* is described by

$$\begin{aligned} x_d(t_{k+1}) &= \Phi x_d(t_k) + \Gamma u(t_k) + v_d(t_k), \\ y(t_k) &= C x_d(t_k) + D u(t_k) + e_d(t_k), \end{aligned}$$

where Φ, Γ, C, and D may be time-varying matrices (see below). The covariance of the discrete-time white noise processes v_d and e_d is given by

$$R_d = \mathbf{E} \left(\begin{array}{c} v_d(t_k) \\ e_d(t_k) \end{array} \right) \left(\begin{array}{c} v_d(t_k) \\ e_d(t_k) \end{array} \right)^T.$$

The input signal u is sampled when the system is updated, and the output signal y is held between updates. The cost of the system is defined as

$$J_d = \lim_{T\to\infty} \frac{1}{T} \int_0^T x_d^T(t) Q_d x_d(t)\, dt,$$

where Q_d is a positive semidefinite matrix. Note that the update instants t_k need not be equidistant in time, and that the cost is defined in continuous time.

The *total system* is formed by appropriately connecting the inputs and outputs of a number of continuous-time and discrete-time systems. Throughout, MIMO formulations are allowed, and a system may collect its inputs from a number of other systems. The total cost to be evaluated is summed over all continuous-time and discrete-time systems:

$$J = \sum J_c + \sum J_d. \tag{1}$$

2.2 Timing Model

The timing model consists of a number of timing nodes. Each node can be associated with zero or more discrete-time systems in the signal model which should be updated when the node becomes active. At time zero, the first node is activated. The first node can be declared to be *periodic* (indicated by an extra circle in the illustrations), which means that the execution will restart in this node every h seconds. This is useful to model

[2]By this we mean that v_c has the spectral density $\phi(\omega) = \frac{1}{2\pi} R_{1c}$.

periodic controllers and also simplifies the cost calculations a lot, see Section 3.3.

Each node is associated with a time delay τ which must elapse before the next node can become active. (If unspecified, the delay is assumed to be zero.) The delay can be used to model computational delay, transmission delay in a network, etc. A delay is described by a discrete-time probability density function

$$P_\tau = [\, P_\tau(0) \quad P_\tau(1) \quad P_\tau(2) \quad \dots \,],$$

where $P_\tau(i)$ represents the probability of a delay of $i\delta$ seconds. The time grain δ is a constant which is specified for the whole model.

Node- and Time-Dependent Execution. The same discrete-time system may be updated in several timing nodes. It is possible to specify different update equations (i.e. different Φ, Γ, C and D matrices) in the different cases. This can be used to model e.g. a filter where the update equations look different depending on whether a measurement value is available or not. An example of this type is given in Section 4.2.

It is also possible to make the update equations depend on the time since the first node became active. This can be used to model e.g. jitter-compensating controllers.

Alternative Execution Paths. For some systems, it is desirable to specify alternative execution paths (and thereby multiple next nodes). In JITTERBUG, two such cases are possible to model, see Figure 2:

(a) A vector n of next nodes can be specified with a probability vector p. After the delay, node $n(i)$ will be activated with probability $p(i)$. This can be used to model e.g. a sample being lost with some probability.

(b) A vector n of next nodes can be specified with a time-vector t. If the total delay since the first node exceeds $t(i)$, node $n(i)$ will be activated next. This can be used to model e.g. time-outs and different compensation schemes.

Periodic vs Aperiodic Systems. For periodic systems (i.e., for systems where the first timing node is periodic), the cost J can be calculated algebraically. The solver is fast and produces an exact solution. For aperiodic systems, the cost must be computed iteratively until it converges (if ever). From this point of view, periodic systems are clearly preferable.

In periodic systems, the execution is preempted if the total delay $\sum \tau$ in the system exceeds the period h. Any remaining timing nodes will be

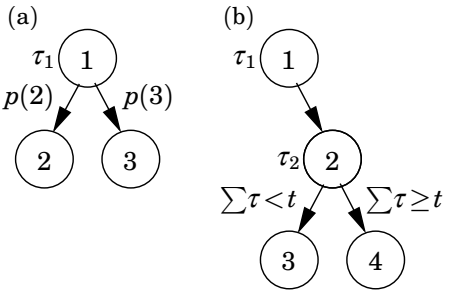


Figure 2. Alternative execution paths: (a) random choice of path, and (b) choice of path depending on the total delay from the first node.

skipped. This models a real-time system where hard deadlines (equal to the period) are enforced and the control task is aborted at the deadline.

An aperiodic system can be used to model a real-time system where the task periods are allowed to drift if there are overruns. It could also be used to model e.g. a controller which samples "as fast as possible" instead of waiting for the next period.

3. Internal Workings

Inside JITTERBUG, the states and the cost are considered in continuous time. The inherently discrete-time states, e.g. in discrete-time controllers or filters, are treated as continuous-time states with zero dynamics. This means that the total system can be written as

$$\dot{x}(t) = Ax(t) + w(t), \tag{2}$$

where x collects all the states in the system, and w is continuous-time white noise process with covariance $\widetilde{R}$. To model the discrete-time changes of some states as a timing node n is activated, the state is instantaneously transformed by

$$x(t^+) = E_n x(t) + e_n(t),$$

where e_n is a discrete-time white noise process with covariance W_n.

The total cost (1) for the system can be written as

$$J = \lim_{T\to\infty} \frac{1}{T} \int_0^T x^T(t)\widetilde{Q}x(t)\,dt, \tag{3}$$

where $\widetilde{Q}$ is a positive semidefinite matrix.

3.1 Sampling the System

JITTERBUG relies on discretized time to calculate the variance of the states and the cost. No approximations are involved, however. Sampling the system (2) with a period of δ (the time-grain in the delay distributions) gives

$$x(k\delta + \delta) = \Phi x(k\delta) + v(k\delta), \tag{4}$$

where the covariance of v is R, and the cost (3) becomes

$$J = \lim_{N\to\infty} \frac{1}{N\delta} \sum_{k=0}^{N-1} \Big(x^T(k\delta) Q x(k\delta) + q \Big).$$

The matrices Φ, R, Q, and q are calculated as

$$\begin{aligned}
\Phi &= e^{A\delta}, \\
R &= \int_0^\delta e^{A(\delta-\tau)} \widetilde{R} e^{A^T(\delta-\tau)}\, d\tau, \\
Q &= \int_0^\delta e^{A^T t} \widetilde{Q} e^{At}\, dt, \\
q &= \operatorname{tr}\Big(\widetilde{Q} \int_0^\delta \int_0^\delta e^{A(t-\tau)} \widetilde{R} e^{A^T(t-\tau)}\, d\tau\, dt \Big),
\end{aligned}$$

or, equivalently, from

$$\begin{pmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{pmatrix} = \exp\left(\begin{pmatrix} -A^T & \widetilde{Q} \\ 0 & A \end{pmatrix} \delta \right),$$

and

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} = \exp\left(\begin{pmatrix} -A & I & 0 \\ 0 & -A & \widetilde{R}^T \\ 0 & 0 & A^T \end{pmatrix} \delta \right),$$

so that

$$\begin{aligned}
\Phi &= P_{22}, \\
Q &= P_{22}^T P_{12}, \\
R &= M_{33}^T M_{23}, \\
q &= \operatorname{tr}\big(Q M_{33}^T M_{13} \big).
\end{aligned}$$

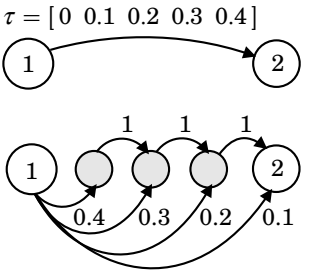


Figure 3. A random delay (above) modeled as a jump linear system (below), where the delay is represented by additional Markov nodes in between the timing nodes.

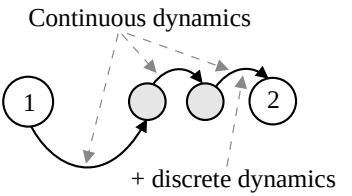


Figure 4. The continuous-time dynamics is active between all Markov nodes, whereas the discrete-time dynamics is activated only before a timing node.

3.2 Timing Representation

As time is discretized, we can transform the system description into a jump linear system, where the Markov state represents the current timing state of the system. Each timing node is represented by one Markov node. In between timing nodes additional Markov nodes representing the delay are inserted as illustrated in Figure 3.

Consider following one path in the Markov chain. For each node which is not a timing node, only the continuous states of the system change. In each time-step, they evolve as in (4), and thus the state covariance $P(k\delta) = \mathbf{E}\{x(k\delta)x^T(k\delta)\}$ evolves as

$$P(k\delta + \delta) = \Phi P(k\delta)\Phi^T + R.$$

At each timing node n, the system is additionally transformed as in (3),

$$P(k\delta^+) = E_n P(k\delta) E_n^T + W_n,$$

where W_n is the covariance of the discrete-time noise $e_n(k\delta)$ in node n. See Figure 4 for an illustration. Combining the above, we define Φ_n as

$$\Phi_n = \begin{cases} \Phi, & \text{if } n \text{ is not a timing node,} \\ E_n\Phi, & \text{if } n \text{ is a timing node,} \end{cases}$$

and similarly R_n as

$$R_n = \begin{cases} R, & \text{if } n \text{ is not a timing node,} \\ E_n R E_n^T + W_n, & \text{if } n \text{ is a timing node.} \end{cases}$$

3.3 Calculating Variance and Cost

Now consider all possible Markov states simultaneously. Let $\pi_n(k\delta)$ be the probability of being in Markov state n at time $k\delta$, and let $P_n(k\delta)$ be the covariance of the state if the system is in Markov state n at time $k\delta$. Furthermore, let the transition matrix of the Markov chain be σ, such that

$$\pi(k\delta + \delta) = \sigma\pi(k\delta).$$

The state covariance then evolves as

$$P_n(k\delta + \delta) = \sum_i \sigma_{ni}\pi_i(k\delta)\Big(\Phi_n P_i(k\delta)\Phi_n^T + R_n\Big), \tag{5}$$

and the immediate cost at time $k\delta$ is calculated as

$$\frac{1}{\delta}\sum_n \pi_n(k\delta)\Big(\operatorname{tr}\big(P_n(k\delta)Q\big) + q\Big).$$

For systems without a periodic node, equation (5) must be iterated until the cost and variance converge. For periodic systems, the Markov state always returns to the periodic timing node every h/δ time steps. As equation (5) is affine in P, we can find the stationary covariance $P_1(\infty)$ in the periodic node by solving a linear system of equations. The total cost is then calculated over the timesteps in one period. The toolbox returns the cost $J = \infty$ if the system is not mean-square stable.

3.4 Calculating Spectral Densities

For periodic systems, the toolbox also computes the discrete-time spectral densities of all outputs as observed in the periodic timing node. The spectral density of an output y is defined as

$$\phi_y(\omega) = \frac{1}{2\pi}\sum_{k=-\infty}^{\infty} r_y(k)e^{-ik\omega}.$$

The covariance function $r_y(k)$ is given by

$$\begin{aligned} r_y(k) &= \mathbf{E}\left\{y(t)y^T(t+kh)\right\} = \mathbf{E}\left\{Cx(t)x^T(t+kh)C^T\right\} \\ &= \mathbf{E}\left\{C\bar{\Phi}^{|k|}x(t)x^T(t)C^T\right\} = C\bar{\Phi}^{|k|}P_1(\infty)C^T, \end{aligned}$$

where $\bar{\Phi}$ is the average transition matrix over a period, and $P_1(\infty)$ is the stationary covariance in the periodic node. The spectral density is returned as a linear system $F(z)$ such that $\phi_y(\omega) = F(e^{i\omega})$.

```
Ptau = [0 1 1 1 0]/3;                          Define delay distribution
load G;                                        Load state-space model of the plant
H1 = tf(1,1,-1);                               Define the sampler
H2 = lqgdesign(G,Q,R1,R2,h,m);                 Design the controller

N = initjitterbug(delta,h);                    Set time-grain and period

N = addtimingnode(N,1,Ptau,2);                 Define timing node 1
N = addtimingnode(N,2);                        Define timing node 2

N = addcontsys(N,1,G,3,Q,R1,R2);               Add plant
N = adddiscsys(N,2,H1,1,1);                    Add sampler to node 1
N = adddiscsys(N,3,H2,2,2);                    Add controller to node 2

N = calcdynamics(N);                           Calculate internal dynamics
J = calccost(N);                               Calculate the cost J
```

Figure 5. An example MATLAB script showing the commands for a JITTERBUG cost calculation.

4. Examples

4.1 Delay and Jitter in a DVD Controller

Delays are a common problem in real-time control systems. In this example we examine the effect of randomly time-varying delays, or *jitter*. Generally, it is straight-forward to compensate for a constant delay, while jitter compensation is a harder issue.

In this example a DVD player focus control loop is considered. The JITTERBUG model of the system is shown in Figure 1. The plant $G(s)$ is given by a resonant sixth order continuous-time model of a DVD focus servo which has been obtained by system identification. The plant should be controlled by a discrete-time LQG controller with a sampling period of h seconds. The system $H_1(z) = 1$ represents a periodic sampler, while the system $H_2(z) = H(z)$ describes the control algorithm and the actuator. There is both process noise and measurement noise.

The sampler is executed at the beginning of each period. Then there is a random delay τ until the control signal is calculated and actuated. The LQG controller is designed to compensate for the mean delay. To study the combined effect of delay and jitter, the probability distribution for τ is given by a uniform distribution in the range $[m - j/2, m + j/2]$, where m is the mean delay and j is the jitter. An example script showing the MATLAB commands for a cost calculation is shown in Figure 5.

A plot showing the cost as a function of the mean delay and the jitter is given in in Figure 6. We can see that, in this case, the controller is

quite sensitive to jitter. Naturally, the results are dependent on all model and design parameters. With JITTERBUG, it is easy to evaluate the effects of delays and jitter exactly for any parameters without resorting to simulations.

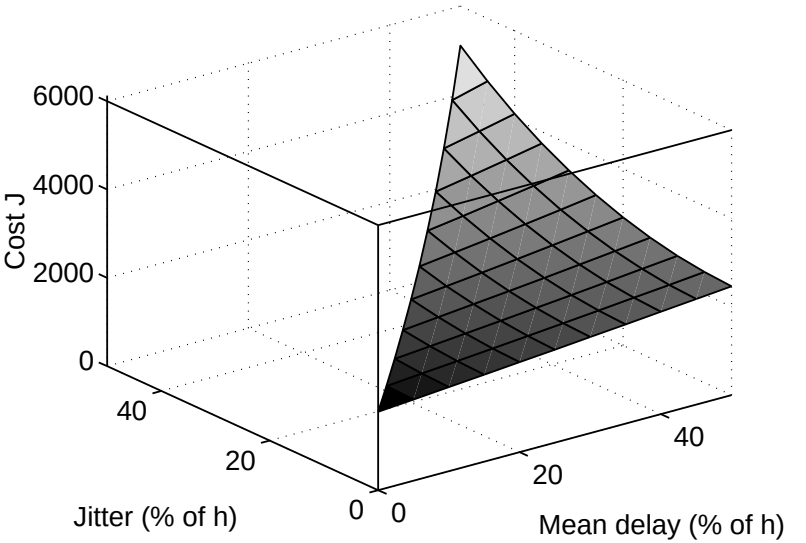


Figure 6. The cost of delay and jitter in the control loop. The controller is designed assuming a constant delay equal to the mean delay.

4.2 Lost Samples in Notch Filters

Removing disturbances from signals using e.g. notch filters is important in many applications. In some cases these filters are very sensitive to lost samples due to the very narrow-band characteristics, and in real-time systems lost samples are sometimes inevitable. In this example JITTERBUG is used to evaluate the effects of this problem on different filters.

The setup is as follows. A good signal x (modeled as low-pass filtered noise) is to be "cleaned" from an additive disturbance e (modeled as band-pass filtered noise), see the signal spectra in Figure 7. An estimate $\hat{x}$ of the good signal should be found by applying a digital notch filter with the sampling interval $h = 0.1$ to the measured signal $x + e$. Unfortunately, a fraction p of the measurements are lost.

A JITTERBUG model of the system is shown in Figure 8. The signals x and e are generated by white noise being filtered through the continuous-time systems G_1 and G_2. The digital filter is represented as two discrete-time systems: *Samp* and *Filter*. The good signal is buffered in the system

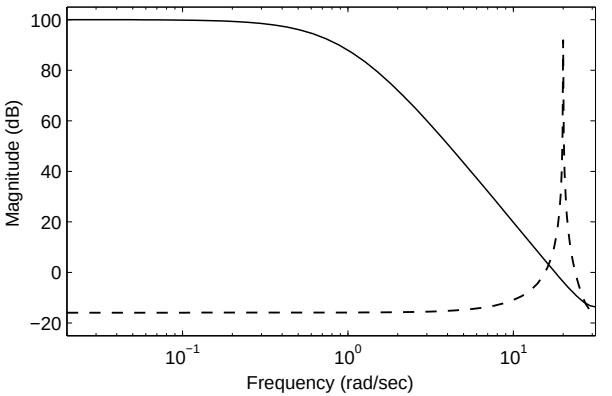


Figure 7. The spectral densities of the good signal x (full) and the disturbance e (dashed).

Delay and is compared to the filtered estimate in the system *Diff*. In the timing model, there is a probability p that the *Samp* system will not be updated. In that case, it is possible to execute an alternate version, *Filter*(2), of the filter dynamics.

Two different filters are compared. The first filter is an ordinary second-order notch filter with two zeros on the unit circle. The same update equations are used regardless if a sample is available or not. The second filter is a second-order Kalman filter based on a simple model of the signal dynamics. In the case of lost samples, only prediction is performed in the filter.

The spectral density of the estimation error $\tilde{x} = x - \hat{x}$ in the two filter cases is shown in Figure 9. It has been assumed that $p = 10\%$ of the samples are lost. It is seen that the ordinary notch filter performs well around the disturbance frequency while the lost samples introduce a large error at lower frequencies. The time-varying Kalman filter is less sensitive towards lost samples and has a more even error spectrum. Overall, the variance of the estimation error is about 40% lower in the Kalman filter case.

5. Conclusion

This paper has presented a MATLAB toolbox called JITTERBUG, which is used to compute a quadratic performance index for a real-time control system. The control (or signal processing) system is described using a number of continuous-time and discrete-time linear systems. A stochastic

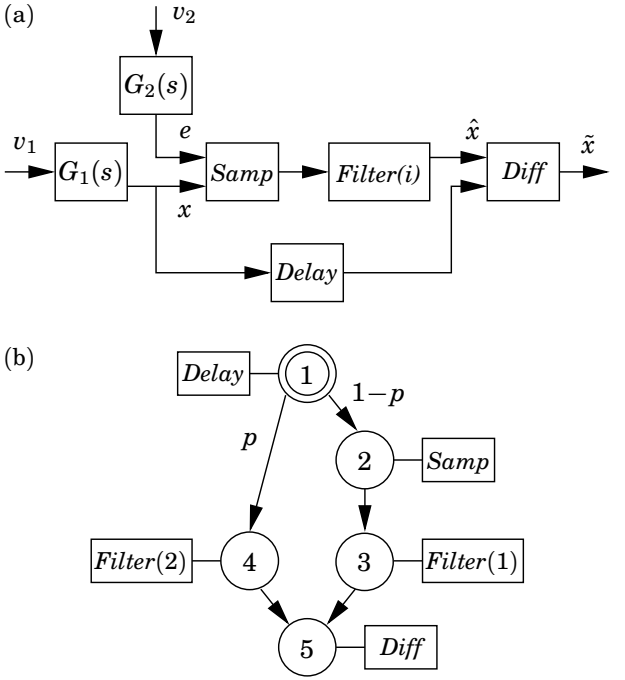


Figure 8. JITTERBUG model of the notch filter: (a) signal model, (b) timing model.

timing model with random delays is used to describe the execution of the system. Some of the things that can be investigated using the toolbox are delays, jitter, jitter compensation, lost samples, aborted computations, aperiodically sampled controllers, and multi-rate controllers.

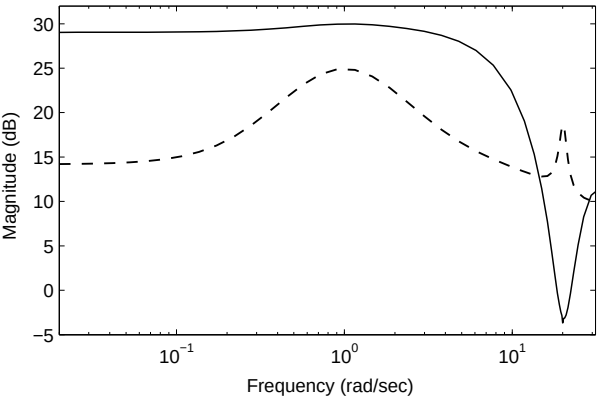


Figure 9. The spectral density of the error output $\tilde{x}$ when 10% of the samples are lost, using a notch filter (full) or a time-varying Kalman filter (dashed).

References

Henriksson, D., A. Cervin, and K.-E. Årzén (2002): "Truetime: Simulation of control loops under shared computer resources." In *Proceedings of the 15th IFAC World Congress on Automatic Control*. Barcelona, Spain.

Ji, Y., H. Chizeck, X. Feng, and K. Loparo (1991): "Stability and control of discrete-time jump linear systems." *Control-Theory and Advanced Applications*, **7:2**, pp. 247–270.

Krasovskii, N. and E. Lidskii (1961): "Analytic design of controllers in systems with random attributes, I, II, III." *Automation and Remote Control*, **22:9–11**, pp. 1021–1025, 1141–1146, 1289–1294.

Nilsson, J. (1998): *Real-Time Control Systems with Delays*. PhD thesis ISRN LUTFD2/TFRT--1049--SE, Department of Automatic Control, Lund Institute of Technology, Sweden.