Feedback Control of Computing Systems

Joseph L. Hellerstein
Yixin Diao
Sujay Parekh
Dawn M. Tilbury

IEEE PRESS

A JOHN WILEY & SONS, INC., PUBLICATION

Feedback Control
of Computing Systems

Feedback Control of Computing Systems

Joseph L. Hellerstein
Yixin Diao
Sujay Parekh
Dawn M. Tilbury

IEEE PRESS

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Feedback control of computing systems / Joseph L. Hellerstein ... [et al.].
p. cm.
"A Wiley-Interscience publication."
Includes bibliographical references and index.
ISBN 0-471-26637-X (cloth)
1. Feedback control systems. 2. Control theory. 3. Electronic data processing. I. Hellerstein, Joseph, 1952-

TJ216.F44 2004
629.8′3–dc22
2004040490

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To Our Families

Contents

Preface

This book is intended primarily for practitioners engaged in the analysis and design of computing systems. Analysts and designers are extremely interested in the performance characteristics of computing systems, especially response times, throughputs, queue lengths, and utilizations. Although steady-state characteristics can be well understood using queueing theory (e.g., as is done with capacity planning), practitioners lack the conceptual tools to address the *dynamics of resource management*, especially changes in workloads and configuration. The focus of this book is to distill and make accessible the essentials of control theory needed by computing practitioners to address these dynamics.

The dynamics of computing systems are important considerations in ensuring the profitability and availability of many businesses. For example, e-commerce sites frequently contend with workloads that change so rapidly that service degradations and failures result. Experienced designers know that leaving the management of dynamics to operators is not acceptable because many changes occur too rapidly for humans to be able to respond in a timely manner. As a result, ad hoc automation is frequently deployed with surprising results, such as wild oscillations or very slow responses to changes in workloads. Our belief is that by understanding the essential elements of control theory, computing practitioners can design systems that adapt in a more reliable manner.

A second audience for this book comprises researchers in the fields of computer science and controls. Today, very few computer science researchers have familiarity with control theory. As a result, many resource management schemes fail to address concepts that are well understood in control, such as the effect of measurement and system delays on stability and other aspects of control performance. Similarly, researchers in control fields rarely appreciate the issues particular to computing systems, such as considerations for policy-based management,

service-level agreements, and the implications of modifying computing systems to provide sensors and actuators that are appropriate for control purposes. To address this second audience, we show through numerous examples how control theoretic techniques can be applied to computer systems, and describe the many challenges that remain.

Much effort has been devoted to making this book accessible to computer scientists. First, the examples focus on computer systems and their components, such as Web servers, caching, and load balancing. Second, our approach to modeling draws heavily on insights from queueing systems and their dynamics (as opposed to Newton's laws). Third, we focus almost entirely on discrete-time systems rather than continuous-time systems (as is traditional in controls books). There are two reasons for this: (1) performance measurements of computer systems are solicited on a sampled basis, which is best described by a discrete-time model; and (2) computer scientists are quite comfortable thinking in terms of difference equations, and much less comfortable thinking in terms of differential equations.

Prerequisites

The book assumes background in series and their convergence, all of which is common in an undergraduate engineering and mathematics curriculum. Some prior exposure to Z-transforms (or Laplace transforms) is also of benefit, although not required. Also helpful is experience with developing statistical models, especially using linear regression.

Having appropriate software tools is immensely helpful in developing statistical models as well as for control analysis and design. Throughout the book, we use MATLAB®, a very powerful analysis environment that is arguably the standard for control engineers[1]. In Appendix E we provide an introduction to MATLAB (including the Control System Toolbox). However, access to MATLAB is not required for the vast majority of the book, only the optional section (indicated by *) at the end of each chapter.

Outline of the Book

The book is divided into three parts. Part I, Background, consists of one chapter introducing the control problem and giving an overview of the area. Part II, Modeling, contains six chapters and covers modeling of dynamic systems in discrete time using difference equations, Z-transforms, block diagrams, transfer functions, and transient analysis. The focus is on single-input, single-output first- and second-order systems, although Chapter 7 is devoted to multiple-input, multiple-output (MIMO) systems. Part III, Control, has four chapters. In the first chapter we describe proportional control and pole placement design. In the next chapter we consider integral and differential control as well, including PID

[1] MATLAB is a registered trademark of The MathWorks, Inc.

(proportional–integral–differential) control tuning techniques. In the third chapter we address state-space feedback control, including the application of pole placement to MIMO systems and design using linear quadratic regulators. In the last chapter we discuss a variety of advanced topics, such as adaptive control, gain scheduling, minimum-variance control, and fuzzy control. In all three parts, examples are used extensively to illustrate the problems addressed, the techniques employed, and the value provided by the techniques.

Several appendixes are provided to make the book more useful as a reference and more self-contained. Appendix A summarizes the mathematical notation used, Appendix B lists key acronyms, and Appendix C contains key results developed in the book. Anothertwo appendixes contain supplemental material. Appendix D describes results from linear algebra that are used in Chapters 7 and 10. In Appendix E we provide an overview of the facilities in `MATLAB` for doing control analysis and design along with a brief `MATLAB` tutorial.

Considerable thought was given to the choice of examples. We sought examples that both aid in communicating key concepts and provide a basis for modeling systems encountered in practice (especially based on our experience at IBM and that of colleagues elsewhere in industry and academia). Our most basic example is a single-server queueing system with exponential interarrival and service times and a finite-size buffer ($M/M/1/K$), which provides a means to study the dynamics of admission control and proportional scheduling. The e-mail example based on the IBM Lotus® Domino™ Server[2] provides insight into challenges faced in system identification. The Apache HTTP Server[3] example serves as a vehicle for studying MIMO control. Additional examples include caching with differentiated service and load balancing.

Roadmaps of the Book

The book may be approached in many ways depending on the interests of the reader. As depicted in Figure P.1, computer scientists interested in the basics of control theory should read Chapters 1 and 4 in detail. Chapters 2, 3, and 5 should be skimmed to gain insight into the nature of control system modeling, and Chapter 8 can be read in modest detail to understand the essence of control system design. Chapter 11 will also be of interest since it discusses other areas of control theory that are potentially applicable to computing systems.

Designers of computing systems who want to apply control theory in practice should proceed as shown in Figure P.2 by readying Chapters 1 through 6 and Chapters 8 and 9. State-space techniques, which are described in Chapters 7 and 10, should be approached only after there is a solid understanding of the other chapters. Considerable effort has been made to include worked examples that can be the basis for more extensive analysis and design studies. Also, all of these chapters include a section of extended examples that should stimulate ideas about the range of applications of control theory to computing systems.

[2]IBM Lotus Domino is a registered trademark of IBM Corporation.

[3]Apache is a trademark of The Apache Software Foundation and is used with permission.

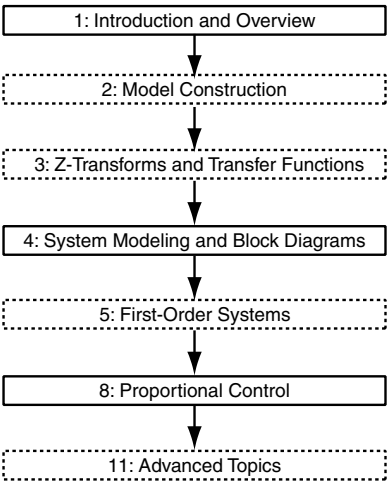

Fig. P.1 *Roadmap for computer scientists interested in the basics of control theory. Dashed boxes indicate chapters that should be skimmed.*

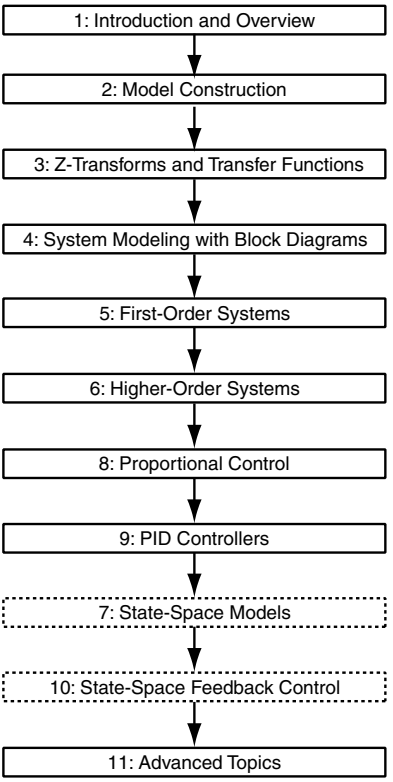

Fig. P.2 *Roadmap for computer scientists interested in applying control theory. Dashed boxes indicate chapters that should be skimmed.*

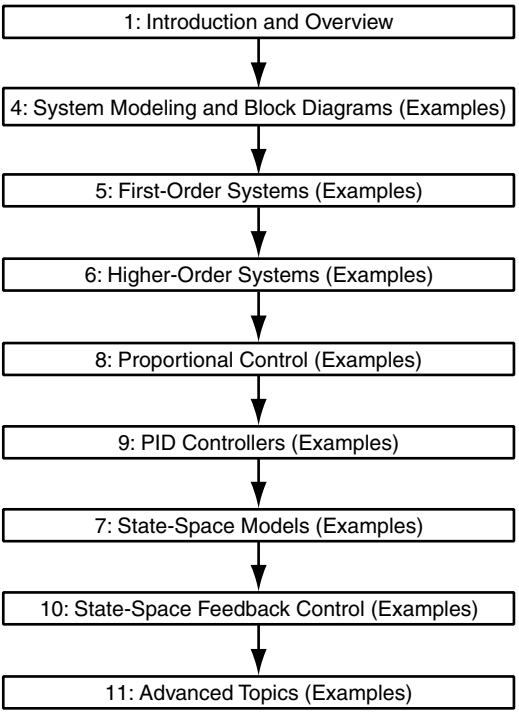

Fig. P.3 *Roadmap for control theorists interested in applications to computing systems. The focus should be on the examples, both the short in-chapter examples and the extended examples at the end of each chapter.*

Control theorists interested in computing system applications should proceed as depicted in Figure P.3. Desirable properties of controllers in computing systems and many examples of computing systems are described in Chapter 1. Chapters 4 through 11 contain a rich set of control problems based on these examples, especially the extended examples at the end of chapters.

Errata and Additions

We intend to post errata and various additions to the book on the Web site *http://www.research.ibm.com/fbcs/*. For example, several of us are currently teaching a class based on the book at Columbia University. This has resulted in a number of new ideas about how to present the material.

Acknowledgments

We wish to acknowledge the many colleagues who have helped with this book in various ways. Xichu Chen at the University of Michigan, Freeman Rawson at IBM Research in Austin, Texas, and Jose Renato Santos at Hewlett-Packard

Laboratory provided detailed comments on the text. David Patterson at the University of California–Berkeley and Armando Fox at Stanford aided us in better focusing the book for a computer science audience. Nagui Halim at IBM Research in Hawthorne, New York, gave strong support for this project from the start and provided constant enthusiasm throughout.

J.L. HELLERSTEIN
IBM Thomas J. Watson Research Center
Hawthorne, New York

Y. DIAO
IBM Thomas J. Watson Research Center
Hawthorne, New York

S. PAREKH
IBM Thomas J. Watson Research Center
Hawthorne, New York

D.M. TILBURY
Mechanical Engineering Department
University of Michigan
Ann Arbor, Michigan

Part I

Background

1

Introduction and Overview

This book is about feedback control of computing systems. The main idea of feedback control is to use measurements of a system's outputs, such as response times, throughputs, and utilizations, to achieve externally specified goals. This is done by adjusting the system control inputs, such as parameters that affect buffer sizes, scheduling policies, and concurrency levels. Since the measured outputs are used to determine the control inputs, and the inputs then affect the outputs, the architecture is called *feedback* or *closed loop*. Almost any system that is considered automatic has some element of feedback control. In this book we focus on the closed-loop control of computing systems and methods for their analysis and design.

1.1 THE NATURE OF FEEDBACK CONTROL

Feedback control is about regulating the characteristics of a system. We begin with some key concepts: the measured output, which is the characteristic to be regulated to a desired value; the control input, which is what influences the measured output; and a disturbance, which affects the way in which the input affects the output. These are illustrated in a later section.

The reader may be familiar with everyday feedback control systems, such as cruise control in an automobile, a thermostat in a house, or the human sensorimotor system. A cruise control system achieves the desired speed by adjusting the accelerator pedal based on a velocity measurement from the speedometer. Here,

Feedback Control of Computing Systems, by Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury
ISBN 0-471-26637-X

the accelerator pedal adjustments are the control input that provides a means to regulate speed, the measured output. The desired speed is maintained even when the car goes up or down hills or encounters head or tail winds, all of which are examples of disturbances that affect the relationship between the control input and the measured output. A thermostat achieves the desired temperature (output) by adjusting the furnace cycle and fan (input). The desired temperature is maintained even when the outside temperature increases or decreases (disturbance). The sensorimotor system achieves the desired hand position (output) to pick up an object by adjusting the muscle tensions (inputs) based on the current position sensed by the eyes and touch.

These concepts of feedback control apply to computing systems as well. Consider a computing system with a desired output characteristic. For example, operators of computing systems, or *administrators*, may want each of three Apache HTTP Servers [24] to run at no greater than 66% utilization, so that if any one of them fails, the other two can immediately absorb the entire load. Here, the measured output is CPU utilization. In computing systems, the measured output typically depends on the nature of the requests being served, or *workload*. Workload is often characterized in terms of the arrival process (e.g., Poisson, self-similar), and the distribution of service times for the resources used (e.g., CPU, memory, and database locks) [20]. In our studies of the Apache HTTP Server, CPU utilization depends on the workload and the control input. The workload is characterized by the request rate and whether the requests are for static or dynamic hypertext pages. The control input is the maximum number of connections that the server permits as specified by the `MaxClients` parameter. The workload is uncontrolled and so can be viewed as a disturbance. The control input `MaxClients` can be manipulated by an administrator or an automatic controller to affect CPU utilization.

Much of feedback control deals with understanding how the control input and disturbance affect the measured output. Continuing with the Apache HTTP Server example, as `MaxClients` increases, the CPU utilization increases. However, the effect is not instantaneous. A larger `MaxClients` only *allows* more users to connect; the system must wait some time for the users to arrive. Similarly, when `MaxClients` is decreased, current users are not disconnected until their sessions have timed out. Further, the value of `MaxClients` that results in a 66% utilization depends on the current workload, which may be unknown *a priori* and/or may change over time. Feedback control provides a method for setting `MaxClients` *automatically* to achieve the desired utilization that takes into account these dynamics and the effects of disturbances.

With this context we can describe feedback control in more detail. However, before doing so, a change in perspective is required. In computing systems we think in terms of the flow of work units or data through a system. Thus, input–output relationships reflect how work is done and/or data are transformed. Control theory also relies heavily on input–output relationships. However, the semantics are different. In control analysis, the inputs and outputs are metric values (e.g., CPU utilization) and/or control settings (e.g., `MaxClients`).

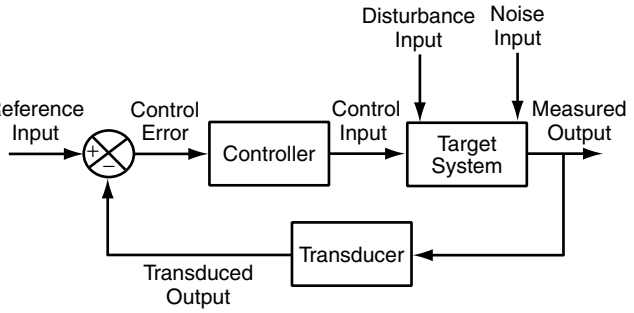

Fig. 1.1 *Block diagram of a feedback control system. The reference input is the desired value of the system's measured output. The controller adjusts the setting of control input to the target system so that its measured output is equal to the reference input. The transducer represents effects such as unit conversions and delays.*

Figure 1.1 is an example of a single-input, single-output *SISO* control system, a control system that has a single control input (i.e., `MaxClients`) and a single measured output (i.e., CPU utilization). More commonly in computing we deal with *MIMO* systems, which have multiple control inputs (e.g., settings of configuration parameters) and multiple measured outputs (e.g., response times and throughputs by service class). For pedagogical purposes, the sequel focuses on SISO systems, although MIMO considerations are addressed in passing.

The essential elements of feedback control system are depicted in Figure 1.1. These elements are:

- *Control error*, which is the difference between the reference input and the measured output.
- *Control input*, which is a parameter that affects the behavior of the target system and can be adjusted dynamically (such as the `MaxClients` parameter in the Apache HTTP Server).
- *Controller*, which determines the setting of the control input needed to achieve the reference input. The controller computes values of the control input based on current and past values of control error.
- *Disturbance input*, which is any change that affects the way in which the control input influences the measured output.
- *Measured output*, which is a measurable characteristic of the target system, such as CPU utilization and response time.
- *Noise input*, which is any effect that changes the measured output produced by the target system. This is also called *sensor noise* or *measurement noise.*
- *Reference input*, which is the desired value of the measured outputs (or transformations of them), such as CPU utilization, should be 66%. Sometimes, the reference input is referred to as *desired output* or the *setpoint*.
- *Target system*, which is the computing system to be controlled (see the examples of target systems in Section 1.6).

- *Transducer*, which transforms the measured output so that it can be compared with the reference input.

The circular flow of information in Figure 1.1 motivates our use of the term *closed-loop system* to refer to a feedback control system. We will use these terms interchangeably.

An appeal of feedback control is that administrators can achieve the desired output in a direct way by specifying the reference input instead of indirectly by manipulating the control input (an approach that is time consuming and requires considerable skill). The focus of this book is on designing feedback controllers to achieve a desired output.

The disturbance inputs are factors that affect the measured output but for which there is no governing control input. The disturbance input is depicted as a second input to the target system in Figure 1.1. An example of a disturbance input in the Apache HTTP Server is the executions of tasks such as backups and virus scans (collectively referred to as *administrative tasks*) that affect the relationship between the control input `MaxClients` and the measured output CPU utilizations and response times. One reason that feedback control is so powerful and so widely used is that it can ensure that the measured output is very close to the reference input even in the presence of disturbances.

The *transducer* transforms the measured output into the values used by the controller. An example of a transducer is a moving-average filter that smooths the stochastics of computer system measurements. Another example is a measurement sensor, especially if the sensor introduces time delays because of the manner in which measurements are collected. A third example is unit conversions, such as converting from queue lengths into response times using formulas such as Little's result [35] in systems that do not measure response times directly. Not all feedback systems contain a transducer. However, in other systems, the transducer is a complicated element that performs multiple functions.

Before closing this section, we note that systematic construction of controllers requires a model of the input–output relationships of the target system. We refer to this as the *system model*. Because of the central role that the system model plays in controller design, a significant fraction of the book is devoted to modeling techniques (especially based on linear system theory) and their application to computing systems.

1.2 CONTROL OBJECTIVES

Controllers are designed for some intended purpose. We refer to this purpose as the *control objective*. The most common objectives are:

- *Regulatory control.* Ensure that the measured output is equal to (or near) the reference input. For example, the utilization of a Web server should be maintained at 66%. The focus here is on changes in the reference input,

such as changing the target utilization from 66% to 75% if a fourth server becomes available. The term *tracking control* is used if the reference input changes frequently.

- *Disturbance rejection.* Ensure that disturbances acting on the system do not significantly affect the measured output. For example, when a backup or virus scan is run on a Web server, the overall utilization of the system is maintained at 66%. This differs from regulatory control in that we focus on changes in the disturbance input, not in the reference input.
- *Optimization.* Obtain the "best" value of the measured output. For example, in Chapter 11 we describe a fuzzy controller that adjusts `MaxClients` in the Apache HTTP Server so as to minimize response times.

Much of the book is about regulatory control with disturbance rejection. The need for regulatory control arises in three ways in computing systems. First, as already noted, regulation arises when there is a need to maintain reserve capacity (sometimes referred to as *head room*). Second, regulatory control is used for a kind of constrained optimization, such as "maximize throughput subject to response time being no greater than 1 second." A common heuristic for such an objective is to accept as many requests as possible without exceeding the response-time constraint (e.g., regulate response time to be 1 second). Third, regulation is important in the enforcement of service-level agreements. Disturbance rejection addresses the fact that the foregoing must be done in the presence of time-varying loads and changes in hardware and software configurations.

To elaborate on the last point, *service-level agreements* (or *SLAs*) are a contract between a service provider and its customers. Such agreements consist of one or more *service-level objectives* (*SLOs*). Examples of service providers include Internet service providers, application service providers, and internal IT organizations. An example of an SLO is: "Gold customer response times should be no greater than 2 seconds." There are three parts to an SLO: the metric (e.g., response time), the bound (e.g., 2 seconds), and a relational operator (e.g., less than). Intuitively, service providers want to have sufficient resources to meet their SLOs. But they do not want to have more resources than required since doing so imposes unnecessary costs. As a result, SLO enforcement often becomes a regulation problem. In terms of the architecture in Figure 1.1, the SLO metric is the measured output, and the SLO bound is the reference input.

The choice of control objective typically depends on the application. Indeed, with multiuse target systems, the same target system may have multiple controllers with different SLOs.

1.3 PROPERTIES OF FEEDBACK CONTROL SYSTEMS

There are several properties of feedback control systems that should be considered when comparing controllers for computing systems. Our choice of metrics is drawn from experience with the commercial information technology systems.

Other properties may be of interest in different settings. For example, [43] discuss properties of interest for control of real-time systems.

Below, we motivate and present the main ideas of the properties considered. More formal definitions are given later in the book.

- A system is said to be *stable* if for any bounded input, the output is also bounded. (We discuss stability in detail in Chapter 3 .) Stability is typically the first property considered in designing control systems since unstable systems cannot be used for mission-critical work.
- The control system is *accurate* if the measured output converges (or becomes sufficiently close) to the reference input. Accurate systems are essential to ensuring that control objectives are met, such as differentiating between gold and silver classes of service and ensuring that throughput is maximized without exceeding response-time constraints. Typically, we do not quantify accuracy. Rather, we measure inaccuracy. For a system in steady state, its inaccuracy, or *steady-state error*, is the steady-state value of the control error.
- The system has *short settling times* if it converges quickly to its steady-state value. Short settling times are particularly important for disturbance rejection in the presence of time-varying workloads so that convergence is obtained before the workload changes.
- The system should achieve its objectives in a manner that *does not overshoot*. Consider a system in which the objective is to maximize throughput subject to the constraint that response time is less than 1 second, which is often achieved by a regulator that keeps response times at their upper limit so that throughput is maximized. Suppose that incoming requests change so that they are less CPU intensive and hence response times decrease to 0.5 second. Then, by avoiding overshoot, we mean that as the controller changes the control input that causes throughput to increase (and hence response time to increase), response times should not exceed 1 second.

Much of the focus of the book is on these *SASO* properties: stability, accuracy, settling time, and overshoot.

We begin with what constitutes a stable system. For computing systems we want the output of feedback control to converge, although it may not be constant due to the stochastic nature of the system. To refine this further, computing systems have operating regions (i.e., combinations of workloads and configuration settings) in which they perform acceptably and other operating regions in which they do not. Thus, in general, we refer to the stability of a system within an operating region. Clearly, if a system is not stable, its utility is severely limited. In particular, the system's response times will be large and highly variable, a situation that can make the system unusable.

Figure 1.2 displays an instability in the Apache HTTP Server that employs an improperly designed controller. The horizontal axis is time, and the vertical axis is CPU utilization (which ranges between 0 and 1). The solid line is the