

# LQG-Based Real-Time Scheduling and Control Codesign

Xu, Yang

2017

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

# LQG-Based Real-Time Scheduling and Control Codesign

Yang Xu


LUND
UNIVERSITY

Department of Automatic Control

PhD Thesis TFRT-1119
ISBN 978-91-7753-515-7 (print)
ISBN 978-91-7753-516-4 (web)
ISSN 0280–5316

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


Printed in Sweden by Media-Tryck.
Lund 2017

# Abstract

Having multiple control tasks concurrently running on a single computing platform increases the processor utilization but degrades the control performance due to delay and jitter in the loops. In scheduling and control codesign, the objective is to optimize the combined performance of all the controllers, subject to a schedulability constraint. The codesign procedure consists of selecting task parameters, e.g., periods and priorities, as well as designing the controllers so that the scheduling-induced delay and jitter are taken into account.

In the thesis, four linear-quadratic-Gaussian (LQG) codesign methods are proposed: stochastic, periodic, harmonic, and robust LQG codesign. In stochastic LQG codesign, the delay distributions are calculated at design-time. Then LQG controllers are designed assuming these delay distributions. The obtained task periods generally give rise to infinite hyperperiods. This can be avoided by perturbing the periods slightly in order to obtain a finite hyperperiod, yielding a periodic delay pattern for the control loops. The periodicity is then accounted for by using periodic LQG control design, resulting in a periodic sequence of feedback gains for each controller. In harmonic LQG codesign, again the task periods are perturbed, but this time to make the periods harmonic. The scheduling-induced delays will be constant and standard LQG design can be applied. Finally, a robust LQG codesign method is presented. The design is based on convex optimization and guarantees system robustness in the presence of delay and jitter. A new rule of thumb for initial sampling period assignment is proposed. We propose a jitter-aware priority and period assignment codesign method to optimize the overall system performance.

A large evaluation of the proposed four codesign methods is performed using the Jitterbug toolbox. All of the four methods lead to improved control performance compared to earlier work. The harmonic scheduling and control codesign shows the largest overall improvements.

# Acknowledgments

I am forever grateful and thankful to my supervisors Prof. Anton Cervin and Prof. Karl-Erik Årzén. I am lucky to have the best supervisors not only academically, but also personally. They spend a lot of time on providing me the general directions and technical details in my study, which improve the quality of both the research and the thesis. Their guidance and support are invaluable in both my professional and personal development.

I would like to thank Prof. Zebo Peng, Prof. Petru Eles, Amir Aminifar, and Rouhollah Mahfouzi at Linköping University for their advice and discussion.

Next, I would like to thank Prof. Enrico Bini at the University of Turin for his guidance and advice. I also would like to thank my colleagues around the world. In particular, I thank Mitra Nasri, Morteza Mohaqeqi, and Bogdan Tanasa. Their insights and suggestions help me to solve some difficult problems in my work.

My work is helped by working with all the research engineers at the Department, especially Anders Nilsson and Leif Andersson.

I also would like to thank all the administrators at the Department for their help. They are Eva Westin, Mika Nishimura, Monika Rasmusson, Cecilia Edelborg Christensen, and Ingrid Nilsson.

I would like to thank my office mates at the Department. They are Philip Reuterswärd, Wilbert Samuel Rossi, Gustav Nilsson, and Manfred Dellkrantz. I have the pleasure to work with them over the last several years. Thanks for providing me such a good working environment.

I also want to thank Sei Zhen Khong and Michelle Chong. I will never forget the good times with them.

I also want to thank Prof. Zonghua Gu at Zhejiang University. He seems to have read all papers about real-time systems and always knows where I should find the relevant literature. I also want to thank a number of my friends and colleagues: Li Zhu, Yuling Li, and Hong Wang.

Thanks to all the colleagues at the Department of Automatic Control. All of you make our department the greatest place to study and work.

Finally, I would like to thank my parents. They are the source of my courage, even though they are thousands of miles away from me. They support any decision

I made, and never ask for feedback. Most of all, I deeply thank Yi. Thanks for her understanding and support.

## Financial Support

Research funding has been provided by the ELLIIT projects “Integrated scheduling and synthesis of networked embedded event-based control systems” and “Co-design of robust and secure networked embedded systems”. The author is a member of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University.

# Contents

# Nomenclature

| Notation | Description |
|---|---|
| $A, B, C$ | Plant state-space matrices |
| $C^b$ | Best-case execution time |
| $C^w$ | Worst-case execution time |
| $D$ | Relative deadline |
| $\delta$ | Delay |
| $\Phi, \Gamma$ | Sampled plant state-space matrices |
| $G(s)$ | General continuous-time transfer function |
| $h$ | Controller sampling period |
| $H$ | Hyperperiod |
| $H(z)$ | General discrete-time transfer function |
| $J$ | Jitter |
| $J_m$ | Jitter margin |
| $K$ | Kalman filter gain |
| $K(z)$ | Discrete-time controller transfer function |
| $L$ | State feedback gain |
| $M_s$ | Maximum sensitivity |
| $n$ | Number of tasks |
| $O$ | Task offset |
| $P(s)$ | Continuous-time plant transfer function |
| $Q_1, Q_2, Q_{12}$ | LQG cost function matrices |
| $R_1, R_2$ | LQG noise intensity matrices |
| $R^b$ | Best-case response time |
| $R^w$ | Worst-case response time |
| $T$ | Task period |
| $\tau$ | Task |
| tr | Trace |
| $u$ | Control signal |
| $U$ | Utilization |

| Notation | Description |
|---|---|
| $v_1$, $v_2$ | Process and measurement noise |
| $w$ | Disturbance input |
| $V$ | Cost function |
| $x$ | Plant state vector |
| $y$ | Measurement signal |
| $z$ | Performance output |

| Abbreviation | Description |
|---|---|
| CPU | Central processing unit |
| EDF | Earliest deadline first |
| FP | Fixed priority |
| LQ | Linear-quadratic |
| LQG | Linear-quadratic-Gaussian |
| MPC | Model predictive control |
| PID | Proportional–integral–derivative |
| RM | Rate monotonic |
| RTOS | Real-time operating system |
| ZOH | Zero-order hold |

# 1

# Introduction

## 1.1 Motivation

The demand for efficient resource utilization in embedded and cyber-physical systems has led to an increased focus on codesign approaches, e.g., [Årzén et al., 2000], [Årzén et al., 2003], [Derler et al., 2013]. Scheduling and control codesign is a fundamental problem that emerges from the combination of real-time computing and feedback control.

For cost-saving reasons, it is not uncommon to let multiple feedback controllers share the same computational platform. The controllers are often implemented as periodic tasks in an RTOS with preemptive fixed-priority scheduling. When the controllers are scheduled on the central processing unit (CPU), preemption from higher-priority tasks will impede the execution of the lower-priority tasks. From a control perspective, there will be extra delay as well as jitter (variable delay) in the corresponding control loops. The delay and jitter, in turn, degrade the closed-loop control performance. Delays can also come from real plant delays (e.g., transportation delays) or from network communication on a distributed platform. The total control loop delay due to implementation consists of the time it takes to sample the plant output, compute the control signal (including preemption and other scheduling effects), actuate the control signal, and transmit the message between the relevant network nodes (including waiting times in the network). In this thesis, we focus on a single CPU as the only shared resource. However, many of the ideas should also be valid for codesign problems involving control over communication networks or multicore systems.

Control performance can be measured in many different ways. In the frequency domain, control performance metrics include the magnitude of the resonance peak, the loop gain crossover frequency, and the closed-loop bandwidth. In the time domain, common control performance metrics are the maximum overshoot, rise time, settling time, steady-state error, or, as in this thesis, a quadratic function of the state variables and the control signal. The most common controller type in industry is the proportional–integral–derivative (PID) controller. It is often tuned heuristically using experiments and tuning rules, although analytical approaches also exist. In this

thesis, we focus entirely on the linear-quadratic-Gaussian (LQG) controller, which represents an optimal linear controller for a given linear plant. The plant is assumed to be disturbed by white noise, and control performance is measured in terms of a quadratic cost function that penalizes deviations in the plant states and the control signal from zero, which is assumed to be the desired steady state. Compared to PID, LQG control is applicable to a wider class of plants, including higher-order, unstable, and oscillatory plants. Moreover, the optimal controller is obtained analytically by solving a couple of Riccati equations. On the downside, LQG control design requires an accurate model of the plant to be controlled, and the resulting controller will be of the same order as the plant model it is based on. Also, an LQG controller is normally not designed taking robustness explicitly into account.

While LQG control is not nearly as common as PID, the basic idea of optimizing a cost function to obtain the best control signal is also employed in model-predictive control (MPC), which is gaining popularity in the process industry [Camacho and Alba, 2013]. MPC commonly employs a quadratic cost function just like LQG but also adds the possibility to specify constraints and limitations on plant states and control signals. Being a nonlinear control strategy, on-line optimization (or table lookup) is needed to obtain a solution at each time step. Also in contrast to LQG, a finite time horizon is typically assumed in the optimization.

Focusing on LQG control allow us to use the same performance criterion—a quadratic cost function—for both controller design and control performance evaluation in real-time control systems. The LQG theory also is readily extended to various cases with delay and jitter in the control loop, and under certain assumptions the resulting control performance can also be evaluated analytically.

Delay and jitter generally have a negative impact on control performance. A typical case is illustrated in Figure 1.1, which shows the LQG cost (i.e., the value of the quadratic cost function) for an inverted pendulum being controlled under delay and jitter. Without going into details on this particular example, it is seen that the cost increases monotonically with both the delay and the jitter, and the increase seems to be near-linear for small delay and jitter values. Depending on the particular plant dynamics and the sampling period, the shape of the cost function can, however, be quite complex [Eker et al., 2000]. Hence, the need to approximate the cost function arises, in particular if some optimization-based codesign approach is to be used. There are different ways to approximate these cost functions, e.g., as affine or quadratic functions [Cervin et al., 2002a].

To simplify the LQG design problems in this thesis as far as possible, we will throughout assume that sampling (i.e., reading of the plant output value) occurs when the control task is released, and actuation (i.e., updating of the plant control signal) happens when the task finishes. Hence, there is only output jitter and no sampling jitter. There are several alternatives to this approach. If the sampling instead takes place at the task starting time, the control delay will on average be shorter, but the resulting sampling periods are not equidistant. If the actuation time instead is selected as the next sampling instant then the delay is constant but it will also

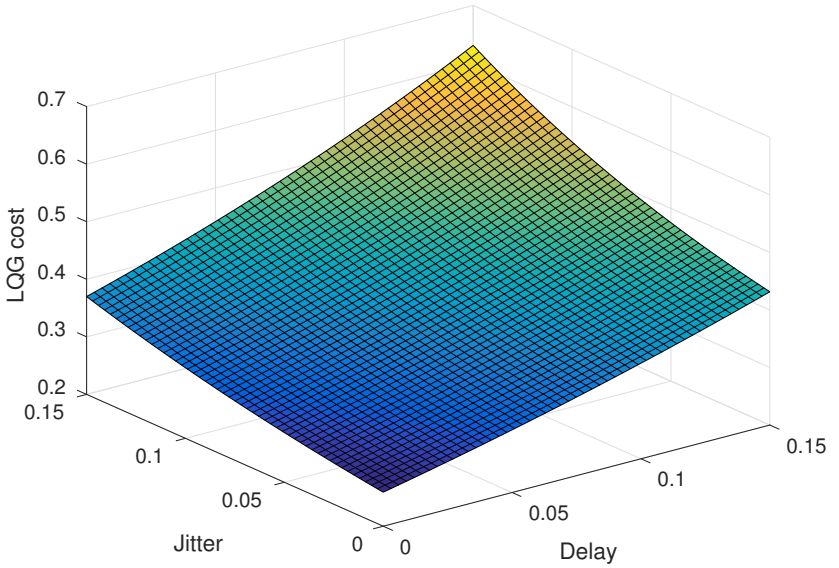



**Figure 1.1** LQG cost as a function of delay and jitter for an inverted pendulum. The sampling period is 0.3.

be longer on average. There are also task-splitting techniques that could be used to schedule the sampling, control signal calculation, actuation, and controller state update as separate (sub)tasks [Cervin, 1999], [Balbastre et al., 2000]. Furthermore, virtualization or isolation techniques could be used to temporally isolate the tasks, hence reducing scheduling-induced jitter. Such techniques could be combined with the proposed LQG design methods to yield even better performance, but that is outside the scope of this work.

In this thesis, we propose a number of different ways to deal with scheduling-induced delay and jitter in multitasking real-time control systems. The first method is named *stochastic LQG codesign*. The core idea is to measure or analyze the delays, treat them as independent random variables, and design LQG controllers that take these delay distributions into account.

In the *periodic LQG codesign* method, the controller periods are tweaked so that a finite hyperperiod is obtained. This means that the delay will vary according to a periodic pattern, which then can be utilized to design a time-varying but periodic LQG controller.

In the *harmonic LQG codesign*, the periods are again tweaked but now so that the periods become harmonic. This then (under certain assumptions) leads to con-

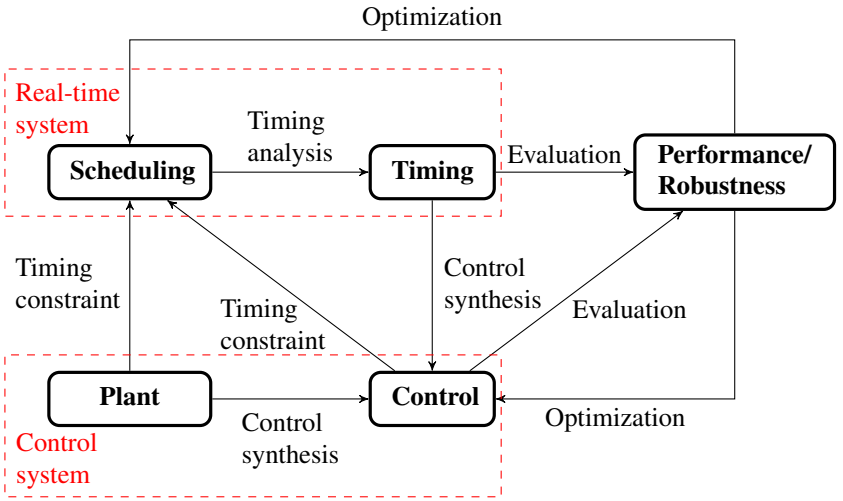



**Figure 1.2** Framework for the codesign problem.

stant control delays, which are straightforward to take into account in the LQG design.

LQG is an optimal control policy, but not necessarily robust, so the above approaches do not give any guarantees on what will happen in the worst-case scheduling scenario. The *jitter margin* describes how much jitter can be tolerated before the feedback loop becomes unstable, and this is used as a robustness measure. In the *robust LQG codesign* method, we design an LQG-optimal controller with a jitter margin constraint using the Youla parameterization and convex optimization and use this as the basis for codesign.

## 1.2 Problem Formulation

The goal of the scheduling and control codesign is to optimize the overall control performance, subject to a real-time system schedulability constraint. In mathematical terms, this is formulated as an optimization problem, where the objective is to minimize the overall cost of the control loops. The variables that we can optimize over are the task priorities, the sampling periods, and the controllers themselves.

The framework of the codesign problem is shown in Figure 1.2. The controller is designed according to the characteristics of the plant. Both the plant and the controller impose certain timing constraints on the scheduling. For example, the task period selection depends on the speed of the plant, and the execution time depends on the time complexity of the control algorithm. The scheduling affects the timing of the real-time system. For a given plant, the control performance and

robustness depend on the timing and the controller. The aim of the codesign is to optimize the control performance or robustness by simultaneously tuning the scheduling parameters and designing the controllers.

In order to achieve optimal control performance for each task, the sampling period, the delay and the jitter should be small, but this requirement cannot be satisfied simultaneously for every task due to the utilization constraint in the real-time system. Furthermore, the relation between the scheduling parameters and the timing parameters, as well as the relation between the timing parameters and the overall control performance, are quite complicated. On the other hand, the overall control performance also depends on the control strategy applied.

In this thesis, the cost function of the optimization problem is the sum of the LQG costs of all the tasks. The inequality constraint is that the sum of utilizations of all the tasks is less than or equal to a utilization bound, $U_b$, where $U_b \leq 1$. The optimization can be done by selecting task parameters, i.e, periods and priorities, and designing the controllers so that they take the timing effects caused by the scheduling into account. The optimization problem is generically formulated as

$$\begin{aligned} &\text{minimize } \sum_i V_i \\ &\text{subject to } \sum_i U_i \leq U_b, \end{aligned} \tag{1.1}$$

where $V_i$ and $U_i$ are the LQG cost and the utilization of task $\tau_i$, respectively.

## Summary of Assumptions and Limitations

As already mentioned above, in the thesis several modeling assumptions and limitations exist. We here summarize the most important limitations. References to related work where some of the limitations are removed are given in Chapter 3.

***Fixed-priority scheduling.*** While fixed-priority scheduling is the most common scheduling policy in embedded systems, there are other scheduling policies with attractive properties. For example, earliest-deadline-first (EDF) scheduling can often achieve higher utilization. EDF is however not considered in the thesis due to the more complicated response-time analysis.

***Single-CPU systems.*** In multiprocessor systems, there is more computational capacity and parallelism available, and hence the potential for better performance. The codesign problem is, however, complex already with a single processor. However, both the periodic LQG and the harmonic LQG methods can be extended to the multiprocessor case, at least if one assumes partitioned scheduling. For global scheduling, it becomes more difficult.

***Independent tasks.*** Only independent control tasks are considered, and it is assumed that each controller is designed independently of the others. Also, for simplicity, no other tasks are assumed to occupy the CPU.

***Only output jitter.*** Sampling jitter is assumed to be eliminated by some hardware or software solution that does not introduce any overhead. The output jitter is due to the execution of the task and interference from higher-priority tasks,

***LQG control.*** Only linear plants with Gaussian noise and quadratic cost functions are considered. Other controller types, nonlinear systems or alternative cost criteria are outside the scope of this work. The only exception is the robustness constraint introduced in Chapter 7, which leads to a non-standard LQG design problem.

***CPU as the only shared resource.*** Resources besides CPU time, such as network bandwidth, memory, or access to shared external devices, are not considered in the thesis.

## 1.3 Contributions and Publications

In this section, the contributions and publications by the author are listed. The thesis is based on the following papers:

***Chapter 4***

Xu, Y., K.-E. Årzén, E. Bini, and A. Cervin (2014). "Response time driven design of control systems". In: *Proceedings of the 19th IFAC World Congress*. Cape Town, South Africa, pp. 6098–6104.

A correct control design must account for the schedule of the control task on the processor. Existing design techniques are based on the assumption that there is no delay, or that the delay is constant over time. However, in practice, almost all controllers have time-varying delay, hence invalidating this assumption. In this paper, we introduce stochastic LQG codesign, in which the controller delay is modeled by the distribution of the task response time. We show, via simulation, that our method can reduce the control cost compared to the state-of-art methods.

Y. Xu formulated the method for assigning optimal sampling period for real-time control tasks under fixed priority scheduling. Further, Y. Xu implemented the codesign method and evaluated it in MATLAB simulations. K.-E. Årzén, E. Bini, and A. Cervin suggested the topic, contributed with feedback on the research and wrote parts of the manuscript. Furthermore, E. Bini provided Figure 4.2.

***Chapter 5***

Xu, Y., K.-E. Årzén, A. Cervin, E. Bini, and B. Tanasa (2015). "Exploiting job response-time information in the co-design of real-time control systems". In: *Proceedings of the 21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Hong Kong, China, pp. 247–256.

In this paper, we exploit the periodicity of the task response time, which corresponds to a periodic delay pattern in the feedback control loop. Perturbed periods are used as a tool to find a finite hyperperiod. We present an analytical procedure to design a periodic LQG controller for tasks with fixed execution times as well as a numerical solution to the periodic–stochastic LQG problem for tasks with variable execution times.

Y. Xu, K.-E. Årzén, and A. Cervin proposed the periodic LQG and periodic–stochastic LQG design methods, and Y. Xu implemented and evaluated them in MATLAB simulations. K.-E. Årzén, and A. Cervin supervised the research and assisted in writing the manuscript. E. Bini proposed the sampling period perturbation method and wrote the corresponding part in the paper. B. Tanasa provided a method to calculate response-time probability distribution and also provided Figure 5.3.

***Chapter 6***

Xu, Y., A. Cervin, and K.-E. Årzén (2016a). "Harmonic scheduling and control co-design". In: *Proceedings of the 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Daegu, South Korea, pp. 182–187.

Xu, Y., A. Cervin, and K.-E. Årzén (2016b). "LQG-based scheduling and control co-design using harmonic task periods". Technical Report TFRT-7646. Department of Automatic Control, Lund University.

The technical report is an extended version of the RTCSA conference paper. In this work, we rely on the fact that harmonic task scheduling has many attractive properties, including a utilization bound of 100% under rate-monotonic scheduling, and reduced jitter. At the same time, it places severe constraints on the task period assignment for the applications. We explore the use of harmonic task scheduling for applications with multiple feedback control tasks and present two algorithms for finding harmonic task periods: one that minimizes the distance from an initial set of non-harmonic periods and one that finds all feasible harmonic periods within a given set of ranges. We apply the algorithms in a scheduling and control co-design procedure, where the goal is to optimize the total performance of a number of control tasks that share a common computing platform.

Y. Xu developed the two methods of harmonic period assignment, implemented them, and performed the simulations. A. Cervin and K.-E. Årzén suggested the research topic, supervised the research and wrote parts of the manuscript.

***Chapter 7***

Xu, Y., A. Cervin, and K.-E. Årzén (2017). "Jitter-robust LQG control and real-time scheduling co-design". In submission to the 2018 American Control Conference.

Standard LQG control design does not give any guarantees on robustness, while robust control design methods often do not handle controller timing uncertainty.

In this paper, we propose a sampled-data controller synthesis method that minimizes an LQG cost function subject to a jitter margin constraint. By robustifying the LQG controller we are able to retain good stability margins under delay and jitter, while only paying a small price in terms of nominal performance. We also present a codesign procedure that assigns optimal priorities and sampling periods to a set of controllers based on their performance characteristics and jitter sensitivity.

The jitter-robust LQG design method was developed by A. Cervin and Y. Xu. The codesign procedure was proposed and evaluated in simulations by Y. Xu. A. Cervin and K.-E. Årzén suggested the research topic, supervised the research and wrote parts of the manuscript.

### Additional Publications

The following publications are related to, but not included in, the thesis:

Mohaqeqi, M., M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén (2016). "On the problem of finding optimal harmonic periods". In: *Proceedings of the 24th Conference on Real-Time and Network Systems (RTNS)*. Brest, France, pp. 171–180.

Mohaqeqi, M., M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén (2017). "Optimal harmonic period assignment: complexity results and approximation algorithms". In submission to *Real-Time Systems*.

The papers above analyze the complexity of harmonic task scheduling and derive bounds on the overall cost degradation when harmonizing a non-harmonic task set.

Xu, Y., K.-E. Årzén, E. Bini, and A. Cervin (2017). "LQG-based control and scheduling co-design". In: *Proceedings of the 20th IFAC World Congress*. Toulouse, France.

This paper presents a summary of the different codesign methods in the thesis, together with an evaluation on randomized examples. It is superseded by the evaluations in Chapter 8.

## 1.4 Outline of Thesis

This thesis begins with a general description of LQG design, real-time system scheduling, and the MATLAB toolboxes Jitterbug and TrueTime in Chapter 2. Following an explanation of the background, Chapter 3 presents the related work on real-time system scheduling, control systems, and their codesign.

*Stochastic LQG codesign* is developed in Chapter 4. The delay distributions are obtained at design-time using schedule simulation and then stochastic LQG control design is employed, i.e., the LQG controllers are designed assuming these delay distributions. The task periods obtained generally give rise to infinite hyperperiods.

*Periodic LQG codesign* is presented in Chapter 5. The idea is to perturb the periods slightly in order to obtain a finite hyperperiod, which will correspond to a periodic delay pattern for the control loops. This periodicity is then explicitly accounted for by using periodic LQG control design, resulting in a periodic sequence of feedback gains for each controller.

*Harmonic LQG codesign* is presented in Chapter 6. Again the task periods are perturbed but this time to make the periods harmonic. The scheduling-induced delays will be constant and again ordinary LQG design with constant delay can be applied.

*Robust LQG codesign* is proposed in Chapter 7. A new rule of thumb for initial sampling period assignment is proposed, and the robust LQG control design problem and its solution are described. Then we give a jitter-aware priority and period assignment codesign method to optimize the overall system performance.

Extensive evaluations are presented in Chapter 8. Finally, Chapter 9 presents a summary of the thesis, and future work is discussed.

# 2

# Background

## 2.1 Control Design

Feedback controllers are ubiquitous in engineering systems. They can be tuned experimentally or designed based on a mathematical model of the plant. Control performance can be evaluated using, for instance, a step response test, where the rise time, the maximum overshoot, the settling time, and the steady-state error are measured. Another approach is to define a cost function for the control loop and evaluate its value for certain disturbances or setpoint changes. In this thesis, we use the LQG cost, which corresponds to the stationary weighted covariance of the plant state and the control signal when white noise disturbances are acting on the system.

In control system implementation, there are several factors that affect the control performance, e.g., the controller type, the controller parameters (alternatively the design parameters), the sampling period, whether some information about the delay and jitter is available at design time so that the controller can be designed for this, whether some information about the delay and jitter is available at run-time so that the controller can online compensate for this, the plant dynamic, the noise acting on the closed-loop system, etc. In this thesis, the focus is how delays and jitter in the delays influence the LQG control performance. A major source of the delays and jitter is the scheduling-induced interference among the tasks which implements the controllers. One approach to minimize this performance degradation is to study control and scheduling codesign, i.e., how we should design the controllers so that they can take the scheduling-induced timing variations into account and how we can select the scheduling parameters to minimize these effects.

### LQG Problem Formulation

The problem formulation and the solution of the LQG problem for sampled-data systems are given in [Åström and Wittenmark, 2013]. The plant is given by

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + v_{1c}(t) \\ y(t) &= Cx(t) + v_{2c}(t) \end{aligned} \tag{2.1}$$

where $v_{c1}$ and $v_{c2}$ are uncorrelated zero-mean continuous-time white-noise processes with intensities $R_{1c}$ and $R_{2c}$, respectively. $(A, B)$ is assumed controllable and $(A, C)$ is assumed observable. The goal is to design a LQG controller that minimizes the expected cost

$$\begin{aligned} V &= \lim_{T\to\infty} \frac{1}{T} \mathrm{E} \int_0^T \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}^T Q_c \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} dt \\ &= \lim_{T\to\infty} \frac{1}{T} \mathrm{E} \int_0^T \left( x^T(t) Q_{1c} x(t) + 2x^T(t) Q_{12c} u(t) + u^T(t) Q_{2c} u(t) \right) dt. \end{aligned} \tag{2.2}$$

The cost matrix $Q_c$ is chosen as a weighted sum of the states and the control signal. A large weight gives a small variation in the corresponding state or control signal. The noise represented by $R_{1c}$ and $R_{2c}$ are unrelated. $R_{2c}$ represents the measurement noise and can be found in specifications of the sensors or experimentally. A large $Q_{1c}$ or a small $Q_{2c}$ lead to a fast state feedback. Similarly, a large $R_{1c}$ or a small $R_{2c}$ lead to a fast Kalman filter. In the thesis, the choice of these four matrices affects the control performance and the performance improvement, so the matrices are taken into account in the control design and the optimization of the real-time system scheduling.

## Sampling of the LQG problem

To design a discrete-time LQG controller that can be implemented as a task in a real-time system, we need to sample the LQG problem formulation. First we zero-order-hold (ZOH) sample the continuous-time LTI system in (2.1) with noise. For details, see [Åström, 1970]. Suppose that the control signal $u(t)$ is constant over the interval $[t_0, t_f]$ and that the initial state $x(t_0)$ is known. The state at time $t_f$ is then given by

$$x\left(t_f\right) = \Phi\left(t_f, t_0\right) x\left(t_0\right) + \Gamma\left(t_f, t_0\right) u\left(t_0\right) + v_1(t_f, t_0) \tag{2.3}$$

where

$$\begin{aligned} \Phi\left(t_f, t_0\right) &= e^{A\left(t_f - t_0\right)} \\ \Gamma\left(t_f, t_0\right) &= \int_0^{t_f - t_0} e^{As} B \, ds \end{aligned} \tag{2.4}$$

and $v_1(t_f, t_0)$ is a mean-zero Gaussian random variable with variance

$$R_1(t_f, t_0) = \int_0^{t_f - t_0} e^{As} R_{1c} e^{A^T s} ds. \tag{2.5}$$

Next we sample the running cost (cf. (2.2))

$$V\left(t_f, t_0\right) = \int_{t_0}^{t_f} \left( x^T(t) Q_{1c} x(t) + 2x^T(t) Q_{12c} u(t) + u^T(t) Q_{2c} u(t) \right) dt \tag{2.6}$$

over the same time interval, yielding

$$V\left(t_f,t_0\right)=x^T\left(t_0\right)Q_1\left(t_f,t_0\right)x^T\left(t_0\right)+2x^T\left(t_0\right)Q_{12}\left(t_f,t_0\right)u^T\left(t_0\right) + u^T\left(t_0\right)Q_2\left(t_f,t_0\right)u^T\left(t_0\right)+V_{const}(t_f,t_0), \quad (2.7)$$

where

$$\begin{aligned} Q_1\left(t_f,t_0\right) &= \int_{t_0}^{t_f}\Phi^T\left(s,t_0\right)Q_{1c}\Phi\left(s,t_0\right)ds \\ Q_{12}\left(t_f,t_0\right) &= \int_{t_0}^{t_f}\Phi^T\left(s,t_0\right)\left(Q_{1c}\Gamma\left(s,t_0\right)+Q_{12c}\right)ds \\ Q_2\left(t_f,t_0\right) &= \int_{t_0}^{t_f}\left(\Gamma^T\left(s,t_0\right)Q_{1c}\Gamma\left(s,t_0\right)+2\Gamma\left(s,t_0\right)Q_{12c}+Q_{2c}\right)ds \\ V_{const}(t_f,t_0) &= \mathrm{tr}\left(Q_{1c}\int_0^{t_f-t_0}\left(\int_0^s e^{As'}R_{1c}e^{A^Ts'}ds'\right)ds\right). \end{aligned} \quad (2.8)$$

Here, tr denotes matrix trace. The term $V_{const}$ represents the cost due to intersample noise and does not depend on the initial state or the control signal.

## Sampling of a System with Constant Delay

A basic case that we will consider in the thesis is sampled-data LQG control of a system with a constant input delay. We denote the sampling period by $h$ and the constant delay by $\delta$. For simplicity we here assume that $0\leq\delta\leq h$, but the method is easily extended to longer delays. Dividing each sampling interval $[kh,kh+h]$, into two parts, $[kh,kh+\delta]$ and $[kh+\delta,kh+h]$, and applying (2.3)–(2.4) yield the sampled state-space model

$$x[k+1]=\Phi x[k]+\Gamma_1u[k-1]+\Gamma_0u[k]+v_1[k], \quad (2.9)$$

where

$$\begin{aligned} \Phi &= \Phi(h,0) \\ \Gamma_1 &= \Phi(h-\delta,0)\Gamma(\delta,0) \\ \Gamma_0 &= \Gamma(h-\delta,0), \end{aligned} \quad (2.10)$$

and where $v_1$ is discrete-time Gaussian white noise with variance $R_1=R_1(h)$. It is seen that the previous control signal, $u[k-1]$, affects the new state. We introduce an augmented state $x_e$ in the discrete-time model that contains the original state and the previous control signal, namely,

$$x_e[k]=\begin{bmatrix}x[k]\\u[k-1]\end{bmatrix}. \quad (2.11)$$

This gives the extended state-space model

$$\begin{aligned} x_e[k+1] &= \Phi_ex_e[k]+\Gamma_eu[k]+\Gamma_e'v_1[k] \\ y[k] &= C_ex_e[k]+v_2[k], \end{aligned} \quad (2.12)$$

with extended matrices

$$\begin{aligned}\Phi_e &= \begin{bmatrix}\Phi & \Gamma_1\\ 0 & 0\end{bmatrix}\\ \Gamma_e &= \begin{bmatrix}\Gamma_0\\ I\end{bmatrix}\\ \Gamma_e' &= \begin{bmatrix}I\\ 0\end{bmatrix}\\ C_e &= \begin{bmatrix}C & 0\end{bmatrix}.\end{aligned} \tag{2.13}$$

Next we sample the cost function when there is an input delay. The running cost from time $kh$ to $kh+h$ is given by

$$V((k+1)h, kh) = \int_{kh}^{(k+1)h} \left(x^T(t)Q_{1c}x(t) + 2x^T(t)Q_{12c}u(t) + u^T(t)Q_{2c}u(t)\right)dt. \tag{2.14}$$

The cost function (2.14) is sampled as

$$V[k] = x_e^T[k]Q_{1e}x_e[k] + 2x_e^T[k]Q_{12e}u[k] + u^T[k]Q_{2e}u[k] + V_{const} \tag{2.15}$$

where

$$\begin{aligned}Q_{1e} &= \begin{bmatrix}Q_1(\delta,0) & Q_{12}(\delta,0)\\ Q_{12}^T(\delta,0) & Q_2(\delta,0)\end{bmatrix}\\ &\quad + \begin{bmatrix}\Phi(\delta,0)\\ \Gamma(\delta,0)\end{bmatrix}^T \begin{bmatrix}Q_1(h-\delta,0) & Q_{12}(h-\delta,0)\\ Q_{12}^T(h-\delta,0) & Q_2(h-\delta,0)\end{bmatrix}\begin{bmatrix}\Phi(\delta,0)\\ \Gamma(\delta,0)\end{bmatrix}\\ Q_{12e} &= \begin{bmatrix}\Phi(\delta,0)\\ \Gamma(\delta,0)\end{bmatrix}^T Q_{12}(h-\delta,0)\\ Q_{2e} &= Q_2(h-\delta,0)\\ V_{const} &= V_{const}(h).\end{aligned} \tag{2.16}$$

With the sampled state-space equations and cost function of the time-delay system, one can use standard discrete-time LQG design to find the optimal controller.

## Optimal State Feedback

For the LQG cost in (2.2), assuming that the augmented state of the plant $x_e$ is available for feedback, the optimal control strategy is

$$u[k] = -Lx_e[k] \tag{2.17}$$

where

$$L = \left(\Gamma_e^T S\Gamma_e + Q_{2e}\right)^{-1}\left(\Gamma_e^T S\Phi_e + Q_{12e}\right) \tag{2.18}$$

and $S$ is the solution to the algebraic Riccati equation

$$S = \Phi_e^T S\Phi + Q_{1e} - \left(\Phi^T S\Gamma + Q_{12e}\right)\left(\Gamma_e^T S\Gamma_e + Q_{2e}\right)^{-1}\left(\Gamma_e^T S\Phi_e + Q_{12e}\right). \tag{2.19}$$

More details on the Riccati equation can be found in, e.g., [Åström and Wittenmark, 2013].

## Optimal State Estimation

Since there is measurement noise and the full plant state $x_e$ is generally not directly measurable, we must design a Kalman filter to produce the estimated state $\hat{x}_e$. The aim of the filter is to minimize the covariance of the steady-state estimation error

$$\lim_{k\to\infty} P[k] = \lim_{k\to\infty} \mathrm{E}\big(x_e[k] - \hat{x}_e[k]\big)\big(x_e[k] - \hat{x}_e[k]\big)^T. \tag{2.20}$$

The optimal estimator contains the prediction step

$$\hat{x}_e[k+1 \mid k] = \Phi_e \hat{x}_e[k \mid k-1] + \Gamma_e u[k] + K\big(y[k] - C_e \hat{x}_e[k \mid k-1]\big) \tag{2.21}$$

and the measurement update

$$\hat{x}_e[k \mid k] = \hat{x}_e[k \mid k-1] + K_f\big(y[k] - C_e \hat{x}_e[k \mid k-1]\big) \tag{2.22}$$

where

$$\begin{aligned} K &= \big(\Phi_e P C_e^T + R_{12e}\big)\big(C_e P C_e^T + R_{2e}\big)^{-1} \\ K_f &= P C_e^T \big(C_e P C_e^T + R_{2e}\big)^{-1} \end{aligned} \tag{2.23}$$

and $P$ is the solution to the algebraic Riccatic equation

$$P = \Phi_e P \Phi_e^T - \big(\Phi_e P C_e^T + R_{12e}\big)\big(C_e P C_e^T + R_{2e}\big)^{-1}\big(\Phi_e P C_e^T + R_{12e}\big)^T + R_{1e}. \tag{2.24}$$

Since we have an infinite-horizon LQG formulation, we only consider the stationary solutions to the Riccati equations (2.19) and (2.24), i.e., there are no time-varying controller and estimator gains. In Chapter 5 we will, however, have a periodic LQG control design, where the solutions to the Riccati equations are different from period to period, but for a specific period within the hyperperiod, the solution is stationary.

## Optimal Output Feedback—LQG

According to the separation theorem, an LQG design problem can be solved by independently designing an optimal state feedback controller and an optimal estimator. Therefore, the LQG controller is

$$u[k] = -L\hat{x}_e[k \mid k] \tag{2.25}$$

where $L$ is given by the optimal feedback gain (2.18), and $\hat{x}_e[k \mid k]$ is given by the optimal estimator (2.21)–(2.22).

The above LQG design method is for a sampled-data system with a constant delay. For a stochastic control delay, which is given in the form of probability mass

function, the $S$ in the optimal state feedback can be calculated by iteratively solving a stochastic Riccati equation [Nilsson, 1998]. The stochastic LQG design will be presented in Chapter 4 and Appendix A.

If the delay appears in a deterministically repeated pattern, we can introduce an augmented state that contains all state variables and all control signals over the periods. The standard optimal state feedback and optimal state estimation then are applied to design a periodic LQG controller. The details are given in Chapter 5 and Appendix B.

## 2.2 Real-Time System Scheduling

### Scheduling Policies

In a real-time system, multiple tasks execute on one or multiple processors, where the number of the processors is less than the number of tasks. A scheduling policy is needed to decide which task, in the case of a uni-processor, or tasks in the case of multiple processors, to execute at a certain time. There are two widely used scheduling policies:

- Off-line scheduling, also known as static scheduling or clock-driven scheduling. Here the scheduling decisions are made off-line and stored in a calendar or table for later use at run-time. The advantage is that all the behaviors of all the jobs in the real-time system are fixed and known, and the complexity of the scheduling algorithm is not important. The disadvantage is that there is no flexibility if the system is non-deterministic.

- On-line scheduling, also known as dynamic scheduling or priority-driven scheduling. Here the scheduling is done at run-time based on information about the arrived jobs, but not on information about future jobs. On-line scheduling is more effective when the parameters of jobs are varying, but there is a reduced ability to make the best use of the system resources. The two most common on-line scheduling algorithms are:

  - Static-priority scheduling, also known as fixed-priority scheduling. All jobs of a task have a static and fixed priority. Two priority assignment methods are rate-monotonic (RM) and deadline-monotonic (DM) priority assignment.

  - Dynamic-priority scheduling, also known as deadline-driven scheduling. Compared to static-priority scheduling, different jobs of a task can be assigned different priorities. Three common scheduling algorithms are earliest deadline first (EDF), latest release time (LRT), and least laxity first (LLF).

Scheduling algorithms can be also characterized by whether they are preemptive or non-preemptive. In preemptive scheduling, the running job can be interrupted by a higher priority job and in non-preemptive scheduling, the running job cannot be interrupted until it finishes or voluntarily yields the processor.

## Fixed-Priority Scheduling

Fixed-priority scheduling is the most widely used scheduling policy in industry. For example, in the embedded operating system standard OSEK, only fixed-priority scheduling is supported. To introduce fixed-priority scheduling, we first make some basic definitions. A real-time system, consisting of $n$ independent tasks running on a single processor under preemptive fixed-priority scheduling, is considered. The $i$th task, denoted by $\tau_i$, is characterized by the following parameters:

- The *worst-case execution time* $C_i^w$ is the largest amount of time it takes to execute a job of task $\tau_i$.
- The *best-case execution time* $C_i^b$ is the smallest amount of time it takes to execute a job of task $\tau_i$.
- The *period* $T_i$ is the constant time interval between two consecutive releases of task $\tau_i$.
- The *offset* $O_i$ is the instant at which the first job of task $\tau_i$ is released. If no offset is specified, then $O_i = 0$ is assumed.
- The *relative deadline* $D_i$ of task $\tau_i$ is the amount of time following the task release after which its execution has to be completed.
- The task *priority* is implicitly given by the task index so that $\tau_i$ has higher priority than $\tau_{i+1}$, unless otherwise stated.

Furthermore, the following task parameters are defined:

- The *response time* $R_i$ is the time interval between the release time and the finish time of task $\tau_i$. Since $R_i$ varies over time depending on the interference from higher priority tasks, we represent it by a random variable with cumulative distribution function $F_i : [0, \infty) \to [0, 1]$. The value $F_i(r)$ is the probability $P\{R_i \leq r\}$ that any job released by $\tau_i$ has response time smaller than or equal to $r$. The distribution $F_i$ depends on the parameters of the tasks $\{\tau_1, \tau_2, \ldots, \tau_i\}$, since they are the only ones which can affect the response time of $\tau_i$. The longest response time is the *worst-case response time* $R_i^w$, and the shortest response time is the *best-case response time* $R_i^b$.
- The *output jitter* $J_i$, is the difference between the worst-case and best-case response times; $J_i = R_i^w - R_i^b$. It represents the variation in time for the update of the control signal in each period.

- The *start latency* $S_i$ is the time interval between the release time and the start time of task $\tau_i$.

- The task *utilization* $U_i = C_i^w / T_i$ measures the worst-case fraction of computational resources required by the task. The *total utilization* of all tasks is $U = \sum_{i=1}^{n} U_i$, which should be less than or equal to 1 for a uni-processor.

- The *hyperperiod* $H$ is the least common multiplier of all the task periods.

Each control task implements a controller, for which we define the following timing parameters:

- The *sampling interval* $h_i$ is the time difference between the sampling operations of task $\tau_i$. We assume that sampling jitter has been eliminated by enforcing sampling at the job release time; hence, $h_i = T_i$.

- The *delay* $\delta_i$ is the time interval between the sampling operation and the output operation of task $i$. We assume that the output operation is performed when the job finishes; hence, $\delta_i = R_i$.

In this thesis, we allow the set of real numbers to represent all time values, including the execution time and the sampling period.

***Response-Time Analysis.*** For a control task, the delay normally varies from job to job. The longest delay and the shortest delay are equal to the worst-case response time and the best-case response time, respectively.

Assuming that $D_i \leq T_i$ for all tasks and that all deadlines can be met, the worst-case response time $R_i^w$ of task $\tau_i$ can be found by the following iterative procedure:

$$\begin{aligned} R_i^w[0] &= C_i^w \\ R_i^w[l+1] &= C_i^w + \sum_{j<i} \left\lceil \frac{R_i^w[l]}{h_j} \right\rceil C_j^w, \quad l = 0, 1, \cdots \end{aligned} \tag{2.26}$$

The procedure is stopped when the same value is found for two successive iterations of $l$ [Joseph and Pandya, 1986]. Similarly, the best-case response time $R_i^b$ of task $\tau_i$ can be found by the following iterative procedure:

$$\begin{aligned} R_i^b[0] &= C_i^b \\ R_i^b[l+1] &= C_i^b + \sum_{j<i} \left\lfloor \frac{R_i^b(l)}{h_j} \right\rfloor C_j^b, \quad l = 0, 1, \cdots \end{aligned} \tag{2.27}$$

The procedure is stopped when the same value is found for two successive iterations of $l$ [Redell and Sanfridson, 2002].

***Schedulability Analysis.*** A schedulability test for fixed-priority scheduling with rate-monotonic priority assignment was given in [Liu and Layland, 1973] as follows. A system of $n$ independent, preemptive periodic tasks with deadlines equal to periods can be feasibly scheduled on one processor according to RM scheduling if the total utilization satisfies

$$U \leq n\left(2^{\frac{1}{n}} - 1\right). \tag{2.28}$$

This is a sufficient, but not necessary, condition. The hyperbolic bound [Bini et al., 2003] is tighter than the bound in [Liu and Layland, 1973]. Let $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a set of $n$ independent, preemptive periodic tasks, where each task $\tau_i$ is characterized by a processor utilization $U_i$. Then, $\Gamma$ is schedulable with the RM algorithm if

$$\prod_{i=1}^{n}(U_i + 1) \leq 2. \tag{2.29}$$

This however requires knowledge of the individual task utilizations.

An exact schedulability analysis can be carried out by computing the worst-case response time of each task and comparing the value with the relative deadline. If $R_i^w \leq D_i$ for each task, then the task set is schedulable.

***Harmonic Scheduling.*** Harmonic RM scheduling has some good properties. A task set is harmonic if for any $i < j$, $T_j$ is an integer multiple of $T_i$. RM is optimal for harmonic periodic systems, i.e., a system of harmonic periodic, preemptive tasks, whose deadlines are at least their periods, is schedulable on one processor using RM if $U < 1$. More details on harmonic task scheduling are given in Chapter 6.

## 2.3 Jitterbug and TrueTime

In this thesis, two MATLAB-based toolboxes, Jitterbug and TrueTime, are used frequently for analysis, synthesis, and control performance evaluation, [Cervin et al., 2003]. Jitterbug is used for LQG controller design and LQG cost evaluation, while TrueTime is used to simulate and evaluate the control system performance in a real-time kernel environment.

### Jitterbug

Jitterbug is an LQG control system analysis and synthesis toolbox [Cervin and Lincoln, 2003]. On the one hand, it is used to design a discrete-time LQG controller for a continuous-time plant with continuous-time LQG cost and with constant or random delay. On the other hand, for any given plant, controller, and timing in one execution period, the LQG cost can be calculated in Jitterbug.

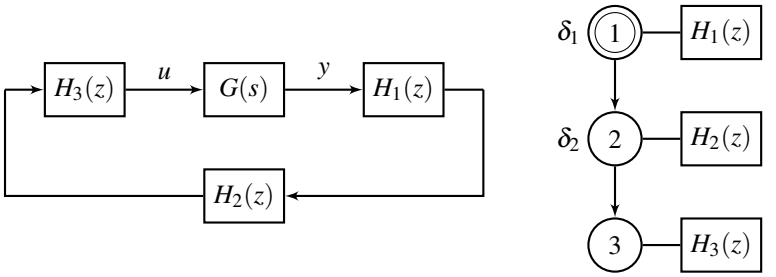



**Figure 2.1** A Jitterbug model of a computer-controlled system: a signal model and a timing model.

For the plant (2.1) and the cost function (2.2), Jitterbug can be used to design a LQG controller. If the plant is given on transfer function or zero-pole-gain form

$$\begin{aligned} y^0(t) &= G(p)\left(u(t-\delta) + v_{1c}(t)\right) \\ y[k] &= y^0[k] + v_2[k] \end{aligned} \tag{2.30}$$

where $G(p)$ is a strictly proper transfer function, and the cost of the system is defined as

$$V = \lim_{T\to\infty} \frac{1}{T} \int_0^T \begin{bmatrix} y^0(t) \\ u(t) \end{bmatrix}^T Q_c \begin{bmatrix} y^0(t) \\ u(t) \end{bmatrix} dt, \tag{2.31}$$

this problem can be reformulated to the problem of (2.1) and (2.2).

If the delay $\delta$ is constant, the LQG controller is designed by solving an algebraic Riccati equation. If the delay $\delta$ is given in the form of delay distribution, the LQG controller is determined by iteratively solving a stochastic Riccati equation.

***LQG Cost Calculation.*** In Jitterbug, a control system is described by two parallel models: a signal model and a timing model. The signal model is given by a number of connected, linear, continuous-time or discrete-time systems. The timing model is given by a number of timing nodes and describes when the different discrete-time systems are updated during the sampling period.

A Jitterbug model of a computer-controlled system is shown in Figure 2.1. The figure on the left is the signal model, which consists of a continuous-time plant $G(s)$, a discrete-time sensor $H_1(z)$, a discrete-time controller $H_2(z)$, and a discrete-time actuator $H_3(z)$. The sensor is a periodic sampler. All of the continuous-time and discrete-time linear systems are driven by white noise. The cost for each system is defined as a stationary, continuous-time quadratic cost function. The total cost of the feedback loop is summed over all the costs of the continuous-time and discrete-time systems.

The figure on the right in Figure 2.1 is the timing model, which consists of a number of timing nodes. Each node with a time delay is associated with zero or more discrete-time systems in the signal model. In this example, at the beginning

of each period, $H_1(z)$ is executed. Then there is a random delay $\delta_1$ before $H_2(z)$ is executed, and there is another random delay $\delta_2$ before $H_3(z)$ is executed. The delays can, e.g., model scheduling delays, computational delays, or network transmission delays. The model in Figure 2.1 could model a networked control loop where $\delta_1$ corresponds to the delay between the sensor node and the controller node and $\delta_2$ corresponds to the delay between the controller node and the actuator node. Alternatively, it could model a control task with $\delta_1$ being the delay between the sampling at the job arrival time and the start of the controller execution and $\delta_2$ corresponds to the computational delay.

The time delay can be defined as a time-independent or time-dependent delay. For a time-independent delay, it is described by a constant or by a probability density function

$$P_\delta = \begin{bmatrix} P_\delta(0) & P_\delta(1) & P_\delta(2) & \cdots \end{bmatrix} \tag{2.32}$$

where $P_\delta(k)$ is the probability of a delay of $k\delta_0$ seconds. The time grain $\delta_0$ is a constant that is specified for the whole model. A delay distribution may be dependent on the time since the most recent execution of the first node in the timing model. The delay is then described by a matrix

$$P_\delta = \begin{bmatrix} P_\delta(0,0) & P_\delta(0,1) & \cdots \\ P_\delta(1,0) & P_\delta(1,1) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{2.33}$$

where $P_\delta(j,k)$ is the probability of a delay of $k\delta_0$ seconds given a previous total delay of $j\delta_0$ seconds.

In order to calculate the LQG cost, timing nodes and systems have to be defined properly. In MATLAB, each timing node is defined by

```
N = addtimingnode(N, nodeid, Ptau, nextnode);
```

where `N` is the system's name, `nodeid` is the positive integer ID of the timing node, `Ptau` is the constant or random variable delay, and `nextnode` is the ID of the next node after the delay in the current node has elapsed.

Each continuous-time plant, or discrete-time sampler and controller is defined by

```
N = addcontsys(N, sysid, sys, inputid, Qc, R1c, R2);
N = adddiscsys(N, sysid, sys, inputid, nodeid);
```

where `sysid` is the name of the added system, `sys` is the state-space representation or transfer function of the added system. The output of the `inputid` system is the input of the added system. `Qc` is the cost matrix, `R1c` is the input noise covariance matrix, `R2` is the discrete-time measurement noise covariance matrix, and `nodeid` is the timing node where the current discrete-time system is executed.

Finally, the commands

```
N = calcdynamics(N);
V = calccost(N);
```

are used to calculate the LQG cost.

***Advantages and Disadvantages.*** Jitterbug has several advantages:

- Jitterbug provides an analytical LQG performance computation. For a given plant and a given controller in continuous-time or discrete-time, the stationary covariance matrix and the quadratic cost can be calculated.
- It is fast to evaluate the cost for a wide range of parameters, e.g., different periods, delays, and jitter. The delay can be given as a constant or a random variable.
- If the obtained cost is finite, then stability (in the mean-square sense) is guaranteed.

There are, however, also some disadvantages of Jitterbug:

- It only supports simplistic timing models. The delays are independent and the delay distributions may not change over time. It cannot handle dependencies between the periods.
- It only supports linear systems and quadratic costs.
- It is only a statistical analysis. The calculated cost is an expected value. All results only hold in a mean-value sense, so it does not provide any worst-case guarantees. Timing scenarios with probability zero are disregarded by the analysis. For instance, cases with switching-induced instability cannot be verified using Jitterbug.

***Use of Jitterbug.*** In this thesis Jitterbug is used for two purposes:

- To derive the LQG controllers, i.e., to solve the corresponding Riccati equations either in the normal case, i.e., with constant delay, the stochastic case, or the periodic case.
- To evaluate the LQG costs analytically in the cases where Jitterbug is applicable, e.g., when the delays are constant or can be reasonably approximated by independent random variables.

### TrueTime

TrueTime is a MATLAB/Simulink-based simulator, which facilitates simulation of controller task execution in real-time kernels, network transmissions, and continuous-time plant dynamics [Cervin et al., 2016]. TrueTime can be used to simulate real-time scheduling for control applications, and hence, for different scheduling settings, the control performance of a control system can be evaluated.

***Control Performance Evaluation.*** In a real-time control system, the TrueTime kernel block, based on Simulink, can be used to simulate a real-time operating system. The following activities are supported by the TrueTime kernel:

- Assigning the scheduling method and the priority assignment method. Truetime supports the following methods:
  - Static-priority scheduling: fixed priority, rate monotonic, deadline monotonic
  - Dynamic-priority scheduling: earliest deadline first

  However, it is also possible to implement user-defined scheduling methods.
- Assigning task parameters, e.g., task period, task offset, and task execution time.
- Specifying the overhead time for a full context switch.

The tasks in the real-time system are added in the initialization script of the TrueTime kernel.

If the output of the plant is assumed to be sampled at the job start time, a periodic task is created by the MATLAB command

```
N = ttCreatePeriodicTask(name,offset,period,codeFcn,data);
```

The first three arguments are the name, the offset, and the period of the created task. `codeFcn` is a user-defined MATLAB function, which models the control algorithm running in the created task. `data` contains the parameters used in `codeFcn`. A typical code function could be

```
function [exectime, data] = controller(segment, data)
switch segment,
  case 1,
     y = ttAnalogIn(data.nbr);
     data.u = data.c*data.x + data.d*y;
     data.x = data.a*data.x + data.b*y;
     exectime = data.exectime;
  case 2,
     ttAnalogOut(data.nbr, data.u);
     exectime = -1;
end
```

In `segment 1`, the control signal is computed and the controller state is updated. This takes `exectime` long time, where `exectime` can be an arbitrary deterministic or random variable. In `segment 2`, the new control signal is sent to analog out port. This takes zero time (indicated by $-1$).

If the output of the plant is assumed to be sampled at the job release time, an interrupt handler, a periodic timer, and a mailbox are needed to create a task. Using this approach, a periodic task is implemented using a periodic timer, an associated interrupt handler, and a mailbox. The timer invokes the interrupt handler periodically and the interrupt handler then samples the process, sends the sampled value to the mailbox, and then triggers a new job of a task where the actual controller code is executed. In the first segment of the code function of the controller task, the new sampled data is retrieved by reading from the mailbox. A simple example is

```
ttCreateInterruptHandler(nametimer_handler,priority, ...
                         samplercodeFcn,data);
ttCreatePeriodicTimer(nametimer,offset,period, ...
                      nametimer_handler);
ttCreateMailbox(nameMailbox);
ttCreateTask(name,deadline,codeFcn,data);
```

The control algorithm code is similar to `controller(segment,data)` with the exception of the first segment where the reading from the mailbox is performed. The code function associated with the interrupt handler could look like

```
function [exectime, data] = samplercode(segment, data)
switch segment,
   case 1,
      y = ttAnalogIn(data.nbr);
      ttTryPost(data.nameMailbox, y);
      ttCreateJob(data.name);
      exectime = 0;
   case 2,
      exectime = -1;
end
```

In `segment 1`, the code reads the process output, puts a sample in the mailbox, and triggers a job of the controller task. This is assumed to take zero execution time, modeling that this is in reality is performed in separate hardware.

***Real-Time System Timing Analysis.*** TrueTime can also be used to do evaluate response times by simulation. Response time, release latency, start latency, execution time, and context switch instances can be logged for each job of each task during the simulation. The distributions of those data are also available. This information can then be used for schedulability analysis or worst-case response time analysis. Hence, in addition to simulating control system performance, TrueTime can be used as a pure scheduling simulator.

The timing information can be created by

```
ttCreateLog(taskname, logtype, variable, size);
```

TrueTime supports five pre-defined log types corresponding to the response time, release latency, start latency, execution time, and context switch instances. The variable `logtype` can also be extended to user-defined logs by using the primitives `ttLogStart`, `ttLogStop`, and `ttLogNow` inside the code functions.

***Real-Time System Implementation.*** TrueTime supports A/D and D/A connection between the kernel and simulated or physical systems using the primitives `ttAnalogIn` and `ttAnalogOut`. If measures are taken to ensure that the Simulink simulations are performed in real-time and these primitives are attached to real A/D and D/A converters then TrueTime can also be used for real-time system implementation.

***Advantages and Disadvantages.*** TrueTime has several advantages, e.g.:

- It can be used to investigate the true, timely behavior of time or event-triggered control loops, subject to sampling jitter, input-output latency and jitter, and lost samples, caused by real-time scheduling and networking effects.
- It supports a variety of different scheduling methods.
- It can also be used as a pure scheduling simulator.

However, it also has some disadvantages, e.g.:

- The abstraction level is low. Modeling a real-time system in TrueTime has close to the same complexity as implementing the system using real software and hardware.
- It is developed as a research tool rather than as a tool for system developers and it is based on MATLAB/Simulink.
- Evaluating the LQG cost numerically using TrueTime simulation can be time-consuming. For the example below, it takes about 1000 seconds to get a converged LQG cost, as shown in Figure 2.3.

## LQG Cost Evaluation Example

This example presents a simple LQG controller for an inverted pendulum where the LQG control cost is evaluated in both Jitterbug and TrueTime. The state-space model of the inverted pendulum is

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(t-\delta) + v_{1c}(t) \\ y[k] &= \begin{bmatrix} 0 & 1 \end{bmatrix} x[k] + v_2[k]. \end{aligned} \tag{2.34}$$

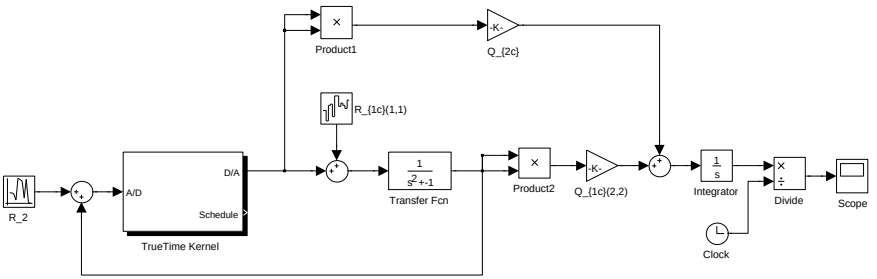



**Figure 2.2** LQG cost evaluation model in TrueTime

The sampling period is set to $h = 0.3$ and the constant delay in every period is $\delta = 0.15$. The cost matrix is

$$Q_c = \begin{bmatrix} Q_{1c} & Q_{12c} \\ Q_{12c}^T & Q_{2c} \end{bmatrix} = \left[\begin{array}{cc|c} 0 & 0 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & 0.01 \end{array}\right], \tag{2.35}$$

where the input noise covariance matrix is

$$R_{1c} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{2.36}$$

and the sampled measurement noise covariance matrix is $R_2 = 0.01$. The discrete-time LQG controller can be designed through the `lqgdesign` command in Jitterbug. The LQG controller is

$$K(z) = \frac{-15.532z^2(z-0.675)}{z(z^2+0.618z+0.157)}.$$

Calculating the LQG cost in Jitterbug gives the value $V = 0.432$.

The corresponding TrueTime model is shown in Figure 2.2. The TrueTime kernel is used to schedule only one task, which implements the LQG controller. The controller is the same as the LQG controller in Jitterbug evaluation, namely $K(z)$. The simulated output signal $y^0(t)$ and control signal $u(t)$ are squared. The square of the output signal is multiplied by $Q_{1c}(2,2)$ and the square of the control signal is multiplied by $Q_{2c}$. Then the square of the output signal and the weighted control signal are summed and the sum is integrated and divided by the running time. The simulation time is 1000 seconds.

The average running cost

$$V\left(t_f\right) = \frac{1}{t_f}\int_0^{t_f} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}^T Q_c \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} dt, \quad t_f > 0 \tag{2.37}$$

is shown in Figure 2.3. In this case the TrueTime cost converges to the Jitterbug cost, if the simulation time in TrueTime is longer than 1000 seconds.

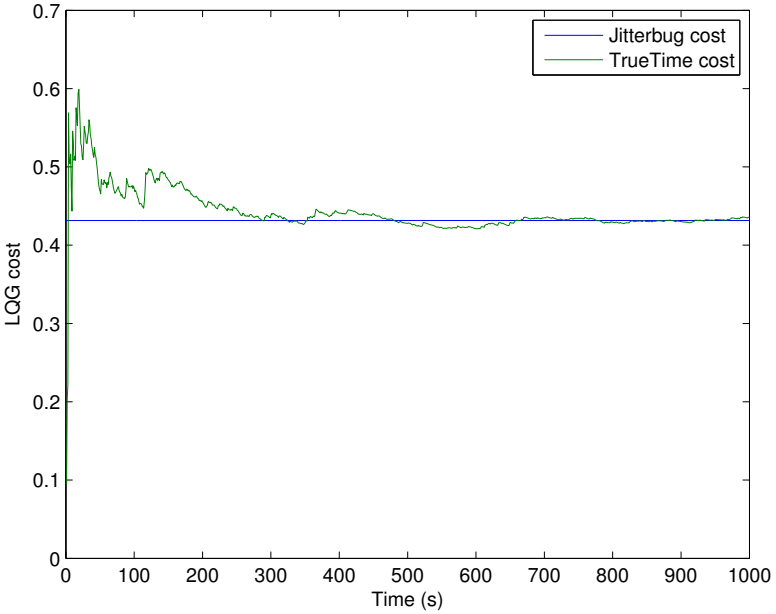



**Figure 2.3** LQG cost in Jitterbug and TrueTime

## Use of TrueTime

In this thesis, TrueTime is used for two purposes.

- To obtain the delay distributions required for the stochastic LQG design method in Chapter 4 and the periodic–stochastic LQG design method in Chapter 5. This is done by simulating the schedule for sufficiently long time and logging the response times.
- To evaluate the LQG costs numerically by simulation in the cases where Jitterbug is not applicable, e.g., when the delays are not independent. The cost evaluations can both be performed as a part of a codesign method, e.g., in the search-based methods in Chapter 4, and in the final control performance evaluation. However, whenever Jitterbug is applicable, it is substantially faster and more accurate to evaluate the cost using Jitterbug than with TrueTime.

# 3

# Related Work

## 3.1 Real-Time System Scheduling

For rate-monotonic (RM) scheduling, the classical utilization bound is provided in [Liu and Layland, 1973]. The bound only depends on the number of tasks in a real-time system. The utilization bound test is only sufficient, but not necessary, so it is in general pessimistic. The hyperbolic bound test proposed in [Bini et al., 2003] has the same complexity as the Liu and Layland bound. It is also a sufficient-only test but it is tighter. It is based on the individual task utilizations rather than the total CPU utilization.

Harmonic task periods have some benefits both in real-time system scheduling and control system design. A harmonic task set with total utilization less than or equal to 1 can be feasibly scheduled by the RM algorithm [Lehoczky et al., 1991]. Testing schedulability and computing the response time is computationally expensive in general, but for harmonic real-time tasks, a polynomial-time algorithm for schedulability tests for FP and EDF scheduling is provided in [Bonifaci et al., 2013]. Two efficient methods to assign harmonic periods to real-time tasks with period ranges have been proposed: the forward approach in [Nasri et al., 2014] and the backward approach in [Nasri and Fohler, 2015].

In [Han and Tyan, 1997], the authors propose a polynomial-time sufficient schedulability test for FP scheduling as long as the total utilization is less than or equal to 1. They also point out that a harmonic task set under RM priority assignment is schedulable if the utilization is less than or equal to 1. For harmonic task sets under preemptive FP scheduling, the potential gains in terms of schedulable utilization are shown for the case of cache memories in [Busquets-Mataix et al., 1996].

A small hyperperiod reduces timing analysis complexity and simulation time. If the periods of all tasks are selected from the natural numbers within given ranges, a heuristic algorithm to calculate the minimum hyperperiod for a set of tasks is proposed in [Brocal et al., 2011]. In [Ripoll and Ballester-Ripoll, 2013], the authors present an algorithm to find the minimal hyperperiod for a set of periodic tasks. The periods are any rational numbers within the given ranges.

An algorithm for calculating the worst-case response time of a task under fixed-priority scheduling is presented in [Joseph and Pandya, 1986], and the corresponding algorithm for the best-case response time is given in [Redell and Sanfridson, 2002]. In [Eisenbrand and Rothvoß, 2008], the authors show that the response time computation for RM scheduling is NP-hard, and there does not exist any polynomial-time approximation algorithms with approximation ratio bounded by a constant. In [Bini and Baruah, 2007], an upper bound on the worst-case response time is proposed. With task release offsets and release jitter, an exact response time analysis is proposed in [Redell and Torngren, 2002]. The authors create the worst-case condition in a hyperperiod, and calculate the worst-case response time by partly iterative algorithms.

Only the worst-case response time and the best-case response time are not enough for detailed performance analysis of control tasks. For this, the response time distributions are needed. For fixed-priority and dynamic-priority scheduling, the response time distribution of each task can be accurately computed [Díaz et al., 2002]. The method can handle arbitrary execution time distributions. The authors derive formulae for the backlog probability mass function (PMF) at any time, in order to calculate the response time. In [Tanasa et al., 2015], the authors calculate the response time probability distribution, with arbitrary execution time distributions. The execution time distributions are approximated by polynomial functions, and then the non-idling scenarios are identified to compute the response time distribution.

In [Davare et al., 2007], a period assignment is accomplished using schedulability analysis with end-to-end latency constraints. This work is based on convex optimization.

## 3.2 Controller Timing

In [Åström and Wittenmark, 2013], it is shown how a continuous-time system with an input delay is sampled under zero-order hold (ZOH) control. In the extended state vector, the past value of the control signal is included. The standard design procedures for linear quadratic (LQ) control and Kalman filters for discrete-time systems are also presented.

When the delay is random with a given delay distribution, the stability of the closed-loop system is studied and a stochastic LQG control design is derived in [Nilsson et al., 1998] and [Nilsson, 1998]. The stationary value of the value function $S(k)$ can be calculated by iteratively solving the stochastic equation, and then the optimal state feedback can be derived. Since the old time delays up to time $k-1$ are known at time $k$, the standard Kalman filter is optimal here. One hypothesis behind this technique is that the delay of each job is independent of the others, which is generally not the case in real-time systems. It is also assumed that the variability in delay is smaller than the sampling period. This limitation is later lifted in

[Lincoln and Bernhardsson, 2000]. The authors derive the LQG-optimal controller for networked control systems with arbitrarily long delays. However, no stationary solution is found, implying that the optimal controller involves quite heavy on-line computations.

Under LQG control, stability of the closed-loop system is guaranteed, but its robustness is not guaranteed [Doyle, 1978]. The control system might be sensitive to modeling errors. The robust control synthesis problem can be solved by the combination of Youla parameter design (also known as Q-parameterization) and convex optimization. Some examples are shown in [Boyd et al., 1998]. Mixed $H_2/H_\infty$ control gives some good properties, such as robustness guaranteed LQG control, e.g. for continuous-time systems [Zhou et al., 1994] and [Doyle et al., 1994], and for discrete-time systems [Muradore and Picci, 2005], but the feedback system is assumed to be linear time-invariant (LTI), meaning that these methods cannot directly deal with output jitter.

Motivated by delay and jitter in real-time control systems, simple sufficient stability criteria for systems with time-varying but bounded delays for both continuous-time and discrete-time systems are given in [Kao and Lincoln, 2004]. Based on this stability theorem, the notion of jitter margin for a control system is proposed in [Cervin et al., 2004]. The jitter margin is defined as a function of the constant delay, and it describes how much additional time-varying delay can be tolerated before the loop becomes unstable. In [Cervin, 2012], the author analyzes the combined effect of input and output jitter on the stability of a linear sampled-data control systems and proposes a new sufficient stability theorem.

For a sampled-data system, periodic sampling is most often used for digital implementation of feedback control laws. However, event-triggered control, [Årzén, 1999] and [Åström and Bernhardsson, 1999], and the self-triggered control approach, [Velasco et al., 2003], have some advantages when energy and computational constraints are considered.

## 3.3 Scheduling and Control Codesign

The codesign problem of real-time system scheduling and feedback control is studied in [Årzén et al., 2000], [Årzén et al., 2003], [Årzén and Cervin, 2005], and [Derler et al., 2013]. The research directions range over system timing and functionality of control engineering and computer science, such as control loop timing, task attribute adjustments, feedback scheduling, etc. Here we discuss some of them.

### Sampling Period Assignment

In [Seto et al., 1996], the authors raise the question of how to optimize the overall control performance subject to an overall utilization constraint. The cost of task $\tau_i$

is defined as a general cost function

$$V_i = S_i\left(x_i\left(t_f\right)\right) + \int_0^{t_f} L_i(x_i(t), u_i(t), t)dt \tag{3.1}$$

and this cost is approximated as a function of sampling period $h_i$ as

$$V_i = \alpha_i e^{-\frac{\beta_i}{h_i}} + \gamma_i. \tag{3.2}$$

Then this optimization problem is analytically solved with the utilization constraint. In this work, which is a milestone in the literature of real-time control codesign, however, the delay between sensing and actuation is assumed to have no impact on the performance of the system. The authors use the utilization upper bound from [Liu and Layland, 1973] as the feasibility constraint.

Delay and jitter due to task scheduling can have a great impact on control performance [Wittenmark et al., 1995]. In [Kim, 1998], the author uses the same cost as in [Seto et al., 1996], but the cost is approximated as a function of sampling period and delay

$$V_i = \alpha_i e^{\beta_i h_i + \varepsilon_i \delta_i} + \gamma_i. \tag{3.3}$$

The initial delay $\delta_i$ is approximated as the worst-case response time and the periods are found. Then, the new delays are computed by simulating the schedule of all the tasks up to the hyperperiod, and iteratively the periods are computed again assuming the new values of the delay.

In [Bini and Di Natale, 2005], the authors propose an algorithm that finds the optimal period assignment of control tasks scheduled by fixed priority. In their work, the delay is guaranteed not to exceed the period for all tasks. Since the optimal method requires a time consuming branch-and-bound algorithm to be executed, they also propose a faster algorithm to find a sub-optimal period assignment taking advantage of some geometrical considerations in the space of feasible activation rates.

In [Bini and Cervin, 2008], the cost is defined as a standard LQG cost and this cost is approximated as a linear function of sampling period and delay. Then the overall cost minimization problem is analytically solved. The details of this method are described in Section 3.4.

The aim in [Samii et al., 2009] is to optimize overall LQG cost, which is affected by delay and jitter, for both static and priority-based scheduling. The approximation of the delay distribution is obtained by simulation. The random variable delay is approximated as the average delay when designing the LQG controller. The delay distribution is used to evaluate the LQG cost. The search process for the periods is based on a genetic algorithm, which assigns controller periods within a fixed and given set.

For RM scheduling, the overall LQ cost is used as control performance metric in [Saifullah et al., 2014]. The LQ cost for each task is approximated as (3.3) with $\gamma_i = 0$. The constraint is that the response time is less than or equal to the deadline

and the period is chosen from a period range. For this non-differentiable, nonlinear, and non closed-form optimization problem, the authors give four ways to obtain sub-optimal periods.

In [Goswami et al., 2012], for joint ECU and bus scheduling in mixed-criticality systems, the overall LQ is used as the control cost. For each task, the period is selected from a finite set. For each period, the LQ cost is approximated as a polynomial function of the delay

$$V_i = \sum_{j=0}^{n} \alpha_j \delta_i^j. \tag{3.4}$$

This integer programming problem is solved by the CPLEX ILP solver.

A codesign simulation tool is developed in [Palopoli et al., 2000]. For both RM and EDF, the authors show the use of soft real-time constraints, by selecting shorter sampling period for control task, leads to significant performance improvement.

In [Zhang et al., 2008], the control performance metric is defined as the overall $H_\infty$ norm or the minimal $H_\infty$ norm. The optimal periods are found such that the Liu and Layland bound is satisfied and all the feedback loops are stable.

In [Ben Gaid et al., 2009], the codesign problem is decomposed into an optimal control problem and an off-line scheduling problem. The sampling periods are optimally chosen. The optimal state feedback control is designed using the lifting technique and an on-line scheduling algorithm (Reactive Pointer Placement) is proposed. Control performance improvements and the stability guarantees are proven.

Periodic LQG control design in real-time systems has been considered in [Ramanathan, 1997], but for the case of variable sampling intervals and not for varying delays. The time-varying state feedback control law is derived by solving a periodic Riccati equation.

## Handling Jitter

In control applications, the jitter may degrade performance and even jeopardize stability. The delay variance is defined as CAI (control action interval) and DAI (data acquisition interval) in [Balbastre et al., 2004]. The CAI and DAI are evaluated offline. Using an IMF (initial, mandatory, and final) task model, a subtask partition method is used to improve control performance.

In [Buttazzo and Cervin, 2007], three methods: task splitting, advancing deadlines, and non preemptive execution; for reducing the jitter in controller task execution are evaluated. The evaluation shows that the deadline advancement method gives the best control performance.

## Online Optimization

In [Eker et al., 2000], the overall LQ cost is used as control performance metric. The authors calculate the relation between sampling period and control performance, and a feedback scheduler is designed based on this relation. First, the cost functions

and their dependence on sampling intervals are calculated. Then a recursive optimization is proposed, based on constrained Newton optimization. It is stable, but computationally expensive. So an approximated solution is given, in which the LQ cost function is approximated as a quadratic function of the sampling period. Both the recursive optimization method and the approximated optimization method can also be used for offline period assignment.

For EDF scheduling, a feedback-feedforward scheduling architecture for a real-time control system is proposed in [Cervin et al., 2002a]. The scheduler uses feedback from execution time measurements and feedforward from workload changes to adjust the sampling periods so that the overall LQG cost is minimized.

In [Abeni et al., 2002], the bandwidths are dynamically assigned to a set of constant bandwidth servers (CBS), by applying feedback control to a reservation-based scheduler.

In [Martí et al., 2004] and [Martí et al., 2009], in order to optimize the overall control performance, the authors present an optimal period assignment based on feedback from the state of the plant. It is formulated as a linear constrained optimization problem and this problem can be analytically solved online. For EDF scheduling, a self-triggered state feedback $H_\infty$ controller is proposed in [Lemmon et al., 2007]. The release time is online adjusted such that the $H_\infty$ norm is bounded in the feedback loop.

In [Henriksson and Cervin, 2005], the expressions relating the LQ cost to the sampling period and the state of the plant are derived. For the case of minimum-variance control applied to an integrator plant, an exact expression is developed. For the general case, an online optimization procedure is obtained.

## Network Scheduling for Control

In [Ben Gaid et al., 2006], the problem of control and scheduling of networked control systems over limited bandwidth deterministic networks is addressed. Multivariable linear systems subject to communication constraints are modeled in the Mixed Logical Dynamical framework. The Mixed Logical Dynamical model is transformed to a Mixed Integer Quadratic Programming formulation. Then the control and scheduling problem is solved. A model predictive controller and an Optimal Pointer Placement scheduler are designed.

In [Ben Gaid et al., 2004], a ride controller, which is used to improve ride comfort by isolating the sprung mass from road disturbances, is simulated in TrueTime. The simulation result shows that the scheduling parameters have an important effect on the robustness of the controlled system.

In [Goswami et al., 2011], using the FlexRay dynamic segment as the communication medium, a scheduling and control codesign method is proposed to guarantee the stability of the control application.

### Robust Stability

In [Palopoli et al., 2002], robustness optimization problem of a control system is defined to maximize the stability radius. The optimization variables are the activation period and the feedback gain. It is solved based on a branch and bound algorithm.

In [Tabuada, 2007], an event-triggered real-time scheduling method is proposed for stabilizing control tasks. The author considers an event-triggered scheduling algorithm that preempts running tasks to execute the control task when a certain error becomes large compared to the state norm. The execution time of the control task is taken into account to show that the proposed scheduling policy provides guarantees on the global asymptotic stability.

In [Aminifar et al., 2012], for the given delay range, the authors propose an integrated approach to assign task periods so that the stability of the plants is guaranteed in the worst case. The jitter margin is used to measure the worst-case control performance. A standard LQG controller is used and the jitter margin constraint is satisfied by designing the scheduling parameters.

### Formal Analysis

In [Frehse et al., 2014], the codesign model is represented as a network of hybrid automata. The timing properties and the closed-loop properties are verified through model checkers, for example, SpaceEx. The verification of timing properties is based on the combination of logical execution times and typical worst-case analysis, and the verification of closed-loop properties is based on reachability analysis of hybrid automata.

## 3.4 Delay-Aware LQG Design

The methods developed in this thesis often refer to the delay-aware LQG design method described in [Bini and Cervin, 2008], either as an initialization method to obtain good starting values for the task periods or as the baseline in the evaluations. For RM scheduling, the authors optimize the sampling period, such that the utilization is equal to 1. Jitter is not considered to have an effect on performance. The cost is defined as the sum of the standard LQG costs for all the tasks. For reasonably short periods, the LQG cost of a control loop is approximated as a linear function of sampling period and delay

$$V_i = \alpha_i h_i + \beta_i \delta_i. \tag{3.5}$$

The delay $\delta_i$ is approximated by a lower bound on the worst-case response time

$$\delta_i = \frac{C_i^w}{1 - \sum_{j=1}^{i-1} U_j} \tag{3.6}$$

which is a constant. With the approximation of the cost and the approximation of the response time, the overall cost is a function of the periods. Then the overall cost

minimization problem is analytically solved under the utilization constraint $U \leq 1$. Since the real cost function is not linear in $h_i$ and $\delta_i$, the authors also propose an iterative period assignment routine based on the analytic solution of the linearized problem. In the sequel the method in [Bini and Cervin, 2008] is also referred to as RiApprox.

# 4

# Stochastic LQG Scheduling and Control Codesign*

## 4.1 Introduction

In earlier work, the codesign problem of assigning optimal control task periods has been approached either by ignoring the effects of the delay on the performance or by approximating the delay by a constant. In this chapter, the delay is instead modeled by the task response time distribution, and the controller is designed to improve performance for a delay that varies according to such a distribution. This is done using the techniques based on [Nilsson et al., 1998] that enable the optimal design of controllers that are subject to a control delay, which is assumed to be a random variable with known distribution. One hypothesis behind this technique is that the delay of each job is *independent* from the others, which is certainly not the case in real systems. However, if the parameters of all tasks are known in advance, it is possible to determine the entire task schedule within the hyperperiod, which is then repeated over time.

The picture, however, drastically changes if the controllers have to be designed, that is, if the task periods are *variables* of an optimization problem. In this case, the hyperperiod may not even exist (if any pair of task periods are incommensurable) and then the full task schedule cannot be computed.

The design of controllers (which includes also the selection of their period) that are aware of the job delay pattern is a challenging problem, which is considered in this chapter.

The work by [Nilsson et al., 1998] assumed that the variability in delay was smaller than the sampling period. This limitation was later lifted by [Lincoln and Bernhardsson, 2000], who derived the LQG-optimal controller for networked control systems with arbitrarily long delays. However, no stationary solution was found, implying that the optimal controller involves quite heavy on-line computations. In

* This chapter is based on [Xu et al., 2014].

this work, therefore, we design controllers based on truncated distributions so that the variability is always smaller than the period, even though this approach is sub-optimal.

### Outline

In Section 4.2 the real-time control system model is presented. Validation for modeling the delay with the response-time distribution is provided in Section 4.3. In Section 4.4 the optimal period assignment problem is defined together a local sequential search-based method. In Section 4.5 this method is compared with a non-linear direct optimization method in a simulation evaluation where the cost is evaluated using the TrueTime simulation tool [Cervin et al., 2003].

## 4.2 Real-Time Control System Model

The basic model of the control cost and the task run-time model are described in Section 2.1 and Section 2.2 respectively. Since the response time $R_i$ varies over time, depending on the interference from higher priority tasks, we here model it as a random variable with cumulative distribution function $F_i : [0, \infty) \rightarrow [0, 1]$. The value $F_i(r)$ is the probability $P\{R_i \leq r\}$ that any job released by $\tau_i$ has response time smaller than or equal to $r$. The distribution $F_i$ depends on the parameters of the tasks $\{\tau_1, \tau_2, \ldots, \tau_i\}$, since they are the only ones that can affect the response time of task $\tau_i$.

If the delay $\delta_i$ of the controller varies as a random variable with a given probability density, it is possible to calculate the corresponding optimal controller that minimizes the cost of (2.2), yielding a so-called stochastic LQG controller. The stochastic LQG solution for a discrete-time system with delays shorter than the period was given in [Nilsson et al., 1998]. We have extended the design method to handle a continuous-time problem formulation as well as delays longer than the period (although with jitter smaller than or equal to the period). The optimal state feedback gain in the controller is obtained through solving a stochastic Riccati equation using iteration. This is combined with a Kalman filter to produce the full LQG controller. The MATLAB toolbox Jitterbug [Lincoln and Cervin, 2002] can be used to compute both the optimal control feedback and the corresponding cost. The details of the stochastic LQG control design procedure are given in Appendix A.

Since the cost $V_i$ of the controller $\tau_i$ depends on the response-time distribution, which in turn depends on the period of the higher priority tasks, then it can be written as

$$V_i(T_1, T_2 \ldots, T_n) \tag{4.1}$$

with

$$\frac{\partial V_i}{\partial T_k} = 0, \qquad \forall k = i+1, \ldots, n. \tag{4.2}$$

The model used assumes that the sampling is performed at the task release times, i.e., without any jitter. This can, e.g., be achieved by performing the sampling in the clock interrupt service routine, or in a dedicated high-priority task that then communicates the sample to the controller task, using, e.g., a mailbox. Modeling the delay of the controller using the task response time assumes that the actuation is performed at the end of the task. Normally, this is not the way a controller is implemented. Instead, the code is structured in two sections: a *CalculateOutput* section that contains the calculations that directly depend on the current sample, and an *UpdateState* section where the internal states of the controller are updated. The actuation is then performed as soon as the *CalculateOutput* section is completed. However, since the approach in this chapter is based on simulating the total task schedule it is straightforward to extend it to this case instead. An additional assumption implicitly made is that the task execution time is relatively constant from one job to the next. Although this is not true for general tasks, the assumption is more valid for controller tasks, since the code size is relatively small and the amount of branches is low. Furthermore, it is assumed the kernel allows the task periods to have arbitrary real-valued values. This implies a so-called tick-free kernel.

## 4.3 Validating the Model of the Task Delay

### Choice of the Response Time Distribution

The design approach in this chapter is based on the possibility to design the controller taking the distribution of the delay caused by the task scheduling into account. This is done by modeling the delay by the response time distribution. However, when the tasks have constant execution times, the delay is actually deterministic, although following a pattern, which is not easily characterizable [Lehoczky, 1990]. It is then necessary to validate the appropriateness of modeling the delay as stochastic.

Consider the following example. Assume a task set consisting of two tasks defined as

| Task | $T_i$ | $C_i^w$ | Priority |
|---|---|---|---|
| $\tau_1$ | 0.24 | 0.12 | High |
| $\tau_2$ | 0.3 | 0.12 | Low |

where task $\tau_2$ implements a LQG-controller with sampling period, $T_2 = 0.3$, controlling an inverted pendulum process modelled by the Laplace-transfer function $P(s) = 1/(s^2 - 1)$. The continuous-time input noise has the covariance $R_{1c} = 1$ and the discrete-time measurement noise has the covariance $R_2 = 0.01$. The cost function that the controller should minimize is given by (2.2), with

$$Q_c = \begin{bmatrix} C^T C & 0 \\ 0 & 0.01 \end{bmatrix}, \tag{4.3}$$

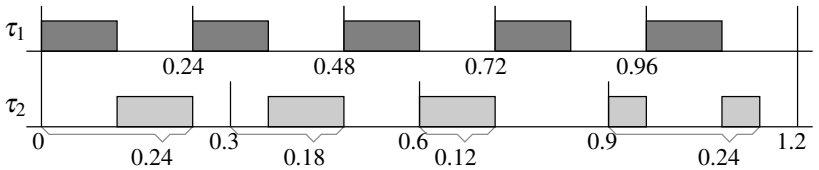



**Figure 4.1** Schedule of the two control tasks.

where $C$ is defined by the controllable canonical state-space representation.

The schedule of the tasks is illustrated in Figure 4.1. The schedule repeats every hyperperiod (1.2 for these tasks). The task execution times are assumed to be constant and given by $C_i^w$.

As it can be noticed in the figure, the response time for task $\tau_2$ will have the following repetitive cycle $R_2 = 0.24, 0.18, 0.12, 0.24, \ldots$ (the interested reader can find more details in the work by [Lehoczky, 1990]). Hence, the worst-case response time is 0.24, the best-case response time is 0.12, and the average-case response time is 0.195.

We now compare the costs for four different LQG-controllers:

LQG$_B$ The controller is designed assuming a constant delay equal to the best-case response time.

LQG$_A$ The controller is designed assuming a constant delay equal to the average-case response time.

LQG$_W$ The controller is designed assuming a constant delay equal to the worst-case response time.

LQG$_S$ The controller is designed assuming that the delay is governed by a stochastic variable with a discrete-time distribution function corresponding to the above cycle, i.e., the likelihood is 50% that the delay is equal to 0.24, 25% that it is equal to 0.18, etc.

The costs will be evaluated for two execution scenarios:

E$_S$ The delay is a stochastic variable with the distribution function defined previously.

E$_D$ The delay varies from job to job according to the deterministic repetitive cycle defined previously. This scenario corresponds to the true execution according to the schedule.

The costs for the eight cases are given below.

| Controller | $E_S$ | $E_D$ |
|---|---|---|
| $LQG_B$ | 0.66 | 0.71 |
| $LQG_A$ | 0.60 | 0.62 |
| $LQG_W$ | 0.64 | 0.62 |
| $LQG_S$ | 0.59 | 0.61 |

In this example, for both execution scenarios, the controller designed using the delay distribution gives the lowest cost, which speaks in favour of the approach adopted in this chapter. Also, the cost for the true deterministic execution scenario is quite close to the cost for the stochastic execution scenario. This also speaks in favour of the proposed approach. Furthermore, when as in the current case the task periods are derived from numerical optimization it is unlikely that there will exist any hyperperiod, i.e., there is no repeating pattern for the delay. As will be shown by the evaluations presented in the following the error introduced by assuming a stochastic delay is in general quite minor.

## Computation of the Response Time Distribution

The above example indicates that adopting the response time distribution as a model for the delay provides results that are quite similar to the ones obtained by considering the exact pattern of job delays. However, to best of our knowledge, today there is no analytical method, which provides the response time distribution as a function of the task parameters. Hence, we can only proceed by simulation.

Our simulation-based computation of the response time distribution must necessarily use a finite number of jobs for which the response time is computed. Next, we investigate the impact of the number of jobs on the accuracy of the response time distribution.

Let $R(k)$ denote the random variable of the job response time extracted among the first $k$ jobs, and $F_{R(k)}(r)$ its cumulative distribution function (CDF). With this notation in mind and for the purpose of measuring how sensitive the distribution of $R(k)$ is to $k$, we define the *incremental normalized distance* as

$$d(k) = \frac{1}{T} \mathrm{E}\left\{ |R(k) - R(k+1)| \right\} \tag{4.4}$$

where $T$ is the period of the task. Intuitively, $d(k)$ represents the diversity of $R(k+1)$ w.r.t. $R(k)$. The factor $1/T$ is added to properly normalize such a distance. We observe that $d(k)$ is zero if and only if $R(k)$ and $R(k+1)$ are coincident almost everywhere. Moreover, we have

$$\lim_{k \to \infty} d(k) = 0$$

since as $k$ grows, the two random variables $R(k)$ and $R(k+1)$ tend to converge.

Such a quantity $d(k)$ can be used to determine the number of jobs after which the simulation can be stopped. In fact, if $d(k)$ is small it means that adding new samples

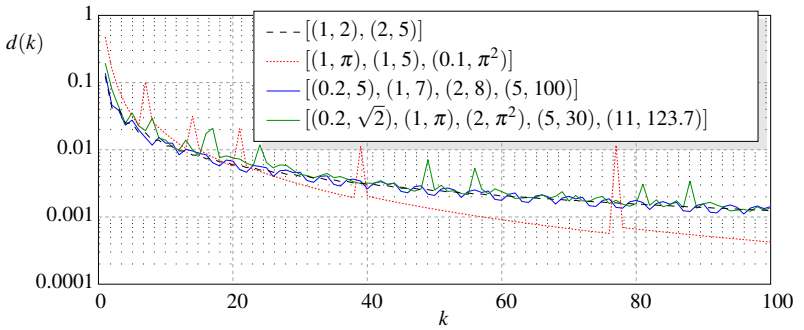



**Figure 4.2** $d(k)$ as a function of the number $k$ of jobs.

of the job response time is not going to affect significantly the new response time distribution. As shown by [Vallender, 1974], the value $d(k)$ of (4.4) can be computed as

$$d(k) = \int_{-\infty}^{\infty} |F_{R(k)}(r) - F_{R(k+1)}(r)|\, dr. \tag{4.5}$$

In Figure 4.2, we report the value $d(k)$ computed for the response time of the lowest priority task, as the number of jobs increases. The plot is reported for several task sets, each denoted in the format $[(C_1^w, T_1), (C_2^w, T_2), \ldots, (C_n^w, T_n)]$ in Figure 4.2. Observe that when the task periods are integers (such as in the 1-st and 3-rd case) the distribution of the response time does not vary rapidly over time. Instead, it is interesting to notice that when any period is irrational (such as in the 2-nd and 4-th cases) the response time of some job does not conform to the distribution computed until job 100.

Finally, we remark that the goal of this chapter is not to determine the most accurate possible distribution of the task response time. Instead, we aim at determining a response time distribution, which models the task delay well enough to allow an improved selection of the task periods. As we will show later in the experiments, selecting the first 100 jobs for computing the response time distribution is a suitable choice for determining task periods which more aggressively reduce the control cost.

## 4.4 Period Assignment

The period selection problem can be formulated as follows:

- find the controller periods $[T_1, T_2, \ldots, T_n]$,
- such that the cost

$$V = \sum_{i=1}^{n} V_i, \tag{4.6}$$

with $V_i$ defined in (2.2), is minimized, and

- the set of controllers is feasible, that is

$$\sum_{i=1}^{n} \frac{C_i^w}{T_i} \leq U_b, \tag{4.7}$$

where $U_b$ is the utilization bound. Since the cost $V_i$ of (2.2) has no explicit form and can only be computed numerically (through TrueTime, in our case), we solve this problem with numerical optimization. Below we provide more details for the optimization procedure.

The delay-aware period assignment method (RiApprox) from [Bini and Cervin, 2008] was used to provide initial task periods. In this approach a linear approximation of the cost function $V = \sum \alpha_i T_i + \beta_i \delta_i$ is minimized assuming that the delay $\delta_i$ equals an approximation of the average response time.

## Cost Calculation

The change of $T_i$ affects the response time distribution of task $j$ when $j \geq i$. And the change of response time distribution affects the cost of the current task. So the overall cost $V$ is a function of each period $T_i$ of tasks. For the given computation time $C_i^w$ and period $T_i$ of each task with fixed priority, the differentiation process for the overall $V$ with respect to each $T_i$ consists of the following steps:

1. Compute the response time distribution of each task $i$ using the method in Section 4.3;
2. Design the LQG controller using the response-time distribution as the delay distribution, for each task $i$ using the method in Section 4.2;
3. Calculate the LQG cost $V_i$ of each task $i$;
4. Calculate the overall cost $V$ by (4.6).

For the example in Section 4.3, we plotted the overall cost $V$ as a function of periods $T_1$ and $T_2$ (Figure 4.3). The function is non-linear and non-convex, even though there are only two tasks.

## Local Optimization

Gradient based optimization methods could be used to find the local optimal overall cost $V$, but they are time-consuming. So instead, we introduce a sequential search-based optimization method, which is derivative free. The method is presented in Algorithm 4.1. Since low priority tasks are disturbed by interference from high priority tasks the algorithm starts by finding the optimal periods for higher priority tasks and fix them, before proceeding to lower priority tasks. Since larger periods

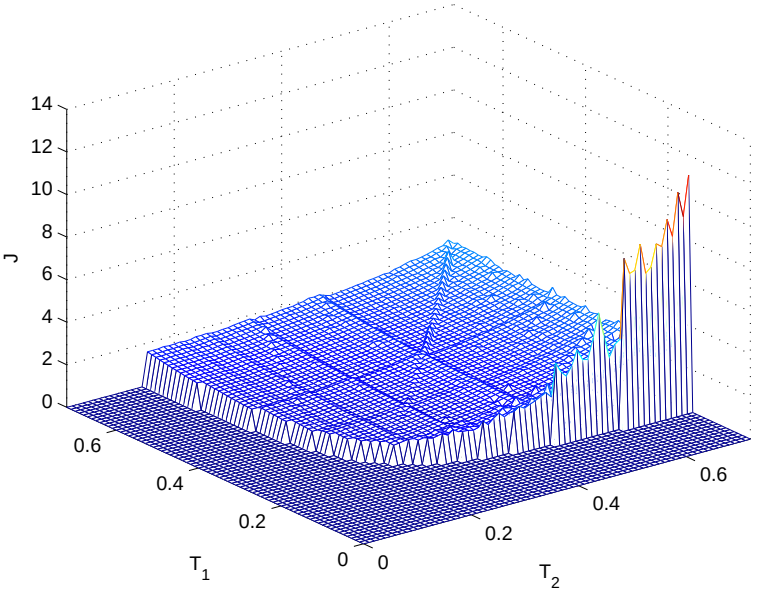



**Figure 4.3** $V$ as a function of periods $T_1$ and $T_2$.

of high priority tasks give more utilization to the low priority tasks, the algorithm begins by trying larger periods and then smaller ones. The utilization requirement $U \leq 1$ is checked before assignment of smaller periods. This heuristic approach is called sequential search in the sequel.

The algorithm starts by calculating the initial overall LQG cost for the initial periods, as shown by line 2 of Algorithm 4.1. The algorithm iterates over all the tasks from task 1 to task $n$. For each task, as shown from line 4 to line 13, we check whether the increased period gives smaller overall LQG cost. If yes, we continue to increase the period until the overall LQG cost increases. If no, we search the period which is shorter than the initial period, as shown from line 14 to line 31. With the decreased period, we first check the overall utilization. If the overall utilization is greater than 1, we skip the current decreased period assignment and stop the search for the current task; if the overall utilization is smaller than or equal to 1, we search the small overall LQG cost until the cost increases.

## Global Optimization

Our interest here is not only in the locally optimal solution. According to the characteristics of the LQG cost as a function of periods, we are certain to say it is a

**Algorithm 4.1** Local optimization

```
1:  procedure SEQUENTIALSEARCH(T_1, T_2, ..., T_n)
2:      calculate V at the initial value T_1, T_2, ..., T_n
3:      for i = 1 : n do
4:          T_i' ← T_i + ΔT_i
5:          calculate V' at T_1, T_2, ..., T_i', ..., T_n
6:          moved ← 0
7:          while V' < V do
8:              T_i ← T_i'
9:              V ← V'
10:             T_i' ← T_i + ΔT_i
11:             calculate V' at T_1, T_2, ..., T_i', ..., T_n
12:             moved ← 1
13:         end while
14:         if moved = 0 then
15:             T_i' ← T_i − ΔT_i
16:             U ← Σ_{j=1}^{n} (C_j^w / T_j) − C_i^w / T_i + C_i^w / T_i'
17:             if U ≤ 1 then
18:                 calculate V' at T_1, T_2, ..., T_i', ..., T_n
19:                 while V' < V do
20:                     T_i ← T_i'
21:                     V ← V'
22:                     T_i' ← T_i − ΔT_i
23:                     U ← Σ_{j=1}^{n} (C_j^w / T_j) − C_i^w / T_i + C_i^w / T_i'
24:                     if U ≤ 1 then
25:                         calculate V' at T_1, T_2, ..., T_i', ..., T_n
26:                     else
27:                         V' ← ∞
28:                     end if
29:                 end while
30:             end if
31:         end if
32:     end for
33:     return T_1, T_2, ..., T_n                    ▷ The local minimum
34: end procedure
```

typical non-convex global optimization problem. So several optimization methods could be applied to its solution. As mentioned, the derivative-based optimization method was avoided, since they are very time-consuming in the current problem.

The DIRECT optimization method [Jones et al., 1993] is a sampling-based search algorithm. It is a global derivative-free method. In the given domain, DIRECT optimization samples some points without an initial value, then decides where to do the next search based on the known information. It is widely used in "black box" problem, in which the relation between inputs and outputs is complex.

## 4.5 Evaluation

In this section the proposed response time driven period assignment method is evaluated comparing with the delay-aware period assignment method (RiApprox) from [Bini and Cervin, 2008]. To better understand the properties of the proposed period assignment method, tasks sets are randomly generated to control randomly generated plants with varying characteristics. The evaluation is performed using TrueTime toolbox.

### Simulation Setup

Plant sets were randomly generated from the following three different plant families.

- Family I: All plants have two stable poles, with each plant drawn with equal probability from

$$\begin{aligned} P_1(s) &= \frac{1}{(s+a_1)(s+a_2)} \\ P_2(s) &= \frac{1}{s^2+2\zeta\omega s+\omega^2} \end{aligned} \tag{4.8}$$

  with $a_1, a_2 \in \text{unif}(0, 1)$, $\omega \in \text{unif}(0, 1)$, $\zeta \in \text{unif}(0, 1)$.

- Family II: All plants have two stable or unstable poles, with each plant drawn with equal probability from

$$\begin{aligned} P_3(s) &= \frac{1}{(s+a_1)(s+a_2)} \\ P_4(s) &= \frac{1}{s^2+2\zeta\omega s+\omega^2} \end{aligned} \tag{4.9}$$

  with $a_1, a_2 \in \text{unif}(-1, 1)$, $\omega \in \text{unif}(0, 1)$, $\zeta \in \text{unif}(-1, 1)$.

- Family III: All plants have three stable or unstable poles, with each plant drawn with equal probability from

$$P_5(s) = \frac{1}{(s+a_1)(s+a_2)(s+a_3)}$$
$$P_6(s) = \frac{1}{(s^2+2\zeta\omega s+\omega^2)(s+a_4)} \tag{4.10}$$

with $a_1, a_2, a_3, a_4 \in \text{unif}(-1, 1)$, $\omega \in \text{unif}(0, 1)$, $\zeta \in \text{unif}(-1, 1)$.

The state-space representation used was the controllable canonical form. For the LQG controllers, $\rho = 0.01$, $R_{1c} = BB^T$, and $R_2 = 0.01\text{tr}\{R_{1c}\}$ were used. Stochastic LQG controllers were designed with the assigned period $T$ and response time distribution. The LQG cost is computed using the same LQG parameters.

The evaluation examined systems of $n \in \{3, 5\}$ control tasks. The nominal task utilization $U_i^{nom}$ were generated using an $n$-dimensional uniform distribution with total utilization 1. The execution time was given by $C_i^w \in \text{unif}(0.04, 0.4)/n$. The task priorities were assigned based on the periods returned by [Seto et al., 1996].

In the optimization procedure, the cost values and the LQG controllers were calculated using Jitterbug toolbox. However, Jitterbug can only design LQG controllers when the delay variation is less than the period. Therefore the obtained delay distributions were truncated from below if the delay variation was larger than the period. The upper limit was set to worst-case response time $R_i^w$, and the lower limit was set to $(R_i^w - T_i)$. The probability of the response time lower than $(R_i^w - T_i)$ was added to the probability at $(R_i^w - T_i)$.

In general, the stability of the plants under control decides the sensitivity towards delay and jitter. Therefore, it can be expected that the cost obtained for Family I has the smallest value, while the cost for Family III has the highest value. This is also shown by the evaluation.

## Evaluation Results

The RiApprox method was used to compute the initial periods. The different period assignment methods were evaluated by Monte Carlo simulation, where the plants (including the disturbances), the controllers, and the scheduler were simulated in parallel using TrueTime. From each family of plants, 10 random plants were generated. After the period assignment the plants and controllers were simulated for 1000 s, and total cost, $J$, was recorded.

The utilization requirement $U \leq 1$ should be always fulfilled. In the sequential search algorithm, when the step has been changed, the utilization is checked to avoid the situation that the utilization is larger than 1. In the DIRECT optimization algorithm, if the utilization is more than 1, the cost is set to a very large number to enforce the optimization result away from this situation.

The evaluation results are based on $n \in \{3, 5\}$ plants and controllers. In each family, the cost values are compared for four period assignment methods:

**Table 4.1** Evaluation of the cost

| | Family I | | Family II | | Family III | |
|---|---|---|---|---|---|---|
| | $n=3$ | $n=5$ | $n=3$ | $n=5$ | $n=3$ | $n=5$ |
| RiApprox | 4.05 | 5.40 | 19.57* | 7.32* | 15.04* | 24.76* |
| Initial | 4.01 | 5.34 | 5.26 | 7.02 | 14.40 | 23.27 |
| Sequential search | 3.86 | 5.25 | 4.92 | 6.63 | 12.32 | 20.01 |
| DIRECT optimization | 3.86 | 5.23 | 4.92 | 6.63 | 12.32 | 19.81 |

- RiApprox by [Bini and Cervin, 2008], in which the LQG controllers are designed for a constant delay and the task periods are obtained from a linear approximation of the cost function;
- Initial, in which the task periods obtained from RiApprox are used but the LQG controllers are redesigned based on the response time distributions;
- Sequential search is the local search algorithm described in Algorithm 4.1, initialized with the periods computed in the previous method; and
- DIRECT optimization that is the global optimization approach proposed by [Jones et al., 1993] and available in MATLAB.

The total costs for the three plant families are shown in Table 4.1.

For Family II and III, the RiApprox method sometimes gives rise to unstable control loops, i.e., infinite cost, shown as *'s in Table 4.1. Therefore, the total cost is divided by the number of plants for which the cost is finite. For Family II, $n = 3$, there is 1 case, out of 10, giving the infinite cost. For Family II, $n = 5$, there are 3. For Family III, $n = 3$, there is 1. For Family III, $n = 5$, there are 3.

The evaluation shows that the response time driven period assignment methods gives better LQG performance than the RiApprox method in all cases. The difference is larger when the task set is large or when the plants are unstable. The sequential search-based method gives results that in all cases are very close to the results obtained by the global DIRECT method.

However, the sequential search-based method requires much fewer computations than the DIRECT method. In order to evaluate this, for each plant family, $n$ plants were randomly generated and $n$ corresponding controllers were designed. The computation times of sequential search and DIRECT optimization were compared using the `tic` and `toc` method in MATLAB. The computation times are shown in Table 4.2 As can be seen the difference is approximately a factor of 2.

## 4.6 Conclusion

In this chapter, we have introduced the response time driven design of the LQG controllers. The major contribution of the chapter is the proposed period assignment

**Table 4.2** Run-time of the methods.

| | | Sequential search | DIRECT method |
|---|---|---|---|
| Family I | $n = 3$ | 8.2912 | 114.2827 |
| | $n = 5$ | 18.4958 | 198.2194 |
| Family II | $n = 3$ | 3.4655 | 123.5941 |
| | $n = 5$ | 11.9846 | 215.4092 |
| Family III | $n = 3$ | 5.0514 | 128.8757 |
| | $n = 5$ | 14.4181 | 266.7887 |

method based on the task response time distribution and the sequential search-based algorithm. Through a simulation-based evaluation, it was shown that the method may lead to very good results. The control performance, as measured by the LQG cost, was improved compared to previous approaches and the local search-based heuristic algorithm gave results that were very close to the global solution, with 50% less computation effort compared to the global optimization method.

# 5

# Periodic LQG Scheduling and Control Codesign*

## 5.1 Introduction

In this chapter, we focus on the response-time variation that arises from preemptive scheduling of a set of periodic tasks. Assuming constant execution times for all tasks, the response times will have a periodic regularity in cases where a hyperperiod exists. If the response-time pattern is known, it can be exploited in the control design.

In reality, task execution times are not constant but may vary due to cache misses, unmodeled hardware interrupts, etc. We can model the execution time of each job of a control task as an independent random variable. The response time of each job will no longer be constant but can be characterized by a probability distribution. There is now a repetitive pattern of the response time distribution over the hyperperiod, and this can also be exploited in the control design.

The approach taken in the current chapter is to exploit the periodic delay pattern that results if the task periods are perturbed slightly, obtaining a finite hyperperiod. This is combined with so-called periodic LQG design techniques, where the periodic delay pattern is taken explicitly into account in the control design.

### Outline

The outline of the rest of this chapter is as follows: Section 5.2 gives a method to achieve a finite hyperperiod. Assuming this method, a periodic LQG control design procedure is given in Section 5.3. Section 5.4 evaluates the periodic LQG control design on three different examples. Section 5.5 discusses statistical response-time analysis vs schedule simulations to find the response-time probability distributions. In Section 5.6, a numerical method to design a periodic–stochastic LQG controller is presented. Section 5.7 evaluates the periodic–stochastic LQG control design on three different examples. Finally, Section 5.8 offers some concluding remarks.

* This chapter is based on [Xu et al., 2015].

## 5.2 Task Period Perturbation

Since the task periods are not necessarily integers, we extend the definition of the hyperperiod of the set of tasks, as follows.

DEFINITION 5.1
The *hyperperiod* of a set of tasks with periods $[T_1, T_2, \ldots, T_n]$, is the smallest $H > 0$ such that

$$\forall i,\ \exists k_i \in \mathbb{N}: \quad k_i T_i = H. \tag{5.1}$$

□

Since we assume that the initial task periods are any real numbers, the hyperperiod $H$ of (5.1) may not exist. In this case we set it to $H = \infty$. We observe that the above definition does not necessarily require the task periods to be integers. For example, if $T_1 = 2\sqrt{2}$ and $T_2 = 3\sqrt{2}$, then according to Definition 5.1 we have that the hyperperiod of $[T_1, T_2]$ is $H = 6\sqrt{2}$. However, Definition 5.1 requires the periods to be commensurate: if $T_1 = 1$ and $T_2 = \sqrt{2}$, then there is no hyperperiod $H$ satisfying (5.1), and we set $H = \infty$.

When the hyperperiod $H$ is large or infinite, it is not practical or feasible to investigate the job response-time pattern over the hyperperiod. We, therefore, propose a method to perturb the periods to obtain a finite and short hyperperiod.

### Finding an Approximate Hyperperiod

When the periods are the solution of an optimization problem such as in scheduling–control codesign, it may indeed happen that the task periods may be real values or, at least, machine-representable "real" values. In these cases, it may be useful to find a suitable substitute for the hyperperiod. We propose the following definition.

DEFINITION 5.2
Given a tolerance $\varepsilon \in (0, 1)$, let $[k_1, k_2, \ldots, k_n] \in \mathbb{N}^n$ be such that

$$1 - \frac{\min_i\{k_i T_i\}}{\max_i\{k_i T_i\}} \leq \varepsilon. \tag{5.2}$$

The *approximate hyperperiod* $\hat{H}$ of the task periods $[T_1, T_2, \ldots, T_n]$ is then

$$\hat{H} = \max_i\{k_i T_i\}. \tag{5.3}$$

□

As $\varepsilon \to 0$, the approximate hyperperiod of Definition 5.2 tends to the actual hyperperiod of Definition 5.1, if the limit exists.

In Table 5.1 we illustrate an example when $T_1 = \sqrt{2}$ and $T_2 = \pi$. The third column reports the values $[k_1, k_2]$ that makes Eq. (5.2) true. It can be observed that, as the tolerance $\varepsilon$ increases the corresponding approximate hyperperiod $\hat{H}$ decreases.

**Table 5.1** Example of approximate hyperperiod $\hat{H}$ when $T_1 = \sqrt{2} \approx 1.4142$ and $T_2 = \pi \approx 3.1416$.

| $\varepsilon$ | $\hat{H}$ | $[k_1, k_2]$ | $[\hat{T}_1, \hat{T}_2]$ |
|---|---|---|---|
| 0 | $\infty$ | $[\infty, \infty]$ | $[\sqrt{2}, \pi]$ |
| 0.001 | 28.284 | [20, 9] | [1.4139, 3.1420] |
| 0.1 | 9.8995 | [7, 3] | [1.3690, 3.1943] |
| 1 | $\max_i\{T_i\} = \pi$ | [1, 1] | [2.5658, 2.5658] |

## Control Task Period Assignment

Let us assume that an optimization-based scheduling–control codesign has been performed, e.g., using the method in [Bini and Cervin, 2008] or in Chapter 4. This leads to a set of real-valued task periods, $\mathbf{T} = [T_1, T_2, \ldots, T_n]$, that give good control performance but for which, typically, no finite hyperperiod exists.

The goal we have is to find some other task periods $\hat{\mathbf{T}} = [\hat{T}_1, \hat{T}_2, \ldots, \hat{T}_n]$ which are "close" to the original values and which have a finite hyperperiod that is not too large.

We propose the following method for period assignment:

1. Set a value of $\varepsilon$ of desired proximity between $\mathbf{T}$ and $\hat{\mathbf{T}}$. A typical value could be between $10^{-3}$ and $10^{-2}$.
2. Compute the set of integers $[k_1, k_2, \ldots, k_n]$ such that (5.2) holds;
3. Calculate the modified periods as

$$\hat{T}_i = \frac{\sum_{j=1}^{n} k_j C_j^w}{k_i}. \tag{5.4}$$

   This choice implies that even the tasks with the modified periods $\hat{T}_i$ are fully utilizing the processor.

4. Redesign the controllers taking the obtained periodicity explicitly into account using a periodic LQG control design scheme. The results of this will be that for each controller the controller parameters will depend on the current job in the hyperperiod.

For the simple example of Table 5.1, if $C_1^w = \sqrt{2}/3$ and $C_2^w = 2\pi/3$, which implies that $\sum_i \frac{C_i^w}{T_i} = 1$, then the task periods $\hat{T}_i$ modified according to (5.4) are reported in the fourth column.

By tuning $\varepsilon$ we can control the length of the hyperperiod and then the length of the pattern of job response times. Also, $\varepsilon$ is used to control the magnitude of the perturbation of the original periods. From (5.2), it follows that

$$\forall i, \quad (1-\varepsilon)\frac{\max_j\{k_j T_j\}}{T_i} \le k_i \le \frac{\max_j\{k_j T_j\}}{T_i}, \tag{5.5}$$

which enables us to find lower and upper bounds to the perturbed $\hat{T}_i$ of (5.4), as follows

$$\forall i, \quad (1-\varepsilon)T_i \sum_{j=1}^{n} U_j \leq \hat{T}_i \leq \frac{1}{1-\varepsilon} T_i \sum_{j=1}^{n} U_j. \tag{5.6}$$

If the original periods $[T_1, T_2, \ldots, T_n]$ are fully utilizing the processor, which is always the case if they are the solution of an optimal real-time control codesign problem [Bini and Cervin, 2008], then the bounds to $\hat{T}_i$ become

$$\forall i, \quad (1-\varepsilon)T_i \leq \hat{T}_i \leq \frac{1}{1-\varepsilon} T_i. \tag{5.7}$$

(5.7) provides an insightful interpretation of $\varepsilon$. It states that by perturbing the periods according to (5.4), with $k_i$ defined by (5.2), then the amount of perturbation can be controlled by $\varepsilon$.

In the real-time control codesign problem, selecting $\varepsilon$ is a trade-off between the tolerable variation of the task periods from the initial periods and the length of the hyperperiod. A long hyperperiod requires more memory to store the controller parameters and a larger (off-line) computational effort to design the controllers.

## 5.3 Periodic LQG Control Design

In a hyperperiod, fixed actual execution times implies that the delays are known, but variable. This section establishes the LQG control design for the resulting periodic system. The modified hyperperiod proposed in the previous section makes it possible to realize this design procedure. Note that existing LQG design methods for time-delay systems do not apply because the delays are either assumed to be constant or in the form of a probability distribution. Here we allow the delays to vary according to a deterministic pattern over a hyperperiod.

### Sampling the Periodic Time Delay System

For task $i$,

$$H = l_i h_i \tag{5.8}$$

where $H$ is the hyperperiod, $h_i$ is the period, and $l_i$ is the number of jobs in one hyperperiod. The controlled plant is given by (2.1). The delays arrive in the deterministic pattern $\delta_{i,1}$, $\delta_{i,2}$, ..., $\delta_{i,l_i}$. The delays can be less than, equal to, or greater than the period. We know, however, that $\delta_{i,j} < \delta_{i,j+1} + h_i$, $j \in \{1, 2, \ldots, l_i - 1\}$, because in a single-CPU system, the finishing times of jobs cannot be disordered. The control signal is assumed to be zero-order hold, i.e. piecewise constant between update instants (see Figure 5.1).

Because the dynamics repeat in every hyperperiod, we only need to investigate the state space model over one hyperperiod. Actually, if there exists a smaller contiguous repeating sub-sequence than the full sequence in the hyperperiod, the

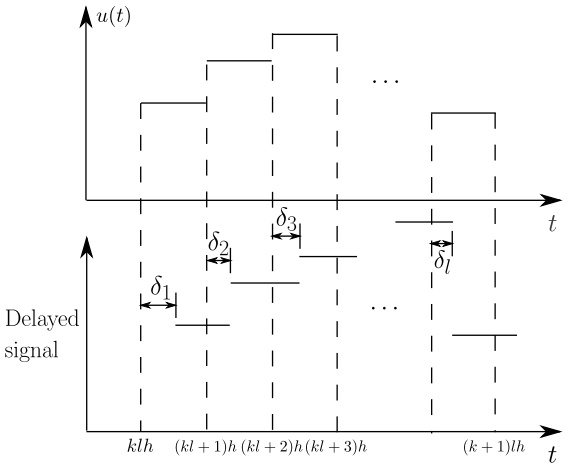



**Figure 5.1** Delayed control signals in one hyperperiod.

controller can be recalculated for the shorter length. The size of the matrices in the state-space model is smaller and the cost reformulation is simpler. However, the controller is the same as the full hyperperiod design. For convenience, the subscripts indicating task $i$ are omitted in the rest of this section.

The integration of (2.1) over one hyperperiod is shown in Appendix B. Following this, an extended state-space model can be introduced,

$$\begin{bmatrix} x((k+1)lh) \\ u'(klh) \end{bmatrix} = \begin{bmatrix} \Phi & \Gamma_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x(klh) \\ u'((k-1)lh) \end{bmatrix} + \begin{bmatrix} \Gamma_0 \\ I \end{bmatrix} u'(klh). \tag{5.9}$$

Here, $l$ control signals over the hyperperiod are included in the extended state. This reformulation changes the system from continuous-time infinite form to discrete-time finite form.

## Sampling the Cost Function

We have so far sampled the plant model, but we also need to sample the cost function. The cost of the $i$´th task is

$$V_i = \sum_{k=0}^{\infty} V_i(k) = \sum_{k=0}^{\infty} V_i(kl_ih_i) \tag{5.10}$$

and

$$\begin{aligned} V(klh) &= \int_{klh}^{(k+1)lh} x^T(t)Q_{1c}x(t) + 2x^T(t)Q_{12c}u(t) + u^T(t)Q_{2c}u(t)dt \\ &:= x^T(klh)Q_1x(klh) + 2x^T(klh)Q_{12}u'(klh) \\ &\qquad + u'^T(klh)Q_2u'(klh) + V_{const}(lh), \end{aligned} \tag{5.11}$$

where $V_{const}(lh)$ is given by (2.16). The discrete-time cost matrices $Q_1$, $Q_2$ and $Q_{12}$ are calculated as shown in Appendix B. Given the extended state-space model, the standard linear quadratic control design method can be applied.

## 5.4 Periodic LQG Control Evaluation

In this section, we first apply the periodic LQG design procedure to a simple example and then to two real-time scheduling and control codesign problems. In the simple example, the design approach will guarantee the correctness of periodic LQG design, without having to consider the exact multitasking behavior in the real-time system. The exact costs can be calculated using Jitterbug. The second example will illustrate how the perturbed period method works. The third evaluation example will provide a systematic approach for periodic LQG design in a real-time multitasking environment. The latter examples involve joint scheduling and control simulations and are evaluated using TrueTime.

### A Simple Example

Assume that a plant under control is given by

$$G(s) = \frac{1}{s^2 - 1}. \tag{5.12}$$

The state-space representation used is the controllable canonical form. The cost function is given by (2.2), in which the continuous-time cost matrix is

$$Q_{1c} = \begin{bmatrix} 0.01 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.13}$$

and $Q_{2c} = 0.01$. The continuous-time state noise covariance matrix is $R_{1c} = BB^T$, and the discrete-time measurement noise covariance matrix is $R_2 = 0.01\text{tr}\{R_{1c}\}$.

The sampling period is $h = 0.3$, and the hyperperiod is assumed to be $H = 4h$. The delays in one hyperperiod are assumed to be $\delta_1 = 0.24$, $\delta_2 = 0.18$, $\delta_3 = 0.12$, $\delta_4 = 0.24$.

LQG controllers are then designed by the following three methods:

- Constant delay: Using Jitterbug, a time-invariant LQG controller is designed assuming a constant delay equal to the average delay.
- Stochastic LQG: Using Jitterbug, a time-invariant LQG controller is designed, viewing the delay as a random variable with known probability mass distribution.
- Periodic LQG: A sequence of $l$ LQG controllers is designed using the periodic LQG control design procedure proposed in Section 5.3.

**Table 5.2** Costs in the simple example.

| | |
|---|---|
| Constant delay design | 3.01 |
| Stochastic LQG design | 2.95 |
| Periodic LQG design | 2.22 |

**Table 5.3** Perturbed periods and costs ($V_{\text{ini}}$ is based on [Bini and Cervin, 2008]).

| | $T_1$ | $T_2$ | $T_3$ | $H$ | $V_{\text{ini}}$ | $V_{\text{LQG}}$ |
|---|---|---|---|---|---|---|
| Initial | 0.2795 | 0.3509 | 0.2792 | $\infty$ | – | – |
| $\varepsilon = 0.1$ | 0.2740 | 0.3653 | 0.2740 | 1.0959 | 5.5697 | 5.2188 |
| $\varepsilon = 0.05$ | 0.2837 | 0.3404 | 0.2837 | 1.7021 | 5.5697 | 5.2061 |

The costs evaluated in Jitterbug are given in Table 5.2. The cost using periodic LQG is better than the other two. This is because it takes all the information about system dynamics, especially about delays, into account in control design procedure. The larger the delay variability, the larger the improvement over previous design techniques will be.

## Three-Plant Example

Assume a set of three tasks controlling three plants

$$\begin{aligned} P_1(s) &= \frac{1}{s^2 + 0.549s - 0.1979} \\ P_2(s) &= \frac{1}{s^2 - 0.9947s + 0.2366} \\ P_3(s) &= \frac{1}{s^2 + 0.711s + 0.0252}. \end{aligned} \tag{5.14}$$

We start by assigning initial periods with the approach proposed in [Bini and Cervin, 2008]. The initial periods are shown in Table 5.3, where all periods have been rounded to four decimal places. The hyperperiod is infinite. Then we modify the periods by selecting the tolerance as $\varepsilon = 0.1$ or $\varepsilon = 0.05$. Now the hyperperiods are finite. As shown in Section 5.2, the lower the $\varepsilon$ value, the longer the hyperperiod. The utilization is kept at $U = 0.98$ in order to avoid numerical errors in the simulation.

We then perturb the periods to obtain a finite hyperperiod, calculate the actual job response times and redesign the controllers according to periodic LQG method. The results are shown in Table 5.3. The initial method from [Bini and Cervin, 2008] uses a constant delay LQG design based on the approximate average response time. Periodic LQG method gives better results than the initial method and even better performance when the tolerance $\varepsilon$ is small. The latter implies a longer hyperperiod but enables a closer-to-optimal resource distribution.

**Table 5.4** Evaluation of costs for codesign for random plants (initial design is based on [Bini and Cervin, 2008]).

| | Family I | Family II | Family III |
|---|---|---|---|
| Initial design | 3.23 | 4.56 | 15.41 |
| Periodic LQG | 3.18 | 4.41 | 12.66 |

## Evaluation on Randomly Generated Plants

In this codesign example, we generate plants randomly from three plant families, which are the same as in Section 4.5. We assume $Q_{1c}$, $Q_{2c}$, $R_{1c}$, and $R_2$ to be the same as those in the simple example above.

The evaluation examines systems with $n = 3$ control tasks. The nominal task utilizations $U_i^{nom}$ are generated randomly using an $n$-dimensional uniform distribution with total utilization 1. The worst-case execution time was drawn from $C_i^w \in \text{unif}(0.04, 0.4)/n$. The task priorities were assigned using rate-monotonic ordering based on the periods returned by [Seto et al., 1996].

The initial task periods are assigned by the method in [Bini and Cervin, 2008] with utilization 0.99, and the initial controllers are designed based on a constant delay equal to the average response time. To avoid numerical errors, the utilization is set to 0.99.

Then perturbed periods are calculated to get a finite hyperperiod, and the delays in a period are derived from a schedule simulation. Finally, a periodic LQG controller is designed for each plant.

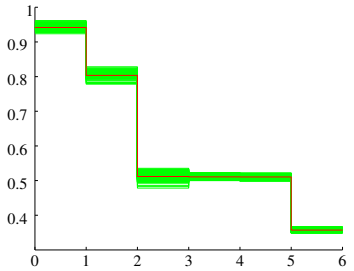
The two control design methods are evaluated by Monte Carlo simulations, where the plants, the controllers, and the scheduler are simulated in parallel using TrueTime. From each family of plants, 10 sets of plants are randomly generated. After the controller design, the plants and controllers are simulated for 1000 s, and the total cost, $J$, was recorded. All the costs are given in Table 5.4.

The cost for periodic LQG control is lower than the cost for the initial design. This is reasonable since the periodic LQG controller takes more information about the system dynamics into account.

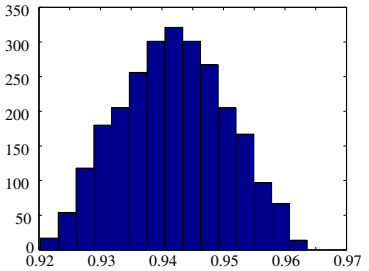
## Limitations of Periodic LQG Control

As shown in the previous evaluations, periodic LQG control gives a good performance as long as the task execution times are constant. However, when the execution times vary, then a periodically repeating delay pattern no longer exists, and the performance obtained using the periodic LQG controller decreases.

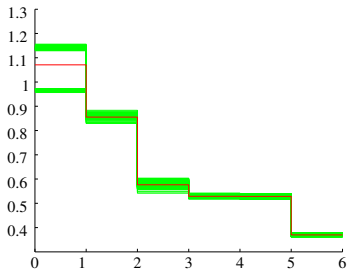
The extent of this performance decrease depends on the shape of the delay distribution function, which in turn depends on the execution time distribution functions of the task under investigation and all higher-priority tasks. Consider the following examples. Assume that a certain task of low priority has six jobs in a hyperperiod. In Figure 5.2(a) the job response times, i.e., delays, are plotted for a large num-



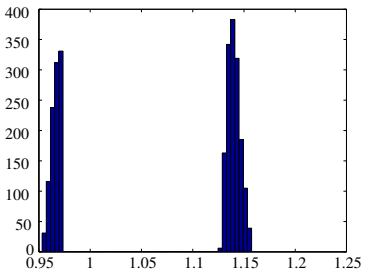

(a) Job response times over a hyperperiod in the first example.




(b) Response time distribution for job 1 in the first example.




(c) Job response times over a hyperperiod in the second example.




(d) Response time distribution for job 1 in the second example.

**Figure 5.2** Two examples showing different job response time distributions.

ber of hyperperiods. The x-axis shows the current job index in the hyperperiod and the y-axis shows the corresponding delay. The red line shows the average response time, and the green lines show the individual response times for each job in the hyperperiods. Figure 5.2(b) shows the response time distribution of the first job in the hyperperiod. With this narrow, unimodal shape of the distribution, it is likely that a periodic LQG controller designed for the average delay in each job would still work reasonably well. However, in another example, using slightly different task parameters, the results may be as presented in Figure 5.2(c) and Figure 5.2(d). Here the response-time distribution is bimodal and the average delay is no longer a good approximation of the true delay and it is less likely that a periodic LQG control will perform well.

In the following sections, we will investigate how a periodic–stochastic LQG control design can be used to cope with this problem. In this type of LQG controller, the delay is modeled by a probability distribution for each job in the hyperperiod. Hence, rather than having delays that repeat periodically, we will have delay distributions that repeat periodically. In order to use this technique, the delay distribution for each job must be known at design time. This is studied next.

## 5.5 Calculation of Job Response-Time Distributions

There are two approaches to obtaining the job response time cumulative distribution functions. The first and most accurate way is to calculate them analytically using some statistical response time analysis tool. The alternative is to simulate the task schedule using a real-time schedule simulator for sufficiently long time and measure the individual delays. Similar to measurement-based WCET analysis the latter approach always runs the risk of not encountering response times that have low probability. Here, both approaches will be investigated. The response time analysis framework considered is the one described in [Tanasa et al., 2015].

### A Framework for Probabilistic Response-Time Analysis

We now describe the framework used to accurately compute the response time distributions of tasks scheduled on uniprocessor platforms based on non-idling, preemptive scheduling policies [Tanasa et al., 2015]. With respect to scheduling policies, the framework is general and covers both static and dynamic scheduling algorithms like fixed priority scheduling and earliest deadline first scheduling.

Besides the scheduling policy, the framework takes as inputs the relevant tasks parameters, which include **i)** periods and **ii)** variable execution times. The main assumption on which the framework is built is that the tasks' jobs are independent of each other. Thus, independent random variables can be used to model the variable execution times of all jobs. As such, jobs belonging to the same task will be modeled with independent identically distributed random variables. The framework also assumes that **i)** the execution times of the tasks are described by probability density functions (PDFs) and that **ii)** the maximum CPU utilization does not exceed 100%. Given all these assumptions the framework accurately computes the response time distributions of the tasks by conducting a job-level analysis over all the jobs in the hyperperiod.

Internally, the framework tightly approximates all the PDFs with polynomial functions. This is because polynomial functions can be integrated analytically in polynomial time given their degrees. Also, for the uniform PDFs used in this chapter, the polynomial approximations are exact. For each job in the hyperperiod, the framework identifies a set of non-idling scenarios where each such scenario covers a continuous interval of possible response times that the job under analysis might have. In this way, the response time distribution of the job under analysis can be evaluated by integrating the joint execution time function of those jobs that it interacts with (i.e. jobs with higher priority) over the points in the current interval. We illustrate the idea with the help of an example below.

### Response-Time Analysis Example

We provide an example describing the functionality of the tool used to compute the response time distributions of the tasks scheduled on uniprocessor platforms based on non-idling, preemptive scheduling policies. Let us assume a task set with three

**Table 5.5** Task parameters in the probabilistic response-time analysis example.

| Task index | $T$ | $D$ | $C^b$ | $C^w$ |
|---|---|---|---|---|
| 1 | 5 | 5 | 1 | 2 |
| 2 | 6 | 6 | 1 | 2 |
| 3 | 9 | 9 | 1 | 2 |

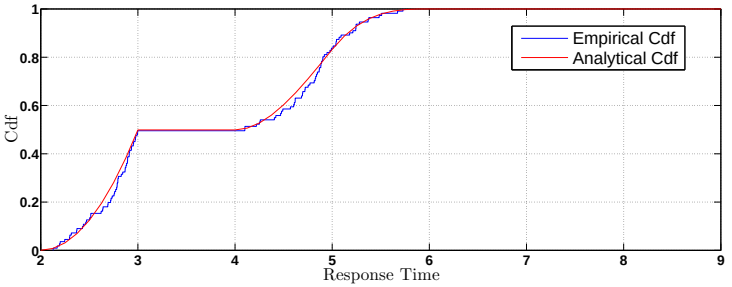



**Figure 5.3** Cumulative response time distribution of the $10^{\text{th}}$ job in the probabilistic response-time analysis example.

tasks. The tasks' parameters are presented in Table 5.5 and it is assumed that the jobs are scheduled with fixed priorities implied by the task ordering and that the random execution times are uniformly distributed. Given the parameters, a total of 43 jobs are released in any hyperperiod. In the following, we show the response time distribution of the $10^{\text{th}}$ job, corresponding to the second job of the third task.

The cumulative response time distribution of the $10^{\text{th}}$ job (see Figure 5.3) has been evaluated in two ways. First, we have conducted simulations in TrueTime and, later, we have used the analytical tool. The red line depicts the response time distribution obtained from task simulations while the blue line denotes the analytical distribution. It can be seen that the two graphs match each other very well apart from one exception. The simulations have shown that the response time distribution of the $10^{\text{th}}$ job spreads over the interval $[2..3] \cup [4..6]$. On the other hand, the analytical tool reports that the same distribution has the domain $[2..3] \cup [4..6] \cup [7..9]$. Such discrepancies are likely to occur as, in our case, the probability for the response time of the $10^{\text{th}}$ job to lie in the interval $[7..9]$ is very small (less than 0.01).

In this section, it has been shown that the response time distribution calculated using TrueTime-based schedule simulation is similar to, but not exactly equivalent to, the analytical solution. With this as motivation, we still will use TrueTime to derive the response time PDFs in the following sections. The reason for not using the analytical calculation framework is that for certain task parameter values the memory requirements are too large.

## 5.6 Periodic–Stochastic LQG Control Design

This section presents a numerical method for LQG control design when the execution time for each task is varying randomly. It is assumed that a hyperperiod exists and that the response time PDF has been derived for each job of the task. As usual, the LQG design can be split into two parts: state feedback design and state estimator design.

### State Feedback Design

In the state feedback design, it is assumed that the full state vector $x$ is known at each sampling instant. The goal is to design a time-varying control law

$$u[k] = -L[k]x[k] \tag{5.15}$$

where the state feedback gain $L$ varies over the hyperperiod. Given knowledge of the PDFs of the delays $L_k$, we formulate a periodic–stochastic Riccati equation as

$$S[k] = \mathop{\mathrm{E}}_{\delta_{k+1}} \Big\{ \Phi^T S[k+1]\Phi + Q_1 - \left(\Phi^T S[k+1]\Gamma + Q_{12}\right) \times \left(\Gamma^T S[k+1]\Gamma + Q_2\right)^{-1} \left(\Gamma^T S[k+1]\Phi + Q_{12}^T\right) \Big\} \tag{5.16}$$

where matrices $\Phi$, $\Gamma$ and $Q$ are all functions of $L_{k+1}$. They can be calculated using the method in Section 5.3. The Riccati equation can be solved iteratively by calculating $S[l]$, $S[l-1]$, …, $S[1]$, and then repeating again from $S[l]$ and so on, until the sequence of matrices $S[k]$, $k = 1, 2, \ldots, l$, converges. The state feedback gain is then obtained as

$$L[k] = \mathop{\mathrm{E}}_{\delta_{k+1}} \left(\Gamma^T S[k+1]\Gamma + Q_2\right)^{-1} \left(\Gamma^T S[k+1]\Phi + Q_{12}^T\right). \tag{5.17}$$

### State Estimator Design

Since the system we are considering in (2.1) is linear and time-varying, and $v_{ci}$, $C_i^w$ are Gaussian, a time-varying Kalman filter is the optimal estimator. Since the sampling period is fixed, the Kalman filter gain $K$ can actually be obtained by solving a regular Riccati equation, yielding the standard update equation

$$\hat{x}[k \mid k] = \hat{x}[k \mid k-1] + K(y[k] - C\hat{x}[k \mid k-1]) \tag{5.18}$$

where $\hat{x}$ is the estimated state vector. The state prediction equation will however be time-varying and is obtained by taking the expected value over the delay for each job:

$$\hat{x}[k+1 \mid k] = \mathop{\mathrm{E}}_{\delta_k} \Phi\hat{x}[k \mid k] + \mathop{\mathrm{E}}_{\delta_k} \Gamma u[k]. \tag{5.19}$$

Combining the state estimator with the state feedback, we finally obtain the periodic LQG control law

$$u[k] = -L[k]\hat{x}[k \mid k]. \tag{5.20}$$

## 5.7 Periodic–Stochastic LQG Control Evaluation

In this section, we first apply the periodic–stochastic LQG design to a simple example and calculate the exact costs using Jitterbug. We then look at two codesign examples where performance was evaluated using simulations in TrueTime.

### A Simple Example

The plant is an inverted pendulum, which is the same as in Section 5.4. We also have the same cost matrix $Q_c$, and noise covariance matrices $R_{1c}$ and $R_2$. The sampling period is $h = 0.3$, and the hyperperiod is $H = 3h$. The control delays of each control task job in one hyperperiod are given in the form of probability mass functions as

$$\begin{aligned} \delta_1 &= \begin{bmatrix} 0.12 & 0.4 \\ 0.15 & 0.2 \\ 0.18 & 0.4 \end{bmatrix} \\ \delta_2 &= \begin{bmatrix} 0.15 & 0.5 \\ 0.18 & 0.5 \end{bmatrix} \\ \delta_3 &= \begin{bmatrix} 0.12 & 1 \end{bmatrix}. \end{aligned} \tag{5.21}$$

Here, the left column is the length of response time, and the right column is the corresponding probability.

LQG controllers are designed by the following three methods:

- Periodic–stochastic LQG: Design a set of controllers using the method proposed in Section 5.6.
- Periodic LQG: Using the average response time as the delay in each period, design a set of LQG controllers using the method from Section 5.3.
- Stochastic LQG: Design a time-invariant controller based on the overall delay distribution.

The costs, evaluated in Jitterbug, are shown in Table 5.6. The periodic–stochastic LQG has a lower cost than the other three methods because it takes all the information about the response times into account in the control design procedure. The other two methods work with different kinds of approximations of the response times and are, hence, suboptimal.

**Table 5.6** Costs in the simple example and the three-plant example.

| Design methods | Simple | Three-plant |
|---|---|---|
| Periodic–stochastic LQG | 0.92 | 5.42 |
| Periodic LQG | 1.01 | 6.84 |
| Stochastic LQG | 0.99 | 16.57 |

## Three-Plant Example

Assume a set of three tasks controlling three plants

$$\begin{aligned} P_1(s) &= \frac{1}{s^2 - 0.068s - 0.007} \\ P_2(s) &= \frac{1}{s^2 + 1.048s - 0.640} \\ P_3(s) &= \frac{1}{s^2 - 0.442s + 0.397} \end{aligned} \tag{5.22}$$

to be controlled in three tasks in a real-time system. $P_1$ has the highest priority, while $P_3$ has the lowest priority.

We start by assigning the initial periods with the approach proposed in [Bini and Cervin, 2008]. We then use perturbed periods to obtain a finite hyperperiod. The utilization is set to 0.95, in order to avoid excessive control delays. We assume that the actual execution times are random variables, where the execution time of each job is drawn from $\text{unif}(0.9C^w, C^w)$. The job response time PDFs are then derived from TrueTime simulations. We evaluate the same three methods as outlined in the previous subsection. The costs obtained from evaluations using TrueTime are shown in Table 5.6. Again it is seen that the periodic–stochastic LQG design is better than the other two methods.

## Evaluation on Randomly Generated Plants

We here consider scheduling and control codesign for sets of randomly generated plants. The same kind of plants as in Section 5.4 are used, and again the evaluation examines systems of $n = 3$ control tasks. The nominal task utilization $U_i^{nom}$ is generated from an $n$-dimensional uniform distribution with total utilization 0.95. The worst-case execution time is generated from $C_i^w \in \text{unif}(0.04, 0.4)/n$. The task priorities are assigned based on the periods returned by [Seto et al., 1996].

First, the initial task periods are assigned by the method in [Bini and Cervin, 2008]. The periods are then perturbed to get a finite hyperperiod. The utilization is throughout kept at 0.95 to avoid excessive control delays. The controllers are again designed by the three methods outlined in Section 5.7.

The different control design methods are evaluated by Monte Carlo simulation, where the plants, the controllers, and the scheduler are simulated in parallel using TrueTime. From each family of plants, 10 sets of random plants are generated. After

**Table 5.7** Evaluation of the costs for random plants.

| | Family I | Family II | Family III |
|---|---|---|---|
| Periodic–stochastic LQG design | 3.71 | 5.36 | 13.07 |
| Periodic LQG design | 4.01 | 9.00(5) | 13.49(3) |
| Stochastic LQG design | 5.48(1) | 7.91(5) | 22.74(2) |

control design, the plants and controllers are simulated for 1000 s, and total cost, $V$, is recorded. All the costs are given in Table 5.7. The actual job execution times are random variables drawn from unif$(0.9C^w, C^w)$.

The numbers in the parenthesis indicate how many times the cost is infinite due to an unstable control loop. The mean value of costs does not include the infinite costs. We can see that the cost with periodic–stochastic LQG controller has a lower value than costs with the other two methods, even when the unstable cases have been discounted. The periodic–stochastic LQG is the only method to obtain 100% stable systems for families II and III.

## 5.8 Conclusion

We have proposed new periodic and periodic–stochastic LQG control designs for minimizing the overall cost in real-time control systems. The approaches rely on knowledge of the response-time pattern of each task. In the case of periodic–stochastic LQG, also knowledge of the PDF of the response time of each job is required. To target large systems, there is a need for more efficient tools for statistical response-time analysis. Also, non-control tasks with possibly unknown phasings should ideally also be included in the analysis.

# 6

# Harmonic LQG Scheduling and Control Codesign*

## 6.1 Introduction

Similar to Chapter 5, in the current chapter the focus is again on perturbing the task periods, but in this case to make the task periods harmonic. Harmonic task sets have many attractive properties. As long as the total utilization is less than or equal to 1, the task set is schedulable under both rate-monotonic (RM) and earliest deadline first (EDF) scheduling. Also, assuming constant execution times, the response times and start latencies are constant, which leads to a particularly simple LQG control design. Consider the simple example shown in Figure 6.1. In both cases we have two controller tasks running under RM priority assignment, where the sampling is performed at the job arrival times and the actuation is performed at the job finishing times, i.e., the control delay equals the response times. In the upper plot, the tasks have periods $\{3, 5\}$ and constant execution times $\{1, 3\}$. We assume that these periods have been chosen to give good total control performance. From the figure, one can see that the response time of task $\tau_2$ varies according to the pattern $5, 4, 4, 5, 4, 4 \ldots$. If one changes the period of task $\tau_2$ to 6, i.e., harmonizes the periods, then, as shown in the lower plot, the response time will be always equal to 5, i.e., more deterministic than in the first case. This will most likely lead to worse control performance since both the period and the average delay are larger than before. However, if one also introduces an offset of 1 for $\tau_2$ then the response time will always be equal to 4. The question is then whether the decrease in control performance of the controller executing as task $\tau_2$ caused by the longer sampling period is compensated for by the increased performance caused by the shorter and constant delay obtained through the period harmonization. This is the essence of the problem that we are investigating in this chapter, and the evaluations performed show that this is generally the case.

* This chapter is based on [Xu et al., 2016a] and [Xu et al., 2016b].

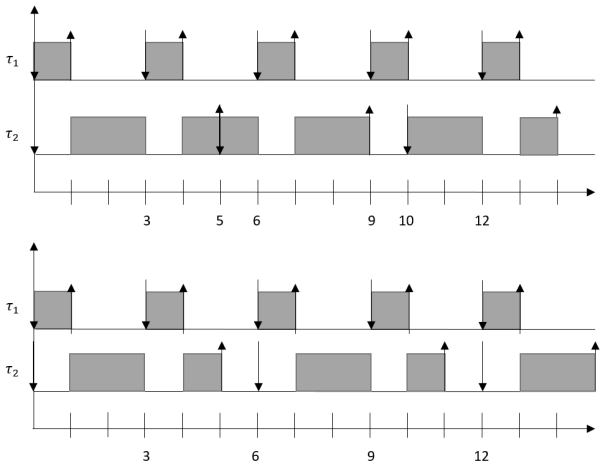



**Figure 6.1** Two tasks with non-harmonic periods (upper plot) and harmonic periods (lower plot). The job arrivals are shown with down-arrows and the job finishing times with up-arrows.

In the chapter, we present some new results on task period harmonization and scheduling analysis for harmonic tasks, with the twin goals of simplifying the control design and improving the performance of multi-loop control systems.

### Outline

Section 6.2 provides the harmonic response-time analysis. The two algorithms to harmonize a non-harmonic task set are presented in Section 6.3. We present the harmonic codesign procedure in Section 6.4. In Section 6.5, the harmonic task period assignment is evaluated with and without task offsets and is compared to the non-harmonic period assignment. Finally, in Section 6.6, some concluding remarks are given.

## 6.2 Scheduling Analysis for Harmonic Tasks

Below we first give an algebraic characterization of the task periods under full utilization. After stating some general scheduling properties, we then give the new response-time analysis for harmonic tasks.

### Characterization of Full-Utilization Harmonic Task Sets

Assume a harmonic task set consisting of $n$ tasks and with full utilization, i.e.,

$$\frac{C_1^w}{T_1} + \frac{C_2^w}{T_2} + \cdots + \frac{C_n^w}{T_n} = 1. \tag{6.1}$$

The fact that the task periods are harmonic can be expressed as $T_{k+1} = m_k T_k$, $m_k \in \mathbb{N}^+$, $k \in \{1, 2, \cdots, n-1\}$. Using this one obtains

$$\frac{C_1^w}{T_1} + \frac{C_2^w}{m_1 T_1} + \frac{C_3^w}{m_1 m_2 T_1} + \cdots + \frac{C_n^w}{m_1 m_2 \cdots m_{n-1} T_1} = 1. \tag{6.2}$$

Solving for $T_1$ gives

$$T_1 = C_1^w + \frac{C_2^w}{m_1} + \frac{C_3^w}{m_1 m_2} + \cdots + \frac{C_n^w}{m_1 m_2 \cdots m_{n-1}}. \tag{6.3}$$

From the harmonicity it follows that

$$\begin{aligned} T_2 &= m_1 C_1^w + C_2^w + \frac{C_3^w}{m_2} + \frac{C_4^w}{m_2 m_3} + \cdots + \frac{C_n^w}{m_2 m_3 \cdots m_{n-1}} \\ &\vdots \\ T_n &= m_1 m_2 \cdots m_{n-1} C_1^w + m_2 m_3 \cdots m_{n-1} C_2^w + \cdots + C_n^w. \end{aligned} \tag{6.4}$$

This can be written on matrix form as

$$T = MC^w \tag{6.5}$$

with $T = \begin{bmatrix} T_1 & T_2 & \cdots & T_n \end{bmatrix}^T$ and with $M$ being the reciprocal matrix given by

$$M = \begin{bmatrix} 1 \\ m_1 \\ \vdots \\ m_1 m_2 \cdots m_{n-1} \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{m_1} & \cdots & \frac{1}{m_1 m_2 \cdots m_{n-1}} \end{bmatrix}. \tag{6.6}$$

Algebraically, $T$ is a linear combination of the execution times $C^w$. Geometrically, $M$ is a linear projection of $C^w$ onto a line $L$ in the coordinate space, running through the origin. The vector $\begin{bmatrix} 1 & m_1 & m_1 m_2 & \cdots & m_1 m_2 \cdots m_{n-1} \end{bmatrix}^T$ lies on $L$. This is not an orthogonal projection. For given execution times $\{C_1^w, C_2^w, \cdots, C_n^w\}$ and $\{m_1, m_2, \cdots, m_{n-1}\}$ the harmonic periods can be calculated using (6.5).

## Scheduling Analysis for Harmonic Tasks

Harmonic tasks have several properties that are appealing from a real-time control perspective [Lehoczky et al., 1989].

1. Schedulability is guaranteed as long as the total utilization is less than or equal to 1.

2. RM and EDF scheduling yields the same start latencies and response times for each task. Under EDF, we assume that the tasks have implicit relative deadlines equal to their periods, and that, if task $\tau_i$ and task $\tau_j$ ($i < j$) have a deadline tie, then EDF will schedule task $\tau_i$.

Furthermore, under the assumption that execution times are constant for each task, it holds that

3. The response time for each task, $R_i$, is constant.
4. The start latency for each task is constant, which is denoted as $S_i$ for task $\tau_i$, and is given by the response time of task $\tau_{i-1}$, with $\tau_i$ being the current task.

The fact that the start latency of task $i$ is equal to the worst-case response time of task $i-1$ implies that the delay can be large, which may degrade the control performance. This can be remedied by adding an offset to each task, so that the job start time always coincides with the job release time.

The following theorem can be used to calculate the response times under both RM and EDF scheduling when the periods are harmonic, using the properties presented above. For convenience, let $m_0 = 1$.

THEOREM 6.1
For RM and EDF scheduling with harmonic periods, given a set of $n$ tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ with utilization $\sum U \leq 1$, there exists a smallest $\hat{m} \in \{1, 2, \ldots, \prod_{i=0}^{j-1} m_i\}$ such that

$$\sum_{i=1}^{j} \left( \left\lceil \frac{m}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) - mT_1 \leq 0 \tag{6.7}$$

for any $m \geq \hat{m}$. Then the response time of task $\tau_j$ ($j \leq n$) is

$$R_j = \sum_{i=1}^{j} \left( \left\lceil \frac{\hat{m}}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right). \tag{6.8}$$

□

***Proof*** To prove the existence of $\hat{m}$, we assume the extreme case $\hat{m} = \prod_{i=0}^{j-1} m_k$ and the utilization bound

$$\sum_{i=1}^{j} U_i = \sum_{i=1}^{j} \frac{C_i^w}{T_i} \leq 1. \tag{6.9}$$

Then

$$\begin{aligned} \sum_{i=1}^{j} \left( \left\lceil \frac{\hat{m}}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) - \hat{m}T_1 &= \left( \sum_{i=1}^{j} \frac{C_i^w}{\left(\prod_{k=0}^{i-1} m_k\right) T_1} - 1 \right) \hat{m}T_1 \\ &= \left( \sum_{i=1}^{j} \frac{C_i^w}{T_i} - 1 \right) \hat{m}T_1 \\ &\leq 0 \end{aligned} \tag{6.10}$$

and, hence, the existence of $\hat{m}$ is proved.

Let $R_i$ be the response time of task $\tau_i$, then when $0 \leq t \leq R_i$, the CPU is fully loaded, so

$$\sum_{i=1}^{j} \left( \left\lceil \frac{m}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) - mT_1 > 0 \tag{6.11}$$

when $m < \hat{m}$. When $m = \hat{m}$,

$$\sum_{i=1}^{j} \left( \left\lceil \frac{m}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) - mT_1 \leq 0 \tag{6.12}$$

and the response time of task $\tau_j$ is

$$R_j = \sum_{i=1}^{j} \left( \left\lceil \frac{\hat{m}}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right). \tag{6.13}$$

□

The procedure to calculate the response time of task $\tau_j$ follows from Theorem 6.1. When $m$ is chosen from $1, 2, \ldots, \prod_{i=0}^{j-1} m_i$, we evaluate

$$\sum_{i=1}^{j} \left( \left\lfloor \frac{m}{\prod_{k=0}^{i-1} m_k} \right\rfloor C_i^w \right) - mT_1 \tag{6.14}$$

and find $\hat{m}$ as the first $m$ that makes this term negative. The procedure is summarized in Algorithm 6.1.

In Algorithm 6.1, a linear search is used. In line 5, the initial value of $m$ is given by

$$m_{\text{initial}} = m_{\text{lower}} = \left\lfloor \frac{\left(1 - \sum_{i=1}^{j-1} \frac{C_i^w}{T_i}\right) T_j}{C_j^w} \right\rfloor \prod_{i=0}^{j-2} m_i \tag{6.15}$$

and it is increased by 1 every step, until $\hat{m}$ is found, as shown in the while loop from line 6 to line 8. By defining

$$m_{\text{upper}} = \left\lceil \frac{\left(1 - \sum_{i=1}^{j-1} \frac{C_i^w}{T_i}\right) T_j}{C_j^w} \right\rceil \prod_{i=0}^{j-2} m_i, \tag{6.16}$$

it follows that $m_{\text{lower}} \leq \hat{m} \leq m_{\text{upper}}$. Furthermore, binary search on $\left[m_{\text{lower}}, m_{\text{upper}}\right]$ can be applied to improve the algorithm performance.

EXAMPLE 6.1

Assume that we have three tasks with execution times $C = \begin{bmatrix} 0.9 & 6.3 & 9.1 \end{bmatrix}$ and that $T_1 = 7.7$, $m_1 = 2$, and $m_2 = 3$, i.e., $T = \begin{bmatrix} 7.7 & 15.4 & 46.2 \end{bmatrix}$. Assume that we want to calculate the response time of task $\tau_3$. Using Algorithm 6.1, one then obtains

$$m_{\text{initial}} = \left\lfloor \frac{\left(1 - \sum_{i=1}^{2} \frac{C_i^w}{T_i}\right) T_3}{C_3^w} \right\rfloor \prod_{i=0}^{1} m_i = 2. \tag{6.17}$$

**Algorithm 6.1** Response time calculation

**procedure** RESPONSETIME
  calculate $R_j$ for $m_0, m_1, \ldots, m_{j-1}, C_1^w, C_2^w, \ldots, C_j^w, T_1$
  $T' \leftarrow MC$
  $T \leftarrow \frac{T_1}{T_1'} T'$
  $m \leftarrow \left\lfloor \frac{\left(1 - \sum_{i=1}^{j-1} \frac{C_i^w}{T_i}\right) T_j}{C_j^w} \right\rfloor \prod_{i=0}^{j-2} m_i$
  **while** $\sum_{i=1}^{j} \left( \left\lceil \frac{m}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) - mT_1 > 0$ **do**
    $m \leftarrow m + 1$
  **end while**
  $\hat{m} \leftarrow m$
  $R_j = \sum_{i=1}^{j} \left( \left\lceil \frac{\hat{m}}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right)$
  **return** $R_j$ ▷ Response time $R_j$
**end procedure**

The search in the while loop in Algorithm 6.1 gives, when $m = 3$,

$$\sum_{i=1}^{3} \left( \left\lceil \frac{m}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) - mT_1 = 2.2 > 0 \tag{6.18}$$

and when $m = 4$,

$$\sum_{i=1}^{3} \left( \left\lceil \frac{m}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) - mT_1 = -5.5 < 0. \tag{6.19}$$

This implies that $\hat{m} = 3$ and that

$$R_3 = \sum_{i=1}^{3} \left( \left\lceil \frac{\hat{m}}{\prod_{k=0}^{i-1} m_k} \right\rceil C_i^w \right) = 25.3. \tag{6.20}$$

□

The time complexity of Algorithm 6.1 can be compared with the fixed priority scheduling response-time calculation algorithm in [Bonifaci et al., 2013]. In [Bonifaci et al., 2013], the authors assume that the task periods and the task execution times are integers. The harmonic periods are defined as $T_i | T_j$ ($T_i$ divides $T_j$) or $T_j | T_i$ ($T_j$ divides $T_i$) for all $i, j = 1, 2, \ldots, n$. The algorithm correctly computes the response time of task $\tau_n$ in $\mathcal{O}(n \log n + n \log P)$ where $n$ is the number of tasks and $P = \max_{i=1}^{n} T_i$. In Algorithm 6.1 task periods and execution times may all be real-valued. The while loop in the algorithm will, in the worst case, run $\prod_{k=0}^{i-1} m_k$ times.

The time complexity is $\mathcal{O}\left(\prod_{k=0}^{i-1} m_k\right)$ in the normal case and $\mathcal{O}\left(\log \prod_{k=0}^{i-1} m_k\right)$ if binary search is used.

Due to the different setups for the response time calculation in [Bonifaci et al., 2013] and this chapter, one has to make adjustments to the proposed method. In order to make a fair comparison one may assume that the periods are all integers. Then it follows that $\prod_{k=0}^{i-1} m_k \leq P$, from which it follows that $\mathcal{O}\left(\log \prod_{k=0}^{i-1} m_k\right) \leq \mathcal{O}(n \log n + n \log P)$ for any $n$, i.e. the complexity of the proposed algorithm is lower than what has been previously presented.

## 6.3 Finding Harmonic Control Task Periods

In scheduling–control codesign, it is typically assumed that the period of each control task can be chosen as a real value within a (possibly infinite) period range. In a prototypical problem formulation, the performance of each controller is described by a cost function $V(T)$, which is assumed to be an increasing function of the task period $T$, i.e., the lower the cost, the better the performance will be. The goal is to optimize the combined performance of all control tasks subject to a utilization constraint $U_b \leq 1$, e.g.,

$$\begin{aligned} &\underset{T_1, T_2, \ldots, T_n}{\text{minimize}}\ J = \sum_{i=1}^{n} V(T_i) \\ &\text{subject to } \sum \frac{C_i^w}{T_i} \leq U_b. \end{aligned} \tag{6.21}$$

If the function $V(T^{-1})$ is convex, efficient numerical methods are available to find the global optimum [Eker et al., 2000].

Here, we are interested in solving a similar codesign problem, but we want to restrict the possible task periods to be harmonic. However, optimization problems involving integers (i.e., the harmonic factors $m_1$, $m_2$, …, $m_{n-1}$ in our case) are in general NP-hard [Papadimitriou, 1981], meaning that the optimal harmonic period assignment codesign problem cannot be solved efficiently. Hence, we propose the following two heuristic approaches to the harmonic control task period assignment: 1) finding the closest harmonic period assignment to a set of initial periods, and 2) finding all possible harmonic period assignments that satisfy given task period ranges. In both cases, all feasible candidate solutions are then evaluated with regards to the combined control performance and the best solution is chosen.

### Finding the Closest Harmonic Task Periods

We assume that a set of full-utilization initial non-harmonic task periods are given as $T^0 = \begin{bmatrix} T_1^0 & T_2^0 & \ldots & T_n^0 \end{bmatrix}^T$. The problem is then to find a set of harmonic periods

that minimizes the Euclidean distance between this set and the initial periods:

$$\begin{aligned} &\underset{T_1,T_2,\ldots,T_n}{\text{minimize}} \ \left\|T - T^0\right\| \\ &\text{subject to } \sum_{i=1}^{n} \frac{C_i^w}{T_i} = 1, \ \frac{T_{k+1}}{T_k} \in \mathbb{N}^+, \ k \in \{1,2,\ldots,n-1\}. \end{aligned} \tag{6.22}$$

THEOREM 6.2
Let

$$T^* = \begin{bmatrix} T_1^* & T_2^* & \cdots & T_n^* \end{bmatrix}^T \tag{6.23}$$

be the solution of the above optimization problem. Then $T^* \in \{MC\}$, where $M$ is defined in (6.5), with $m_i \in \left\{ \left\lfloor \frac{T_{i+1}^0}{T_i^0} \right\rfloor, \left\lceil \frac{T_{i+1}^0}{T_i^0} \right\rceil \right\}$. □

***Proof*** Let $f : \mathbb{R}^{n-1} \to \mathbb{R}^n$ be defined as

$$T = MC = f\left(\begin{bmatrix} m_1 & m_2 & \ldots & m_{n-1} \end{bmatrix}\right) \tag{6.24}$$

where the matrix $M$ is given in (6.5), and let the initial harmonic periods be $T^* = f(m^*_{\text{vector}})$, in which $m^*_{\text{vector}} = \begin{bmatrix} m_1^* & m_2^* & \ldots & m_{n-1}^* \end{bmatrix}$. Further define

$$\begin{aligned} \overline{m}_{\text{vector},\,j} &= \begin{bmatrix} m_1^* & m_2^* & \ldots & \left\lceil \frac{T_{j+1}^0}{T_j^0} \right\rceil + 1 & \ldots & m_{n-1}^* \end{bmatrix} \\ \underline{m}_{\text{vector},\,j} &= \begin{bmatrix} m_1^* & m_2^* & \ldots & \left\lceil \frac{T_{j+1}^0}{T_j^0} \right\rceil - 1 & \ldots & m_{n-1}^* \end{bmatrix}. \end{aligned} \tag{6.25}$$

Let $\underline{T} = f\left(\underline{m}_{\text{vector},\,j}\right)$. Now consider Figure 6.2 where the curve represents the utilization bound. Since this curve is convex, in the triangle $T^*T^0\underline{T}$, the angle between $T^*T^0$ and $T^*\underline{T}$ is greater than 90°. Then

$$\left\|T - T^0\right\| < \left\|f\left(\underline{m}_{\text{vector},\,j}\right) - T^0\right\| \tag{6.26}$$

and, similarly,

$$\left\|T - T^0\right\| < \left\|f\left(\overline{m}_{\text{vector},\,j}\right) - T^0\right\|. \tag{6.27}$$

The two inequalities above also apply to the high-dimensional case. By choosing different value of $m_i$, one can prove that for each $i \le n-1$, the two inequalities are valid. In the $n-$dimensional case, we need to check $2^{n-1}$ inequalities. □

EXAMPLE 6.2
Assume that we have three tasks with the same execution times as in Example 1, i.e., $C = \begin{bmatrix} 0.9 & 6.3 & 9.1 \end{bmatrix}$. Let the full-utilization initial periods be

$$T^0 = \begin{bmatrix} 12.3 & 13.7 & 19.4 \end{bmatrix}, \text{ with } \frac{T_2^0}{T_1^0} = 1.1, \ \frac{T_3^0}{T_2^0} = 1.4, \tag{6.28}$$

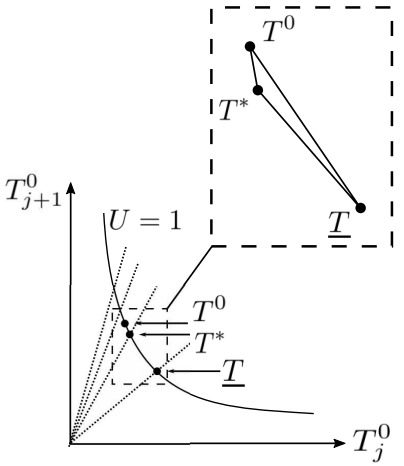



**Figure 6.2** Finding the closest periods set to the initial periods set.

so $m_1 = 1$ or 2 and $m_2 = 1$ or 2. Four sets of harmonic periods can be calculated from (6.5). The 2-norms of $T - T^0$ are given as follows:

$$\left\|T - T^0\right\| = \begin{cases} 5.62 & \text{if } m_1 = 1,\ m_2 = 1 \\ 4.60 & \text{if } m_1 = 1,\ m_2 = 2 \\ 5.57 & \text{if } m_1 = 2,\ m_2 = 1 \\ 8.52 & \text{if } m_1 = 2,\ m_2 = 2. \end{cases} \tag{6.29}$$

The closest harmonic periods are given by $m_1 = 1$, $m_2 = 2$ with $T^* = \begin{bmatrix} 11.75 & 11.75 & 23.5 \end{bmatrix}$. □

The time complexity for calculating all feasible candidates for the closest harmonic periods is $\mathcal{O}\left(2^{n-1}\right)$.

## Harmonic Period Assignment with Period Ranges

In many real-time control applications, there exist lower and upper bounds on the possible task periods. For instance, a commonly quoted rule of thumb [Åström and Wittenmark, 2013] states that the period $T$ of a control task should be chosen such that

$$0.2 \leq \omega_b T \leq 0.6 \tag{6.30}$$

where $\omega_b$ is the bandwidth of the closed-loop system. Another rule of thumb, based on the jitter margin of the control loop, is given in the next chapter.

In more general terms, we require that $T_i \in \left[T_i^l, T_i^u\right]$ where $T_i^l$ and $T_i^u$ are the lower and upper period bounds of task $\tau_i$ and we assume that $T_i^l \leq T_{i+1}^u$. We then want to find all harmonic periods that satisfy the requirements above for all tasks. We have the following theorem.

THEOREM 6.3
There exist $\tilde{m}_i$, $i \in \{1, 2, \ldots, n-1\}$, satisfying the harmonic period requirement above if and only if

$$\left\lceil \frac{T_j^l}{T_i^u} \right\rceil \leq \prod_{k=i}^{j-1} \tilde{m}_k \leq \left\lfloor \frac{T_j^u}{T_i^l} \right\rfloor \text{ for all } i < j; \tag{6.31}$$

and

$$\sum_{i=1}^{n} \frac{C_i^w}{\alpha \prod_{j=0}^{i-1} \tilde{m}_j} \leq 1, \text{ where } \alpha = \min_i \frac{T_i^u}{\prod_{j=0}^{i-1} \tilde{m}_j}. \tag{6.32}$$

□

***Proof*** Condition 1) can be understood as an existence condition. As shown in Section 6.2, the harmonic periods are a function of $\tilde{m}_i$, $i \in \{1, 2, \ldots, n-1\}$, for given execution times $[C_1^w, C_2^w, \ldots, C_n^w]$. Thinking in the $n$-dimensional space spanned by $T_1$, $T_2$, ..., $T_n$, each set of harmonic periods corresponds to a line through the origin. The existence condition can be understood as an $n$-dimensional cuboid in that space. Any line that intersects with the cuboid can be represented by a function of $\tilde{m}_i$.

Condition 2) states that the total utilization should be less than or equal to 1. Part of the intersection between the line representing the harmonic condition and the cuboid must satisfy this utilization condition. □

Using Theorem 6.3 to find all harmonic periods, we can start with 1) to find all possible sets of $\tilde{m}_i$, and then use condition 2) to verify all of them. However, if

$$\left\lceil \frac{T_j^l}{T_i^u} \right\rceil \ll \left\lfloor \frac{T_j^u}{T_i^l} \right\rfloor \tag{6.33}$$

then there exists a large number of sets of $\tilde{m}_i$ satisfying condition 1). We could choose an equivalent to condition 1), which is

$$\left\lceil \frac{T_{i+1}^l}{T_i^u} \right\rceil \leq \tilde{m}_i \leq \left\lfloor \frac{T_{i+1}^u}{T_i^l} \right\rfloor \text{ and } \alpha \prod_{j=0}^{i-1} \tilde{m}_j \geq T_i^l \tag{6.34}$$

where $\alpha$ has been defined in condition 2).

We give an intuitive explanation for Theorem 6.3 when $n = 2$ in Figure 6.3. The curve represents the utilization bound. For given $C_1^w$ and $C_2^w$, $\frac{C_1^w}{T_1} + \frac{C_2^w}{T_2} \leq 1$ is the region on the right upper side of this curve. The lines through the origin represent different harmonic relations between $T_1$ and $T_2$, while the rectangle represents the allowed period ranges. The line segment between the two marked points give all possible harmonic period assignments with utilization less than or equal to 1.

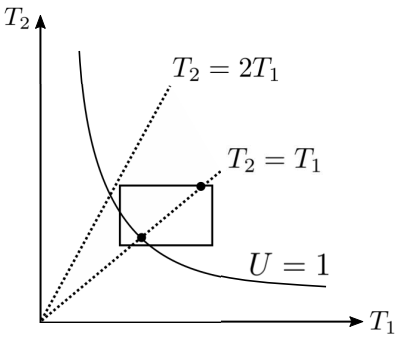



**Figure 6.3** Harmonic periods selection for $n = 2$.

Considering $n$ tasks, the point that is closest to the origin on each line segment is $T_0 = \tilde{M}C$, where $\tilde{M}$ is calculated by (6.5) with $\tilde{m}_i$. The utilization when $T = T_0$ is 1. The other point is

$$T_f = \alpha \begin{bmatrix} 1 & \tilde{m}_1 & \tilde{m}_1\tilde{m}_2 & \dots & \tilde{m}_1\tilde{m}_2\dots\tilde{m}_n \end{bmatrix}. \tag{6.35}$$

The utilization when $T = T_f$ is less than 1. Hence, the harmonic periods can be selected anywhere in $(1-a)T_0 + aT_f$, with $0 \le a \le 1$.

EXAMPLE 6.3
Assume that we have three tasks with the same execution times as in Examples 6.1 and 6.2, i.e., $C = \begin{bmatrix} 0.9 & 6.3 & 9.1 \end{bmatrix}$. Further assume that we have specified the allowable period ranges as

$$\begin{bmatrix} T_1^l, T_1^u \end{bmatrix} = \begin{bmatrix} 6, 12 \end{bmatrix}, \ \begin{bmatrix} T_2^l, T_2^u \end{bmatrix} = \begin{bmatrix} 7, 21 \end{bmatrix}, \ \begin{bmatrix} T_3^l, T_3^u \end{bmatrix} = \begin{bmatrix} 9, 27 \end{bmatrix}. \tag{6.36}$$

We then have

$$\begin{aligned} \left\lceil \frac{T_2^l}{T_1^u} \right\rceil = 0.58, \ \left\lfloor \frac{T_2^u}{T_1^l} \right\rfloor = 3.5 \Rightarrow m_1 \in \{1, 2, 3\} \\ \left\lceil \frac{T_3^l}{T_2^u} \right\rceil = 0.43, \ \left\lfloor \frac{T_3^u}{T_2^l} \right\rfloor = 3.86 \Rightarrow m_2 \in \{1, 2, 3\} \\ \left\lceil \frac{T_3^l}{T_1^u} \right\rceil = 0.75, \ \left\lfloor \frac{T_3^u}{T_1^l} \right\rfloor = 4.5 \Rightarrow m_1 m_2 \in \{1, 2, 3, 4\}. \end{aligned} \tag{6.37}$$

We should choose $[m_1, m_2] \in \{[1,2], [2,1], [2,2], [3,1]\}$. When $[m_1, m_2] \in \{[1,1], [1,3]\}$ the utilization condition is not satisfied. We then use (6.5) and (6.35)

to calculate the harmonic periods $T_0$ and $T_f$. The corresponding period sets are

$$\begin{aligned}
&a\begin{bmatrix}0.25 & 0.25 & 0.5\end{bmatrix} + \begin{bmatrix}11.75 & 11.75 & 23.5\end{bmatrix}\\
&a\begin{bmatrix}0.4 & 0.8 & 0.8\end{bmatrix} + \begin{bmatrix}8.6 & 17.2 & 17.2\end{bmatrix}\\
&a\begin{bmatrix}0.425 & 0.85 & 1.7\end{bmatrix} + \begin{bmatrix}6.325 & 12.65 & 25.3\end{bmatrix}\\
&a\begin{bmatrix}0.967 & 2.9 & 2.9\end{bmatrix} + \begin{bmatrix}6.033 & 18.1 & 18.1\end{bmatrix}
\end{aligned} \tag{6.38}$$

for any $a \in [0, 1]$.

Note that the first solution with $a = 0$ corresponds to the closest period solution in Example 6.2. Here we have obtained more solutions that could be further evaluated with regard to, e.g., overall system performance. □

The harmonic period assignment using period ranges has previously been solved in [Nasri et al., 2014] and in [Nasri and Fohler, 2015]. In [Nasri and Fohler, 2015] which uses the so-called forward calculation approach, the computational complexity is $\mathcal{O}(n)$ when all integer multiples intersect, while the complexity is pseudo-polynomial when all integer multiples do not intersect. In [Nasri and Fohler, 2015], which uses an alternative backward calculation approach, the complexity is pseudo-polynomial when all integer multiples intersect, and when all integer multiples do not intersect the complexity is $\mathcal{O}(n \log n)$.

Theorem 6.3 provides a unified method for both the case when all integer multiples intersect and when all integer multiples do not intersect, or a combination of those two situations. Since the algorithm is only run once from 1 to $n$ to calculate $\tilde{m}_i$ and to evaluate

$$\sum_{i=1}^{n} \frac{C_i^w}{\alpha \prod_{j=0}^{i-1} \tilde{m}_j}, \tag{6.39}$$

the time complexity is $\mathcal{O}(n)$ for each such calculation.

## 6.4 Codesign Procedure

In this section, we apply harmonic period assignment in scheduling-control codesign. As a starting point for the codesign, we find a set of non-harmonic, real-valued task periods and corresponding controllers using the sequential search optimization method in Chapter 4. We then harmonize these periods using Theorem 6.2 or Theorem 6.3 and enumerate all possible combinations of the harmonic factors. For each harmonic period assignment, we redesign each controller based on the new period and the new (now constant) control delay. Finally, we evaluate the combined control performance of all cases and select the best result.

All LQG controllers are designed using the `lqgdesign` command in Jitterbug [Cervin et al., 2002b]. To make a fair comparison between harmonic and non-harmonic designs, we make the following assumptions for the LQG control design:

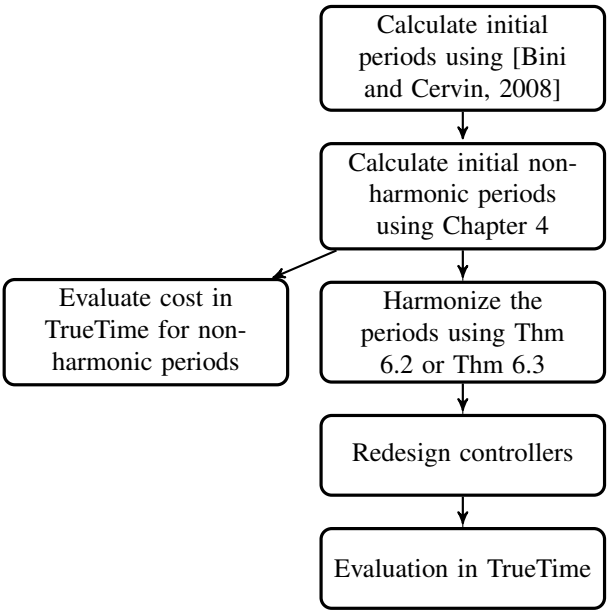



**Figure 6.4** Codesign and evaluation procedure for non-harmonic and harmonic designs.

- The harmonic control design takes the constant delay into account.
- The non-harmonic control design takes the delay distribution due to task scheduling into account, resulting in a jitter-robust controller with fixed parameters (Chapter 4). The delay distribution is found through a schedule simulation in TrueTime.
- The CPU utilization is 1 for both harmonic scheduling and non-harmonic scheduling. In non-harmonic scheduling, if the response time is greater than the period, the response time distribution used in the control design is truncated to the period length.

The cost function (2.2) for each controller under each scheduling scenario is evaluated with TrueTime. The design and evaluation procedure is summarized in the flow diagram in Figure 6.4.

## 6.5 Evaluation

In this section, we compare the resulting performance of harmonic codesign to state-of-the-art non-harmonic codesign.

## A Simple Codesign Example

For a simple codesign example, we choose three plants

$$P_1 = \frac{2}{s^2}, \quad P_2 = \frac{1}{s^2-3}, \quad P_3 = \frac{1}{s(s+1)} \tag{6.40}$$

to be controlled by three tasks $\tau_1$, $\tau_2$, $\tau_3$. $\tau_1$ has the highest priority and $\tau_3$ has the lowest priority. The execution times are given as $C_1^w = 0.1$, $C_2^w = 0.12$, $C_3^w = 0.14$. The state-space representation used is the controllable canonical form. The LQG control cost parameters are given as $Q_{1c} = C^T C$, $Q_{2c} = 0.01\mathrm{tr}\,(Q_{1c})$, $R_{1c} = BB^T$, and $R_{2c} = 0.01\mathrm{tr}\,(R_{1c})$.

The initial non-harmonic periods are calculated by the initialization procedure described in Chapter 4:

1. For task $i$, assume that the LQG cost can be approximated by a linear function of the period $T_i$ and of the delay $\delta_i$,

   $$V_i = \alpha_i T_i + \beta_i \delta_i. \tag{6.41}$$

   Evaluate the sensitivity coefficients $\alpha_i$ and $\beta_i$ at the point where $T_i = C_i^w$ and $\delta_i = C_i^w$ using numerical linearization and Jitterbug. Then use the period assignment method in [Bini and Cervin, 2008] to minimize the LQG cost under the simplifying assumption that these are the true cost functions.

2. Use the Sequential Search method in Chapter 4 to find the stochastic LQG periods.

The initial non-harmonic periods are $T_1^* = 0.3017$, $T_2^* = 0.4089$, $T_3^* = 0.4478$ with initial cost $V^* = 2.01$. Using Theorem 2, we find the harmonic factors

$$\begin{aligned} m_1 &\in \left\{ \left\lfloor \frac{T_2^*}{T_1^*} \right\rfloor, \left\lceil \frac{T_2^*}{T_1^*} \right\rceil \right\} = \{1, 2\} \\ m_2 &\in \left\{ \left\lfloor \frac{T_3^*}{T_2^*} \right\rfloor, \left\lceil \frac{T_3^*}{T_2^*} \right\rceil \right\} = \{1, 2\}. \end{aligned} \tag{6.42}$$

The LQG cost is then evaluated in TrueTime for the following cases:

- **No offset.** Assume a constant delay equal to the job response time $R_i$ to design and evalute the LQG controllers.
- **Offset.** Add the start latency $S_i$ as a release offset to each task; then design and evalute the LQG controllers for the constant delay $R_i - S_i$.

The resulting periods and LQG costs are shown in the table below. No offset means sampling happens at job release time, while offset means that sampling happens at the job start.

| $\{m_1, m_2\}$ | $T_1$ | $T_2$ | $T_3$ | $V_{\text{no offset}}$ | $V_{\text{offset}}$ |
|---|---|---|---|---|---|
| $\{1, 1\}$ | 0.36 | 0.36 | 0.36 | 1.99 | 1.57 |
| $\{1, 2\}$ | 0.29 | 0.29 | 0.58 | 2.11 | 1.57 |
| $\{2, 1\}$ | 0.23 | 0.46 | 0.46 | 1.90 | 1.33 |
| $\{2, 2\}$ | 0.20 | 0.39 | 0.78 | 2.56 | 1.75 |

The result shows that when we approximate the initial non-harmonic task periods with harmonic ones, the costs can be better or worse than the initial cost, i.e., $V^* = 2.01$. However, when we add release offsets, the cost is clearly better than the initial cost in all the cases, with the best case obtained for $\{m_1, m_2\} = \{2, 1\}$.

Continuing the same example, we also show how to find harmonic periods when there are constraints on the allowable period ranges, and then evaluate the LQG control performance, using the same plants with the same execution times as before. Assuming that the period $T_i$ can only be chosen from $[0.6T_i^*, 1.7T_i^*]$, it follows from Theorem 6.3 that the possible periods are when $\{m_1, m_2\}$ is $\{1, 1\}$, $\{1, 2\}$, $\{2, 1\}$ (as shown above), or $\{3, 1\}$ (as shown below).

| $\{m_1, m_2\}$ | $T_1$ | $T_2$ | $T_3$ | $V_{\text{no offset}}$ | $V_{\text{offset}}$ |
|---|---|---|---|---|---|
| $\{3, 1\}$ | 0.19 | 0.56 | 0.56 | 3.36 | 2.25 |

It should be noted, though, that the harmonic period assignment with the lowest control cost is not necessarily restricted to the above cases. The control cost function could have a form so that the harmonic period assignment with the lowest cost is not among those assignments that are close to the initial non-harmonic periods in the Euclidean sense. However, as shown in the general evaluation the proposed approach obtains gives considerably better control performance than the non-harmonic case.

### Randomly Generated Example

To see if the good results shown in the simple example above holds in more general cases, we randomly generate sets of three plants from three plant families, which are the same as in Section 4.5.

We randomly generate 20 sets of plants for each family. The transfer functions are converted to a state-space representation on controllable canonical form. For the LQG controllers, we use the design parameters $Q_{1c} = C^T C$, $Q_{2c} = 0.01\text{tr}(Q_{1c})$, $R_{1c} = BB^T$, and $R_{2c} = 0.01\text{tr}(R_{1c})$. The task execution times were randomly generated from $C_1^w \in \text{unif}(0.09, 0.11)$, $C_2^w \in \text{unif}(0.11, 0.13)$, $C_3^w \in \text{unif}(0.13, 0.15)$. The nominal task utilizations $U_i^{nom}$ were generated using an $n$-dimensional uniform distribution with total utilization 1. Task 1 has the highest priority, while task 3 has the lowest priority.

The optimization procedure to assign initial non-harmonic periods is the same as in the previous section. We find the four closest harmonic period sets to the initial

**Table 6.1** The overall LQG costs, averaged over randomly generated plant sets.

| Family | I | II | III |
|---|---|---|---|
| Non-harmonic tasks | 2.92 | 8.61 | 29.46 |
| Harmonic tasks | 2.53 | 5.17 | 17.76 |
| Harmonic tasks with offsets | 2.03 | 3.91 | 15.43 |

periods using Theorem 6.2. For the harmonic periods case, the response time of each task is constant. Using this constant response time as delay, the LQG controllers are designed and the corresponding costs are evaluated. In the table below, the minimum cost, out of the four cases with harmonic periods, is given. We then add an offset to each task in order to obtain a shorter delay for task 2 and 3. The length of the offset is the start latency of each task. The constant delay is $R_i - S_i$. We design a controller and evaluate the LQG costs for this constant delay. The obtained LQG costs, averaged over 20 generated plant sets for each family, are summarized in Table 6.1.

The LQG costs are evaluated as follows:

- **Non-harmonic tasks.** The delay distribution is truncated to the interval $\left[R_i^b, R_i^b + T_i\right]$. The probability of a response time greater than $R_i^b + T_i$ is added to the probability mass function (PMF) at $R_i^b + T_i$. We then use the truncated delay distribution to design stochastic LQG controllers. The LQG cost is evaluated in TrueTime.

- **Harmonic tasks.** We calculate $2^{n-1}$ sets of harmonic periods. For each set, LQG controllers with constant delays equal to $R_i$ are designed and evaluated in TrueTime. The period set giving the smallest cost is selected.

- **Harmonic tasks with offsets.** For each set of harmonic periods, LQG controllers with constant delays equal to $R_i - S_i$ are designed and evaluated in TrueTime. The period set giving the smallest cost is selected.

We normalize the costs for non-harmonic tasks to 1 for each plant set, then normalize each cost for harmonic tasks without or with offsets, compared with corresponding non-harmonic tasks cost. The box plot is shown in Figure 6.5.

In Family III, the likelihood that the plants are unstable, and, hence, more sensitive to delays and delay jitter, is larger, and therefore the control cost is considerably higher than for Family I and II. As shown in Table 6.1, when the periods are harmonic without offset, the costs are lower than in the non-harmonic case. The reason for this is that the increase in cost caused by the period perturbation is compensated for by the decrease in cost caused by the jitter-free, but possibly large delays. The

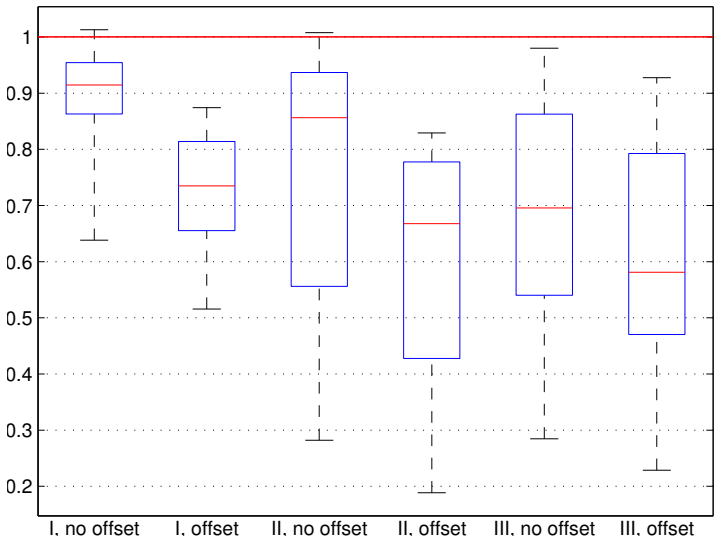



**Figure 6.5** Normalized costs for harmonic tasks without and with offsets.

best results are obtained for the harmonic tasks with offsets. In this case, the increase in cost caused by the period perturbation is small compared to the decrease in cost caused by the smaller and jitter-free delays.

The evaluation above is based on the assumption that the execution time is constant. However, in reality, this is seldom the case. To investigate the effect of varying execution times we design the controllers using the harmonic task with offset method assuming that the execution times are constant. When we evaluate the performance we let the execution time vary from job to job according to $\text{unif}(0.9C_i^w, C_i^w)$. The costs now become

| Family | I | II | III |
|---|---|---|---|
| Cost | 2.01 | 3.88 | 15.33 |

i.e., even smaller than before. The box plot of the costs for non-constant execution times compared with non-harmonic tasks costs are shown in Figure 6.6. The result is, however, not surprising since the average delay is always shorter than the constant delay in the constant execution time case.

## 6.6 Conclusion

In this chapter, we have investigated the harmonic scheduling and control codesign problem. Through an extensive evaluation, it was shown that codesign using

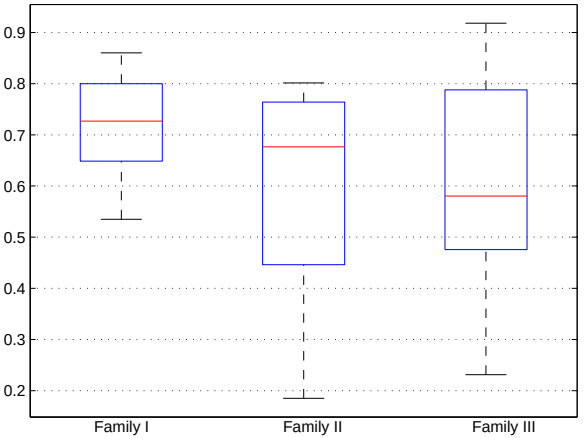



**Figure 6.6** Normalized costs for non-constant execution times (different scale compared to Figure 6.5).

harmonic controller task periods gives better control performance than using non-harmonic periods when task offsets are added. The reason for this is the fact that under harmonic scheduling the response times and start latencies for each task are constant, assuming that the task execution times are constant. This can be exploited in the LQG control design through the constant control delays that it gives rise to. However, as shown in the evaluation also in the case when the task execution times vary slightly from job to job the proposed codesign method gives good results.

In order to implement the codesign method, it is necessary to be able to find the harmonic periods and to calculate the task response times. A new method for calculating the response times has been presented that has lower complexity than earlier methods. Also, two heuristic approaches to harmonic task period assignment have been presented. One method for finding the closest harmonic periods to a set of initial periods and one method for finding all possible harmonic period assignments that satisfy constraints on allowable task period ranges.

# 7

# Robust LQG Scheduling and Control Codesign*

## 7.1 Introduction

The design methods in Chapters 4, 5, and 6 are all based on standard LQG design, albeit with extensions to both stochastic and periodic delays. It is however well known that good LQG performance does not mean that the system is robust [Doyle, 1978]. While there are several methods proposed to improve the robustness of LQG, very few have focused on uncertainties related to controller timing. This is the reason for exploring LQG control synthesis with a jitter robustness constraint.

In this chapter, we present an approach to jitter-robust LQG control synthesis together with priority assignment and period selection methods for the real-time system. We propose a convex optimization-based sampled-data LQG control design method with a jitter robustness constraint. The constraint guarantees that the closed-loop system has a specified *jitter margin* [Kao and Lincoln, 2004; Cervin et al., 2004]. We also propose a new rule of thumb for initial sampling period selection based on the jitter margin. Finally, we give a codesign method to assign priorities and sampling periods to obtain optimal robust LQG performance for a set of controllers that share a single CPU.

### Outline

In Section 7.2, we give the control system model, including the performance metric we use. In Section 7.3, a new rule of thumb for initial sampling period assignment is proposed. The robust LQG control design problem and its solution are described in Section 7.4. In Section 7.5, we give a jitter-aware priority and period assignment codesign method to optimize the overall system performance. In Section 7.6, an example is presented to show the entire procedure of scheduling and control codesign. In the same section, the method is evaluated on randomized plant sets, showing an improvement over state-of-the-art methods. Conclusions are given in Section 7.7.

* This chapter is based on [Xu et al., 2017b].

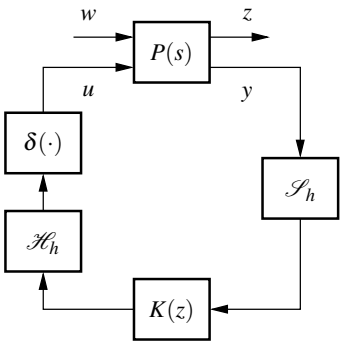



**Figure 7.1** Control loop with continuous plant $P(s)$, discrete LQG controller $K(z)$, periodic sampler $\mathscr{S}_h$ and hold $\mathscr{H}_h$, and time-varying delay $\delta(t)$.

## 7.2 System Model

The control loop is illustrated in Figure 7.1. For the system defined by (2.1), a time-invariant LQG controller should be designed to minimize the quadratic cost function

$$V = \lim_{T\to\infty} \frac{1}{T} \mathrm{E} \int_0^T \left( x^T(t) Q_{1c} x(t) + 2x^T(t) Q_{12c} u(t) + u^T(t) Q_{2c} u(t) \right) dt = \mathrm{E}\, z^2. \tag{7.1}$$

The sampling period of the controller is $h = T_i$. As before, sampling is performed at the task release time, e.g., using external hardware, and the actuation is performed when the task finishes. Hence, the actuation (or output) is subject to a time-varying delay $\delta(t)$, $C_i^b \le \delta(t) \le C_i^w + J_i$, due to the task output jitter where $J_i$ is the jitter. For a given constant delay or known delay distribution (assuming independent delays between periods), the LQG controller and its corresponding cost (7.1) can be designed and evaluated using, e.g., Jitterbug.

As first shown in [Doyle, 1978], an LQG controller has no guaranteed robustness. In order to gain robustness of the real-time control system, we would like to design a controller that has a specified *jitter margin* $J_m$ [Kao and Lincoln, 2004; Cervin et al., 2004]. This means that the closed-loop system should be stable for any time-varying delays $\delta \in [C_i^b, C_i^w + J_m]$. Naturally, we will require that $J_m > J_i$. For a continuous-time control system, the sufficient condition for stability of the closed-loop system is

$$|T(i\omega)| = \left| \frac{P(i\omega)K(i\omega)}{1 + P(i\omega)K(i\omega)} \right| < \frac{1}{J_m \omega}, \quad \forall \omega, \tag{7.2}$$

where $P(s)$ is the plant and $K(s)$ is the controller [Kao and Lincoln, 2004]. In a Bode plot, this corresponds to the magnitude curve of the complementary sensitivity function $T(s)$ lying below a line with slope $-1$ and gain $1/J_m^c$ (see Figure 7.2).

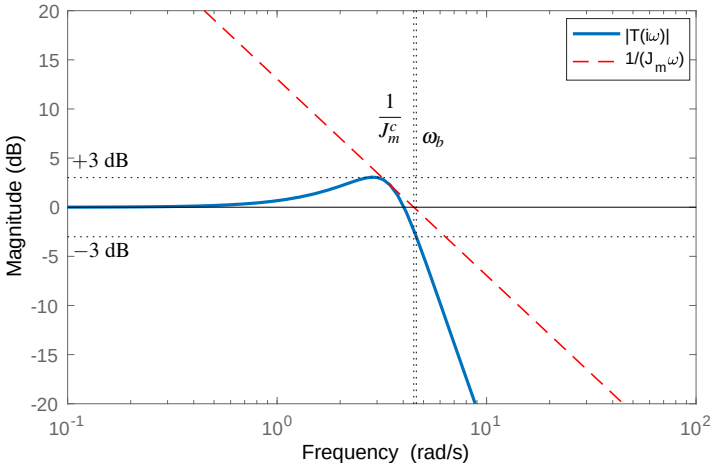



**Figure 7.2** The continuous-time jitter margin $J_m^c$ is given by the intersection of the closed-loop transfer function $|T(i\omega)|$ and the line $1/(J_m^c\omega)$. For a typical robust design, the 3 dB bandwidth $\omega_b$ is close to $1/J_m^c$.

For a sampled-data system, the stability criterion is slighly more complicated (see Section 7.4 below).

The goal of the control and scheduling codesign is to optimize the overall control performance, subject to the real-time system utilization constraint. This can be expressed as

$$\begin{aligned} &\text{minimize } V = \sum_i V_i \\ &\text{subject to } \sum_i U_i \leq 1 \end{aligned} \tag{7.3}$$

where the cost $V_i$ for each controller is defined in (7.1). Moreover, we would like to give guarantees on the robustness for each control loop. The parameters that we can optimize over are the task priorities, the sampling periods, and the controllers themselves.

In order to achieve a small LQG cost and good robustness, the sampling period and the jitter should be small, but this requirement cannot be satisfied simultaneously for every task due to the fixed-priority scheduling algorithm used. This makes the codesign problem nontrivial.

## 7.3 Initial Sampling Period Selection

Sampling period selection for controllers is typically done based on the properties of the closed-loop system. The sampling rate should be fast enough, so that disturbance rejection and robustness are not affected too much by the sampling and hold

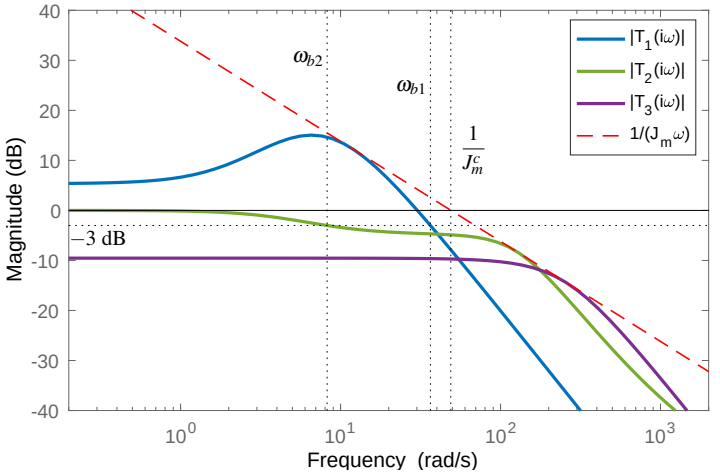



**Figure 7.3** In degenerate cases, the 3 dB bandwidth $\omega_b$ may be significantly smaller than $1/J_m^c$ (systems $T_1$, $T_2$) or even undefined (system $T_3$).

operations, while slow enough to avoid numerical problems and allow implementation in resource-constrained systems.

There are several rules of thumb that relate the sampling rate with the speed of the closed-loop system. Franklin *et al.* recommend sampling about 10–40 times faster than the bandwidth [Franklin et al., 1994]. Åström and Wittermark recommend 4 to 10 samples per rise-time of the closed-loop system [Åström and Wittenmark, 2013]. These recommendations (and other similar ones [Levine, 1996]) roughly translate into the relation

$$0.15 \leq \omega_b h \leq 0.6 \tag{7.4}$$

where $\omega_b$ is the 3 dB closed-loop bandwidth and $h$ is the sampling interval.

Basing the sampling period selection on a single point $\omega_b$ of the closed-loop frequency response, however, makes this rule of thumb sensitive to degenerate cases. As an alternative, we propose a new rule of thumb that is based on the continuous-time jitter margin $J_m^c$ of the control system:

$$0.15 \leq h/J_m^c \leq 0.6. \tag{7.5}$$

For a typical, robust closed-loop system with a maximum complementary sensitivity of 3 dB, we have $\omega_b \approx 1/J_m^c$, and the new rule produces similar sampling intervals as the old rule (7.4). This nominal case is illustrated in Figure 7.2.

For degenerate cases, however, the new rule will typically recommend shorter sampling intervals than the old rule. Three such cases are illustrated in Figure 7.3. For closed-loop system $T_1$, the maximum complementary sensitivity is close to

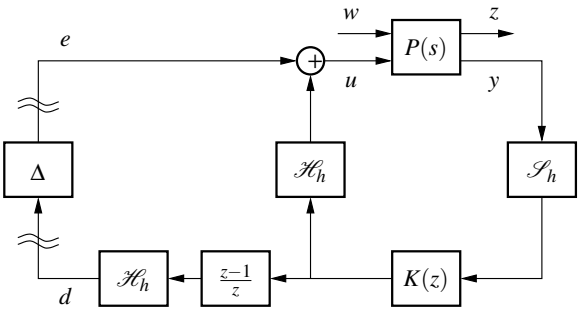



**Figure 7.4** Transformation of control loop with time-varying output delay. The jitter is represented by the uncertainty Δ.

15 dB, which implies that the system is not robust. For system $T_2$, the 3 dB bandwidth is quite low, while $|T(i\omega)|$ does not roll off until much later. For system $T_3$ the closed-loop gain is low throughout and the bandwidth is not well-defined. As seen in the figure, all of these systems actually have the same jitter margin, and the new rule will hence recommend the same sampling interval for all of them.

The new rule aims for a sampled closed loop that should be robust towards delay and jitter amounting to more than one and a half sampling interval in total. This aligns with the worst-case situation in typical multirate applications: the sample-and-hold operation can be approximated by a delay of $h/2$, and the output jitter is typically upper bounded by $h$. Using the rule does not give any hard stability guarantees, however, since the exact performance degradation due to sampling cannot be captured using simple expressions. Once a discrete-time controller has been designed, its jitter margin should be verified using the sampled-data analysis in [Kao and Lincoln, 2004].

## 7.4 Jitter-Robust LQG Control Synthesis

### Robust LQG Problem Formulation

In order to design a LQG controller with guaranteed robustness against jitter, we use the stability criterion in [Kao and Lincoln, 2004] as the constraint. As shown in [Kao and Lincoln, 2004] the sampled-data control loop with output jitter in Figure 7.1 can be transformed into the block diagram shown in Figure 7.4. The time-varying operator Δ represents the uncertainty due to jitter and has the worst-case gain

$$\|\Delta\| = \sqrt{(2\lfloor N\rfloor + 1)N - \lfloor N\rfloor^2 - \lfloor N\rfloor}, \tag{7.6}$$

where $N = J_i/h$.

Referring to Figure 7.4, the jitter-robust control design problem can now be stated as the optimization problem

$$\begin{aligned} &\underset{K(z)}{\text{minimize}} \ \|G_{zw}\|_2^2 \\ &\text{subject to } \|G_{de}\|_\infty < b, \end{aligned} \tag{7.7}$$

where

$$\begin{aligned} b &= \frac{1}{\sqrt{(2\lfloor N_m \rfloor + 1)N_m - \lfloor N_m \rfloor^2 - \lfloor N_m \rfloor}}, \\ N_m &= J_m^{req}/h, \end{aligned} \tag{7.8}$$

where $J_m^{req}$ is the required jitter margin. The objective function in the optimization problem, namely the square of the $H_2$ norm of $G_{zw}$, is equal to the LQG cost (7.1).

Since both the criterion and the constraint in (7.7) are nonconvex, it is hard to solve the problem directly. We, therefore, reformulate it as a convex optimization problem below.

## Youla Parameterization and Convex Optimization

Using the Youla parameterization [Boyd and Barratt, 1991], the optimization problem (7.7) can be reformulated as

$$\begin{aligned} &\underset{\Theta}{\text{minimize}} \int_{-2\pi}^{2\pi} \left| P_{zw}(e^{j\omega}) - P_{zu}(e^{j\omega})\Theta(e^{j\omega})P_{yw}(e^{j\omega}) \right|^2 d\omega \\ &\text{subject to } \left\| \frac{e^{j\omega}-1}{e^{j\omega}} P_{yu}\left(e^{j\omega}\right)\Theta\left(e^{j\omega}\right) \right\|_\infty < b, \end{aligned} \tag{7.9}$$

where the arbitrary stable finite order LTI $\Theta$ is defined as

$$\Theta(e^{j\omega}) = \frac{K(e^{j\omega})}{1 + P_{yu}(e^{j\omega})K(e^{j\omega})}. \tag{7.10}$$

For the sampled plant $P_d(z)$ with noise covariance matrices $R_{1d}$ and $R_{2d}$, and weighting matrices $Q_{1d}$ and $Q_{2d}$, the transfer function matrix used in the optimization (7.9) is

$$\left[\begin{array}{c|c} P_{zw}(z) & P_{yw}(z) \\ \hline P_{zu}(z) & P_{yu}(z) \end{array}\right] = \left[\begin{array}{cc|c} \sqrt{Q_{1d}}P_d(z)\sqrt{R_{1d}} & 0 & \sqrt{Q_{1d}}P_d(z) \\ 0 & 0 & \sqrt{Q_{2d}} \\ \hline P_d(z)\sqrt{R_{1d}} & \sqrt{R_{2d}} & P_d(z) \end{array}\right]. \tag{7.11}$$

The optimization problem can be solved numerically using, e.g., the CVX toolbox [Grant and Boyd, 2014]. Here we choose a pulse response representation of the Youla parameter,

$$\Theta(z) = \sum_{i=0}^{n-1} \frac{\theta_i}{z^k}, \tag{7.12}$$

with $\{\theta_i\}$ being the set of scalar optimization variables. The magnitude constraint is checked over a dense grid of frequency points. Once the problem is solved for $\Theta(z)$, the corresponding controller can be calculated by

$$K(z) = \frac{\Theta(z)}{1 - \Theta(z)P_{yu}(z)}. \tag{7.13}$$

## Example

Consider control of an inverted pendulum with realization

$$A = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{7.14}$$

The continuous-time cost and noise matrices are

$$Q_{1c} = \begin{bmatrix} 100 & 0 \\ 0 & 0 \end{bmatrix}, \; Q_{2c} = 1, \; R_{1c} = \begin{bmatrix} 0 & 0 \\ 0 & 100 \end{bmatrix}, \; R_{2c} = 1. \tag{7.15}$$

A standard LQG design gives the continuous-time jitter margin $J_m^c = 0.195$. The recommended sampling period range is hence $0.03 < h < 0.12$, and we choose $h = 0.1$. The minimum delay is assumed to be zero, and the output delay is assumed to be independent between periods and uniformly distributed between 0 and $J$. For zero delay and jitter, the sampled-data jitter margin is $J_m^0 = 0.190$.

For different values of the jitter, $0 \le J \le h$, three different designs are compared:

- Plain LQG design, assuming zero delay and jitter.
- Stochastic LQG design, with perfect knowledge of the delay distribution. This is the optimal design with regards to the LQG cost.
- Jitter-robust LQG design, with the constraint of keeping the remaining jitter margin at the original value $J_m^0$. This is achieved by setting $J_m^{req} = J_m^0 + J$.

The expected LQG cost under uniform jitter is calculated using Jitterbug and the remaining jitter margin is calculated using [Kao and Lincoln, 2004]. The results, normalized to 1 for the case of zero jitter, are given in Figure 7.5. It is seen that the stochastic LQG performs best in terms of average-case performance (it should since it is an optimal design), while its jitter margin degrades as the jitter increases. The situation is worse for the plain LQG controller, which suffers from both performance deterioration and decreasing jitter margin. The jitter-robust LQG is able to keep the jitter margin at its original value (0.190) while paying only a small price in terms of performance degradation.

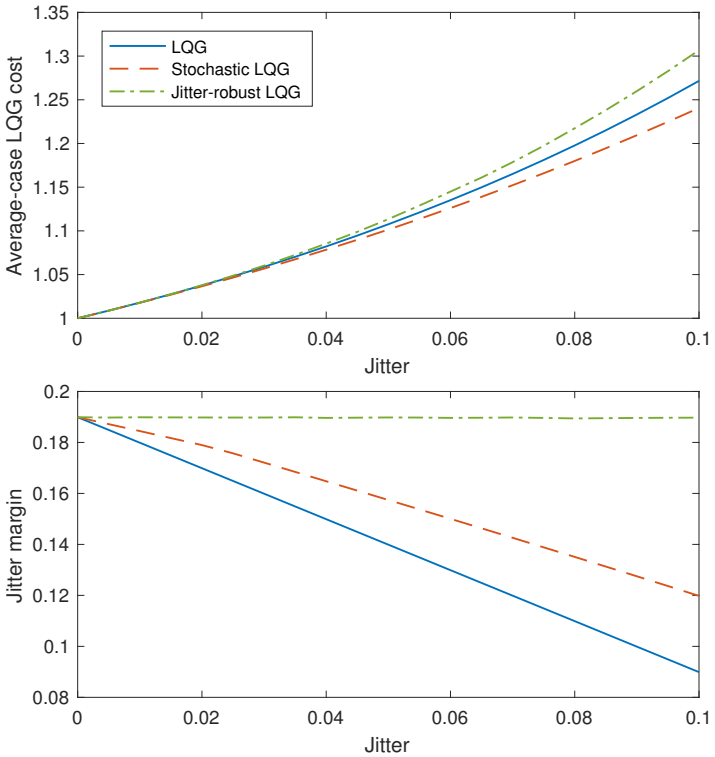



**Figure 7.5** Comparison of design methods for the inverted pedulum: average-case normalized performance (top) and remaining jitter margin (bottom) vs amount of jitter in the control loop.

## 7.5 Real-Time System Scheduling Codesign

We now turn to the problem of implementing several digital controllers on the same CPU. Using the procedure described above, we can design a robust controller with a specified jitter margin for a given sampling period, but it remains to optimize the combined performance of a set of controllers by assigning suitable periods and priorities.

### Affine Cost Function Approximation

We begin by characterizing the jitter-robust LQG cost $V_i$ as a function of the sampling period and the jitter. We first calculate the initial sampled-data jitter margin $J^0_{m,i}$ assuming a constant delay $C^w_i$ and zero jitter. For a pair $(h_i, J_i)$ we then design a jitter-robust LQG controller as described in Section 7.4 with the required jitter margin $J^{req}_{m,i} = J^0_{m,i} + J_i$. Solving the convex optimization problem (7.9) for a given period

$h_i$ and jitter $J_i$, we obtain one data point of the cost function $V_i = f_i(h_i, J_i)$. We store the result of each evaluation as a data point $p_j = (T, J, V)$, $j \in \{1, 2, \ldots, m\}$, with $m$ being the number of points.

In order to facilitate an analytical solution for the optimal periods below, we approximate the cost $V_i$ as an affine function of sampling period $T_i$ and jitter $J_i$, namely,

$$V_i = a_i h_i + b_i J_i + c_i. \tag{7.16}$$

We want the square sum of the orthogonal distances between the affine approximation and the points as small as possible. Let $c$ be a point on the plane, and $n$ be a unit normal vector to the plane. The optimization problem is

$$\begin{aligned} &\underset{c,n}{\text{minimize}} \sum_{j=1}^{m} \left( (p_j - c)^T n \right)^2 \\ &\text{subject to } \|n\| = 1. \end{aligned} \tag{7.17}$$

This plane fitting problem can be efficiently solved by the singular value decomposition (SVD) method [Arun et al., 1987]. The optimization solution contains two parts, $c$ and $n$. The point on the plane $c$ can be calculated by $c = 1/m \sum_{j=1}^{m} p_j$. Let $A = [p_1 - c, p_2 - c, \ldots, p_m - c]$. The unit normal vector $n$ is the last column of $U$, where $A = USV^T$. Having obtained $c$ and $n$, the equation of the plane is

$$\left( \begin{pmatrix} h_i & J_i & V_i \end{pmatrix}^T - c \right)^T n = 0, \tag{7.18}$$

from which we obtain the coefficients in (7.16).

## Period Assignment

For a given task priority ordering, we now derive a solution to the optimal period assignment problem. To facilitate an analytical solution, we use simple bounds on the best-case and worst-case response times and conservatively over-estimate the jitter.

The best-case response time of task $i$ is trivially lower bounded by $R_i^b \geq C_i^w$. For the worst-case response time, we use the simplistic upper bound (see [Bini and Baruah, 2007])

$$R_i^w \leq \frac{\sum_{j=1}^{i} C_j^w}{1 - \sum_{j=1}^{i-1} \frac{C_j^w}{T_j}}, \tag{7.19}$$

where $j < i$ indicates that task $j$ has higher priority than task $i$. It then follows that the jitter is upper bounded by

$$J_i = R_i^w - R_i^b \leq \frac{\sum_{j=1}^{i} C_j^w}{1 - \sum_{j=1}^{i-1} \frac{C_j^w}{T_j}} - C_i^w. \tag{7.20}$$

In order to obtain a solution that is guaranteed not to violate the jitter margin requirement for any controller, we assume that the upper bound in (7.20) can actually be reached during execution. Using the upper bound for the value of $J_i$ and the relation $U_i = C_i^w/h_i$, the cost function (7.16) for task $i$ can be restated as

$$V_i = \frac{a_i C_i^w}{U_i} + \frac{b_i \sum_{j=1}^{i} C_j^w}{1 - \sum_{j=1}^{i-1} U_j} - b_i C_i^w + c_i. \tag{7.21}$$

The period assignment problem (7.3) can now be expressed in terms of task utilizations as

$$\begin{aligned} &\underset{\{U_i\}}{\text{minimize}} \sum_i V_i \\ &\text{subject to} \sum_i U_i \leq 1. \end{aligned} \tag{7.22}$$

A similar optimization problem of this form was solved in [Bini and Cervin, 2008] and we reuse that solution here. Let

$$\alpha_i = a_i C_i^w, \quad \beta_i = b_i \sum_{j=1}^{i} C_j^w, \tag{7.23}$$

and recursively define $\mu_k$ and $\lambda_k$ as

$$\begin{aligned} \mu_k &= \sqrt{\alpha_k}, \quad k = 1, 2, \cdots, n-1 \\ \lambda_{n-1} &= \sqrt{\alpha_n + \beta_n} \\ \lambda_{k-1} &= \sqrt{\beta_k + (\lambda_k + \mu_k)^2}, \quad k = 2, 3, \cdots, n-1. \end{aligned} \tag{7.24}$$

The optimal utilizations of each task are then given by

$$\begin{aligned} U_1 &= \frac{\mu_1}{\lambda_1 + \mu_1} \\ U_k &= U_1 \frac{\mu_k}{\mu_1} \prod_{j=1}^{k-1} \frac{\lambda_j}{\lambda_{j+1} + \mu_{j+1}}, \quad k = 2, 3, \cdots, n. \end{aligned} \tag{7.25}$$

Finally, the optimal periods are recovered as $h_i = C_i^w/U_i$. The solution always achieves full utilization ($\sum_i U_i = 1$). For details, see [Bini and Cervin, 2008].

## Priority Assignment

As discussed in [Mancuso et al., 2014], optimal priority assignment in real-time control systems is, in general, a combinatorial problem with exponential complexity in terms of the number of tasks. Assigning correct priorities is however crucial, since the amount of jitter (and hence also the performance degradation) depends heavily on the task priority (cf. (7.20)).

In this chapter, we have used the following two methods to assign priorities to a set of controller tasks:

- **Global solution.** Using exhaustive search, we solve the optimal period assignment problem for all permutations of the task priorities and select the priority ordering that gives the smallest overall cost. Because of its complexity, this method can only be used for small task sets in practice.

- **Heuristic solution.** We propose to order the tasks by the value of $C_i^w/\sqrt{b_i}$ in ascending order. The idea is to give high priority to tasks with short execution times and high jitter sensitivity. This method can be used for arbitrarily large task sets and only requires the period assignment problem to be solved once.

## 7.6 Evaluation

This section illustrates the jitter-robust LQG control and scheduling codesign procedure in a simple example and in a larger evaluation using randomly generated plants. In summary, the different steps of the codesign method are:

1. Initial sampling period selection. Using the rule of thumb (7.5), initial sampling periods are calculated based on the continuous-time jitter margins. With these periods, the sampled-data closed-loop systems will have some basic robustness against scheduling-induced delay and jitter.

2. Characterization of the LQG cost. Designing a jitter-robust LQG controller and evaluating the resulting cost for a number of different values of the period and jitter, we obtain an affine cost function for each control loop.

3. Priority assignment and sampling period selection. A heuristic method and a global search method are used for the task priority ordering. For each priority assignment, optimal periods are calculated based on the affine cost functions. If any period is outside of the given range, it is clamped at the limitation of the range.

### A Simple Example

The simple example consists of three tasks sharing one CPU. Each task implements an LQG controller that controls one of the following open-loop unstable plants:

$$\begin{aligned} P_1(s) &= \frac{1}{(s-0.71)(s-0.40)} \\ P_2(s) &= \frac{1}{(s-0.21)(s+0.08)} \\ P_3(s) &= \frac{1}{(s-0.95)(s+0.76)}. \end{aligned} \tag{7.26}$$

**Table 7.1** Initial parameters in the simple example.

| | $h_i^0$ | $U_i^0$ | $C_i^w$ | $J_{m,i}^0$ |
|---|---|---|---|---|
| $i=1$ | 0.074 | 0.35 | 0.026 | 0.24 |
| $i=2$ | 0.082 | 0.18 | 0.015 | 0.27 |
| $i=3$ | 0.048 | 0.47 | 0.022 | 0.16 |

For the LQG design, the three plants are realized in controllable canonical form. The cost weighting matrices and noise covariance matrices are chosen as

$$\begin{aligned} Q_{1,i} &= \begin{bmatrix} 0 & 0 \\ 0 & \alpha_i \end{bmatrix}, \quad Q_{2,i} = 1, \\ R_{1,i} &= \begin{bmatrix} \beta_i & 0 \\ 0 & 0 \end{bmatrix}, \quad R_{2,i} = 0.01, \end{aligned} \tag{7.27}$$

where $\alpha_1 = 1.89$, $\alpha_2 = 72.38$, $\alpha_3 = 102.66$, $\beta_1 = 14.48$, $\beta_2 = 1818.80$, $\beta_3 = 17.86$.

For the given parameters, a standard continuous-time LQG controller $K_i(s)$ for each plant is designed, and the continuous-time jitter margin $J_{m,i}^c$ is calculated. Initial sampling periods are then chosen as

$$h_i^0 = 0.3 J_{m,i}^c, \tag{7.28}$$

i.e., the recommendation (7.5) is fulfilled. The initial utilizations $U_i^0$ are randomized using the UUniFast algorithm [Bini and Buttazzo, 2005]. The execution times are then given by $C_i^w = U_i^0 h_i^0$. The initial jitter margin $J_{m,i}^0$ is calculated using the initial period $h_i^0$ and with $C_i^w$ as the best-case response time. The initial periods, utilizations, execution times, and jitter margins are summarized in Table 7.1.

To obtain cost function approximations, for each task $\tau_i$ we choose the sampling period $h_i$ from 7 evenly spaced points between $0.5h_i^0$ and $2h_i^0$ and choose the jitter $J_i$ from 9 evenly spaced points between 0 and $2h_i^0$. For each pair $(h_i, J_i)$ that satisfies $R_i^b + J_i \leq h_i$, we design a robust LQG controller with the required jitter margin $J_m^{req} = J_{m,i}^0 + J_i$. The corresponding LQG cost is evaluated in Jitterbug, assuming a uniform response time distribution, unif$\left(R_i^b, R_i^b + J_i\right)$. The resulting costs functions $V_i(h_i, J_i)$ for $P_1(s)$, $P_2(s)$, $P_3(s)$ are plotted in Figure 7.6, showing that affine approximations are reasonable. Using the SVD plane fitting method, the affine LQG cost functions are

$$\begin{aligned} V_1(h_1, J_1) &= (0.01h_1 + 0.06J_1 + 0.03) \times 10^3, \\ V_2(h_2, J_2) &= (0.57h_2 + 1.16J_2 + 0.51) \times 10^3, \\ V_3(h_3, J_3) &= (0.12h_3 + 0.80J_3 + 0.26) \times 10^3. \end{aligned} \tag{7.29}$$

As can be seen from the coefficients, the cost is more sensitive to the size of the jitter than to the value of the sampling period.

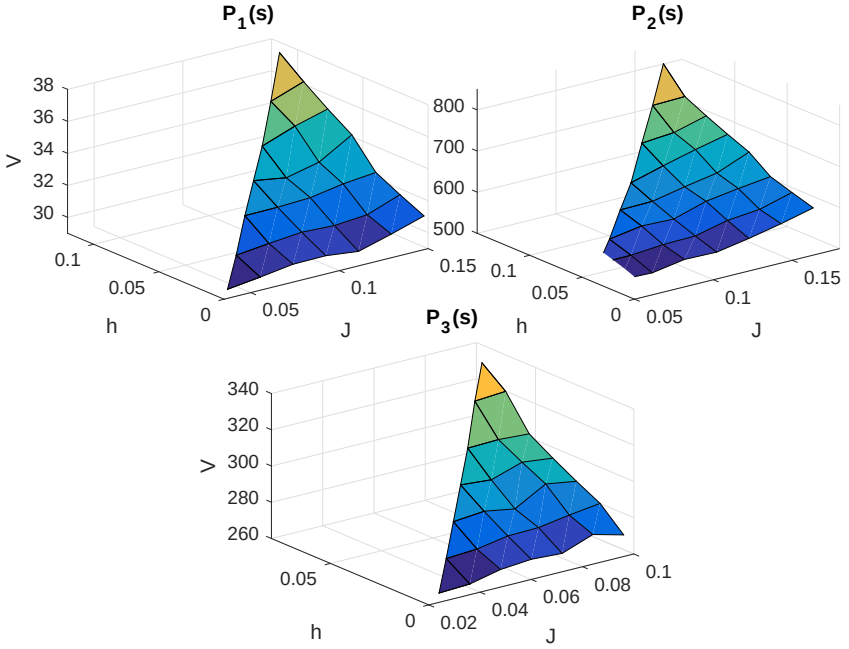



**Figure 7.6** Cost functions $V_i = f_i(h_i, J_i)$ for three example plants. It is seen that each $V_i$ can be quite well approximated by an affine function.

**Table 7.2** Enumeration of all possible priority orderings.

| | $\sum_{i=1}^{3} V_i^0$ | $h_1^*$ | $h_2^*$ | $h_3^*$ | $\sum_{i=1}^{3} V_i^*$ |
|---|---|---|---|---|---|
| $\tau_1, \tau_2, \tau_3$ | 1000.8 | 0.15 | 0.06 | 0.04 | 951.3 |
| $\tau_1, \tau_3, \tau_2$ | 1285.1 | 0.15 | 0.04 | 0.10 | 973.9 |
| $\tau_2, \tau_1, \tau_3$ | 945.7 | 0.15 | 0.08 | 0.04 | 914.0 |
| $\tau_2, \tau_3, \tau_1$ | 881.0 | 0.07 | 0.05 | 0.07 | 870.4 |
| $\tau_3, \tau_1, \tau_2$ | 1246.8 | 0.15 | 0.04 | 0.10 | 946.9 |
| $\tau_3, \tau_2, \tau_1$ | 926.1 | 0.06 | 0.04 | 0.10 | 883.0 |

In order to optimize the overall cost, priorities and periods need to be assigned. We first enumerate all the permutations of priority assignment and calculate the corresponding cost $\sum_{i=1}^{3} V_i^0$ using the initial periods $h_i^0$. The results are shown in the first two columns of Table 7.2, where the first column lists the tasks in descending priority order.

Next, for each priority assignment, the periods are optimized using the method

in Section 7.5. Any period that falls outside the range $\left(0.15J^0_{i,m}, 0.6J^0_{i,m}\right)$ is clamped at the limit, and then the optimization for the remaining periods is repeated. The optimal sampling periods $h_1^*$, $h_2^*$, $h_3^*$ and the corresponding total cost $\sum_{i=1}^3 V_i^*$ are shown in the four last columns of Table 7.2.

In all six cases, the overall cost using optimal periods is smaller than the cost using the initial periods. The global optimal solution to the priority and period assignment problem is found when the priority ordering is $\tau_2$, $\tau_3$, $\tau_1$, which gives the optimal total cost $\sum_{i=1}^3 V_i^* = 870.4$. This can be compared to the initial priority ordering $\tau_3$, $\tau_1$, $\tau_2$ with initial periods, which has the total cost $\sum_{i=1}^3 V_i^0 = 1246.8$.

Applying the heuristic priority assignment method, the tasks are sorted by $C_i^w/\sqrt{b_i}$ in ascending order. It turns out that $\tau_2$, $\tau_3$, $\tau_1$ is the priority order, which is same as the global optimal priority assignment order for this example.

## Randomly Generated Examples

To further investigate the performance of the codesign method, 10 sets of 3 plants are randomly generated from three plant families, which are the same as in Section 4.5.

For each $P_i(s)$, the cost weighting matrices and noise covariance matrices are randomly generated as

$$\begin{aligned} Q_1 &= 10^{3p} C^T C, \quad Q_2 = 1, \\ R_1 &= 10^{3q} B B^T, \quad R_2 = 0.01, \end{aligned} \tag{7.30}$$

where $B$ and $C$ are state-space matrices of the plant $P_i(s)$ in controllable canonical form, $p$ and $q$ are random numbers on unif$(0, 1)$. We then design a standard LQG controller $K_i(s)$ for each $P_i(s)$ assuming a constant delay $C_i^w$. The initial sampling period is chosen using (7.28). The initial utilization $U_i^0$ for each task is assigned by the UUniFast method, and the execution times are given by $C_i^w = h_i^0 U_i^0$.

For the given execution time $C_i^w$ and periods $h_i^0$, and with the initial priority assignment and using the affine cost function approximations, we calculated the following four overall costs for each set of randomly generated plants:

- Initial overall cost $\boldsymbol{V}_{\text{ini}}$. This cost is calculated for the initial periods $h_i^0$ and with rate-monotonic priority assignment. This is our baseline approach.
- Overall cost after re-assigning the priorities $\boldsymbol{V}_{\text{pri}}$. This cost is calculated for the initial periods $h_i^0$ with heuristic priority assignment.
- Overall cost after re-assigning priorities and periods $\boldsymbol{V}_{\text{heuristic}}$. We re-assign the priorities with the heuristic method and calculate the optimal period assignment and then the overall cost is evaluated.
- Global optimal overall cost $\boldsymbol{V}_{\text{global}}$. Here we calculate the optimal periods for all the permutations of priority assignments and find the minimal overall cost.

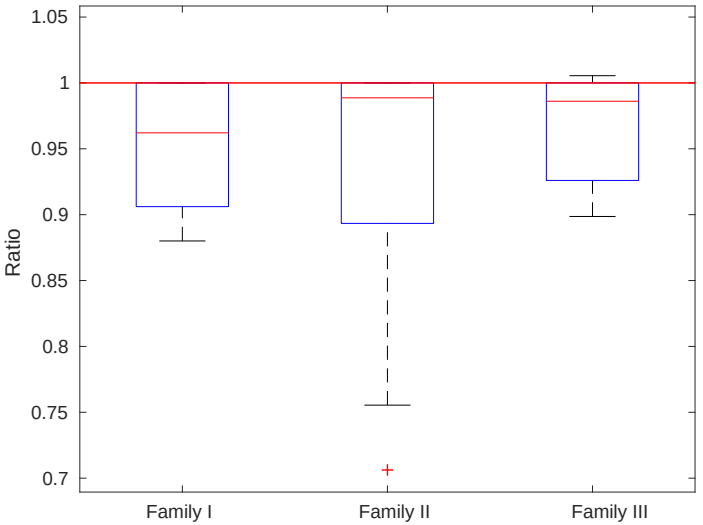



**Figure 7.7** Ratio of $V_{\text{pri}}/V_{\text{ini}}$ for the randomly generated examples with heuristic priority assignment.

We compare the ratio of $V_{\text{pri}}/V_{\text{ini}}$ and the ratio of $V_{\text{heuristic}}/V_{\text{ini}}$ and $V_{\text{global}}/V_{\text{ini}}$ in Figure 7.7 and 7.8, respectively.

Figure 7.7 shows the performance improvement by priority assignment. In most cases, the overall cost is decreased using the heuristic priority assignment, compared to the initial rate-monotonic priority assignment. Hence, using rate-monotonic priority assignment which is often done in the real-time control community is not necessarily the best. Here only the priorities were re-assigned using the heuristic method (keeping the same sampling periods), and it gives the lower overall cost.

Figure 7.8 shows the performance improvement by period assignment. In all cases, the overall cost is decreased using optimal period assignment. It also shows that $V_{\text{heuristic}}$ is very close to $V_{\text{global}}$ in most randomly generated sets of plants. The results imply that the period assignment improves the overall LQG performance, and the performance obtained using the heuristic method gives almost as good results as a much more expensive exhaustive enumeration.

In summary, the results show that the heuristic solution to the priority assignment problem gives almost the same result as the global solution, and both of them are better than the initial priority assignment, which is the rate-monotonic priority assignment based on the initial sampling periods. In most cases, the priority assignments by the global solution and by the heuristic solution coincide, but the global solution requires evaluating all the possible priority assignments, which is unfeasi-

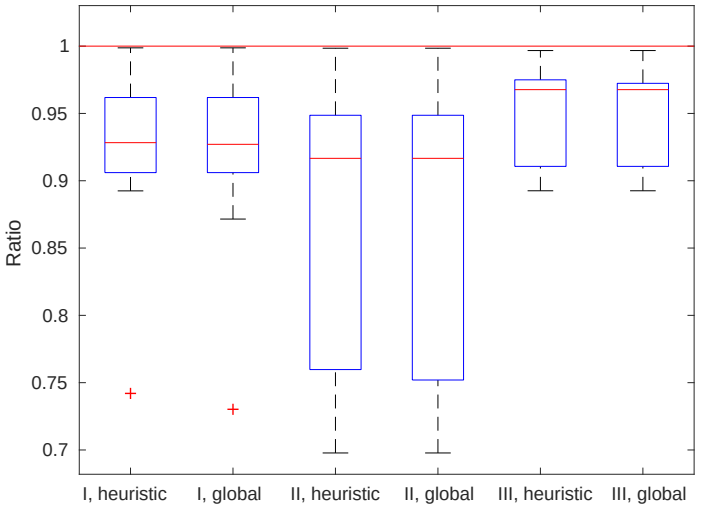



**Figure 7.8** Ratio of $V_{\text{heuristic}}/V_{\text{ini}}$ and $V_{\text{global}}/V_{\text{ini}}$ for the randomly generated examples with optimized periods.

ble for large task sets. The overall LQG performance can be further improved by using the proposed period assignment method. In the evaluation the performance improvement is visible by comparing Figure 7.7 and Figure 7.8. The median cost reduction approximately doubles when the period optimization is performed as well.

## 7.7 Conclusion

In this chapter, we proposed an LQG control synthesis method with a robustness constraint, in the context of real-time system scheduling and control codesign. The initial sampling period assignment and the robustness constraint are both based on the jitter margin of the control system. The approximated affine cost function shows that the cost is more sensitive to jitter than to the value of the sampling periods. A priority and period assignment method is given based on the affine cost functions. The evaluation shows that both priority assignment and period assignment are important in obtaining the overall best performance while retaining the initial robustness of the control loops.

# 8

# Evaluation

In this chapter, the codesign methods from previous chapters are compared and evaluated in more design examples. First, the methods from Chapters 4 to 7 are evaluated in a simple codesign example. Then the methods from Chapters 4 to 6 are compared in a large study on randomly generated plants, tasks, and associated LQG controllers. To allow for a large number of cases to be investigated, Jitterbug is used throughout to evaluate the resulting performance.

## 8.1 A Simple Codesign Example

As a simple codesign example, assume three plants,

$$\begin{aligned} P_1(s) &= \frac{2}{s^2-1}, \\ P_2(s) &= \frac{2}{s^2}, \\ P_3(s) &= \frac{1}{s(s+1)}, \end{aligned} \tag{8.1}$$

that should be controlled by three tasks $\tau_1$, $\tau_2$, $\tau_3$. The task execution times are assumed constant and given by $C_1^w = 0.10$, $C_2^w = 0.09$, $C_3^w = 0.06$. For each controller, the goal is to minimize the quadratic cost function

$$V_i = \lim_{T\to\infty} \frac{1}{T} \mathrm{E} \int_0^T \left( y_i^2(t) + u_i^2(t) \right) dt \tag{8.2}$$

when the plant is subject to white input noise with intensity 1 and white measurement noise with intensity 0.1. This translates to the LQG design matrices

$$Q_{1ci} = C_i^T C_i, \quad Q_{2ci} = 1, \quad R_{1ci} = B_i B_i^T, \quad R_{2ci} = 0.1 \tag{8.3}$$

where $(A_i, B_i, C_i, 0)$ is the controllable canonical state-space realization of the plant $P_i$.

**Table 8.1** Key control loop parameters for the simple example.

| | Loop 1 | Loop 2 | Loop 3 |
|---|---|---|---|
| 3 dB bandwidth ($\omega_b$) | 3.22 rad/s | 2.78 rad/s | 0.98 rad/s |
| Maximum sensitivity ($M_s$) | 2.36 | 1.91 | 1.38 |
| Jitter margin ($J_m^c$) | 0.302 | 0.398 | 1.41 |
| Recommended periods | [0.045, 0.181] | [0.060, 0.239] | [0.212, 0.847] |

Prior to implementation, a continuous-time LQG controller $K_i(s)$ has been designed for each plant to give satisfactory performance. The cost functions are normalized so that $V_i = 1$ corresponds to continuous control. Running all continuous controllers hence gives the overall cost $V = \sum_{i=1}^{3} V_i = 3$.

For each control loop, some key parameters are reported in Table 8.1. The 3 dB bandwidth, $\omega_b$, is the frequency at which the closed-loop gain has dropped to 0.707, and is a measure of the speed of the controlled system. The maximum sensitivity, $M_s$, is calculated as

$$M_s = \max_{\omega} \left| \frac{1}{1 + P(i\omega)K(i\omega)} \right|, \tag{8.4}$$

and is an indicator of the robustness of the control loop. Typical values for robust control systems are in the range $[1.4, 2.0]$; the lower the value of $M_s$, the larger the stability margin is. Finally the continuous-time jitter margin is calculated as

$$J_m^c = \left( \max_{\omega} \left| \frac{\omega P(i\omega)K(i\omega)}{1 + P(i\omega)K(i\omega)} \right| \right)^{-1}. \tag{8.5}$$

It is seen that the first control loop has the worst robustness (largest $M_s$), while the third loop has the best robustness. This is natural since the last plant is open-loop stable. The ranges of recommended sampling periods are calculated using the new rule of thumb (7.5) proposed in Chapter 7. The execution times have been selected so that, selecting initial sampling periods $T^{\text{init}}$ near the upper recommended values results in full processor utilization ($U = 1$).

Priorities are assigned rate-monotonically according to the initial periods and are kept constant throughout all the different methods below. (The heuristic priority assignment rule in Chapter 7 was proposed for jitter-robust LQG and may not necessarily give good results for other LQG design techniques.)

We start by evaluating three different LQG design methods for a fixed set of sampling periods given by $T = T^{\text{init}}/U_b$, where $U_b$ is the chosen utilization bound:

- **Initial LQG design.** Each controller is designed without regard for the scheduling, assuming zero jitter and a constant delay $\delta_i = C_i^w$.

- **Delay-aware LQG design.** The delay of each controller is assumed constant and given by the approximate response-time formula (3.6) from [Bini and

**Table 8.2** Design results for the simple example with utilization bound $U_b = 0.99$.

| | $T_1$ | $T_2$ | $T_3$ | $V$ |
|---|---|---|---|---|
| Initial LQG design | 0.183 | 0.241 | 0.856 | 4.73 |
| Delay-aware LQG design | 0.183 | 0.241 | 0.856 | 4.43 |
| Jitter-robust LQG design | 0.183 | 0.241 | 0.856 | 4.68 |
| Stochastic LQG codesign | 0.260 | 0.261 | 0.436 | 3.76 |
| Periodic LQG codesign | 0.227 | 0.227 | 0.396 | 4.00 |
| Harmonic LQG codesign | 0.160 | 0.320 | 0.640 | 3.57 |

Cervin, 2008]. Based on this, a standard LQG controller compensating for the fixed delay is designed.

- **Jitter-robust LQG design.** The delay of each controller is assumed to vary between $C_i^w$ and $R_i^w$, where $R_i^w$ is the worst-case response time; the jitter is hence $J_i = R_i^w - C_i^w$. A jitter-robust controller is designed using the method from Chapter 7 with the required jitter margin $J_i^{\text{req}} = J_{mi}^c + J_i$.

We further evaluate three different codesign methods, where both the periods and the controllers are redesigned:

- **Stochastic LQG codesign.** Following the procedure in Chapter 4, we use the initial sampling periods as a starting point and calculate the first 100 job response times to obtain the delay probability distributions for each controller. This data is then used to design a set of stochastic LQG controllers. The whole procedure is repeated using the DIRECT global search method for the smallest total cost, using a maximum of 10 iterations.

- **Periodic LQG codesign.** Using the optimized stochastic LQG periods as a starting point, the solution is perturbed with the tolerance $\varepsilon = 0.05$ to yield a finite hyperperiod as described in Chapter 5. A set of time-varying LQG controllers are then designed based on the resulting cyclic schedule.

- **Harmonic LQG codesign.** We again take the stochastic LQG periods as a starting point and enumerate the $2^{n-1} = 4$ closest harmonic task period sets as proposed in Chapter 6. For each resulting task set, we assign task release offsets to minimize the control delay and design a standard LQG controller for the remaining constant delay. All four sets of controllers are evaluated and the one with the smallest total cost is selected.

The resulting periods and the total LQG cost are shown in Table 8.2 and Table 8.3, where all numbers have been rounded to three significant digits. In the evaluation, two utilization bounds, $U_b = 0.99$ and $U_b = 0.78$, have been included. In the first case, we used the bound 0.99 instead of 1 to avoid numerical errors in

**Table 8.3** Design results for the simple example with utilization bound $U_b = 0.78$.

| | $T_1$ | $T_2$ | $T_3$ | $V$ |
|---|---|---|---|---|
| Initial LQG design | 0.232 | 0.306 | 1.09 | 4.01 |
| Delay-aware LQG design | 0.232 | 0.306 | 1.09 | 3.99 |
| Jitter-robust LQG design | 0.232 | 0.306 | 1.09 | 4.06 |
| Stochastic LQG codesign | 0.330 | 0.332 | 0.544 | 3.78 |
| Periodic LQG codesign | 0.290 | 0.290 | 0.483 | 3.89 |
| Harmonic LQG codesign | 0.160 | 0.320 | 0.640 | 3.57 |

response-time calculations. In the second case, we use Liu and Layland's sufficient schedulability bound for three tasks under RM scheduling.

As shown in Table 8.2 and Table 8.3, the delay-aware LQG cost is smaller than the initial LQG cost, because the delay-aware LQG design takes the RiApprox delay into the design procedure, which is a better approximation of the true delay. The stochastic LQG has a lower cost than the initial LQG and the delay-aware LQG methods, because more response time information is taken into account in the design procedure and the optimization is conducted for the period assignment. In this particular example, the periodic LQG cost is worse than the delay-aware cost, because the selected periods of periodic LQG deviate from the stochastic LQG periods, which are the result of the global search. But periodic LQG cost is still lower than the initial and the delay-aware LQG cost. The harmonic LQG method has better performance than all the other methods. The reason is that, by using harmonic periods and task offsets, the schedule will give rise to both constant and short delays. This positive effect dominates over the negative effect of having to deviate quite far from the optimal, real-valued periods.

It is also seen that using the lower utilization bound gives better results for many of the design methods. The reason seems to be that jitter can become extreme when utilization is close to 1, and this can cause large performance degradation for the low-priority tasks under $U_b = 0.99$. With $U_b = 0.78$ the jitter is always smaller than one sampling period, and the performance degradation of the lower-priority tasks are kept in check.

In the next section, a larger evaluation is performed using randomly generated plant dynamics, showing the variability and confidence of the results obtained.

## 8.2 Randomly Generated Codesign Examples

To see whether the results for the simple example above hold in more general cases, sets of three plants have been randomly generated for evaluation from the following four plant families:

- Family A: All plants have two real or complex stable poles. They are drawn

from

$$P_A(s) = \frac{K}{T^2 s^2 + 2\zeta T s + 1}, \tag{8.6}$$

where $T = 10^{-2\alpha}$, $K = 2 \times 10^{\beta}$, $\zeta = 2\gamma$, and $\alpha, \beta, \gamma \in \text{unif}(0, 1)$.

- Family B: All plants have two real or complex stable poles and one integrator. They are drawn from

$$P_B(s) = \frac{K}{Ts(T^2 s^2 + 2\zeta T s + 1)}, \tag{8.7}$$

where $T = 10^{-2\alpha}$, $K = 0.3 \times 10^{\beta}$, $\zeta = 2\gamma$, and $\alpha, \beta, \gamma \in \text{unif}(0, 1)$.

- Family C: All plants have two real or complex stable poles, one stable or unstable zero and one integrator. They are drawn from

$$P_C(s) = \frac{K[1 - \text{sgn}(\lambda - 0.5)TT_z s]}{Ts(T^2 s^2 + 2\zeta T s + 1)}, \tag{8.8}$$

where $T = 10^{-2\alpha}$, $K = 1.3 \times 10^{\beta}$, $\zeta = 2\gamma$, $T_z = 10^{\eta - 1}$, and $\alpha, \beta, \gamma, \eta, \lambda \in \text{unif}(0, 1)$.

- Family D: All plants have two real or complex stable poles, one stable or unstable pole and one integrator. They are drawn from

$$P_D(s) = \frac{K}{Ts(T^2 s^2 + 2\zeta T s + 1)[1 - \text{sgn}(\lambda - 0.5)TT_p s]}, \tag{8.9}$$

where $T = 10^{-2\alpha}$, $K = 10^{\beta}$, $\zeta = 2\gamma$, $T_p = 10^{\eta + 1}$, and $\alpha, \beta, \gamma, \eta, \lambda \in \text{unif}(0, 1)$.

Compared to the random plants in earlier chapters, these plant families represent more realistic cases, which are not only dominated by unstable dynamics. Integrators and zeros have also been included. The plant parameters have been chosen to give reasonable robustness (with typical maximum sensitivity values between 1.2 and 2.6). Also a random time constant $T$ has been included to generate plants with possibly very different time scales.

25 sets of three plants are randomly generated for each family. All plants are converted to controllable canonical state-space form, and continuous-time LQG controllers are designed using the parameters $Q_{1ci} = C_i^T C_i$, $Q_{2ci} = 1$, $R_{1ci} = B_i B_i^T$, $R_{2i} = 0.01 \times 10^{2\sigma}$, where $\sigma \in \text{unif}(0, 1)$. Box plots of the maximum sensitivity, $M_s$, of the control loops from each family are given in Figure 8.1. It is seen that control loops based on Family A have the highest robustness, while loops from Family D have worse robustness (indicated by the larger $M_s$ values).

For the initial sampling period assignment, we first calculate the continuous-time jitter margin, $J_{mi}^c$. Using the rule of thumb for the period selection, the initial

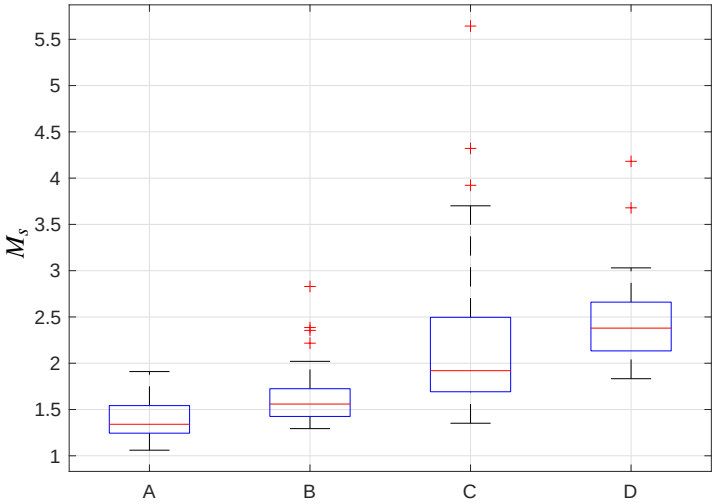



**Figure 8.1** Maximum sensitivity values for the randomly generated plants in Families A–D.

sampling period is calculated by a coefficient multiplied by the continuous-time jitter margin of the plant

$$T_i^{\text{init}} = cJ_{mi}^c. \tag{8.10}$$

To investigate a wide range of sampling periods, from fast sampling to very slow sampling (even violating the rule of thumb), we choose the constant $c \in \{0.15, 0.3, 0.45, 0.6, 0.75, 0.9\}$. Because each plant is controlled by a LQG controller with similar complexity, we assume, in a set of plants, that the greatest execution time, out of three, is not larger than twice the execution time of the smallest one. To reach the utilization bound, $U = U_b$, where $U_b \in \{0.99, 0.78\}$, when the initial sampling periods are used, we assign a constant execution time to each task as

$$C_i^w = \frac{U_b\,(1+e_i)}{\sum_{j=1}^{3}\left(\frac{1+e_j}{T_j^{\text{init}}}\right)}, \tag{8.11}$$

where $e_i \in \text{unif}(0,\,1)$ for $i \in \{1, 2, 3\}$.

Using this setup, we evaluate the following costs. In the initial, delay-aware, stochastic, and periodic LQG evaluations, we use the utilization bound $U_b$.

- Continuous-time LQG cost $V^{\text{ct}}$. For each plant, we design a continuous-time LQG controller, and the continuous-time LQG cost, $V_i^{\text{ct}}$, is evaluated in Jitterbug. In order to more easily compare the results between different plants, we normalize $V_i^{\text{ct}}$ to 1 for each control loop. $V^{\text{ct}}$ is the sum of the normalized cost value, so $V^{\text{ct}} = 3$ after normalization.

- Execution time LQG cost $V^{\text{et}}$. For each plant, we design a discrete-time LQG controller when the sampling period is chosen as the execution time ($T_i = E_i$) and the constant delay is the same as the execution time ($\delta_i = E_i$). This represents an ideal case, where the controller can fully utilize its own processor and execute as frequently as possible. The LQG cost is evaluated in Jitterbug as $V_i^{\text{et}}$. We define

$$V^{\text{et}} = \sum_{i=1}^{3} \frac{V_i^{\text{et}}}{V_i^{\text{ct}}}. \tag{8.12}$$

- Initial LQG cost $V^{\text{init}}$. The LQG controllers are designed for $T_i^{\text{init}}$ as the period and $E_i$ as the constant delay. For each task, we calculate the first 1000 response times to estimate the response time distribution. Using this distribution as the delay distribution in Jitterbug, the LQG cost $V_i^{\text{init}}$ is evaluated. $V^{\text{init}}$ is calculated by

$$V^{\text{init}} = \sum_{i=1}^{3} \frac{V_i^{\text{init}}}{V_i^{\text{ct}}}. \tag{8.13}$$

- Delay-aware LQG cost $V^{\text{da}}$. We again design the LQG controller with $T_i^{\text{init}}$ as the period but now with $R_i^{\text{RiApprox}}$ as the constant delay. The first 1000 response times are used to calculate the response time distribution. Then the distribution is used in Jitterbug evaluation. We calculate $V^{\text{da}}$ by using

$$V^{\text{da}} = \sum_{i=1}^{3} \frac{V_i^{\text{da}}}{V_i^{\text{ct}}}. \tag{8.14}$$

- Stochastic LQG cost $V^{\text{s}}$. First we use the DIRECT optimization described in Chapter 4 with the utilization bound $\sum_{i=1}^{3} U_i \leq U_b$. In the optimization, we calculate the delay distribution from the first 1000 response times, and use the delay distribution to design the LQG controller. The LQG cost is evaluated using the delay distribution. The period we get from the stochastic LQG design is $T_i^{\text{s}}$. $V^{\text{s}}$ is defined as

$$V^{\text{s}} = \sum_{i=1}^{3} \frac{V_i^{\text{s}}}{V_i^{\text{ct}}}. \tag{8.15}$$

- Periodic LQG cost $V^{\text{p}}$. Using $T_i^{\text{s}}$ as the initial value, the period perturbation method in Chapter 5 is used to calculate the periods with a short hyperperiod. Then the periods are scaled such that $U = U_b$. In the evaluation procedure, for each task, if a hyperperiod contains more than 20 periods or the largest delay is greater than the period, we use the stochastic LQG controller and use the delay distribution in the evaluation; otherwise, a set of controllers

are designed using the periodic LQG method, and the deterministic repeated pattern of the delays in a hyperperiod is used to evaluate the cost. We use

$$V^{\mathrm{p}} = \sum_{i=1}^{3} \frac{V_i^{\mathrm{p}}}{V_i^{\mathrm{ct}}}. \tag{8.16}$$

- Harmonic LQG cost $V^{\mathrm{h}}$. We use $T_i^{\mathrm{s}}$ as the initial period and the harmonic LQG design method in Chapter 6 to evaluate the four closest harmonic task period sets with full utilization. The LQG controller is designed using a constant delay, equal to the difference between the response time and the start latency. The LQG cost is evaluated with an offset which is equal to the start latency. $V^{\mathrm{h}}$ is defined as

$$V^{\mathrm{h}} = \sum_{i=1}^{3} \frac{V_i^{\mathrm{h}}}{V_i^{\mathrm{ct}}}. \tag{8.17}$$

We consider two options for the overall utilization bound $U_b$: 0.99 and 0.78. In the full utilization case, we use 0.99 to avoid numerical errors in the evaluation. 0.78 is the Liu and Layland sufficient utilization bound for a system with three tasks.

The costs, averaged over 25 generated plant sets for each family and each utilization bound $U_b$, are shown in Figures 8.2, 8.4, 8.6, 8.8, 8.10, 8.12, 8.14, and 8.16. To see the variability of the results we also show the box plots of the costs over the 25 sets of the randomly generated plants for the case $T^{\mathrm{init}} = 0.6J_m^c$ in Figures 8.3, 8.5, 8.7, 8.9, 8.11, 8.13, 8.15, and 8.17.

As seen in the figures, $V^{\mathrm{ct}} = 3$ is used as the baseline in the evaluation. $V^{\mathrm{et}}$ is evaluated when a task fully uses a CPU, so there is not any uniprocessor scheduling method that can outperform it, as shown in the figures. $V^{\mathrm{da}}$ is lower than $V^{\mathrm{init}}$. The reason is that, in the initial and the delay-aware LQG, although the sampling periods are the same and both use a constant delay to design the controller, the delay-aware LQG uses the RiApprox method to better approximate the non-constant delay. $V^{\mathrm{init}}$ is evaluated using the execution times as delays, which is not a good approximation.

The stochastic LQG design always gives better performance than the delay-aware design, because the stochastic uses the delay distribution to design the controllers, and uses global search to assign sampling periods, which leads to a lower total cost. When $U_b = 0.78$, the periodic LQG outperforms the stochastic LQG, because the periodic LQG takes the deterministic repeated pattern of response times into account in the controller design, which is better than a stochastic interpretation of the delay. However, the periodic LQG is more sensitive to the increased utilization. So when $U_b = 1$, the performance of the stochastic LQG is better than the periodic LQG. Harmonic LQG always has a better performance compared to stochastic and periodic LQG, the reason being that in harmonic LQG, the delay is always constant and a release offset is assigned to shorten the delay.

For the given examples, it is seen that the cost increases monotonically with the sampling periods in all cases. This is to be expected, since the controllers can act less

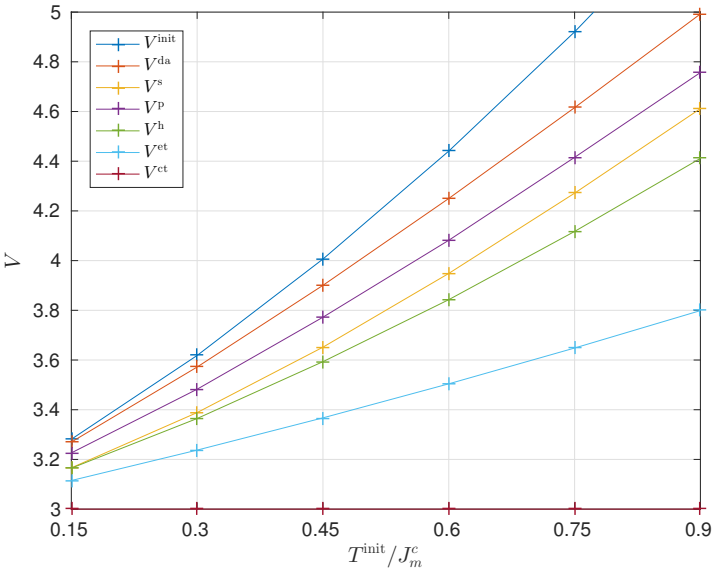



**Figure 8.2** Average costs for Family A, $U_b = 0.99$.

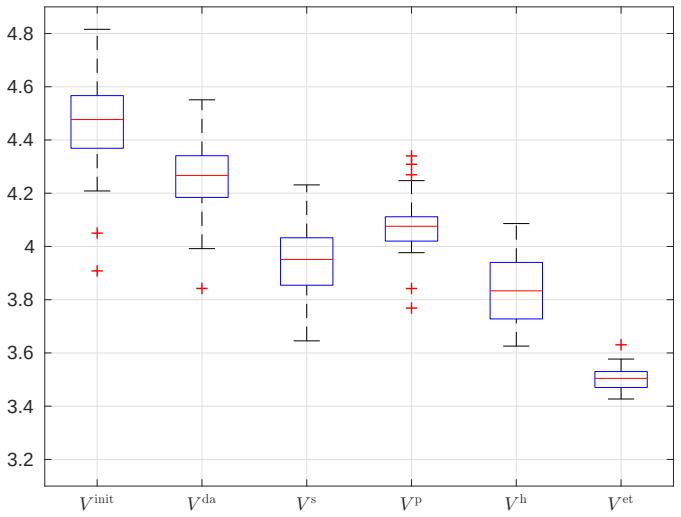



**Figure 8.3** Box plots of the costs for Family A, $U_b = 0.99$, $T^{\text{init}} = 0.6J_m^c$.

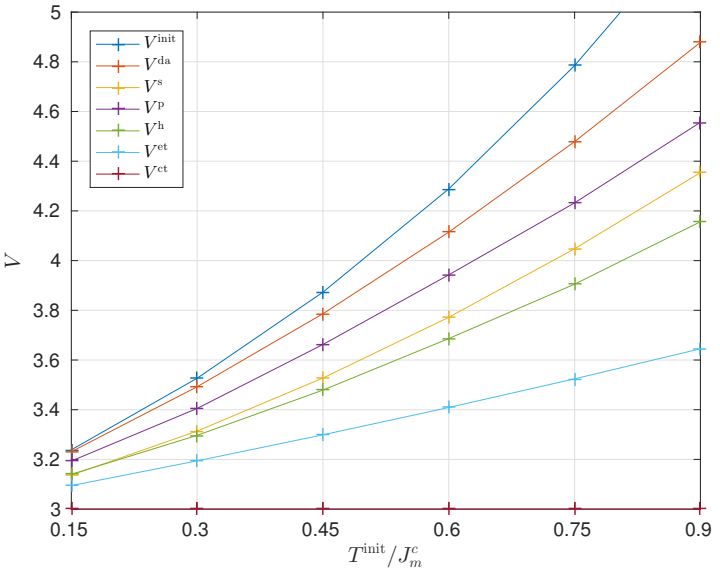



**Figure 8.4** Average costs for Family B, $U_b = 0.99$.

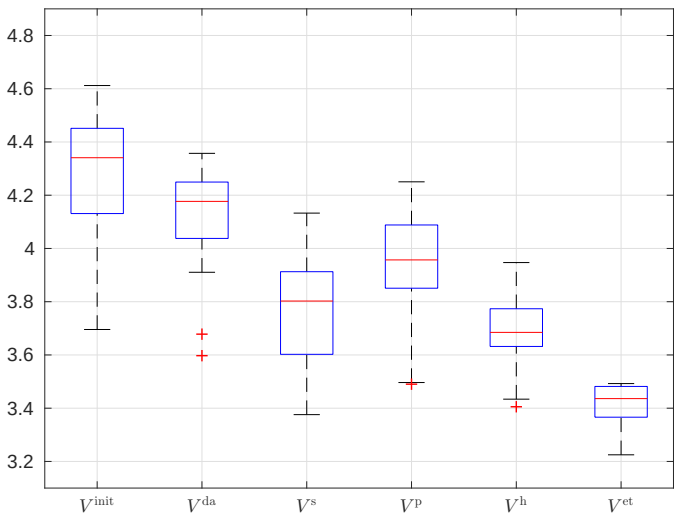



**Figure 8.5** Box plots of the costs for Family B, $U_b = 0.99$, $T^{\text{init}} = 0.6J_m^c$.

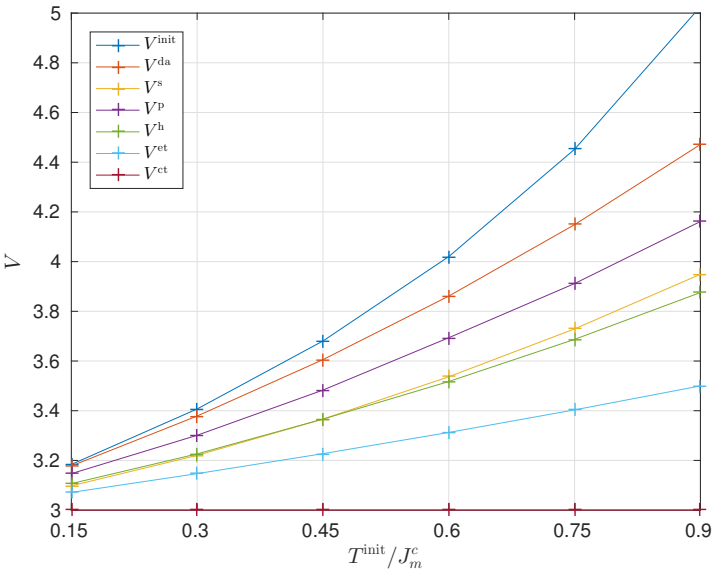



**Figure 8.6** Average costs for Family C, $U_b = 0.99$.

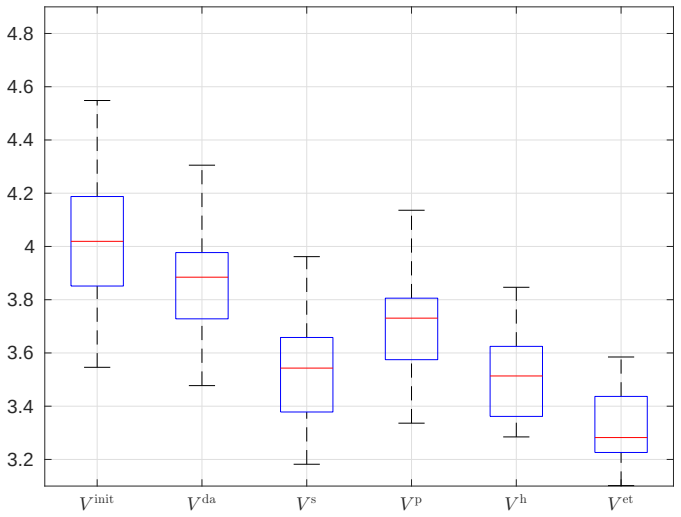



**Figure 8.7** Box plots of the costs for Family C, $U_b = 0.99$, $T^{\text{init}} = 0.6J_m^c$.

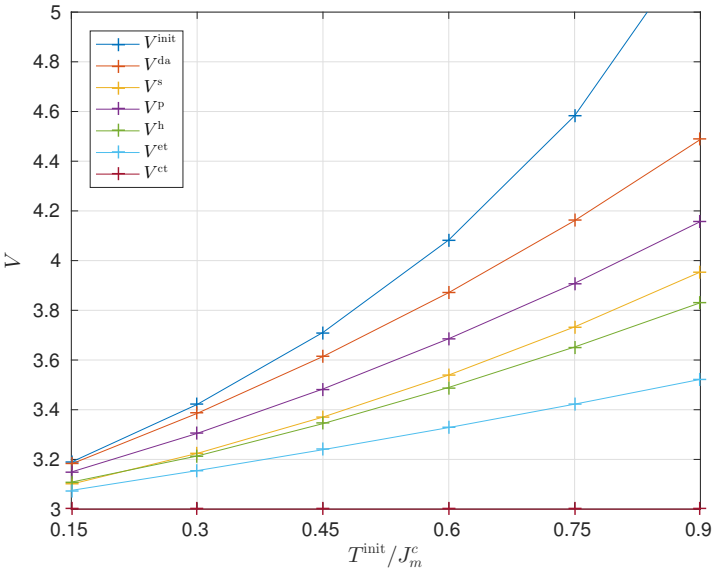



**Figure 8.8** Average costs for Family D, $U_b = 0.99$.

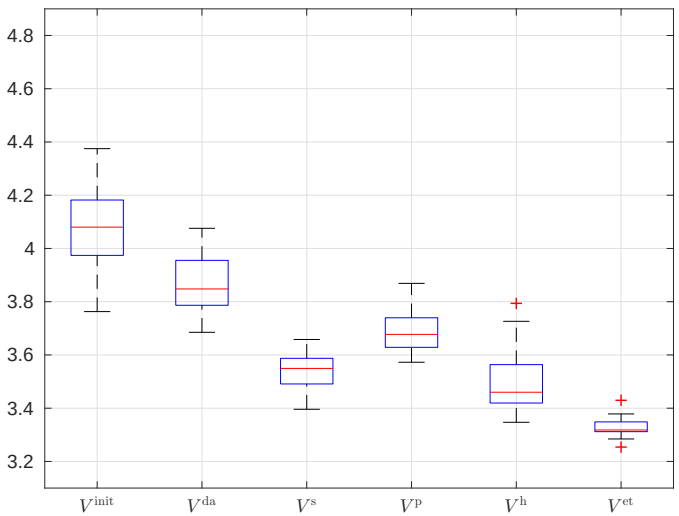



**Figure 8.9** Box plots of the costs for Family D, $U_b = 0.99$, $T^{\text{init}} = 0.6J_m^c$.

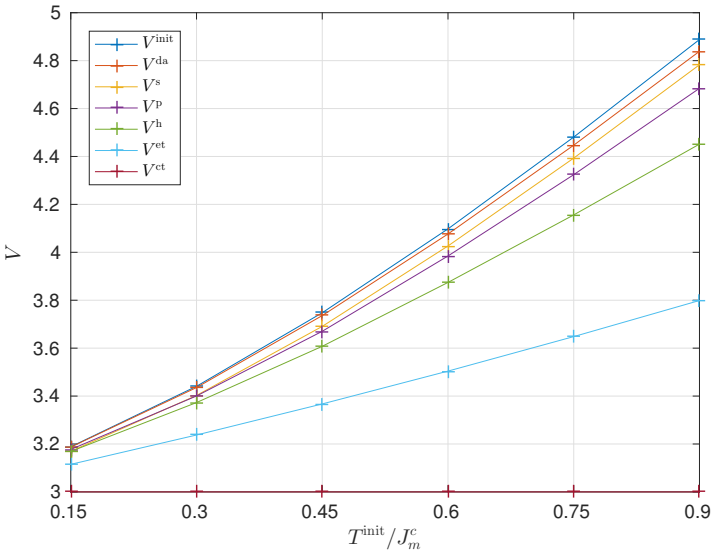



**Figure 8.10** Average costs for Family A, $U_b = 0.78$.

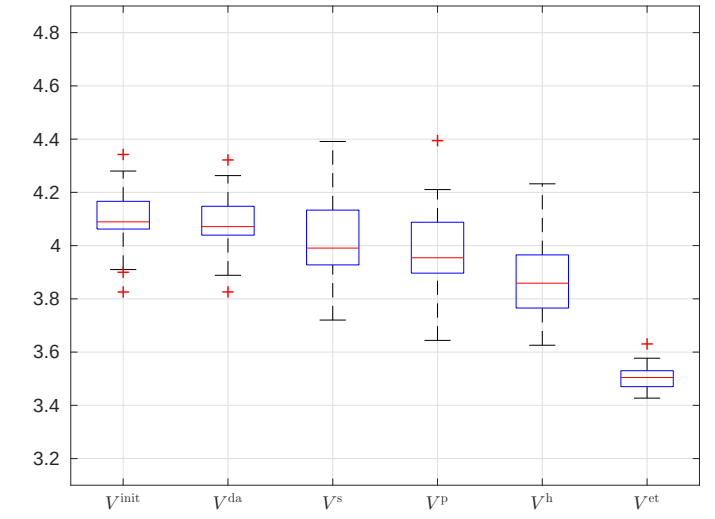



**Figure 8.11** Box plots of the costs for Family A, $U_b = 0.78$, $T^{\text{init}} = 0.6J_m^c$.

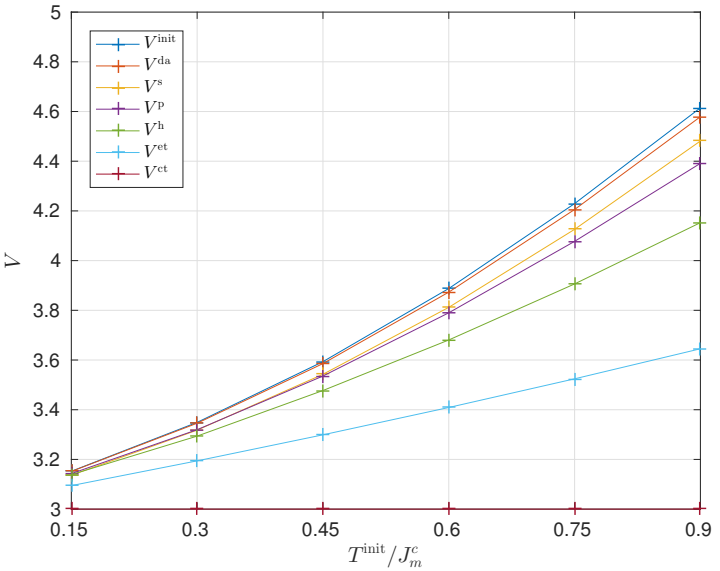



**Figure 8.12** Average costs for Family B, $U_b = 0.78$.

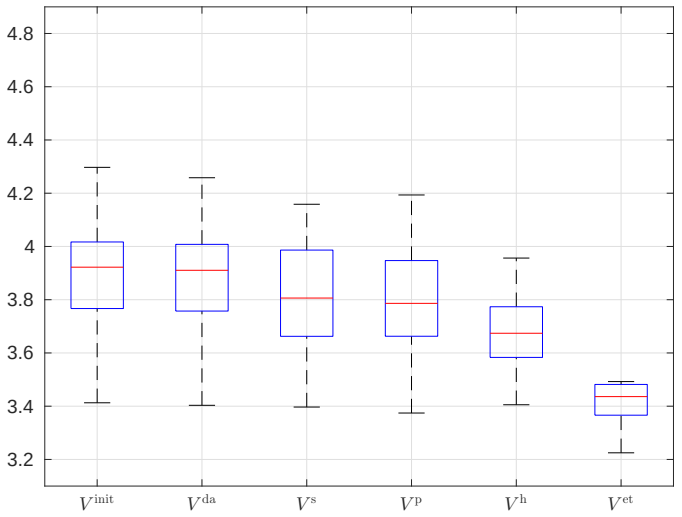



**Figure 8.13** Box plots of the costs for Family B, $U_b = 0.78$, $T^{\text{init}} = 0.6J_m^c$.

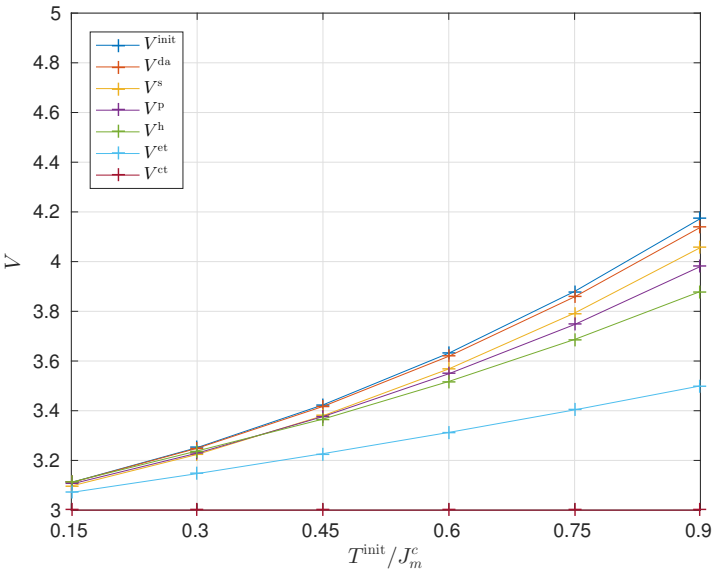



**Figure 8.14** Average costs for Family C, $U_b = 0.78$.

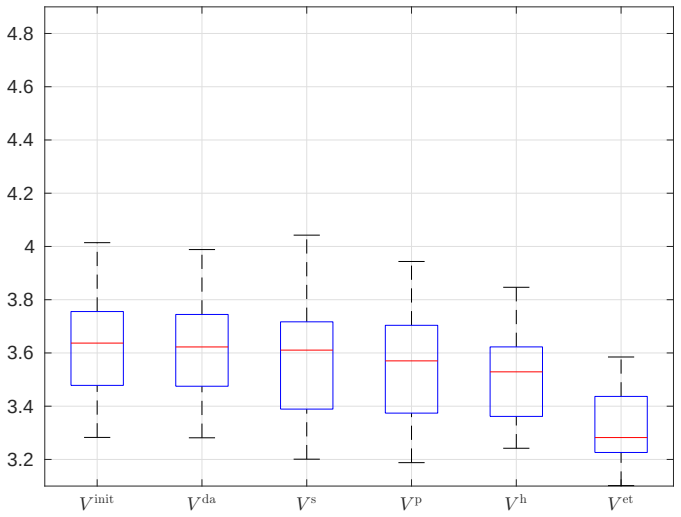



**Figure 8.15** Box plots of the costs for Family C, $U_b = 0.78$, $T^{\text{init}} = 0.6J_m^c$.

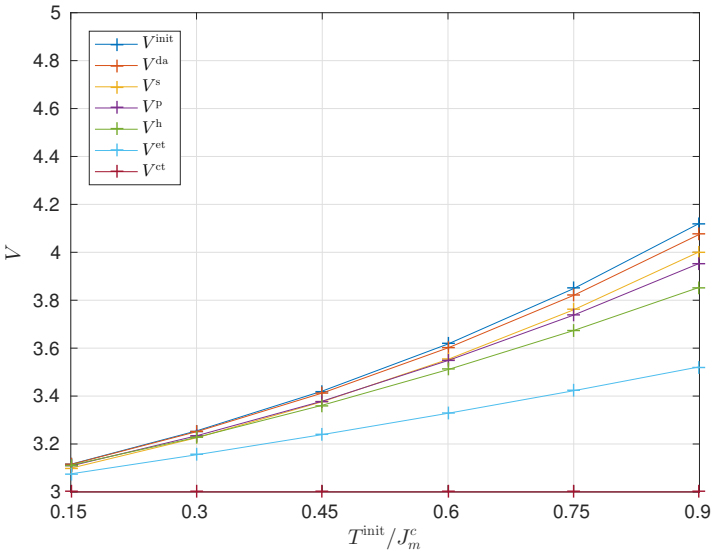



**Figure 8.16** Average costs for Family D, $U_b = 0.78$.

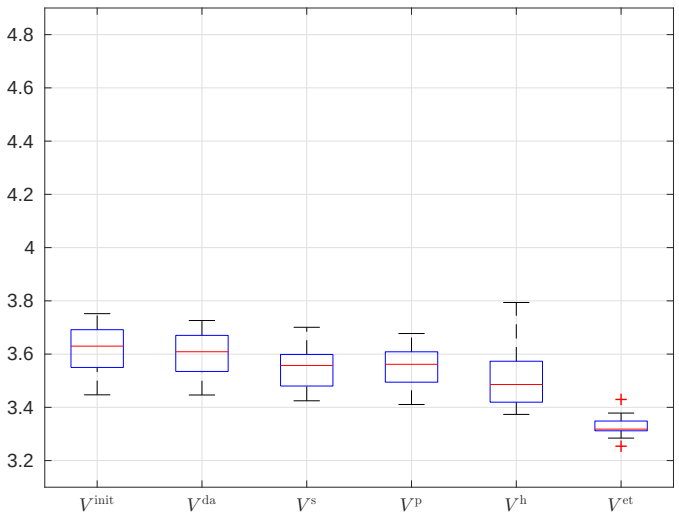



**Figure 8.17** Box plots of the costs for Family D, $U_b = 0.78$, $T^{\text{init}} = 0.6J_m^c$.

often to reject disturbances when the period is large. The performance degradation is not however particularly large; even as slow sampling as $T^{\text{init}} = 0.9J_m^c$ only gives a cost increase of at most 60% for all of the codesign methods. This is an indication of that the rule of thumb for initial period selection is quite robust.

From Family A to Family D, the sensitivity of the plants to delays and jitter increases. As shown in Figure 8.1, the maximum sensitivity increases, and, hence, the robustness decreases. This actually makes the rule of thumb for sampling interval selection choose smaller periods for Family D, which ends up making those examples more robust towards scheduling-induced jitter.

Comparing the two cases of $U_b = 0.99$ and $U_b = 0.78$, the costs are higher when $U_b = 0.99$, because the high utilization leads to long average delays and also large jitter for the low-priority tasks. However, the relative performance improvement for $U_b = 0.99$ is higher when applying the codesign methods.

One final conclusion from the evaluation is that, if computing resources are abundant enough to allow choosing all the sampling periods to be short, e.g., $T^{\text{init}} = 0.15J_m^c$, then scheduling and control codesign becomes a non-issue and any implementation will give satisfactory performance.

# 9

# Conclusion

## 9.1 Summary

This thesis presented four scheduling and control codesign methods for optimizing the LQG performance of multitasking control applications, such that the scheduling constraints are satisfied.

We began by presenting a stochastic LQG codesign method (Chapter 4) for designing LQG controllers, taking the delay distributions into account. The major contribution of the stochastic LQG codesign is the proposed period assignment method based on the task response time distribution and the sequential search-based algorithm. Through a simulation-based evaluation, it was shown that the control performance, as measured by the LQG cost, was improved compared to previous approaches and the local search-based heuristic algorithm gave results that were very close to the solution obtained with a global sampling-based optimization method, with 50% less computation effort compared to the latter method. In Chapter 8, the method performed very well for a wide range of sampling periods and plant dynamics in randomized design examples.

In Chapter 5 we described how the periodic pattern of response times can be used in periodic LQG codesign. We presented periodic and periodic–stochastic LQG control designs for minimizing the overall cost in real-time control systems. In the case of periodic–stochastic LQG, also knowledge of the probability distributions of the response times of each job is required. Since both approaches rely on knowledge of the response-time pattern of each task, they require more analysis and design effort than stochastic LQG codesign. The resulting controller has a time-periodic gain, meaning that the implementation is also more complicated. In Chapter 8, it was seen that the method performed at its best at processor utilizations lower than 1, where in many cases it could beat the stochastic LQG codesign method.

In Chapter 6 we presented the harmonic LQG scheduling and control codesign method, in which the sampling periods are perturbed to be harmonic. In order to implement the codesign method, it is necessary to be able to find the harmonic periods and to calculate the task response times. Two heuristic approaches to har-

monic task period assignment were presented. One method for finding the closest harmonic periods to a set of initial periods and one method for finding all possible harmonic period assignments that satisfy constraints on allowable task period ranges. The resulting controllers are easy to design and implement, since they are based on a constant control delay. Evaluations showed that the method often gives very good results, despite the fact that having harmonic periods is a quite severe design restriction. In Chapter 8 it yielded better performance than both stochastic and periodic LQG codesign for almost all cases.

Robustness is not considered in Chapters 4–6, so Chapter 7 addressed codesign of jitter-robust LQG controllers. We proposed an LQG control synthesis method with a robustness constraint. The initial sampling period assignment and the robustness constraint were both based on the jitter margin of the control system. The approximated affine cost function shows that the cost is more sensitive to jitter than to the sampling period. A priority and period assignment method was given based on the affine cost functions. The evaluation shows that both priority assignment and period assignment are important in obtaining the overall best performance while retaining the initial robustness of the control loops. The performance with this method is not as good as under for instance stochastic LQG, but the main goal here was robustness and not performance.

Finally, Chapter 8 gave further numerical evidence of the relative strength of the stochastic, periodic, harmonic, and robust LQG codesign methods. The results showed that all of the proposed LQG codesign methods achieve better overall control performance than methods that do not take the jitter into account.

The thesis presented multiple methods on LQG-based control and scheduling codesign, and each of these approaches has its benefits. Which one is the best? Generally, harmonic LQG design should be used in control and scheduling codesign. This is motivated by the good properties of harmonic LQG design, such as schedulability with full utilization, constant response time (under the assumption of constant execution times), and constant job start time. Furthermore, harmonic LQG design can be used together with release offsets, in order to shorten the constant response times and hence further improve the control performance.

However, these four methods are independent, so each of them can be chosen for a specific control aim or a specific scheduling timing. For example, if robustness of the control system a major requirement when optimizing the performance, we can use robust LQG design. When the response time pattern is known, the periodic LQG design can provide better performance than stochastic LQG design.

## 9.2 Future Work

Many different directions for future research and extensions of the thesis are possible. These range from continued investigation of real-time scheduling analysis to further development of the control synthesis methods. Although all the limitations

listed in Chapter 1 could have been discussed, here we will only highlight a few of them.

First, development of multicore scheduling and control codesign methods is needed. Several of the codesign methods discussed throughout the thesis can quite easily be extended to partitioned multiprocessor scheduling, because the local schedulers are independent and the single core scheduling can be reused. For example, in the periodic LQG case, the problem becomes how to partition the task set in the best way in order to obtain finite and short hyperperiods for each of local processors while keeping the task periods close to the initial periods. Similarly, in the harmonic LQG case, the problem is how to partition the task set so that harmonic task periods can be obtained for each local processor that keep the task periods close to the initial periods. Since global scheduling suits dynamic configurations and provides optimal schedules, it is also necessary to investigate the codesign methods in the case of global scheduling.

Second, methods for efficient codesign under EDF scheduling could be developed. EDF is an optimal dynamic scheduling strategy for preemptive uniprocessors, and its utilization bound is 1. It allows better usage of limited computational resources, and the jitter also tends to be more evenly distributed between tasks, which may lead to better control performance.

Third, further exploration of other types of controllers types besides LQG may lead to other control and scheduling codesign problems. Another popular design criterion is to minimize the weighted $H_\infty$ norm of the closed-loop system. This approach is popular in robust control and could possibly be extended to handle jitter as well.

Fourth, the assumptions on sampling at the job release time and actuation at the job finishing time could be relaxed. In this case, the delay will be shorter than with periodic sampling. This, however, leads to systems with varying sampling intervals. The LQG theory could be extended to also cover this case, although it is currently not known whether a stochastic Kalman filter and a stochastic state feedback can be combined using the separation principle to form an optimal LQG controller.

# Bibliography


Abeni, L., L. Palopoli, G. Lipari, and J. Walpole (2002). "Analysis of a reservation-based feedback scheduler". In: *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)*. Austin, TX, USA, pp. 71–80.

Aminifar, A., S. Samii, P. Eles, Z. Peng, and A. Cervin (2012). "Designing high-quality embedded control systems with guaranteed stability". In: *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS)*. San Juan, Puerto Rico, pp. 283–292.

Arun, K. S., T. S. Huang, and S. D. Blostein (1987). "Least-squares fitting of two 3-D point sets". *IEEE Transactions on pattern analysis and machine intelligence* 5, pp. 698–700.

Årzén, K.-E. (1999). "A simple event-based PID controller". In: *Proceedings of the 14th IFAC World Congress*. Beijing, China, pp. 423–428.

Årzén, K.-E. and A. Cervin (2005). "Control and embedded computing: survey of research directions". In: *Proceedings of the 16th IFAC World Congress*. Prague, Czech Republic, pp. 191–202.

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 (CDC)*. Sydney, Australia, pp. 4865–4870.

Årzén, K.-E., A. Cervin, and D. Henriksson (2003). "Resource-constrained embedded control systems: possibilities and research issues". In: *Proceedings of the Co-design for Embedded Real-time Systems (CERTS)*. Porto, Portugal.

Åström, K. J. (1970). *Introduction to stochastic control theory*. Academic Press.

Åström, K. J. and B. Wittenmark (2013). *Computer-controlled systems: theory and design*. Courier Corporation.

Åström, K. J. and B. Bernhardsson (1999). "Comparison of periodic and event based sampling for first-order stochastic systems". In: *Proceedings of the 14th IFAC World Congress*. Beijing, China, pp. 5006–5011.

Balbastre, P., I. Ripoll, and A. Crespo (2000). "Control tasks delay reduction under static and dynamic scheduling policies". In: *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. Cheju Island, South Korea, pp. 522–526.

Balbastre, P., I. Ripoll, J. Vidal, and A. Crespo (2004). "A task model to reduce control delays". *Real-Time Systems* **27**:3, pp. 215–236.

Ben Gaid, M. E. M., A. Cela, and Y. Hamam (2006). "Optimal integrated control and scheduling of networked control systems with communication constraints: application to a car suspension system". *IEEE Transactions on Control Systems Technology* **14**:4, pp. 776–787.

Ben Gaid, M. E. M., A. Cela, and Y. Hamam (2009). "Optimal real-time scheduling of control tasks with state feedback resource allocation". *IEEE Transactions on Control Systems Technology* **17**:2, pp. 309–326.

Ben Gaid, M. E. M., A. Cela, and R. Kocik (2004). "Distributed control of a car suspension system". In: *Proceedings of the 5th Eurosim Congress on Modelling and Simulation (EUROSIM)*. Paris, France.

Bini, E. and S. K. Baruah (2007). "Efficient computation of response time bounds under fixed-priority scheduling". In: *Proceedings of the 15th Conference on Real-Time and Network Systems (RTNS)*. Nancy, France, pp. 95–104.

Bini, E. and G. C. Buttazzo (2005). "Measuring the performance of schedulability tests". *Real-Time Systems* **30**:1, pp. 129–154.

Bini, E., G. C. Buttazzo, and G. M. Buttazzo (2003). "Rate monotonic analysis: the hyperbolic bound". *IEEE Transactions on Computers* **52**:7, pp. 933–942.

Bini, E. and A. Cervin (2008). "Delay-aware period assignment in control systems". In: *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS)*. Barcelona, Spain, pp. 291–300.

Bini, E. and M. Di Natale (2005). "Optimal task rate selection in fixed priority systems". In: *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)*. Miami, FL, USA, pp. 399–409.

Bonifaci, V., A. Marchetti-Spaccamela, N. Megow, and A. Wiese (2013). "Polynomial-time exact schedulability tests for harmonic real-time tasks". In: *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*. Vancouver, Canada, pp. 236–245.

Boyd, S. P. and C. H. Barratt (1991). *Linear controller design: limits of performance*. Prentice Hall Englewood Cliffs, NJ.

Boyd, S., C. Crusius, and A. Hansson (1998). "Control applications of nonlinear convex programming". *Journal of Process Control* **8**:5-6, pp. 313–324.

Brocal, V., P. Balbastre, R. Ballester, and I. Ripoll (2011). "Task period selection to minimize hyperperiod". In: *Proceedings of the 16th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE. Toulouse, France, pp. 1–4.

Busquets-Mataix, J. V., J. J. Serrano, R. Ors, P. Gil, and A. Wellings (1996). "Using harmonic task-sets to increase the schedulable utilization of cache-based preemptive real-time systems". In: *Proceedings of the 3rd International Workshop on Real-Time Computing Systems and Applications (RTCSA)*. Seoul, South Korea, pp. 195–202.

Buttazzo, G. and A. Cervin (2007). "Comparative assessment and evaluation of jitter control methods". In: *Proceedings of the 15th Conference on Real-Time and Network Systems (RTNS)*. Nancy, France, pp. 163–172.

Camacho, E. F. and C. B. Alba (2013). *Model predictive control*. Springer Science & Business Media.

Cervin, A. (1999). "Improved scheduling of control tasks". In: *Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS)*. York, UK, pp. 4–10.

Cervin, A. (2012). "Stability and worst-case performance analysis of sampled-data control systems with input and output jitter". In: *Proceedings of the American Control Conference (ACC)*. IEEE. Montreal, Canada, pp. 3760–3765.

Cervin, A., J. Eker, B. Bernhardsson, and K.-E. Årzén (2002a). "Feedback–feedforward scheduling of control tasks". *Real-Time Systems* **23**:1, pp. 25–53.

Cervin, A., D. Henriksson, B. Lincoln, and K.-E. Årzén (2002b). "Jitterbug and TrueTime: analysis tools for real-time control systems". In: *Proceedings of the 2nd Workshop on Real-Time Tools*. Copenhagen, Denmark.

Cervin, A., D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén (2003). "How does control timing affect performance? analysis and simulation of timing using Jitterbug and TrueTime". *IEEE control systems* **23**:3, pp. 16–30.

Cervin, A., D. Henriksson, and M. Ohlin (2016). *TrueTime 2.0—Reference Manual*. Technical Report.

Cervin, A. and B. Lincoln (2003). *Jitterbug 1.1—Reference Manual*. Technical Report 7604. Department of Automatic Control, Lund University, Sweden.

Cervin, A., B. Lincoln, J. Eker, K.-E. Årzén, and G. Buttazzo (2004). "The jitter margin and its application in the design of real-time control systems". In: *Proceedings of the 10th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. Gothenburg, Sweden, pp. 1–9.

Davare, A., Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli (2007). "Period optimization for hard real-time distributed automotive systems". In: *Proceedings of the 44th annual Design Automation Conference (DAC)*. ACM. San Diego, CA, USA, pp. 278–283.

Derler, P., E. A. Lee, M. Törngren, and S. Tripakis (2013). "Cyber-physical system design contracts". In: *Proceedings of the 4th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE. Philadelphia, PA, USA, pp. 109–118.

Díaz, J. L., D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella (2002). "Stochastic analysis of periodic real-time systems". In: *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)*. Austin, TX, USA, pp. 289–300.

Doyle, J. (1978). "Guaranteed margins for LQG regulators". *IEEE Transactions on Automatic Control* **23**:4, pp. 756–757.

Doyle, J., K. Zhou, K. Glover, and B. Bodenheimer (1994). "Mixed $H_2$ and $H_\infty$ performance objectives II. optimal control". *IEEE Transactions on Automatic Control* **39**:8, pp. 1575–1587.

Eisenbrand, F. and T. Rothvoß (2008). "Static-priority real-time scheduling: response time computation is NP-hard". In: *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS)*. Barcelona, Spain, pp. 397–406.

Eker, J., P. Hagander, and K.-E. Årzén (2000). "A feedback scheduler for real-time controller tasks". *Control Engineering Practice* **8**:12, pp. 1369–1378.

Franklin, G. F., J. D. Powell, A. Emami-Naeini, and J. D. Powell (1994). *Feedback control of dynamic systems*. Vol. 3. Addison-Wesley Reading, MA.

Frehse, G., A. Hamann, S. Quinton, and M. Woehrle (2014). "Formal analysis of timing effects on closed-loop properties of control software". In: *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*. Rome, Italy, pp. 53–62.

Goswami, D., M. Lukasiewycz, R. Schneider, and S. Chakraborty (2012). "Time-triggered implementations of mixed-criticality automotive software". In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. European Design and Automation Association. Dresden, Germany, pp. 1227–1232.

Goswami, D., R. Schneider, and S. Chakraborty (2011). "Co-design of cyber-physical systems via controllers with flexible delay constraints". In: *Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. Yokohama, Japan, pp. 225–230.

Grant, M. and S. Boyd (2014). *CVX: matlab software for disciplined convex programming, version 2.1*. `http://cvxr.com/cvx`.

Han, C.-C. and H.-Y. Tyan (1997). "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms". In: *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*. San Francisco, CA, USA, pp. 36–45.

Henriksson, D. and A. Cervin (2005). "Optimal on-line sampling period assignment for real-time control tasks based on plant state information". In: *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*. Seville, Spain, pp. 4469–4474.

Jones, D. R., C. D. Perttunen, and B. E. Stuckman (1993). "Lipschitzian optimization without the Lipschitz constant". *Journal of optimization Theory and Applications* **79**:1, pp. 157–181.

Joseph, M. and P. Pandya (1986). "Finding response times in a real-time system". *The Computer Journal* **29**:5, pp. 390–395.

Kao, C.-Y. and B. Lincoln (2004). "Simple stability criteria for systems with time-varying delays". *Automatica* **40**:8, pp. 1429–1434.

Kim, B. K. (1998). "Task scheduling with feedback latency for real-time control systems". In: *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. Hiroshima, Japan, pp. 37–41.

Lehoczky, J. P. (1990). "Fixed priority scheduling of periodic task sets with arbitrary deadlines". In: *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS)*. Lake Buena Vista, FL, USA, pp. 201–209.

Lehoczky, J. P., L. Sha, J. K. Strosnider, and H. Tokuda (1991). "Fixed priority scheduling theory for hard real-time systems". *Foundations of Real-Time Computing: Scheduling and Resource Management* **1**, pp. 1–30.

Lehoczky, J., L. Sha, and Y. Ding (1989). "The rate monotonic scheduling algorithm: exact characterization and average case behavior". In: *Proceedings of the 10th IEEE Real-Time Systems Symposium (RTSS)*. Santa Monica, CA, USA, pp. 166–171.

Lemmon, M., T. Chantem, X. Hu, and M. Zyskowski (2007). "On self-triggered full-information H-infinity controllers". *Hybrid Systems: computation and control*, pp. 371–384.

Levine, W. S. (1996). *The control handbook*. CRC press.

Lincoln, B. and B. Bernhardsson (2000). "Optimal control over networks with long random delays". In: *Proceedings of the 14th International Symposium on Mathematical Theory of Networks and Systems (MTNS)*. Perpignan, France.

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 (CDC)*. Las Vegas, NV, USA, pp. 1319–1324.

Liu, C. L. and J. W. Layland (1973). "Scheduling algorithms for multiprogramming in a hard-real-time environment". *Journal of the ACM (JACM)* **20**:1, pp. 46–61.

Mancuso, G. M., E. Bini, and G. Pannocchia (2014). "Optimal priority assignment to control tasks". *ACM Transactions on Embedded Computing Systems (TECS)* **13**:5s, p. 161.

Martí, P., C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes (2004). "Optimal state feedback based resource allocation for resource-constrained control tasks". In: *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS)*. Lisbon, Portugal, pp. 161–172.

Martí, P., C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes (2009). "Draco: efficient resource management for resource-constrained control tasks". *IEEE Transactions on Computers* **58**:1, pp. 90–105.

Mohaqeqi, M., M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén (2016). "On the problem of finding optimal harmonic periods". In: *Proceedings of the 24th Conference on Real-Time and Network Systems (RTNS)*. Brest, France, pp. 171–180.

Mohaqeqi, M., M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén (2017). "Optimal harmonic period assignment: complexity results and approximation algorithms". In submission to *Real-Time Systems*.

Muradore, R. and G. Picci (2005). "Mixed $H_2/H_\infty$ control: the discrete-time case". *Systems & control letters* **54**:1, pp. 1–13.

Nasri, M. and G. Fohler (2015). "An efficient method for assigning harmonic periods to hard real-time tasks with period ranges". In: *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*. Lund, Sweden, pp. 149–159.

Nasri, M., G. Fohler, and M. Kargahi (2014). "A framework to construct customized harmonic periods for real-time systems". In: *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*. Madrid, Spain, pp. 211–220.

Nilsson, J. (1998). *Real-time control systems with delays*. PhD thesis. TFRT–1049–SE, Department Automatic Control, Lund University, Lund, Sweden.

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1998). "Stochastic analysis and control of real-time systems with random time delays". *Automatica* **34**:1, pp. 57–64.

Palopoli, L., L. Abeni, G. Buttazzo, F. Conticelli, and M. Di Natale (2000). "Real-time control system analysis: an integrated approach". In: *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS)*. Orlando, FL, USA, pp. 131–140.

Palopoli, L., C. Pinello, A. S. Vincentelli, L. Elghaoui, and A. Bicchi (2002). "Synthesis of robust control systems under resource constraints". In: *Proceedings of the 5th Workshop on Hybrid Systems: Computation and Control (HSCC)*. Springer. Stanford, CA, USA, pp. 337–350.

Papadimitriou, C. H. (1981). "On the complexity of integer programming". *Journal of the ACM (JACM)* **28**:4, pp. 765–768.

Ramanathan, P. (1997). "Graceful degradation in real-time control applications using $(m, k)$-firm guarantee". In: *Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS)*. IEEE. Seattle, WA, USA, pp. 132–141.

Redell, O. and M. Sanfridson (2002). "Exact best-case response time analysis of fixed priority scheduled tasks". In: *Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS)*. Vienna, Austria, pp. 165–172.

Redell, O. and M. Torngren (2002). "Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter". In: *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. San Jose, CA, USA, pp. 164–172.

Ripoll, I. and R. Ballester-Ripoll (2013). "Period selection for minimal hyperperiod in periodic task systems". *IEEE Transactions on Computers* **62**:9, pp. 1813–1822.

Saifullah, A., C. Wu, P. B. Tiwari, Y. Xu, Y. Fu, C. Lu, and Y. Chen (2014). "Near optimal rate selection for wireless control systems". *ACM Transactions on Embedded Computing Systems (TECS)* **13**:4s, p. 128.

Samii, S., A. Cervin, P. Eles, and Z. Peng (2009). "Integrated scheduling and synthesis of control applications on distributed embedded systems". In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. European Design and Automation Association. Nice, France, pp. 57–62.

Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin (1996). "On task schedulability in real-time control systems". In: *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS)*. Washington, DC, USA, pp. 13–21.

Tabuada, P. (2007). "Event-triggered real-time scheduling of stabilizing control tasks". *IEEE Transactions on Automatic Control* **52**:9, pp. 1680–1685.

Tanasa, B., U. D. Bordoloi, P. Eles, and Z. Peng (2015). "Probabilistic response time and joint analysis of periodic tasks". In: *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*. Lund, Sweden, pp. 235–246.

Vallender, S. (1974). "Calculation of the Wasserstein distance between probability distributions on the line". *Theory of Probability & Its Applications* **18**:4, pp. 784–786.

Velasco, M., J. Fuertes, and P. Martí (2003). "The self triggered task model for real-time control systems". In: *Work-in-Progress Session of the 24th IEEE Real-Time Systems Symposium (RTSS)*. Vol. 384. Cancun, Mexico.

Wittenmark, B., J. Nilsson, and M. Torngren (1995). "Timing problems in real-time control systems". In: *Proceedings of the American Control Conference (ACC)*. IEEE. Seattle, WA, USA, pp. 2000–2004.

Xu, Y., K.-E. Årzén, E. Bini, and A. Cervin (2014). "Response time driven design of control systems". In: *Proceedings of the 19th IFAC World Congress*. Cape Town, South Africa, pp. 6098–6104.

Xu, Y., K.-E. Årzén, E. Bini, and A. Cervin (2017a). "LQG-based control and scheduling co-design". In: *Proceedings of the 20th IFAC World Congress*. Toulouse, France.

Xu, Y., K.-E. Årzén, A. Cervin, E. Bini, and B. Tanasa (2015). "Exploiting job response-time information in the co-design of real-time control systems". In: *Proceedings of the 21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Hong Kong, China, pp. 247–256.

Xu, Y., A. Cervin, and K.-E. Årzén (2016a). "Harmonic scheduling and control co-design". In: *Proceedings of the 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Daegu, South Korea, pp. 182–187.

Xu, Y., A. Cervin, and K.-E. Årzén (2016b). "LQG-based scheduling and control co-design using harmonic task periods". Technical Report TFRT-7646. Department of Automatic Control, Lund University.

Xu, Y., A. Cervin, and K.-E. Årzén (2017b). "Jitter-robust LQG control and real-time scheduling co-design". In submission to the 2018 American Control Conference.

Zhang, F., K. Szwaykowska, W. Wolf, and V. Mooney (2008). "Task scheduling for control oriented requirements for cyber-physical systems". In: *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS)*. Barcelona, Spain, pp. 47–56.

Zhou, K., K. Glover, B. Bodenheimer, and J. Doyle (1994). "Mixed $H_2$ and $H_\infty$ performance objectives I: robust performance analysis". *IEEE Transactions on Automatic Control* **39**:8, pp. 1564–1574.

# A

# Stochastic LQG Design

We here derive the discrete-time stochastic LQG controller for a continuous-time plant with a random control delay $\delta_k$, used in the stochastic scheduling and control codesign in Chapter 4. It is assumed that the difference between the maximum delay, $\delta^{\max}$, and the minimum delay, $\delta^{\min}$, is smaller than or equal to the sampling interval $h$. (Without this assumption, further information would be needed about the order of arrival of the control signals to the plant, and the calculations would also be much more involved.) For ease of notation, we further assume that $\delta^{\min} \leq h$. Extra integer delays can easily be added to the sampled plant before the state feedback is calculated.

To handle the long control delays in the control design, we treat the constant part of the delay, $\delta^{\min}$, as a measurement delay, while the time-varying part of the delay, $\delta_k - \delta^{\min}$ is handled as an input delay. The setup is illustrated in Figure A.1. At time $kh$, the plant output $y[k]$, sampled at time $kh - \delta^{\min}$, arrives to the controller. The newly calculated control signal $u[k]$ is then applied somewhere in the interval $[kh, kh+h]$. As usual, the LQG controller design can be separated into optimal state feedback and Kalman filtering.

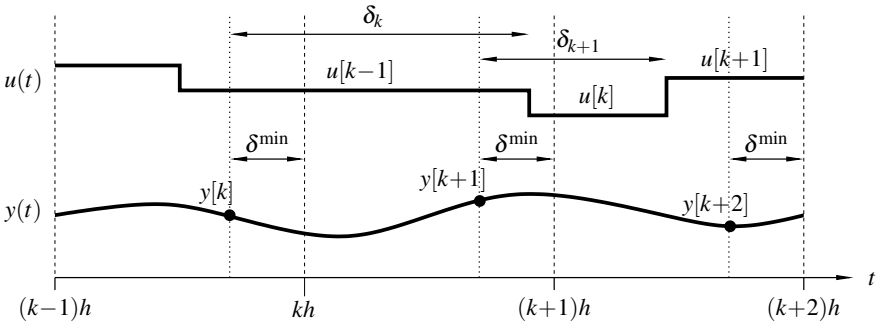



**Figure A.1** The long control delays are handled using a combination of constant measurement delays and a time-varying input delay.

## Optimal State Feedback

Introduce the notation $\bar{\delta}_k = \delta_k - \delta^{\min}$ for the time-varying part of the delay. The evolution of the state is then given by the discrete-time state-space model

$$x[k+1] = \Phi x[k] + \bar{\Gamma}_1(\bar{\delta}_k)u[k-1] + \bar{\Gamma}_0(\bar{\delta}_k)u[k] + v[k], \tag{A.1}$$

where

$$\begin{aligned} \Phi &= \Phi(h,0) \\ \bar{\Gamma}_1(\bar{\delta}_k) &= \Phi(h-\delta_k,0)\Gamma(\delta_k,0) \\ \bar{\Gamma}_0(\bar{\delta}_k) &= \Gamma(h-\delta_k,0), \end{aligned} \tag{A.2}$$

compare (2.10). After introducing the augmented state vector $x_e$ as in (2.11), the sampled cost in the time interval $[kh, kh+h]$ given by

$$V[k] = x_e^T[k]Q_{1e}(\bar{\delta}_k)x_e[k] + 2x_e^T[k]Q_{12e}(\bar{\delta}_k)u[k] + u^T[k]Q_{2e}(\bar{\delta}_k)u[k] + V_{const}, \tag{A.3}$$

where $Q_{1e}(\bar{\delta}_k)$, $Q_{12e}^T(\bar{\delta}_k)$, and $Q_{2e}^T(\bar{\delta}_k)$ are given by the time-varying counterparts of the cost matrices (2.16). To solve for the optimal state feedback gain, let

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{12}^T & X_{22} \end{bmatrix} \tag{A.4}$$

be a symmetric matrix of appropriate size. The optimal static feedback $L$ is then given by the stationary solution to the stochastic Riccati equation

$$X = \mathop{\mathrm{E}}_{\bar{\delta}_k} \left\{ \begin{bmatrix} \Phi_e^T(\bar{\delta}_k) \\ \Gamma_e^T(\bar{\delta}_k) \end{bmatrix} S \begin{bmatrix} \Phi_e^T(\bar{\delta}_k) \\ \Gamma_e^T(\bar{\delta}_k) \end{bmatrix}^T + \begin{bmatrix} Q_{1e}(\bar{\delta}_k) & Q_{12e}(\bar{\delta}_k) \\ Q_{12e}^T(\bar{\delta}_k) & Q_{2e}(\bar{\delta}_k) \end{bmatrix} \right\} \tag{A.5}$$

where

$$\begin{aligned} S &= X_{11} - L^T X_{22} L \\ L &= X_{22}^{-1} X_{12}^T. \end{aligned} \tag{A.6}$$

In MATLAB, this can be solved by using iteration.

## Kalman Filtering

The Kalman filter is based on an extended model that includes the measurement delayed by $\delta^{\min}$ time units. The delayed measurement to be used in the next step is captured by an extra state variable $y_0$, yielding the extended state-space model

$$\begin{aligned} x[k+1] &= \Phi x[k] + v_1[k] \\ y_0[k+1] &= C\Phi(h-\delta^{\min})x[k] + v_y[k] \\ y[k] &= y_0[k] + v_2[k]. \end{aligned} \tag{A.7}$$

The extended process noise vector $(v_1, v_y)$ has covariance

$$\mathrm{E}\begin{bmatrix} v_1[k] \\ v_y[k] \end{bmatrix}\begin{bmatrix} v_1[k] \\ v_y[k] \end{bmatrix}^T = \begin{bmatrix} R_1 & R_{12y} \\ R_{12y}^T & R_{2y} \end{bmatrix} \tag{A.8}$$

where

$$\begin{aligned} R_{12y} &= \Phi(\delta^{\min})R_1(h-\delta^{\min})C^T \\ R_{2y} &= CR_1(h-\delta^{\min})C^T. \end{aligned} \tag{A.9}$$

Solving the associated algebraic Riccati equation yields the Kalman filter gain vector $[K^T \; K_y^T]^T$, and the Kalman filter becomes

$$\begin{aligned} \hat{x}[k+1] &= \Phi\hat{x}[k] + \hat{\Gamma}_1 u[k-1] + \hat{\Gamma}_0 u[k] + K(y[k] - \hat{y}_0[k]) \\ \hat{y}_0[k+1] &= C\Phi(h-\delta^{\min})\hat{x}[k] + \hat{\Gamma}_{1y} u[k-1] + \hat{\Gamma}_{0y} u[k] + K_y(y[k] - \hat{y}_0[k]), \end{aligned} \tag{A.10}$$

where $\hat{\Gamma}_1$ and $\hat{\Gamma}_0$ represent the average (expected value) $\Gamma_1$ and $\Gamma_0$ matrices over a sampling period, while $\hat{\Gamma}_{1y}$ and $\hat{\Gamma}_{0y}$ are the corresponding matrices but only up to time $h-\delta^{\min}$ in the period, where the new measurement sample will be taken.

## Optimal Output Feedback

Since there is no sampling jitter, the time-invariant Kalman filter will be optimal. It is combined with the static state feedback gain from the stochastic Riccati equation to form an optimal time-invariant LQG controller.

# B

# Periodic LQG Design

To derive the periodic LQG control for deterministic job response times presented in Chapter 5, we describe here the procedure that leads us to the result.

## Sampling the Periodic Time Delay System

For convenience, all the subscripts indicating the $i$th task are omitted. Integration of (2.1) over one hyperperiod is given as

$$\begin{aligned}
x((k+1)lh) &= e^{Alh}x(klh) + \int_{klh}^{klh+\delta_1} e^{A((k+1)lh-s)}dsBu((kl-1)h) \\
&\quad + \sum_{j=1}^{l-1} \int_{(kl+j-1)h+\delta_j}^{(kl+j)h+\delta_{j+1}} e^{A((k+1)lh-s)}dsBu((kl+j-1)h) \\
&\quad + \int_{((k+1)l-1)h+\delta_l}^{(k+1)lh} e^{A((k+1)lh-s)}dsBu(((k+1)l-1)h) \\
&= \Phi x(klh) + \Gamma_0 \begin{bmatrix} u(klh) \\ u((kl+1)h) \\ \vdots \\ u(((k+1)l+1)h) \end{bmatrix} + \Gamma_1 \begin{bmatrix} u((k-1)lh) \\ u(((k-1)l+1)h) \\ \vdots \\ u((kl-1)h) \end{bmatrix} \\
&= \Phi x(klh) + \Gamma_0 u'(klh) + \Gamma_1 u'((k-1)lh)
\end{aligned} \tag{B.1}$$

where

$$\Phi = e^{Alh} \tag{B.2}$$

$$\Gamma_0 = \begin{bmatrix} \int_{klh+\delta_1}^{(kl+1)h+\delta_2} e^{A((k+1)lh-s)}dsB \\ \int_{klh+\delta_2}^{(kl+1)h+\delta_3} e^{A((k+1)lh-s)}dsB \\ \vdots \\ \int_{((k+1)l-2)h+\delta_{l-1}}^{((k+1)l-1)h+\delta_l} e^{A((k+1)lh-s)}dsB \end{bmatrix}^T \tag{B.3}$$

$$\Gamma_1 = \begin{bmatrix} 0 & 0 & \cdots & 0 & \int_{klh}^{klh+\delta_1} e^{A((k+1)lh-s)}dsB \end{bmatrix}.$$

Here, the vector $u'(klh)$ contains all the control signals from time $klh$ to time $(k+1)lh$, and the length of it is $l$.

## Sampling the Cost Function

In order to calculate $Q_1$, $Q_{12}$ and $Q_2$, the cost matrices of the cost function are sampled using zero-order hold. For (2.1), when $v_{ci}$ are zero, and from time $b$ to time $a$ control $u_i(t)$ is constant, the discrete time cost matrices are

$$\begin{aligned} Q_1^0(a,b) &= \int_b^a \Phi^T(s,b)Q_{1c}\Phi(s,b)ds \\ Q_{12}^0(a,b) &= \int_b^a \Phi^T(s,b)\left[Q_{1c}\Gamma(s,b)+Q_{12c}\right]ds \\ Q_2^0(a,b) &= \int_b^a \left[\Gamma^T(s,b)Q_{1c}\Gamma(s,b)+2\Gamma^T(s,b)Q_{12c}+Q_{2c}\right]ds. \end{aligned} \tag{B.4}$$

Due to the periodicity, the cost matrices calculation is only performed from time 0 to time $lh$, which is one hyperperiod. Then

$$Q_1 = Q_1^0(lh,0). \tag{B.5}$$

We further have

$$Q_2 = \begin{bmatrix} Q_2^{(1,1)} & Q_2^{(1,2)} & \cdots & Q_2^{(1,l)} \\ Q_2^{(1,2)T} & Q_2^{(2,2)} & \cdots & Q_2^{(2,l)} \\ \vdots & \vdots & \ddots & \vdots \\ Q_2^{(1,l)T} & Q_2^{(2,l)T} & \cdots & Q_2^{(l,l)} \end{bmatrix} \tag{B.6}$$

where

$$\begin{aligned} Q_2^{(1,1)} = Q_2^0(h+\delta_2,\delta_1) + \sum_{i=1}^{l-2} \Big\{ & \Gamma^T(h+\delta_2,\delta_1)\,\Phi^T(ih+\delta_{i+1},h+\delta_2) \\ & \times Q_1^0[(i+1)h+\delta_{i+2},ih+\delta_{i+1}]\,\Phi(ih+\delta_{i+1},h+\delta_2)\,\Gamma(h+\delta_2,\delta_1)\Big\} \\ & + \Gamma^T(h+\delta_2,\delta_1)\,\Phi^T[(l-1)h+\delta_l,h+\delta_2]\,Q_1^0[lh,(l-1)h+\delta_l] \\ & \times \Phi[(l-1)h+\delta_l,h+\delta_2]\,\Gamma(h+\delta_2,\delta_1) \end{aligned} \tag{B.7}$$

$$\begin{aligned} Q_2^{(1,2)} = \Gamma^T(h+\delta_2,\delta_1) \Bigg\{ & \sum_{i=1}^{l-1} \Gamma^T(ih+\delta_{i+1},h+\delta_2)\,Q_1^0[(i+1)h+\delta_{i+2},ih+\delta_{i+1}] \\ & \times \Gamma(ih+\delta_{i+1},h+\delta_2) + \Gamma^T[(l-1)h+\delta_l,h+\delta_2]\,Q_1^0[lh,(l-1)h+\delta_l] \\ & \times \Gamma[(l-1)h+\delta_l,h+\delta_2] \Bigg\} \end{aligned} \tag{B.8}$$

$$Q_2^{(1,l)} = \Gamma^T(h+\delta_2,\delta_1)\,\Phi^T((l-1)h+\delta_l,h+\delta_2)\,Q_{12}^0(lh,(l-1)h+\delta_l) \tag{B.9}$$

$$\begin{aligned} Q_2^{(2,2)} = Q_2^0(2h+\delta_3,h+\delta_2) + \Gamma^T(2h+\delta_3,h+\delta_2) \Bigg\{ & \sum_{i=2}^{l-2} \Phi^T(ih+\delta_{i+1},2h+\delta_3) \\ & \times Q_1^0((i+1)h+\delta_{i+2},ih+\delta_{i+1})\,\Phi(ih+\delta_{i+1},2h+\delta_3) \\ & + \Phi^T((l-1)h+\delta_l,2h+\delta_3)\,Q_1^0(lh,(l-1)h+\delta_l) \\ & \times \Phi((l-1)h+\delta_l,2h+\delta_3) \Bigg\} \Gamma(2h+\delta_3,h+\delta_2) \end{aligned} \tag{B.10}$$

$$Q_2^{(2,l)} = \Gamma^T(2h+\delta_3,h+\delta_2)\,\Phi^T((l-1)h+\delta_l,2h+\delta_3)\,Q_{12}^0(lh,(l-1)h+\delta_l) \tag{B.11}$$

$$Q_2^{(l,l)} = Q_2^0(lh,(l-1)h+\delta_l) \tag{B.12}$$

and

$$Q_{12} = \begin{bmatrix} Q_{12}^1 \\ Q_{12}^2 \\ \vdots \\ Q_{12}^l \end{bmatrix} \tag{B.13}$$

where

$$Q_{12}^{1} = \begin{bmatrix}\Phi(\delta_1,0) & \Gamma(\delta_1,0)\end{bmatrix}^T \Bigg\{ Q_{12}^{0}(h+\delta_2,\delta_1) + \Bigg[\sum_{i=2}^{l-1} \Phi^T((i-1)h+\delta_i,\delta_1) \times Q_1^0(ih+\delta_{i+1},(i-1)h+\delta_i)\,\Phi((i-1)h+\delta_i,h+\delta_2) + \Phi^T((l-1)h+\delta_l,\delta_1) \times Q_1^0(lh,(l-1)h+\delta_l)\Phi((l-1)h+\delta_l,h+\delta_2)\Bigg]\Gamma(h+\delta_2,\delta_1)\Bigg\} \tag{B.14}$$

$$Q_{12}^{2} = \begin{bmatrix}\Phi(\delta_1,0) & \Gamma(\delta_1,0)\end{bmatrix}^T \Phi^T(h+\delta_2,\delta_1)\Bigg\{ Q_{12}^{0}(2h+\delta_3,h+\delta_2) + \Bigg[\sum_{i=3}^{l-1} \Phi^T((i-1)h+\delta_i,h+\delta_2)\,Q_1^0(ih+\delta_{i+1},(i-1)h+\delta_i) \times \Phi((i-1)h+\delta_i,2h+\delta_3) + \Phi^T((l-1)h+\delta_l,h+\delta_2)\,Q_1^0(lh,(l-1)h+\delta_l) \times \Phi((l-1)h+\delta_l,2h+\delta_3)\Bigg]\Gamma(2h+\delta_3,h+\delta_2)\Bigg\} \tag{B.15}$$

$$Q_{12}^{l} = \big(\Phi(\delta_1,0) \quad \Gamma(\delta_1,0)\big)^T \Phi^T[(l-1)h+\delta_l,\delta_1]\,Q_{12}^{0}[lh,(l-1)h+\delta_l]. \tag{B.16}$$

The matrix $Q_1$ is positive semidefinite, and $Q_2$ is positive definite, which will be relaxed next. The cross term $Q_{12}$ is general not zero even if $Q_{12c} = 0$.

## Linear Quadratic Control Design

Using the extended state space matrices $\Phi$, $\Gamma$ from Section 5.3 and cost matrix $Q$ from Section 5.3, we solve the standard discrete time algebraic Riccati equation

$$Q_1 + \Phi_e^T P \Phi_e - \left(\Phi_e^T P \Gamma_e + Q_{12}\right)\left(\Gamma_e^T P \Gamma_e + Q_2\right)^{-1}\left(\Gamma_e^T P \Phi_e + Q_{12}^T\right) - P = 0 \tag{B.17}$$

where

$$\Phi_e = \begin{bmatrix}\Phi & \Gamma_1 \\ 0 & 0\end{bmatrix}, \quad \Gamma_e = \begin{bmatrix}\Gamma_0 \\ I\end{bmatrix}. \tag{B.18}$$

The linear quadratic state feedback vector is given by

$$L = \left(Q_2 + \Gamma_e^T P \Gamma_e\right)^{-1}\left(2Q_{12}^T + \Gamma_e^T P \Phi_e\right). \tag{B.19}$$

$Q_2 + \Gamma_e^T P \Gamma_e$ needs to be positive definite. It means that the requirement in Section 5.3 about $Q_2$ is relaxed. The control signals over the hyperperiod are obtained

as

$$u'(klh) = \begin{bmatrix} u(klh) \\ u((kl+1)h) \\ \vdots \\ u((k+1)lh) \end{bmatrix} = -Lx(klh) = -\begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_l \end{bmatrix} x(klh). \tag{B.20}$$

The state feedback vector is based on the extended state $x(klh)$, which means the sampling happens every hyperperiod. But in fact, it should happen every period. So the state feedback vector must be reformulated according to

$$\begin{bmatrix} u(klh) \\ u((kl+1)h) \\ \vdots \\ u((k+1)lh) \end{bmatrix} = -\begin{bmatrix} L_1'x(klh) \\ L_2'x((kl+1)h) \\ \vdots \\ L_l'x((k+1)lh) \end{bmatrix}. \tag{B.21}$$

For any $i \in \{1, 2, \ldots, l\}$, the extended state at time $(kl+i)h$ is

$$\begin{aligned} x((kl+i)h) &= \left(\Phi_{e,i-1} - \Gamma_{e,i-1}L_{i-1}'\right) x((kl+i-1)h) \\ &= \prod_{j=1}^{i-1} \left(\Phi_{e,i-j} - \Gamma_{e,i-j}L_{i-j}'\right) x(klh) \end{aligned} \tag{B.22}$$

where $\Phi_{e,i}$ and $\Gamma_{e,i}$ are the $i$th extended state space matrices in a hyperperiod. So

$$\begin{aligned} L_i x(klh) &= L_i' x((kl+i-1)h) \\ &= L_i' \prod_{j=2}^{i-1} \left(\Phi_{e,i-j} - \Gamma_{e,i-j}L_{i-j}'\right) x(klh). \end{aligned} \tag{B.23}$$

The reformulated state feedback vectors are finally calculated recursively by

$$L_i' = \begin{cases} L_1, & \text{if } i = 1 \\ L_i \prod_{j=1}^{i-2} \left(\Phi_{e,j} - \Gamma_{e,j}L_j'\right)^{-1}, & \text{if } i > 1. \end{cases} \tag{B.24}$$