UNIVERSITAT POLITÈCNICA DE CATALUNYA

Deptartament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial

Programa de doctorat:

Enginyeria en Informàtica Industrial / Tecnologies Avançades de la Producció

Tesi doctoral

Analysis and Design of Real-Time Control Systems with Varying Control Timing Constraints

Pau Martí Colom

Directors:

Dr. Josep M. Fuertes
Dept. d'Enginyeria de Sistemes,
Automàtica i Informàtica Industrial
Universitat Politècnica de Catalunya

Dr. Gerhard Fohler
Dept. of Computer Engineering
Mälardalen University

Barcelona, juny de 2002

A la Belén

Agraïments

No hauria estat possible realitzar aquesta tesi sense l'ajuda de moltes persones. A totes elles, moltíssimes gràcies!

It would not have been possible to produce this thesis without the help of many people. To all of them, thanks a lot!

Primerament, vull donar les gràcies als dos directors, el Dr. Josep Mª Fuertes (del departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial de la Universitat Politècnica de Catalunya) i el Dr. Gerhard Fohler (del Computer Engineering Department de la Mälardalen University, Sweden) per haver-me dirigit, tant a nivell científic com humà, el procés d'aprenentatge en la realització d'aquesta tesi. Moltes gràcies!

En el camí que m'ha conduït fins aquí, són moltes les persones que m'he anat trobant. Cada una d'elles forma part d'aquesta tesi.

En un ordre cronològic i sense pretendre ser exhaustiu:

Gràcies Pep per confiar en mi i ajudar-me en aquest món del control, tan complex per a mi.

Thanks Gerhard for making me feel so welcome and guiding me throughout my real-time systems research.

Thanks a lot to everybody living up there, in Sweden, for making me feel at home. Thanks Sashi, thanks Radu and Camilla, and thanks Thomas and Damir.

Els meus agraïments també estan dirigits al Dr. Ricard Villà: moltes gràcies per la teva agudesa i les teves valuoses i imprescindibles aportacions.

I also want to spend a few words to thank Prof. Krithi Ramamritham. Thanks Krithi for all the stimulating discussions we had, and for agreeing to collaborate in the work done in this thesis. Thanks for your contributions.

Thanks also to Anton Cervin. Without your help, this thesis wouldn't be like it is.

Gràcies també a totes les persones que ara no cito i sense les quals aquest treball no hauria estat una realitat. Especialment, gràcies a la gent del departament, que d'una manera o altra, sempre m'han ajudat.

No m'oblido dels meus amics i amigues: gràcies per estar sempre al meu costat!

Pels ànims i suport que sempre m'heu donat, gràcies família!

I finalment, gràcies a tu Belén, per tot!

El treball realitzat en aquesta tesi ha estat parcialment financiat per:

- Projecte *Metodologias de Diseño y Evaluación de Prestaciones en los Sistemas Distribuidos de Control*. Ref. CICYT TAP98-0585-C03-01
- Projecte *Arquitecturas de Control Distribuido. Análisis y Diseño de Estructuras con Soporte de Tecnologias Internet-Intranet*. Ref. CICYT DPI2000-1760-C03-01
- Network of Excellence. *Advanced Real Time Systems (ARTIST)*. Information Society Technologies. Ref. IST-2001-34820
- Beques per a la recerca a fora de Catalunya (Generalitat de Catalunya), convocatòries 2000 i 2001
- Pla de mobilitat externa del professorat UPC (Universitat Politècnica de Catalunya), convocatòries 1999, 2000, 2001 i 2002

Aquesta tesi s'ha realitzat durant la meva contractació com a professor associat en el Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial de la Universitat Politècnica de Catalunya. Tanmateix, vull destacar la importància que en la realització d'aquesta tesi han tingut les meves estades de recerca (Juliol de 1999, Juny de 2000, Gener i Juliol de 2001, Gener de 2002) al Real-Time Systems Lab (SDL) del Mälardalen Real-Time Research Centre (MRTC), Department of Computer Engineering, Mälardalen University, Sweden.

Analysis and Design of Real-Time Control Systems with Varying Control Timing Constraints

Summary

The analysis and design of real-time control systems is a complex task, requiring the integration and good understanding of both control and real-time systems theory. Traditionally, such systems are designed by differentiating two separate stages: first, control design and then its computer implementation, leading to sub-optimal solutions in terms of both system schedulability and controlled systems performance.

Traditional discrete-time control models and methods consider implementation constraints only to a very small extent. This is due to the fact that in the control design stage, controllers are assumed to execute in dedicated processors and processors are assumed to be fast and deterministic enough not to worry about the timing that the controlling activities may have on the implementation. However, when resources (e.g., processors) are limited, timing variations in the execution of control algorithms occur. Specifically, a control algorithm in traditional real-time scheduling is implemented as a periodic task characterized by standard timing constraints such as period and deadline. In real-time scheduling, timing variations in task instance executions (i.e., jitters) are allowed as far as the schedulability constraints are preserved. Consequently, the resulting jitters for control task instances do not comply with the strict timing demanded by discrete-time control theory.

This has two pervasive effects: the presence of jitters for control tasks degrades the controlled system performance, even causing instability. On the other hand, minimizing the likelihood of jitters for control tasks by over-constraining the control task specification reduces the schedulability of the entire task set.

It is worth mentioning that control theory offers no advice on how to include, into the design of controllers, the effects that implementation constraints have in the timing of the control activities (e.g., scheduling inherent jitters). Also, real-time theory lacks task models and timing constraints that can be used to guarantee a periodic task execution free of jitters without over-constraining system schedulability.

In this thesis we present a *flexible integrated scheduling and control* analysis and design framework for real-time control systems that solves the problems outlined above: poor system schedulability and controlled systems performance degradation. We show that by merging the activities of the control and real-time communities, that is, by integrating control design with computer implementation, both system schedulability and controlled systems performance are improved.

We present a new approach to discrete-time controller design that takes implementation constraints into account and relaxes the equidistant sampling and actuation assumptions of traditionally designed discrete-time controllers. Instead of specifying a single value for the sampling period and a single value for the time delay at the design stage, we specify a set of

values for both the sampling period and for the time delay. This new approach for the controller design relies on the idea of adjusting controller parameters at run time according to the specific implementation timing behaviour, i.e., scheduling inherent jitters. The resulting closed-loop systems are based on irregularly sampled discrete-time system models with varying time delays. We have used state space formulation to present a complete stability and response analysis for such models.

We also show how to derive more flexible timing constraints for control tasks by exploiting the timing properties imposed by this new approach to discrete-time controller design. Real-time scheduling standard timing constraints for periodic tasks are constant for all task instances. That is, a single value of a constraint (e.g., period or deadline) holds for all task instances. Our flexible timing constraints for control tasks do not set specific values. Rather, they provide ranges and combinations to choose from (at each control task instance execution), taking into account, for example, schedulability of other tasks.

That is, these more flexible timing constraints for control tasks allow us to obtain feasible schedules and stable control systems from task sets (including control and non-control tasks) that are not feasible using traditional real-time scheduling and discrete-time control design methods. In addition, by associating control performance information with these new timing constraints for control tasks, we show how scheduling decisions, going beyond meeting timing constraints, can be taken to improve the performance of the controlled systems when they are affected by perturbations.

Anàlisi i Disseny de Sistemes de Control de Temps Real amb Restriccions Temporals Variables de Control

Resum

L'anàlisi i el disseny dels sistemes de control de temps real és una tasca complexa, que requereix la integració de dues disciplines, la dels sistemes de control i la dels sistemes de temps real. Tradicionalment però, els sistemes de control de temps real s'han dissenyat diferenciant, de forma independent, dues fases, primerament el disseny del controlador, i després, la seva implementació en un computador. Això ha desembocat en solucions no òptimes tant en termes de planificabilitat del sistema i com en el rendiment dels sistemes controlats.

Normalment, els mètodes i models de la teoria de control de temps discret no consideren durant la fase de disseny dels controladors les limitacions que es puguin derivar de la implementació. En la fase de disseny s'assumeix que els algorismes de control s'executaran en processadors dedicats i que els processadors seran prou ràpids i determinístics per no haver-se de preocupar del comportament temporal que aquests algorismes de control tindran en temps d'execució. Tot i així, quan els recursos - per exemple, processadors - són limitats, apareixen variacions temporals en l'execució dels algorismes de control. En concret, en els sistemes de planificació de tasques de temps real, un algorisme de control s'implementa en una tasca periòdica caracterizada per restriccions temporals estàndards com períodes i terminis. És sabut que, en la planificació de tasques de temps real, les variacions temporals en l'execució d'instàncies de tasques és permesa sempre i quan les restriccions de planificabilitat estiguin garantides. Aquesta variabilitat per tasques de control viola l'estricte comporament temporal que la teoria de control de temps discret pressuposa en l'execució dels algorismes de control.

Això té dos efectes negatius: la variabilitat temporal en l'execució de les tasques de control degrada el rendiment del sistema controlat, fins i tot causant inestabilitat. A més, si es minimitza la probabilitat d'aparició d'aquesta variabilitat en l'execució de les tasques de control a través d'especificacións més limitants, la planificabilitat del conjunt de tasques del sistema disminueix.

Cal tenir en compte que la teoria de control no dóna directrius de com incloure, en la fase de disseny dels controladors, aquesta variabilitat en l'execució de tasques que es deriva de les limitacions d'implementació. A més, la teoria de sistemes de temps real no proporciona ni models de tasques ni restriccions temporals que puguin ser usats per garantir l'execució periòdica, i sense variabilitats temporals, de tasques sense sobrelimitar la planificabilitat dels sistema.

En aquesta tesi es presenta un entorn *integrat i flexible de planificació i de control* per a l'anàlisi i el disseny de sistemes de control de temps real que dóna solucions als problemes

esmentats anteriorment (baixa planificabilitat en el sistema i degradació del rendiment dels sistemes controlats). Mostrem que, fusionant les activitats de la comunitat de temps real amb les de la comunitat de control, això és, integrant la fase de disseny de controladors amb la fase d'implementació en un computador, es millora tant la planificabilitat del sistema com el rendiment dels sistemes controlats.

També es presenta una nova aproximació al disseny de controladors de temps discret que té en compte les limitacions derivables de la implementació i relaxa les tradicionals assumpcions dels controladors de temps discret (mostreig i actuació equidistants). En lloc d'especificar, en la fase de disseny, únics valors pel període de mostreig i pel retard temporal, especifiquem un conjunt de valors tant per l'un com per l'altre. Aquesta nova aproximació al disseny de controladors es basa en la idea d'ajustar, en temps d'execució, els paràmetres del controlador d'acord amb el comportament temporal específic de la implementació (per exemple, d'acord amb la variabilitat en l'execució de les tasques deguda a la planificació). Els llaços de control resultants esdevenent sistemes variants en el temps, amb mostreig irregular i retards temporals variables. Per a aquests sistemes, i utilitzant formulació en l'espai d'estat, presentem una anàlisi completa d'estabilitat, així com l'anàlisi de la resposta.

També mostrem com, a partir de les propietats temporals d'aquesta nova aproximació al disseny de controladors, podem obtenir restriccions temporals més flexibles per a les tasques de control. Les restriccions temporals estàndards, per a les tasques periòdiques en els sistemes de temps real, són constants per a totes les instàncies d'una tasca. Això és, només un sol valor per a una restricció és aplicable a totes les instàncies. Les noves restriccions temporals que presentem per a tasques de control no forcen a aplicar un valor específic, sinó que permeten aplicar valors diferents a cada instància d'una tasca, tenint en compte, per exemple, la planificabilitat d'altres tasques.

Aquestes restriccions temporals flexibles per a tasques de control ens permeten obtenir planificacions viables i sistemes de control estables a partir de conjunts de tasques (incloent tasques de control i d'altres) que no eren planificables en usar mètodes estàndards tant de planificació de temps real com de disseny de controladors. A més, associant informació de rendiment de control a aquestes noves restriccions temporals per a tasques de control, mostrem com podem prendre decisions de planificació que, anant més enllà de complir amb les restriccions temporals, milloren el rendiment dels sistemes controlats quan aquests sofreixen perturbacions.

Contents

List of Figures

List of Tables

List of symbols

Symbol	Description
$task_i$	Periodic task *i*, where *i* identifies the task
$task_{i,k}$	k^{th} instance (or invocation) of a periodic task $task_i$
D_i	Task $task_i$ relative deadline
T_i	Task $task_i$ period
C_i	Task $task_i$ worst case execution time
c_i	Task $task_i$ exact execution time
P_i	Task $task_i$ priority
O_i	Task $task_i$ offset
U	Processor utilization
$s(task_{i,k})$	Start time of the k^{th} instance of task $task_i$
$f(task_{i,k})$	Finishing time ot the k^{th} instance of task $task_i$
$r(task_{i,k})$	Release time of the k^{th} instance of task $task_i$
$h(task_{i,k})$	Sampling interval of the k^{th} instance of the control task $task_i$ due to sampling jitter
$\tau(task_{i,k})$	Sampling-actuation delay of the k^{th} instance of the control task $task_i$ due to sampling-actuation jitter
g	Basic time granularity length (in the system model)
y(t)	Output of a controlled process (system response)
u(t)	Input of a controlled process (control signal)
r(t)	Reference signal
e(t)	Closed-loop system error
x(t)	State vector
t_k	Sampling instant
a_k	Actuation instant
h	Sampling period
τ	Time delay
FH_i	Set of feasible sampling intervals for a given control task $task_i$
FT_i	Set of feasible sampling-actuation delays for a given control task $task_i$
$h_{i,j}$	Feasible sampling interval belonging to FH_i, where *i* identifies the control task $task_i$ and *j* identifies the specific value
$\tau_{i,j}$	Feasible sampling-actuation delay belonging to FT_i, where *i* identifies the control task $task_i$ and *j* identifies the specific value
A	Continuous-time system matrix
B	Continuous-time input matrix
C	Continuous-time output matrix
D	Continuous-time direct matrix
Φ	Discrete-time system matrix
Γ	Discrete-time input matrix
L	Discrete-time gain matrix
K	Discrete-time observer matrix
ω_n	Natural frequency
ζ	Damping ratio

List of acronyms

Acronym	Description
AD	Analogue-to-Digital
DA	Digital-to-Analogue
DC	Direct Current
EDF	Earliest Deadline First
FPS	Fixed Priority Scheduling
GCD	Greatest Common Divisor
LCM	Least Common Multiple
PID	Proportional, Integral and Derivative
QoC	Quality-of-Control
RM	Rate Monotonic
SFC	State Feedback Controller
WCET	Worst-Case Execution Time

Chapter 1

Introduction

The objective of *computer control* is to use computers to manipulate the available inputs of a dynamic system in order to cause the system to behave in a manner more desirable than it otherwise would [LUE79]. Computer control is used in many application areas, such as factory automation, process control, robotics, automotive systems, and others. In such applications, computers are used to control processes, and are expected to react within precise time constraints to external events according to the application requirements. Computer control systems[1] are expected to behave correctly both in the value and timing domains: they process inputs and provide the adequate outputs with an accurate timing.

Computing systems in which meeting *timing constraints* is essential to correctness, i.e., their correctness depends not only on the logical results of the computations but also on the time at which these results are produced [STA88], are called *real-time systems*. Therefore, computer-controlled systems must be considered real-time systems. In real-time systems, by means of algorithms, task *scheduling* deals with the problem of meeting the timing constraints of the tasks. These timing constraints are derived from the application timing requirements. During the last three decades, real-time scheduling has been a very active research area and many different scheduling models and methods have been presented. Scheduling approaches are based on standard task timing constraints such as *periods* and *deadlines* [BUT97].

Computer-controlled systems theory assumes a highly deterministic timing of an implementation [AST97]. The classic mathematical models for computer-controlled systems transform a *continuous-time system* into a *discrete-time system* by considering the behaviour of the signals (measured and control signals) at the *sampling instants* only. That is, continuous-time signals are replaced by sequences of numbers, which represent the values of the signals at certain synchronised times. As a consequence, computers for control applications are expected to behave as the mathematical models demand. For that reason, the most stringent timing constraints for real-time systems have their origin in the timing requirements imposed by discrete-time control theory.

The application of standard timing constraints for control tasks impairs system schedulability because those timing constraints, which are artificial constraints rather than constraints able to comply with the control timing requirements, over-constrain the schedule [FOH97]. On the other hand, real-time scheduling introduces variability in the starting and completion times of successive instances of a same task, i.e., *jitters* [BAR97]. The jitters inherent to scheduling for control tasks prevent controllers from fulfilling the control

[1] Also known as computer-controlled systems, sampled data systems or discrete-time systems [AST97].

performance requirements, thus degrading the controlled system response. This degradation appears because these jitters in the control tasks violate the strict timing assumptions that discrete-time control theory imposes.

In this thesis we present a flexible *integrated scheduling and control* analysis and design framework for real-time control systems that improves both system schedulability and controlled systems performance. We present a new *controller design method* for control tasks that takes scheduling inherent jitters into account, thus removing the control performance degradation that would otherwise occur. In addition, by exploiting the timing properties imposed by this new controller design method, we show how to derive more *flexible timing constraints* for control tasks. These allow us to obtain feasible schedules from task sets (including control and non-control tasks) that are not schedulable using traditional scheduling and control design methods, thus improving system schedulability. Finally, we formulate a novel scheduling problem, *Quality-of-Control scheduling*, in which improving both system schedulability and controlled systems performance is of main concern. Specifically, we show that the control performance information that can be associated to each control task timing constraint can be used to improve the performance of the controlled processes in the presence of perturbations.

1.1 Motivation

The development of real-time control systems is a complex task, requiring the integration and good understanding of both control and real-time systems theory. However, control theory and real-time scheduling theory have been relatively independent research areas [TOR98]. This fact arises from the traditional way real-time control systems have been developed; that is, differentiating two separate stages, each in isolation [SET96]: first, control design and then its computer implementation.

This has allowed the control community to focus on its own problem domain without being really concerned about how the implementation is being done. The control community sees the computing platform as providing the determinism that discrete-time control theory requires. Control theory has considered implementation other than dedicated processors systems only to a very small extent. Consequently, when computing resources (processor time and communication bandwidth) are limited, control theory rarely advises on how to design controllers to take these limitations into account [ARZ00].

On the other hand, this has released the scheduling community from the need to understand what impact scheduling inherent jitters have on the stability and performance of control systems. Real-time scheduling generally assumes that a control algorithm implemented as a periodic task with standard timing constraints such as period and deadline will meet the control requirements (in terms of stability and control performance). However, since a periodic task execution in scheduling theory is an execution that takes place anywhere within its deadline, small time variations (i.e., jitters) occur at each task instance execution. Consequently, the resulting jitters for control task instances do not comply with the deterministic timing demanded by discrete-time control theory in the implementation.

Specifically, as we show in this thesis, the negative effects of bringing together the separate results of control and scheduling theories for computer-based control systems are:

- ***Poor computing system schedulability:*** in general, when discrete-time control theory timing requirements are expressed with traditional periodic task timing constraints (periods and deadlines), the task set schedulability becomes unfeasible.
- ***Controlled system performance degradation*****:** *scheduling inherent jitters* for periodic control tasks (which implement controllers designed using classic[2] discrete-time control theory) degrade the performance of the controlled system, and can even cause a critical failure (instability).

The integrated control and scheduling framework we present in this thesis bridges the gap between the two communities, joining control and scheduling theory so as to provide solutions to both problems.

1.2 Objectives

In the thesis we show how to use a combination of control and scheduling principles in order to design controllers that deal with new and more flexible timing constraints and to allow scheduling approaches to take scheduling decisions considering both system schedulability and control performance. Specifically, *we demonstrate that combining offline scheduling analysis and offline control analysis with online scheduling and dynamic control compensations we obtain better system schedulability, and better control performance*.

The integrated control and scheduling framework that we present as an analysis and design methodology for real-time control systems is based on two novel paradigms:

- ***Flexible control design:*** we present a new controller design method, ***compensation approach***, that goes beyond the classic discrete-time control theory timing assumptions of equidistant sampling and equidistant actuation given by the constant sampling period and constant time delay specified at the design stage. Instead of specifying a single value for the sampling period and a single value for the time delay, we design controllers to account for a set of *feasible sampling intervals* and for a set of *feasible sampling-actuation delays*. This controller design method, which includes new stability and response analysis, relies on the idea of adjusting controller parameters at runtime (compensations) according to the specific implementation timing behaviour, i.e., scheduling inherent jitters. In addition, design decisions are also taken regarding implementation details such as space and time overheads. We consider two alternatives: (a) performing the compensation calculations online - if these incur only negligible overheads - or (b) determining offline the compensation parameters for table look-up at runtime. We characterise when each of these alternatives is suitable.
- ***Flexible control task scheduling:*** we present new ***flexible timing constraints*** for control tasks. Controllers based on the compensation approach assume a closed-loop implementation with *irregular sampling* and *varying time delays*. The new timing assumptions behind the compensation approach give the potential to derive more

[2] *Classic* discrete-time control theory refers generically to well known discrete-time control methods and models based on *regularly sampled discrete-time systems* (regardless of whether they belong to *classical* or *modern* control theory, a distinction made within the control community to differentiate methods based on *transfer function* or *state-space* models. See section 2.2.1 for further details).

flexible timing constraints for control tasks, beyond task periods and deadlines necessary to apply standard periodic task scheduling. Control tasks are no longer seen as classic real-time tasks with fixed values assigned to their timing constraints (such as periods and deadlines). They are characterised by flexible timing constraints in terms of *feasible instance separation* and *feasible response time* sets. That is, at each control task instance execution, scheduling approaches can choose for the instance separation and response time constraints different values from these sets, taking into account, for example, schedulability of other tasks or performance improvement of the controlled processes. These constraints are defined on a per control task instance basis, as opposed to fixed values, such as periods and deadlines, applicable to all instances – as assumed by standard scheduling schemes such as Rate Monotonic (RM) [LIU73], Earlier Deadline First (EDF) [LIU73] and Fixed Priority Scheduling (FPS) [TIN94]. Thus, our methods provide more flexibility than the one obtained by using fixed timing constraints.

Using these novel paradigms, we show how to solve the two problems outlined in the previous section:

- *With the flexibility given by the compensation approach controller design method, we* ***eliminate the control performance degradation*** *that control tasks subject to scheduling inherent jitters introduce in the controlled processes.* Although the scheduling community has tried to minimise jitters by designing specific purpose real-time task models and algorithms, jitter is an inherent scheduling problem and cannot be completely removed. Nevertheless, we show that by accepting jitters in the control design, control task implementing controllers designed with the compensation approach solve the degradation that would otherwise occur. In summary, we solve the problems posed by scheduling inherent jitters for periodic control tasks, which in general are not addressable using traditional offline and online scheduling based approaches nor by previous real-time and control integration approaches.
- *With the new flexible timing constraints for control task scheduling we provide the instruments that can be used* ***to transform unfeasible schedules and instable control systems into feasible schedules and stable control systems***. The application of fixed timing constraints for control tasks impairs system schedulability by over-constraining the schedule. However, the compensation approach affords us the possibility of relaxing the strict periodicity and deadline requirements for traditional control task scheduling (based on fixed timing constraints); instead, we demonstrate how we can take advantage of the new flexible timing constraints for control tasks scheduling in order to improve system schedulability. Note that we do not propose a specific scheduling approach, rather a new set of flexible control timing constraints for control tasks that we show can be used to achieve stable control systems when the same control tasks characterized by fixed timing constraints were not schedulable.

In addition, by taking advantage of the compensation approach and flexible timing constraints for control tasks, we define a ***Quality-of-Control*** (QoC) metric that associates with each feasible flexible timing constraint a quantitative value expressing control performance in terms of the controlled system error resulting from the use of that timing constraint. This offers the possibility of taking scheduling decisions at each control task

instance execution considering this control information, thus demanding novel scheduling approaches. We present a new scheduling paradigm, ***QoC scheduling***, in which we demonstrate that the QoC information of task timing constraints can be used **to improve the performance of the controlled processes in the presence of perturbations**. After formulating the QoC scheduling problem, we categorise the main scheduling issues and identify feasible solutions. Specifically, we show how the problem of reacting to perturbations with control tasks specified with control timing constraints expressing QoC can be achieved applying standard guarantee techniques

1.3 System model

In this section we describe the system model for the approach to scheduling and control co-design we present. We consider a *distributed control system* that consists of a set of *processing nodes* that run one or several *tasks*, some of them in charge of controlling *physical systems* (plants/processes), which communicate data across a *communication network*. See Figure 1.1 for a full view of the system model.

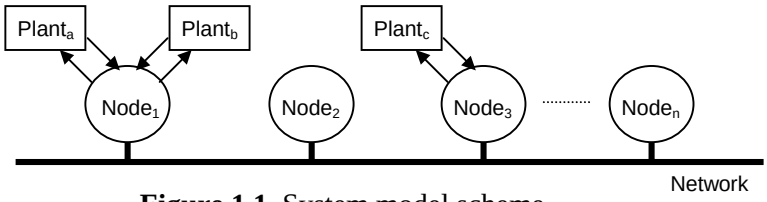


Figure 1.1. System model scheme

We assume a *discrete time* model [KOP92]: An external observer counts the ticks of the globally synchronized clock (which is based on the metrics of the physical second, as a time unit) with *granularity* of length *g* and assigns natural numbers from 0 to ∞ to them. All the nodes and the communication network of the system have the same time granularity. Digital-to-Analogue (DA) and Analogue-to-Digital (AD) converters (that interface processing nodes and the continuous-time physical systems that are controlled) have the same time granularity. We assume that the DA and AD operation times are negligible compared to *g* (granularity length).

At a node level, each node runs a set of tasks. We distinguish two types of tasks according to their functionality: *control tasks* and *non-control tasks*. Control tasks are in charge of controlling the physical systems (through measuring the controlled variables, *y(t)*, and transmitting the control signals, *u(t)*), if any. A physical system is controlled by one or several control tasks. Non-control tasks are the remaining tasks. See Figure 1.2 for an illustrative scheme of the node level.

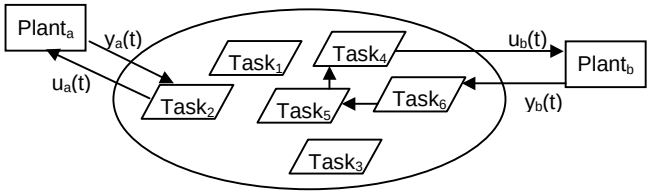


Figure 1.2. Node level scheme

In a node, at the operating system level, a *dispatcher* (also called scheduler), at each time it is invoked, either assigns a task to the processor or leaves it idle. A task is executed uninterruptedly during the granularity length *g*. The task execution lengths are a multiple of the granularity. We regard the actual time to perform the dispatching and the task context switching to be negligible with respect to *g*[3]. If the dispatching and the task context switching are too long, it can be included in the worst-case execution time of the task.

The allocation of tasks to processors is carried out using any of the allocation published methods, such as in [FOH95] or [RAM90]. Therefore, the scheduling strategies we present are in a node level context.

1.4 Thesis structure

The thesis is organized as follows:

Chapter 2 gives a brief overview of basic but important concepts of both real-time and control systems theory and practice. Also, the state of the art concerning this thesis is reviewed.

In *Chapters 3* and *4* we identify the two main problems this thesis solves. In *Chapter 3* we explore the impact of classic control timing requirements on real-time scheduling. We show that in general this leads to unfeasible scheduling scenarios. *Chapter 4* explains the impact of scheduling inherent jitters on control tasks. We show that jitters in control task instance executions degrade the controlled system response, even causing instability.

In *Chapter 5* we discuss the necessity of developing a) new flexible control design methods and b) more flexible timing constraints for control task scheduling for solving the problems identified in the previous two chapters.

In *Chapter 6* we define the compensation approach controller design method. We discuss its completeness in terms of coping with all possible closed-loop implementations. We then formulate the new controller design method problem based on state-space models.

After the problem formulation, in *Chapter 7* we present the new controller design method that includes new stability and response analysis, all based on state-space models. We also address practical aspects such as code implementation details and different strategies for the controller parameter adjustment required for the application of the compensation approach.

In *Chapter 8* we explain the use of the compensation approach as a control-based solution to eliminate the degradation that scheduling inherent jitters introduce in the controlled system response. In this context, we also present a performance evaluation of the application of the compensation approach.

In *Chapter 9* we present new flexible timing constraints for control task scheduling: firstly we demonstrate how to use them to obtain feasible schedules of task sets that were not feasible using fixed timing constraints; secondly, after presenting the QoC metric and

[3] For example, in the real-time kernel S.Ha.R.K. (Soft and Hard Real-time Kernel) [GAI01], the task context switch and dispatcher execution is 15μs. We regard such times for our system model negligible because the granularity we use is of the order of milliseconds.

formulating the QoC scheduling problem, we demonstrate that scheduling decisions can be taken accounting for both schedulability and control performance improvement.

Finally, *Chapter 10* draws the conclusions of this thesis, lists the main contributions and points out directions for future work.

At the end all *references* are listed. We also include three appendices: *Appendix A* with code details of the controllers we design and use, *Appendix B* with a numerical stability analysis of two of the examples we use and *Appendix C* with the closed loop matrices we found in the system evolution for one of the examples we use.

Chapter 2

Background and state of the art

The successful design and implementation of real-time control systems requires the interaction between two technical disciplines: real-time systems and control systems. However, as pointed out in [TOR98], control theory and real-time theory have been relatively independent research areas. The aim of this chapter is to provide a brief overview of basic but important concepts of both real-time and control systems theory and practice. In this way, we introduce the fundamental ideas for understanding the integrated control and scheduling framework we present while bridging the existing gap between both real-time and control communities.

In this chapter we also review related work. We show that the practical problems posed by:

a) scheduling inherent jitter in control tasks, i.e., controlled system degradation,

b) fixed timing constraints to meet the stringent timing requirements assumed by discrete-time control theory, i.e., poor system schedulability,

have not been formally addressed. In addition, neither controlled systems performance nor system schedulability have been, in any of the previous works, jointly improved as we do with the application of more flexible controller designs and more flexible timing constraints for control tasks scheduling.

2.1 Real-time systems

Real-time systems can be constructed out of sequential programs, but are typically built from concurrent programs, called *task*s. A real-time task is an executable entity of work that, at a minimum, is characterized by a *worst-case execution time* (WCET) [PUS89] and a *time constraint* [RAM96]. The WCET is an estimation of the maximum time required by the processor to execute the task. A typical timing constraint on a real-time task is the *relative deadline*, i.e. the time interval within which the task must complete its execution. The objective of real-time computing is to meet the individual timing constraints of tasks. Real-time systems are computing systems in which the correctness of the computations depends not only on the logical results but also on the time at which the results are produced [STA88]. These systems have a unique set of requirements that are not always taken into consideration, leading to serious misconceptions about real-time computing [STA88]. The main objective of real-time computing is not fast computing, it is predictability [STA90]. The fast computing objective is to minimize the average response time of a given set of tasks. Predictability implies that it has to be possible to prove that timing requirements are met, in accordance to system specifications. Regarding timing requirements, real-time scheduling is the main concern.

In real-time systems, scheduling theory addresses, by means of algorithms, the problem of meeting the specified timing requirements in order to have understandable and predictable system timing behaviour. Therefore, scheduling involves the allocation of resources and time in such a way that certain performance requirements are met [RAM94].

2.1.1 Tasks constraints

Real-time tasks are computing entities that must process inputs and provide the adequate outputs (see Figure 2.1) within a time interval, i.e. relative deadline, which is dictated by the requirements of the application.

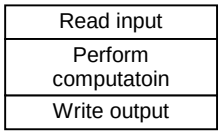


Figure 2.1. Task structure

Depending on the consequences of a missed deadline, real-time tasks can be characterized according to its criticality as:

- *Hard real-time tasks*: the completion of the task must be within its deadline, otherwise serious consequences occur.
- *Soft real-time tasks*: a task is soft if missing its deadline decreases the performance of the system but no serious consequences occur if there is a late completion.

Accordingly, although we can differentiate two types of real-time systems, hard and soft, many systems consist of both hard and soft real-time tasks. Sometimes, if the consequences of missing a deadline are catastrophic, the target system is called a critical (or safety-critical) real-time system instead of a hard real-time system. In fact, it has to be pointed out that there is a current debate on the previous definitions.

Another timing characteristic that can be specified in a real-time task concerns the regularity of its activation. Depending on this, a task is defined as a *periodic*, *aperiodic* and *sporadic*. Periodic tasks consist of a sequence of identical activities, called *instances*, which are recurrently activated on a regular basis. We denote the k^{th} instance of a periodic task $task_i$ by $task_{i,k}$. Figure 2.2 shows an example of tasks instances for a periodic task.

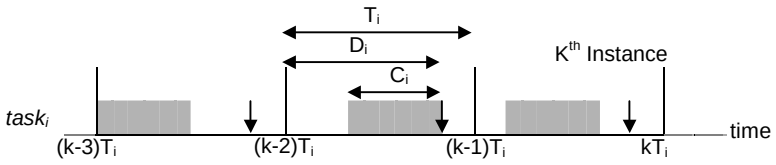


Figure 2.2. Sequence of instances for a periodic task

The activation of the k^{th} instance of a periodic task $task_i$ is given by $(k-1)T_i$, where T_i is called the *period* of the task. In many practical cases, a periodic task $task_i$ can be completely characterized by its WCET C_i (which is an estimation of the maximum time required by the

processor to execute the task), its relative deadline D_i (which is the time interval within which the task should be completed), which is often considered to coincide with the end of the period.

On the other hand, a task that is not invoked at regular intervals is an aperiodic task. Aperiodic tasks characterized by a minimum inter-arrival time are called sporadic. For further details on tasks characterization, see for example [BUT97].

Standard *timing constraints* for a periodic task (such as *period* and *deadline*) are fixed, i.e. a single value holds for all instances of the task [FOH94]. We have to point out that some works, for example [DOB01], distinguish between *simple* constraints, i.e. period and deadlines, and *complex* constraints such as end-to-end deadlines. We do not make this distinction. For a discussion on task constraints, see [RAM96].

Other typical constraints that can be specified in real-time tasks, apart from timing constraints, are *precedence relations* and *resource constraints*. Precedence constraints refer to the fact that computational activities cannot be executed in arbitrary order but have to respect some precedence relations defined at the design stage. Resource constraints refer to the fact that computational activities that share resources have to be synchronized.

2.1.2 Real-time Scheduling

When a processor has to execute a set of concurrent tasks, the processor has to be assigned to the various tasks according to a predefined criterion, called a *scheduling polic*y. There are a great variety of algorithms proposed for scheduling of real-time systems today. A *schedule* is an assignment of tasks to the processor, so that each task is executed until completion.

The *dispatcher* allocates the processor to the task selected by the scheduling policy. Consequently, a task that could potentially be executed by the processor (*active* task) can be either in execution (*running* task) or waiting (*ready* task) in the ready queue if another task is executing. *Context switches* allow the running task exchange on the processor.

In this context, we introduce the following notation to facilitate the description of schedules (and scheduling policies):

$r(task_{i,k})$ denotes the *release time* of the k^{th} instance of task $task_i$, i.e. the time at which a task instance becomes ready for its execution

$s(task_{i,k})$ denotes, the *start time* of the k^{th} instance of task $task_i$, i.e. the time at which a task instance starts its execution

$f(task_{i,k})$ denotes the *finishing time* (also called *completion time*) of the k^{th} instance of task $task_i$, i.e. the time at which a task instance completes its execution

In Figure 2.3, these concepts are portrayed.

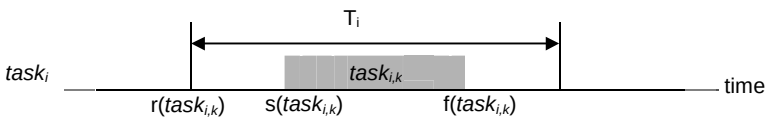


Figure 2.3. Task instance description

As described in [BUT97], in general, to define a scheduling problem we need to specify three sets: a set of tasks, a set of processors and a set of types of resources. Moreover, precedence relations among tasks can be specified through a direct acyclic graph, and timing constraints can be associated with each task. In this context, scheduling means to assign processors and resources to tasks in order to complete all tasks under the imposed constraints. This problem, in this general form, has been shown to be NP-complete [GAR79] and hence computationally intractable. Note that the complexity of scheduling algorithms is highly relevant when scheduling decisions must be taken on-line, during task execution.

In order to reduce the complexity of constructing a feasible schedule, one may simplify the computer architecture (e.g., by restricting it to the case of uniprocessor systems), or may make simplifying assumptions on the tasks (e.g., remove precedence constraints). A schedule is said to be *feasible* if all tasks can be completed according to a set of specified constraints. In order to check the feasibility of the schedule before tasks execution, the system has to plan its actions by looking ahead to the future and by assuming a worst-case scenario. Recall that in hard real-time applications that require highly predictable behaviour, the feasibility of the schedule should be guaranteed in advance, that is, before tasks execution. Feasibility tests (see [JEF93] for further discussion) can be based on the processor utilization approach (which measures the fraction of processor time spent in the execution of the task set) and/or on response time analysis techniques [JOS98] (which uses recurrent formulas to calculate the worst-case finishing time of any task).

For example, a necessary condition for achieving schedulability using the processor utilization (U) approach on a single processor is given by (2.1).

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.1}$$

where C_i and T_i are the task $task_i$ worst-case execution time and period, respectively. Known sufficient schedulability conditions for scheduling algorithms such as RM and EDF [LIU93] are given by (2.2), where U=1 for EDF and $U=n(2^{1/n}-1)$ for RM.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} < U \tag{2.2}$$

In the following, we give a short description of the main types of scheduling algorithms (this classification is based on the assumptions made about the system or the tasks). Rather than providing an exhaustive description, we pick up the most important concepts and properties that are of special interest for the work of this thesis.

Offline vs. online scheduling

Among the great variety of real-time scheduling approaches that have been presented, real-time scheduling algorithms fall into two categories [STA95], depending on the time scheduling decisions are taken: *offline* and *online* scheduling.

In offline scheduling, the scheduler has complete knowledge of the task set and its constraints, such as deadlines, computation times, precedence constraints and so on. The schedule construction is based on fixed parameters, assigned to tasks before their activation. The entire offline guaranteed schedule is stored in a table (which contains all guaranteed tasks arranged in the proper order) and dispatched later during system runtime. The main advantage of offline scheduling is that it provides determinism, as all times for task executions are determined and known in advance. In addition, the runtime overhead does not depend on the complexity of the algorithm. This allows very sophisticated algorithms to be used to solve complex problems (e.g., control constraints). However, as all actions have to be planned before startup, run-time flexibility is lacking. Offline scheduling is also referred as *static* or *pre-runtime* scheduling.

In contrast, online scheduling algorithms make their scheduling decisions at runtime. Online schedulers are flexible and adaptive, but they can incur significant overheads because of runtime processing. Besides, online scheduling algorithms do not need to have complete knowledge of the task set or its timing constraints. For example, with an external event that arrives at the runtime of the system, we need to deal with it upon its arrival. In online scheduling algorithms, scheduling decisions are taken every time a new task enters the system, when a task becomes ready and/or when a running task terminates. Although online scheduling can provide more flexibility, it is limited with respect to predictability, as actual start and completion times of execution depend on run-time events. It is worth noting that depending on the algorithm, the guarantee must be done on-line (e.g., when a new task enters the system). In such cases, since the guarantee algorithm is based on worst-case assumptions, a task could be unnecessarily rejected. On the other hand, the benefit of having an online guarantee mechanism is that a potential overload situation can be detected in advance, thus avoiding negative effects on the systems. Online scheduling is often referred to as *dynamic* or *runtime* scheduling.

These are also scheduling approaches that fall into both categories. For example [FOH95] combines offline and online scheduling, taking advantage of the determinism provided by offline scheduling, and the flexibility provided by online scheduling.

To illustrate the determinism provided by offline and online scheduling policies in terms of knowing the exact task executions times before run-time, consider the following task set (Table 2.1) with two periodic tasks, where T_i is the task period and C_i is the WCET. We assume deadlines equal to periods.

	T_i	C_i
$task_2$	5	2
$task_1$	4	1

Table 2.1. Task set

Applying an offline scheduling strategy, we can construct, according to task timing constraints, the feasible offline schedule over the tasks periods LCM (Least Common

Multiple)[1] that we show in Figure 2.4 (where boxes mark task periods and shaded areas mark worst-case execution times).

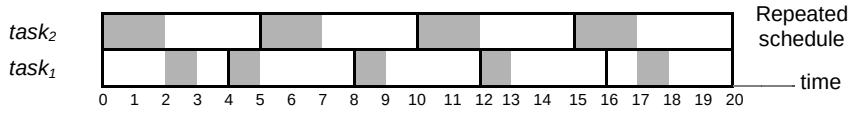


Figure 2.4. Offline schedule

Given this offline schedule, at run time, the dispatcher will execute task instances ($task_{i,k}$) according to the ordering provided by the constructed table (Table 2.2) for each LCM.

Time	0	2	4	5	8	10	12	15	17
Task instance	$task_{2,1}$	$task_{1,1}$	$task_{1,2}$	$task_{2,2}$	$task_{1,3}$	$task_{2,3}$	$task_{1,4}$	$task_{2,4}$	$task_{1,5}$

Table 2.2. Dispatcher table

Therefore, looking at the schedule (Figure 2.4) and dispatcher table (Table 2.2), at run time, the execution start time of each instance will coincide with the offline instances start times. However, looking at the completion times, since the offline schedule is based on worst-case execution time, the run time completion times may be different to the offline completion times, because at run time, each task instance can execute less than the assumed worst-case (as we show in Figure 2.5 for one of the LCM executions, where shaded areas mark now actual execution times).

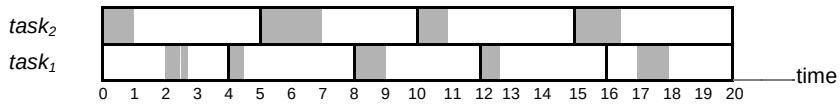


Figure 2.5. Run time execution of the offline schedule of Figure 2.4

Note for example that the first, third and fourth instances of task $task_2$ (starting at times 0, 10 and 15 respectively, Figure 2.5) executes less than its WCET. Instead of executing for 2 time units (as assumed by the worst case scenario, Table 2.1), the first and third instances execute 1 time unit while the fourth instance executes 1.5 time units. Similar phenomena occur in the finishing times of instances of task $task_1$ (Figure 2.5) Therefore, although the starting times of each task instance is known before run time, the actual finishing times, if the offline schedule is constructed in terms of the tasks WCET, may not coincide with the offline finishing times.

However, if what characterises each task (in Table 2.1) is the exact execution time (which is a reasonable assumption for most real-time control systems [BUT98]) instead of the worst-case execution time, the run time completion times will also coincide with the offline completion times.

[1] Although theoretically, offline scheduling can construct non-periodic schedules, for practical purposes, offline schedules are constructed over the LCM of the task's periods, which implies a LCM periodic pattern.

However, none of these properties (known start and completion times) apply to online scheduling. For example, we apply EDF [LIU73] to the task set specified in Table 2.1. Recall that EDF is an online scheduling policy where tasks instances are dispatched at run time according to the earlier deadline.

The set of tasks complies the EDF schedulability test (2.2) as detailed in (2.3).

$$U = \sum_{i=1}^{2} \frac{C_i}{T_i} = \frac{2}{5} + \frac{1}{4} = 0.65 \leq 1 \tag{2.3}$$

An example of the schedule that we obtain over the tasks periods LCM is shown in Figure 2.6, where boxes mark task periods and shaded areas mark WCET.

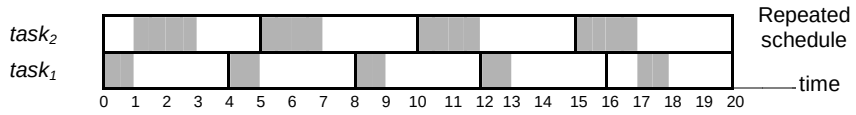


Figure 2.6. Example of schedule produced by EDF

In this case, at run time, every time a task instance terminates its execution (which can execute its worst execution time or less), the scheduler assigns the processor to another task instance. Consequently, before run time, the exact start and completion times of each task instance is not known.

The previous problem (knowing task start and completion times before run time) increases when the online scheduling policy takes scheduling decisions upon arrival of aperiodic or sporadic tasks.

Pre-emptive vs. non pre-emptive

Looking at the run-time behaviour of different scheduling policies, two modes can be distinguished: *pre-emptive* and *non pre-emptive* scheduling. With pre-emptive scheduling, the running task can be interrupted at any time by another task, according to some predefined scheduling policy. In non pre-emptive scheduling, a task, once started, is executed by a processor until its completion.

When the application tasks have different levels of criticalness expressing task importance, i.e., priority, pre-emption permits us to anticipate the execution of the most critical tasks, producing more efficient schedules in terms of system responsiveness. However, this also implies that predictability in terms of knowing times for task executions decreases.

To illustrate the decrease of determinism that pre-emptive scheduling policies implies in terms of knowing task executions times before run-time, we again consider the task set we used before (Table 2.1). For example, we apply to the task set RM [LIU73] scheduling approach, which is a pre-emptive scheduling policy based on the following priority assignment scheme: to give the tasks a priority level based on its period: the smaller the period, the higher the priority; that is, $T_i < T_j$, $P_i > P_j$, where T_i and P_i denotes the task period and priority of each task $task_i$. Table 2.3 also gives the priority assignment for the task set we are considering (recall that task deadlines are assumed to be equal to task periods).

	T_i	C_i	P_i
$task_2$	5	2	2
$task_1$	4	1	1

Table 2.3. Priority assignment for RM

The set of tasks complies with the RM schedulability test (2.2) as detailed in (2.4)

$$U = \sum_{i=1}^{2} \frac{C_i}{T_i} = \frac{2}{5} + \frac{1}{4} = 0.65 < 2(2^{1/2} - 1) \approx 0.83 \quad (2.4)$$

An example of the schedule that we obtain over the tasks periods LCM is shown in Figure 2.7, where boxes mark task periods and shaded areas worst-case execution times.

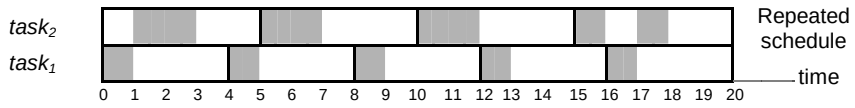


Figure 2.7. Example of schedule produced by RM

As can be seen in Figure 2.7, the fourth instance of task $task_2$ is pre-empted at time 16 by the fifth instance of task $task_1$. This interruption adds more variability in the completion time of the fourth instance of task $task_2$.

Time triggered vs. event triggered

There are two fundamentally different principles that determine the activation of tasks in a real-time system, *event-triggered* and *time-triggere*d. In *event-triggered* systems, all activities are activated in reaction to relevant events external to the system. When a significant event in the outside world happens, it is detected by some sensor, which then causes the attached device (processor) to get an interrupt signal. For soft real-time systems with lots of computing power to spare, this approach is simple, and works well. The main problem with event-triggered systems is that they can fail under heavy load conditions, i.e., when many events are happening at once. Event-triggered designs give a faster response at low load but more overhead and chance of failure at high load. This approach is more suitable for dynamic environments, where dynamic activities can arrive at any time.

In a *time-triggered* system, all activities are activated at certain points in time that are known *a priori*. Accordingly, all nodes in time-triggered systems have a common notion of time, based on synchronised clocks. One of the most important advantages of time-triggered systems is the deterministic temporal behaviour of the system, which eases system validation and verification considerably. Time-triggered systems are suitable in static environments in which the system behaviour can be completely known in advance.

Summary

As we have seen in the previous scheduling examples, as far as task timing constraints are fulfilled, the periodic task execution given by scheduling theory is an execution that takes place some time within the task relative deadline. This has the effect of introducing start-

time delays in task instance executions, as well as introducing variable completion times. We call the variability caused by scheduling policies on task instance executions *scheduling inherent jitters*.

Although a great variety of scheduling policies have been presented (for handling periodic, aperiodic and sporadic tasks, on a single and multiprocessor architecture, with simple and complex constraints, etc) in this work we will mainly focus on offline constructed schedules as well as traditional Earliest Deadline First (EDF) [LIU73] and Fixed Priority Scheduling (FPS) [TIN94] based scheduling approaches. The main reason is that if the methods we present work well for these standard approaches, they should improve further if applied to more sophisticated scheduling algorithms based on the former.

Note that in the following chapters, we do not propose any scheduling algorithm. Rather, we provide instruments to be used to improve the controlled systems responses and system schedulability, which we show to be applicable to standard scheduling approaches. This explains why we will not compare any scheduling algorithms.

In this section, specific features and types of scheduling algorithms have been described. For further reading, see [SHI94], [AUD95], [BUR97], [BUT97], [TIN97] and [STA98].

2.2 Control systems

A *control system* is an interconnection of components forming a system configuration that will provide a desired system response [DORF95]. The basis for analysis of a system is the foundation provided by linear system theory, which assumes a cause-effect relationship for the components of the systems. A component or process to be controlled (also called physical system or plant) can be represented by a block (as shown in Figure 2.8) where the input-output relationship represents the cause-effect relationship.

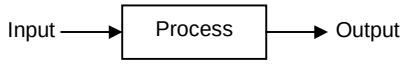


Figure 2.8. Process to be controlled

In control terms, the *controlled* variable is the quantity or condition that is measured and controlled (i.e., process output). The *manipulated* variable is the quantity or condition (i.e., process input) that is varied by the controller so as to affect the value of the controlled variable. Depending on the type of information that the controller uses to vary the manipulated variable, we distinguish between *open-loop* control and *closed-loop* control. The defining feature of an open-loop control is that the controller function that varies the manipulated variable is determined completely by an external process that accounts only for the desired output response (Figure 2.9)

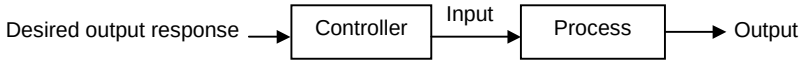


Figure 2.9. Open-loop (also called feedforward) control system

A *closed-loop* control system (also called closed-loop system) utilizes an additional measure of the actual output (controlled variable) to compare it with the desired output. Consequently, the controller function is determined on a continuing basis by the behaviour of the system itself (as expressed by the behaviour of the outputs). The measure of the output is called the *feedback signal*, because it is fed back (possibly in modified form due to the controller action) to the process. In summary, in closed-loop systems, *to control* means measuring the value of the controlled variable of the system and applying the manipulated variable to the system to correct or limit deviation of the measured value from a desired value. A simple closed-loop control system is shown in Figure 2.10.

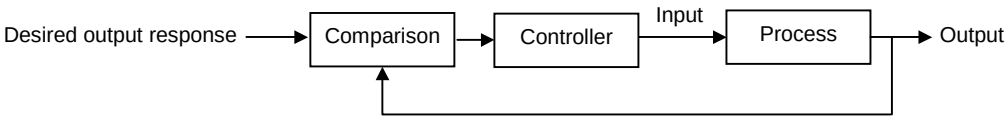


Figure 2.10. Closed-loop (also called feedback) control system

There are many reasons why closed-loop control is often preferable to open-loop control. Feedback is often superior to open-loop from a performance standpoint and it can automatically adjust to unforeseen system changes or to unanticipated disturbance inputs. A *disturbance* is a signal that tends to adversely affect the value of the output of the system.

A distributed control system is a control system whose processors (sensors, controllers and actuators), which run one or several tasks, are distributed geographically and the processors communicate data through some communication medium. The key for distributed control systems is that almost no local control action can be taken in isolation from the rest of the system.

Broadly speaking, a control system basically has three main subsystems: a sensory subsystem, a controller subsystem and an actuator subsystem (Figure 2.11). In a distributed control system, each of these subsystems can be physically divided into separate units, and control loops are closed over communication networks.

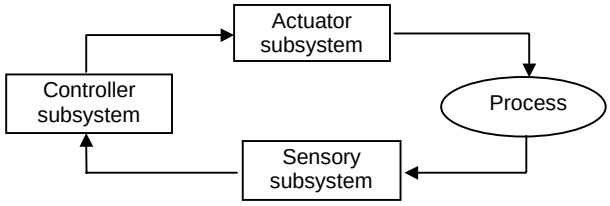


Figure 2.11. Subsystems in a closed-loop control system

The general functionality of control systems can be described as follows: firstly, the sensory system collects data from the process to be controlled. Secondly, the control system, by means of a control law, processes this data and calculates the control signal, considering the desired process behaviour. Finally, the actuator system performs the action on the process according to the control signal.

The *feedback* concept has been the foundation for control systems analysis and design, which enables us to control the desired output and improve *accuracy* while maintaining *stability*. Traditionally, closed-loop systems were *analogue-control systems* (also called *continuous-time systems*); practically all the control systems implemented today are based on computer control, i.e., they are *computer-controlled systems* (also called *discrete-time systems*).

2.2.1 Analysis and design of control systems

To understand and control systems (or processes), one must obtain quantitative mathematical models of these systems by analysing the relationships between the system variables. Because the systems are dynamic in nature, when describing the dynamic behaviour of a physical system by physical laws, the descriptive equations are usually differential equations. Physical systems are inherently *non-linear* (i.e., the relation between their variables is not linear). However, in many systems, if the system signals do not vary over too wide a range, the system responds in a *linear* manner (satisfies the properties of superposition and homogeneity). Consequently, even though we deal with non-linear systems, in order to design the control law, the usual procedure that we follow is to work with a linearised approximation model (if it is not already linear) of the system, concerning the functional parameters. In addition, depending on whether the properties of the system change with time, a system is *time-varying* or *time-invariant* (i.e., whether the coefficients of the equations that describe the system behaviour vary with time or not). In a time-varying control system, the response of the system will depend on the time at which an input is applied.

In the following subsections, we introduce important concepts for the analysis and design of control systems. For further details, see for example [AST97], [DORF95], [OGA97], [PHI95] and [VAC95].

Mathematical models

The first step in the analysis of a dynamic system is to derive its mathematical model. We must always keep in mind that deriving a reasonable mathematical model is the most important part of the entire analysis. In control theory, two types of mathematical models are used to describe the system dynamics: *transfer function* models and *state-space* models. Although the different models are equivalent (in the sense that there are methods of changing from one representation to another and vice versa), depending on the particular system and the particular circumstances (e.g., in optimal control problems, it is advantageous to use state-space models, while for single input, single-output systems, the transfer function may be more convenient), one mathematical model may be better suited than other models. Once a mathematical model of the system is obtained, various tools can be used for purposes of analysis and synthesis.

In addition, *classical control theory* utilizes the transfer function concepts extensively. The transfer function describes the dynamics of the system, and represents the relationship of the input and output variables (cause-effect relationship). The transfer function replaces the differential equations that describe the system by algebraic equations in terms of a complex

variable (*s* for linear time-invariant continuous-time systems or *z* for linear time-invariant discrete-time systems) that are easier to solve. Analysis and design are done in the *s*, *z* and/or frequency domain. *Modern control theory*, which is based on state-space concepts (where the state of the system is represented by the *state variables*), extensively utilizes the vector-matrix analysis. State-space models allow us to describe the future response of a system, given the present state (characterized by the state variables), the excitation inputs and the equations describing its dynamics. These models can be used for both continuous-time and discrete-time systems, the latter only considering the system at the sampling points. Analysis and design are done in the time domain.

Controller design

The goal of controller design is to achieve certain desired closed-loop system characteristics. The desired characteristics, or *performance specifications*, generally relate to the *controlled system response* (such as *transient response* and *steady-state accuracy*) and *stability*. The interplay of performance and stability can be somewhat subtle, and there is often a trade-off to be made in this regard. The desired controlled system characteristics, in a formal specification of the problem, are given through the design parameters, which can be met by specifying the *closed-loop poles* location for both analogue and discrete-time control systems. However, rather than specifying design parameters, usually it is more meaningful to specify quantities, such as at which time the controlled system recovers from a perturbation, or the allowed error of the controlled system response to certain anticipated inputs. As a consequence, the relation between these quantities and the formal design parameters has been extensively studied.

Closed-loop poles

Closed-loop poles represent the system's autonomous behaviour. A system subject to a perturbation follows a specific trajectory. The characteristics of the trajectory (shape, velocity, etc) depend on the closed-loop poles. That is, the closed-loop poles location determines the *stability* and *response* of the system. The poles of a closed-loop system can be mathematically obtained either from the transfer function or state-space model of the system. For a transfer function, the closed-loop poles are the roots of the polynomial denominator; for state-space models, they are the eigenvalues of the system matrix. Poles of a continuous-time system have a direct mapping to the poles of a discrete-time system, for a given sampling period.

Stability analysis of closed-loop systems

Stability is a basic requirement of all control systems, and therefore it is the first design specification to account for in the analysis and design of control systems. The concept of stability is very important when analysing dynamic systems. A control system is said to be in *equilibrium* if in the absence of any perturbation or input, the system output remains in the same state. A linear time-invariant control system is *stable* if the system output returns to the equilibrium state when the control system is subject to an initial condition (e.g., perturbation). If the system output neither converges nor becomes unbounded (e.g., bounded oscillation), the system is said to be *marginally stable*. A linear time-invariant control

system is *unstable* when the system output unboundedly diverges from the equilibrium state when the control system is subject to an initial condition (e.g., perturbation). Figure 2.12 illustrates these stability concepts:

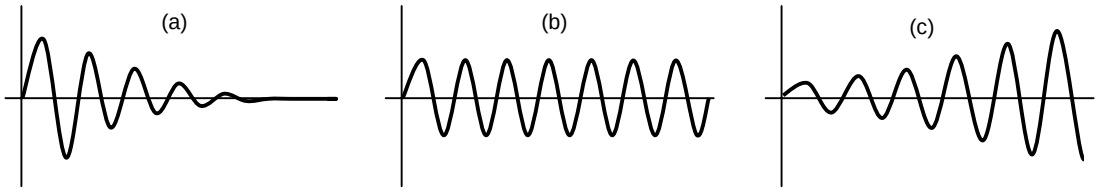


Figure 2.12. (a) Stable, (b) marginally stable or (c) unstable linear time-invariant control system

The stability of a closed-loop system can be determined by the location of its closed-loop poles. A linear time-invariant control system is stable when in the (see Figure 2.13):

- continuous-time domain, all its closed-loop poles lie in the left half plane of the s-plane
- discrete-time domain, all its closed-loop poles lie within the unit circle in the z-plane

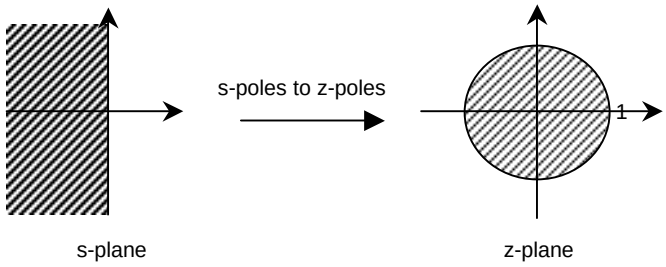


Figure 2.13. Linear time-invariant control system stability according to the closed-loop poles location in the (a) s-plane and (b) z-plane (the crossed area marks the stability zones)

There are several techniques to analyse the stability of closed-loop systems.

Response analysis of closed-loop systems

Apart from stability, *transient response* (also called relative stability) and *steady-state accuracy* are also a focus of attention in the design (whether continuous-time or discrete-time) of control systems. The transient response appears because systems cannot react instantaneously when they are subject to inputs or perturbations. Analytically, it refers to that portion of response due to the closed-loop poles of the system. The steady-state response refers to the ability of a control system to follow or track certain inputs (such as classic pulse, step, ramp or sinusoidal inputs) with minimum error. Analytically, it refers to that portion of the response due to the input (or forcing function). Figure 2.14 illustrates these concepts.

In many practical cases, the desired performance characteristics of control systems, either for continuous-time or discrete-time design methods, are specified in terms of time domain quantities (note that in most of the cases, the controlled plants are continuous, thus the plant

output signals are also continuous). Specifically, the performance characteristics of a control system are specified in terms of the transient response to a unit step input (this is due to the fact that the unit step input is easy to generate and is sufficiently drastic to provide useful information on both the transient and steady-state response characteristics of the system). Figure 2.14 shows the unit step response of a standard second order system[2], with the following descriptors:

- *Rise time (t_r)*: is the time required for the response to rise from 10% to 90% of its final value.
- *Maximum overshoot*: is the maximum peak value of the response curve measured form unity[3]
- *Peak time (t_p)*: is the time required for the response to reach the maximum overshoot
- *Settling time (t_s)*: is the time required for the response curve to reach and stay within a range about the final value of a size specified as an absolute percentage of the final values (usually a δ of 2% or 5%)

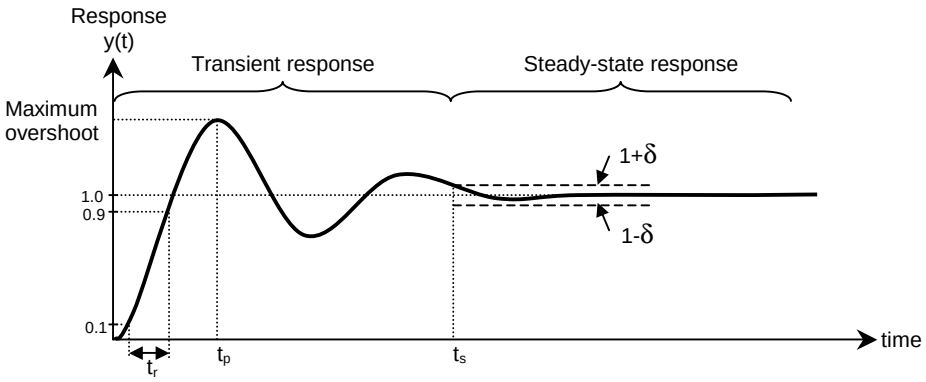


Figure 2.14. Unit step response of a standard second order system.

The transient response characteristics of a closed-loop system, which often exhibits damped oscillations before reaching the steady-state, can be determined by the location of its closed-loop poles. Frequently, we define the closed-loop system with a pair of dominant poles as one that can be modelled with reasonable accuracy by a second order transfer function. Consider then the standard second order transfer function for a linear time invariant continuous-time system (2.5):

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.5}$$

It has the poles at (2.6):

[2] Practical control systems responses often reduce to first or second order like responses
[3] If the final steady-state value of the response differs from unity, then it is common to use the maximum percentage overshoot

$$s_{1,2} = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} \tag{2.6}$$

where ζ is the damping ratio and ω_n is the natural frequency. Closed-loop poles are characterized by parameters ζ and ω_n. Parameter ζ influences the relative damping of the response (overshoot), and ω_n influences the response speed (settling time). It is also known that by increasing ω_n, the magnitude of the control signal is increased. Therefore, we cannot increase the speed of the response without limit. In summary, the damping ratio and the natural frequency are directly related to the transient response characteristics.

Observe that not all specifications necessarily apply to any given system and they can be easily accommodated to other inputs (such as pulse or ramps) rather than unit-step.

The steady-state performance of a stable control system is generally judged by the steady-state error due to different system inputs. Depending on the type of system and input, different steady-state errors apply. A compromise between steady-state accuracy and transient response characteristics is always required.

Closed-loop system performance evaluation

In classic control theory, several properties are used to evaluate the performance of closed-loop systems. The primary evaluation is concerned with meeting the closed-loop response performance specifications and stability. Beyond these requirements, looking at the closed-loop response, since controller designs attempts to minimize the *system error* to certain anticipated inputs or perturbations, traditional performance criteria focus on the system error. The system error is defined as the difference between the desired response of the system and the actual response of the system. Figure 2.15 illustrates these concepts.

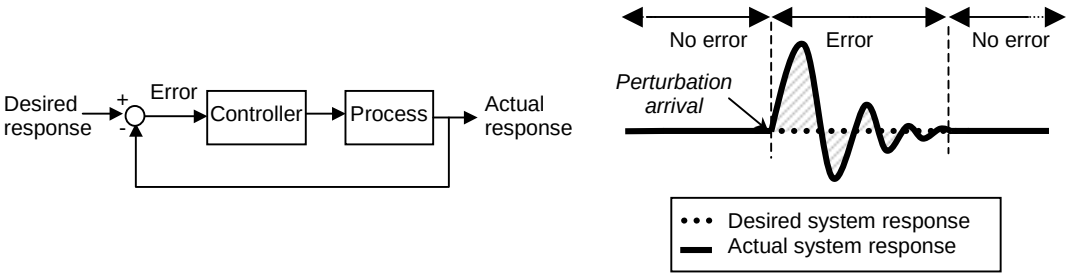


Figure 2.15. Right - Closed-loop system. Left - System error (shaded area)

The usual criteria used to evaluate (and design) controllers that give quantitative measures of closed-loop system responses in terms of errors are the performance criteria ISE (Integral of Square Error), ITSE (Integral of Time-weighted Square Error), IAE (Integral of the Absolute Error) or ITAE (Integral of Time-weighted Absolute Error) [DOR95], defined as follows (where e(t) denotes the closed-loop system error):

- Integral of the Absolute value of the Error (IAE): $IAE = \int_0^\infty |e(t)|dt$
- Integral of Square Error (ISE): $ISE = \int_0^\infty e^2(t)dt$

- Integral of Time-weighted Absolute Error (ITAE): $ITAE = \int_0^\infty t|e(t)|dt$
- Integral of Time-weighted Square Error (ITSE): $ITSE = \int_0^\infty te^2(t)dt$

All performance criteria measure the system error, but in different ways (absolute or square error and either time-weighted or not). Criteria that weight errors with time penalize later errors (steady-state errors) more heavily and discount the transient response errors, whereas the other weights all errors equally. Depending on the application and on what the performance evaluation should focus on, one of the previous criteria is chosen (see for example [LIA02]).

Classic control problems

Control problems can broadly speaking be classified in *regulation problems* and *command signal following* (tracking). The major issue in regulation problems is to react to disturbances or perturbations (that can be modelled as impulses that occur irregularly and so widely spread that the system settles between impulses). The major issue in command signal following is to track changes of the input signals (which can be modelled using the standard step, ramp or sinusoidal inputs).

2.2.2 Computer control

When a closed-loop is closed with a digital computer, i.e., *computer-controlled systems*, models and/or designs must be transferred to digital form (discrete-time domain). A computer-controlled system can be described schematically as in Figure 2.16.

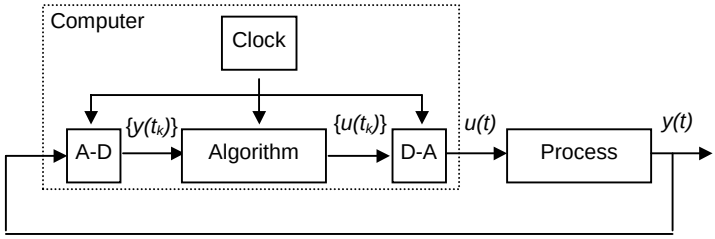


Figure 2.16. Diagram of a computer-controlled system

The output of the process, y(t) is a continuous-time signal. The output is converted to digital form by the AD converter. The conversion is done at the sampling instants, t_k. The time between successive sampling instants is called the sampling period and is denoted by h. The computer interprets the converted signal, $\{y(t_k)\}$, as a sequence of numbers, processes the measurements using an algorithm, and gives a new sequence of numbers, $\{u(t_k)\}$. This sequence, (called control signal or process input) is converted to an analogue signal, u(t), by a DA converter. A real-time clock in the computer synchronizes the events. Therefore, a computer-controlled system contains both continuous-time signals and sampled, or discrete-time signals.

When transferring models and/or designs to digital form, two different design approaches can be used: *discrete-time* design or *discretization of a continuous-time* design. In both cases the interface to the process consists of AD and DA converters. The AD converter acts as a sampler that returns a snapshot value of a continuous-time signal, and the DA converter acts as a hold circuit that takes a discrete-time signal and converts it into a continuous-time signal. Usually, zero-order hold is used, in which case the resulting continuous-time signal is piecewise constant between successive DA conversions.

Discrete-time (or sampled) control theory only considers the system through the values of the system inputs and outputs at the sampling instants (from the point of view of the computer). To do this, a sampled version of the continuous system model is derived. Usually, strictly periodic sampling is assumed. By doing this, well-known discrete-time system descriptions are obtained [AST97] and a wide range of discrete-time controller design methods (pole placement design, linear quadratic design, or model predictive control) can be applied in order to obtain the desired discrete-time controller.

Discretization of continuous-time design means to design the controller in the continuous-time domain, and then to approximate this design by an implementation (computer-based controller) through fast sampling. Using this approach it is not necessary to employ any special sampled control design theory. The price that one pays for this is higher requirements on fast sampling, because in the first approximation, the faster the sampling, the better the match between the discretization and the continuous-time system.

The sampling period for discrete-time control designs, beyond conforming to the Shannon theorem, can be chosen following one of various so-called rules-of-thumb, depending on the desired performance of the closed-loop system and the dynamics of the process to be controlled. An accepted rule-of-thumb is that the sampling frequency should be 4 to 20 times the system's cut-off frequency (which is usually approximated by the system natural frequency ω_n) [AST97]. This gives the possibility of choosing relatively long sampling intervals, compared to what is used when discretization-based design is used. We further discuss this issue later in this section.

In the end, in both cases, the discrete-time controller is a control computation algorithm to be executed at every *sampling period* h. In addition, the controller can be designed to account for a *time delay* τ. This time delay, traditionally assumed to be constant [AST97], would correspond to the execution time of the controller, given that it has to be computer implemented. It has to be pointed out that the resulting controller is characterized by several design parameters that are highly dependent on the constant sampling period (and constant time delay, if any) assumed at the design stage.

Examples

In the rest of this thesis we will use two examples to illustrate all the concepts we introduce:

- *DC (Direct Current) servo* problem, controlled by a discretization of a continuous-time designed PID (Proportional, Integral and Derivative) controller
- *Inverted pendulum* problem, controlled by a discrete-time state feedback controller (SFC) obtained using pole placement observer design.

We have chosen these two examples for the following reasons:

a) The control approach: the DC servo problem we present is based on input-output models while the inverted pendulum is based on state-space models.

b) The computer control approach: the state feedback controller obtained by pole placement observer design is designed in the discrete-time domain while the PID controller is obtained through discretization of a continuous-time design.

c) The controller design method we use: nowadays, PID controller design is widely used in many practical control problems [AST97]. Therefore, PID controllers are one of the most popular types of controllers. On the other hand, state-space controller design is considered the most modern approach for the design and synthesis of discrete-time systems [PHI95]. And specifically, pole placement observer design is a realistic approach (in the sense of state estimation [PHI95]) based on estimating the present condition of a system, and using this additional information to achieve better control.

d) The control problem we deal with: The DC servo problem falls into the tracking type of problems while the inverted pendulum, which is one of the most demanding benchmarks for the control community, falls into the regulation type.

Both examples are considered *standard processes* [AST97] for computer control. Although in this section we introduce these two control problems, which will be used throughout this work to illustrate the different concepts we introduce, usually we will focus on one of them (because the results we present are not application specific, being applicable to both examples). Only when the concepts we explain require comparisons of both examples will we use both.

Example 1

In the servo problem, the major goal is to follow the command signal, which in this case we model as a squared pulse signal. Consider the PID control of a DC servo described by the following continuous-time transfer function (2.7):

$$G(s) = \frac{1000}{s(0.5 \cdot s + 1)} \tag{2.7}$$

Following the specified requirements in order to have a percentage overshoot of less than 15%, we heuristically tune the PID parameters to the following values $K_p = 1.8$, $K_i = 0.1$ and $K_d = 0.09$ (see [AST97] for PID controller design). Once the PID controller has been designed in the continuous-time domain and with the appropriate sampling period (h=2ms, see [AST97] for sampling period selection), we obtained its discrete approximation by approximating the integral part by a forward approximation and the derivative part by taking backward differences (see [AST97] for PID discrete approximation).

The resulting discrete approximation can be seen in Figure 2.17, where e(t) is the error (difference between r(t), the reference signal (command signal), and y(t), the actual response (process output or measured variable)) , K_p, K_i, and K_d are the PID primary parameters, h is the sampling period, u(t) is the control signal (process input) and k denotes the k^{th} PID execution.

$$
\begin{aligned}
e(kh) &= r(kh) - y(kh) \\
p(kh) &= K_p e(kh) \\
i(kh) &= i(kh-h) + \frac{K_i h}{2}(e(kh) + e(kh-h)) \\
d(kh) &= \frac{K_d}{h}(e(kh) - e(kh-h)) \\
u(kh) &= p(kh) + i(kh) + d(kh)
\end{aligned}
$$

Figure 2.17. PID discretization

We implemented the obtained controller as a control task named *task$_{PID}$* using a single task with period T_{PID}=2ms and estimated worst case execution time C_{PID}=0.2ms (in this case, the task deadline, D_{PID}, is assumed equal to period). In Figure 2.18 we show the task *pseudo code*: at the sampling instant t_k after reading the reference signal $r(t_k)$ and the actual response $y(t_k)$, we calculate the control signal $u(t_k)$ from the error $e(t_k)$. Note that the sampling period h is a constant parameter used in the calculations. For a more detailed code, see Section A1 in Appendix A.

```
task_PID
{
        read_inputs (y(t_k), r(t_k));
        e(t_k)= r(t_k) - y(t_k);
        u(t_k) = calculate_output (p_k(e(t_k)), i_k(h, e(t_k)), d_k(h, e(t_k)));
        write_output (u(t_k));
}
```

Figure 2.18. PID controller implementation in a control task

We show in Figure 2.19 the response of the DC servo controlled by *task$_{PID}$* executing in isolation on a single processor given a squared pulse signal as input. Due to the fact that PID controller design problem is beyond the scope of this work, we assume that our PID is good enough for illustrative purposes. Note that the response we obtain fulfils the overshot performance requirement, as illustrated in Figure 2.19. That is, the response tracks the reference signal without crossing the marked *maximum overshoot*.

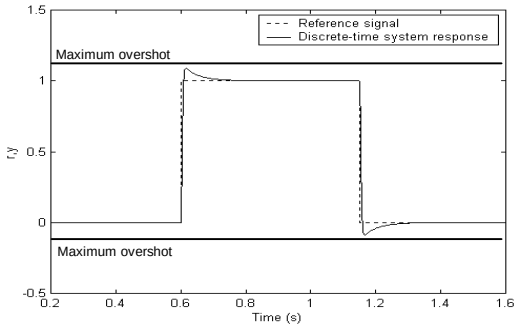


Figure 2.19. DC servo system response showing the reference signal tracking

Example 2

The inverted pendulum control problem can be stated as follows: the inverted pendulum (of length l and mass m) can only swing in a vertical plane parallel to the direction of the cart (of mass M), where g is gravity. In the presence of a perturbation, to balance the pendulum, the cart is pushed back and forth on a track of limited length. Balancing fails when the inclination of the pendulum exceeds preset limits, or when the cart hits the stops at the end of the track. The aim is to find a controller to balance the inverted pendulum, preventing it from failing, and to bring the cart to the centre of the track. A sketch of an inverted pendulum mounted on a motor driven cart is shown in Figure 2.20.

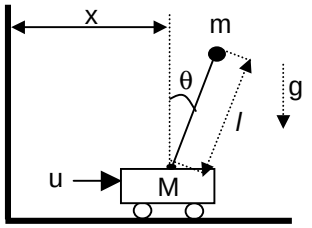


Figure 2.20. Inverted pendulum

The state of the inverted pendulum on a cart is described by the cart position (x), its velocity (v), the pendulum angle (θ) and the angular velocity (ω). The force provided to the cart (u) is the controlling action calculated according to the actual angle and position (y, controlled variables). A linear time invariant state-space model of the inverted pendulum, used for the control design is (2.8) (where for the example M=2kg, m=0.1kg, l=0.5m and g=9.81m/s^2):

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)\cdot g}{M\cdot l} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{m\cdot g}{M} & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \omega \\ x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{M\cdot l} \\ 0 \\ \frac{1}{M} \end{bmatrix} u(t) \qquad y(t) = \begin{bmatrix} \theta \\ x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \omega \\ x \\ v \end{bmatrix} \tag{2.8}$$

For the sake of simplicity, we will focus only on the angle (θ). Therefore, the goal of our controller is to maintain the desired vertical position of the inverted pendulum at all times. It can be seen that the open-loop system is unstable, i.e., the open-loop system poles do not lie within the unit circle in the z-plane.

To stabilize the inverted pendulum, we close the loop by designing a state feedback controller using pole placement observer design [AST97], taking into account the performance requirement: to recover from a perturbation (modelled by a pulse) in less than two seconds. In control terms, the settling time is 2s. By

a) setting ω_n=6rad/s and ζ=0.707 for the closed-loop dominant poles[4],

[4] Apart from the pair of dominant poles ($d_{1,2}$) the remaining three poles (notice that the time delay adds an extra state variable which represents the past value of the control signal) apart from the observer poles are set to real(d_1)/2, real(d_1)/3 and real(d_1)/4. The observer poles are set to the closed-loop poles divided by 10, in order to be faster than the closed-loop poles (see [AST97])

b) with the appropriate sampling period h (80ms) and taking into account an estimated time delay τ of 20ms, and

c) through pole placement observer design (using Ackerman formula, see [AST97])

we obtain a state feedback controller that locates the closed-loop poles in such a way that the specified closed-loop requirements are met.

We implemented the state feedback controller in a control task named $task_{sfc}$ (with period T_{sfc} = 80ms and estimated worst case execution time C_{sfc} = 20ms, assuming as before deadline D_{sfc} equal to period). Figure 2.21 shows the code of the task: at the sampling instant t_k, after reading the controlled variable (angle) $y(t_k)$ and the reference signal $r(t_k)$, we calculated the control signal $u(t_k)$ according to the current state $x(t_k)$, the reference signal $r(t_k)$ and the gain matrix $L(h,\tau)$. Afterwards, the state $x(t_{k+1})$ is updated according to the actual state, output, input, and closed-loop matrices ($\Phi(h,\tau)$, $\Gamma(h,\tau)$ and C) and observer matrix $K(h,\tau)$. Almost all the matrices depend on the sampling *period* h and time delay τ, which are constant parameters of the controller algorithm, specified at the design stage. Therefore, h and τ are supposed to remain constant at run time (see Section A3 in Appendix A for a more detailed code).

```
task_sfc
{   read_input (y(t_k), r(t_k));
    u(t_k) = calculate_output (x(t_k), -L(h,τ), r(t_k));
    write_output (u(t_k));
    x(t_k+1) = update_state (x(t_k), Φ(h, τ), Γ(h, τ), C, u(t_k), K(h, τ), y(t_k ));
}
```

Figure 2.21. State feedback controller implementation in a control task

The response of the task $task_{sfc}$ executing in isolation on a single processor in the presence of a perturbation (modelled as a discrete pulse) can be seen in Figure 2.22. Since it is beyond the scope of this work to specifically discuss state feedback controller design with pole placement techniques, we assume that our controller is good enough for illustrative purposes. Note that the performance requirement is met; that is, the pendulum recovers from the perturbation in less than two seconds.

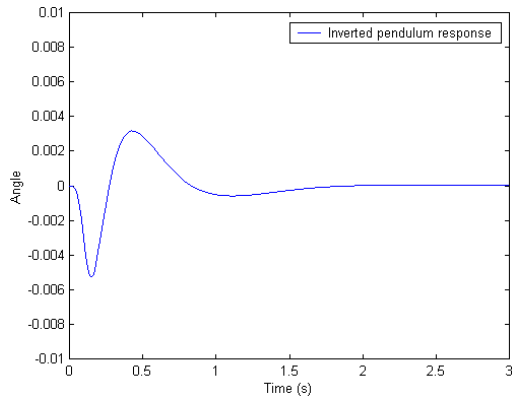


Figure 2.22. Inverted pendulum system response

Sampling period and time delay effects on the system response

Previously we concluded that a discrete-time controller depends on the constant sampling period h and constant time delay τ assumed in the controller design. We have also explained that the stability and transient response characteristics of a closed-loop system can be determined by the location of its closed-loop poles (specifically, by its dominant poles, recall Equations (2.5) and (2.6)). The equivalent *z*-plane poles (in the discrete-time domain) occur at (2.9).

$$z_{1,2} = e^{sh}\Big|_{s_{1,2}} \tag{2.9}$$

where h is the sampling period. Note that the location of the closed-loop poles for discrete-time control systems depends on the sampling period. Consequently, the sampling period selection is of importance with respect to the closed-loop stability and system response. The sampling period selection is described next.

According to [PHI95] it is reasonable to choose at least N > 10 (preferable N=25 to 75) and $\omega_n h < 1$, where N gives the number of samples per period of dominating mode of the closed-loop system (2.10), h is the sampling period, ω_n is the natural frequency and ζ is the damping ratio.

$$N = \frac{2\pi}{\omega_n h\sqrt{1-\zeta^2}} \tag{2.10}$$

In [AST97] it is suggested that the sampling period must be chosen to give $\omega_n h$=0.1 to 0.6. In [PHI95] it is also suggested that for discrete-time control systems, pole locations should be placed in the vicinity of *z*=1 if the system constraints allow a sufficiently high sampling rate to be chosen.

Closed-loop specification (ω_n=4 rad/s and ζ=0.707)			
h (ms)	**Dominant poles ($d_{1,2}$)**	**N**	**$\omega_n h$**
30	0.9154 ± 0.0779i	74	0.12
40	0.8873 ± 0.1008i	55	0.16
50	0.8595 ± 0.1224i	44	0.20
60	0.8318 ± 0.1426i	37	0.24
70	0.8044 ± 0.1614i	31	0.28
80	0.7772 ± 0.1790i	27	0.32
90	0.7503 ± 0.1953i	24	0.36
100	0.7237 ± 0.2104i	22	0.40
110	0.6975 ± 0.2243i	20	0.44
120	0.6716 ± 0.2372i	18	0.48
130	0.6461 ± 0.2489i	17	0.52
140	0.6210 ± 0.2596i	16	0.56
150	0.5963 ± 0.2694i	15	0.60

Table 2.4. Closed-loop system properties

For example, given as performance requirements ω_n=4rad/s (which gives a reasonably small settling time of approx. 2s.) and ζ=0.707 (which gives reasonable damping while minimizing the settling time), the previous equations give us a sampling period choice of

h={30 to 150} ms[5]. Table 2.4 shows for each sampling period the location of the pair of closed-loop dominant poles, the N parameter and the relation $\omega_n h$. Table 2.4 also indicates that any sampling period from 30 to 150 ms will fulfil the sampling period selection requirements referred above while meeting the performance requirements. It has to be pointed out that other closed-loop poles (apart from the pair of dominant poles) of the closed-loop system (if any) should be placed on the left of the pair of dominant poles, having then little effect on the system response [AST97].

Observe from Equation (2.9) that the location of the closed-loop poles in the z-plane does not depend on the time delay. However, since a time delay in a controller implies that each control signal is sent after this time delay, the effect on the closed-loop system response are to delay the response, which, for example, implies that the response settles later. In the following, we show the effects of the previous different poles locations and sampling periods as well as different time delays on the closed-loop system response. We use the inverted pendulum controlled by control task $task_{sfc}$ (see Figure 2.21) (but with the state feedback controller designed with ω_n=4rad/s and ζ=0.707 for the closed-loop dominant poles) with different sampling periods and time delays. First of all, we look at the inverted pendulum responses for each acceptable sampling period h=30 to 150ms (acceptable in term of fulfilling the restrictions set on parameters N and product $\omega_n h$). For this set-up, a constant time delay of τ=1ms is assumed. With these specifications, for each sampling period we obtain a specific controller, which gives a different system response in terms of performance. Figure 2.23 shows each particular system response for selected h=40, 60, 80, 100, 120 and 140ms.

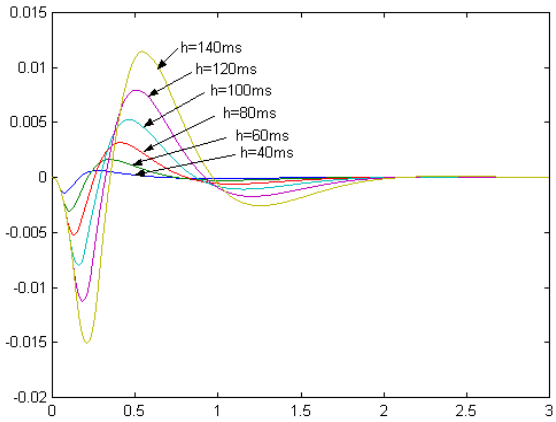


Figure 2.23. Inverted pendulum system response for different sampling periods

Looking at the different responses, we see that the performance of the transient system response (rise time, overshoot, settling time, etc) clearly depends on the sampling period, given the same performance requirements (ω_n=4rad/s and ζ=0.707). Therefore, when selecting the sampling period, we have to take into account its effects on the system response (provided system stability is also guaranteed).

[5] Notice that the theoretical longest sampling period (derived from Shanon's theorem) is $h=2\pi/\omega_s$, where $\omega_s=2\omega_N$, where the Nyquist frequency ω_N can be approximated by ω_n. Therefore, the theoretical longest sampling period is approximately 800 ms with poles at $-0.0664 \pm 0.0801i$.

Focusing on the effects of time delays, given a sampling period, in Figure 2.24, we show different inverted pendulum responses depending on the time delay, with the same specification as before, with h=80 ms. Time delays vary from τ=1ms to 80ms (in a 20ms granularity).

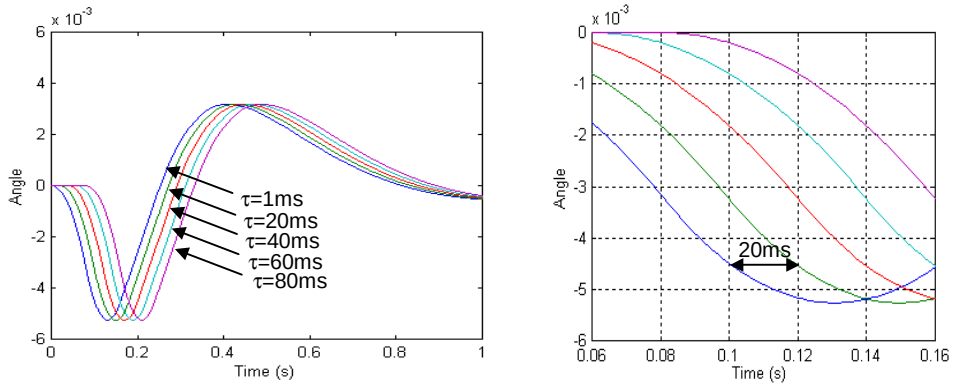


Figure 2.24. Inverted pendulum system response for different time delays (complete view - left- and detailed view –right.)

We can see in Figure 2.24 (left) that the response of the inverted pendulum is delayed proportionally to the time delay assumed in the controlled design stage. It can also be seen in the detailed view (Figure 2.24 - right) that each response is delayed according to the time delay we assumed in the controller.

Summary

In the preceding sections we have described fundamental concepts of control systems analysis and design: given a plant, we must first obtain its mathematical model. Then, using the mathematical models obtained, we design a controller such that the closed-loop system will satisfy the given specifications (stability and specific response characteristics). It has been shown that the sampling period and time delay that we use in the design of discrete-time controllers is of prime importance, because the performance of the closed-loop systems

2.3 State of the art

In this section, we give a brief survey of the so-called integrated real-time control systems approaches. In [ÅRZ99] there is a good, extended account of the state of the art in the field of control and real-time systems. Here, we survey existing relevant work in this area that relates to the results presented in this thesis. We categorize the existing work of this area in four major tendencies (without clear demarcation): feedback control real-time scheduling, control approaches, scheduling approaches, and integrated control and scheduling.

2.3.1 Feedback control real-time scheduling

The main idea behind these approaches is to treat the scheduling problem as a feedback control problem. Scheduling algorithms traditionally work in open-loop. That is, once schedules (or scheduling rules) are created, they are not adjusted based on continuous

feedback. While open-loop scheduling algorithms can perform well in static or dynamic systems in which the workloads can be accurately modelled, they can perform poorly in unpredictable dynamic systems. For these latter cases, feedback control real-time scheduling aims to look at the scheduling problem as a feedback control problem, defining error terms for schedules, monitoring the error, and continuously adjusting the schedules in order to maintain stable performance. The main goal of these approaches is to present solutions for a computer-based problem such as task scheduling using control techniques instead of using the traditional scheduling techniques, which are computer-based.

Important contributions in this area can be found in [STA99], [LU99], and [LU00]. In [STAN99] a general architecture for feedback control real-time scheduling and a new real-time scheduling algorithm called Feedback Control Earliest Deadline First (FC-EDF) is presented. The new scheduling algorithm has proved to be robust in overload situations. Proportional, Integral and Derivative (PID) control loop is used. In [LU99], a performance evaluation of the FC-EDF using deadlines-based metrics is presented. [LU00] presents a control-theory-based framework that enables system designers to specify the performance specifications (stability, transient and steady-state performance), and apply existing control methods to analytically design an adaptive real-time system to achieve the specifications.

Although in our approach we take advantage of the feedback behind control systems, we do not treat scheduling as a feedback control problem. We use the feedback paradigm to adjust control tasks periods according to the controlled system performance in the Quality-of-Control (QoC) scheduling approach we present.

2.3.2 Control approaches

These approaches have the common feature of interpreting the control task scheduling effects on the system performance from a control viewpoint. Firstly, they detect the effects of jitters and other computer implementation problems in the controlled system performance. Afterwards, they model, study and propose control design based solutions, using different techniques in order to compensate for the introduced degradation.

For example, in [TÖR95] many fundamental issues in implementing real-time control applications in distributed control systems are discussed. [TÖR97] derives timing requirements and constraints for implementation of multirate control applications. [SHI96] and [WIT95] give interpretations of time varying delays as computer induced disturbances. Specifically, [WIT95] investigated the effects of time varying delays on control system stability and performance. It is shown that time varying delays can cause instability and diminish the controlled systems performance. [WIT98] have treated deficiencies in the computer system implementation of the control system with respect to time-variations and time-restrictions in control-networked systems. In a similar way, [NIL98a] discusses some problems that can be found in real-time control. In [SCH01] an optimal controller design for sampled data control is suggested where the sampling time varies due to the control task scheduling. [WIT01] study sampled-induced delays in synchronous multirate control systems. In [NIL98b] a stochastic analysis and control approach of real-time systems with random time delays is presented. As pointed out in [MIT01], real-time (time delays) issues and feedback in communications are questions that have received inadequate attention. In

[ZHA01], a stability analysis for networked control systems is presented where time delays are assumed to be constant.

Although particular solutions have been presented to deal with irregularly sampled discrete-time control systems [WIT80], [ALB90], [ALB99], [ÅRZ00] and [SCH01], or systems with varying time delays (which may include communication-induced delays) [CHA95] [SHI96], [NIL98], [WIT98], [MOT00], no integrated controller design method has provided a solution for both problems, as we do with the compensation approach.

2.3.3 Scheduling approaches

As jitter is an inherent scheduling problem, several works have treated the jitter problem from a scheduling viewpoint. That is, by designing specific real-time purpose task models and algorithms, they try to minimize the jitter itself.

[STA94] has studied jitter minimization in the context of providing end-to-end timing guaranties in distributed real-time systems. In [BAR97] a model for periodic tasks is proposed that explicitly incorporates jitter. Feasibility analysis of systems of such tasks is studied in the context of dynamic-priority, pre-emptive, uniprocessor scheduling. In [BAR99] a formal quantitative model for output jitter for periodic task scheduling is proposed.

With our approach, we do not aim to minimize scheduling inherent jitters. We accept jitters by characterizing them and designing controllers that take into account the jitters that control tasks are subject to (note that even though jitter is minimized in previous works, it continues to exist, thus causing the problems that we solve by accepting jitters in the control design).

2.3.4 Control and scheduling integration

The goal of these integrated approaches is to combine control systems and real-time systems theory is such a way that they achieve control performance optimization using different techniques.

Focusing on the jitter problem itself, some works have presented specific scheduling-based solutions when the scheduled tasks are control tasks [KIM98, CRE99, BAL02, ALB00, CER99]. A new task scheduling problem with a suitable control performance index that includes scheduling and control perspectives is formulated in [KIM98]. [CRE99] propose a method to determine the minimum interval where the control action or the data acquisition has to be allocated avoiding the jitter effects on control tasks. [BAL02] extend the previous results providing also new schedulability analysis. [ALB00] suggest a method to reduce output jitter variability and its degrading effects on control performance by splitting control tasks with a new priority assignment. In a similar way, [CER99] show that by scheduling two parts of a control algorithm as separate tasks the computational delay can be significantly reduced and, as a consequence, the system performance is improved.

The optimization of control system performance subject to schedulability has been also treated in [SET96] and [REH00]. In [SET96], at the design stage, tasks frequencies are

optimized in order to make all tasks schedulable and to enhance control system performance. In [REH00] an offline scheduling method is proposed based on optimal control theory.

Runtime control performance and schedulability optimization is treated in [CAC00], [BUT98], [SHA00], [PAL00] and [BUT02]. In these approaches, the major goal has been to adapt properties of the schedule at runtime, modifying the scheduling algorithm in order to improve schedulability and optimize the control performance. In [CAC00], the main idea is that tasks' computation times are allowed to range from average to worst case computation times and periods are adjusted at runtime to optimize control performance and enhance schedulability by using server approaches. In [BUT98], an elastic task model for control tasks is presented. Using this task model, periodic tasks can intentionally change their execution rate at runtime to provide a different quality of service, and the other tasks can automatically adapt their periods to keep the system under-loaded. In [BUT02], a smooth rate adaptation through impedance control is presented for the elastic model. In [SHA00], an integration of load driven online scheduling with direct digital design to optimize control performance as a function of varying workload is presented. An integrated real-time control design approach is also presented in [PAL00]. In [PAL02], synthesis of real-time embedded controllers taking into account constraints derived form the implementation platform is addressed.

In all the previous approaches, the problem posed by jitters for control task scheduling, controlled system degradation, and the problems posed by fixed timing constraints for control task scheduling, over constrained schedules, are not addressed. We address them by combining the compensation approach we present along with the flexible timing constraints.

In summary, existing approaches have either studied the deleterious effects of computer implementations on control system performance and stability, or have attempted quite successfully to interpret such effects from a control viewpoint or have tried to optimize control system performance through the optimized selection of periods, by designing special purpose task models, or scheduling algorithms. However, practical problems posed by scheduling inherent jitter in control tasks have not been formally addressed, and both control performance and systems schedulability have not been, in any of the previous works, jointly improved.

It is important to stress that several works have presented computer-aided tools for the analysis and design of real-time control systems. In [MAR00a] a Matlab™ based framework for the design of distributed control systems was presented. It covers all the required steps (from specification to system maintenance) for the analysis and design of real-time control systems, with special emphasis on distributed architectures based on fieldbus systems (as shown in [MAR99]). In [EKER99] a Matlab™ toolbox for real-time and control systems co-design was presented. The basic idea is to simulate a real-time kernel in parallel with continuous plant dynamics. The toolbox allows the user to explore the timely behaviour of control algorithms, and to study the interaction between the control tasks and the scheduler. Most of the simulations presented in this thesis have been carried out using the latter simulator.

2.4 Summary

In this chapter we have reviewed key concepts of real-time and control systems. Some formal definitions have been given. Note however, that we have focused on specific concepts that are relevant to the work we present in this thesis. Also, a brief survey of the state of the art on the field of real-time control systems has been presented, pointing out what make our methods and solutions different and better.

Chapter 3

Control impact on schedulability

In this chapter we study the *timing assumptions* made in discrete-time control theory. When transferring continuous-time control models and/or designs to a digital form, a specific timing is assumed. Consequently, the computer-controlled system implementation must preserve this timing in order to achieve the specified closed-loop characteristics (stability and desired controlled system response). From these timing assumptions made by discrete-time control theory, we derive the *timing requirements* that the implementation must deterministically fulfil [AST97]. However, we show in this chapter that the timing requirements imposed by discrete-time control theory on a closed-loop implementation are not realistic regarding the timing behaviour of possible implementations. Moreover, these timing requirements derived from discrete-time control theory assumptions pose real-time demands on a closed-loop implementation. As pointed out in [KOP97], the most stringent timing constraints for real-time systems have their origin in the timing requirements of closed-loops. In this chapter, we also show how to express these real-time demands using traditional timing constraints for control tasks, such as periods and deadlines. Furthermore, we discuss the different ways of implementing closed-loops in real-time systems using periodic tasks.

We finally show that when using traditional real-time task models to express discrete-time control theory timing requirements, the system schedulability decreases. That is, scheduling algorithms, when trying to fulfil the timing requirements posed by closed-loops, over-constrain the schedule (because task models are based on fixed timing constraints that lack flexibility) resulting in poor system schedulability. Therefore, the implementation of discrete-time controllers in a multitasking real-time system impairs schedulability.

The work explained in this chapter has been partially presented in [MAR00b].

3.1 Discrete-time control theory timing analysis

The general functionality of a computer-controlled system was described in Section 2.2.2. The functional scheme of a closed-loop system can be split into three main activities: *sampling*, *control algorithm computation* and *actuation*. Discrete-time control theory regards these three main activities with specific timing. That is, models and methods used in discrete-time control theory implicitly impose the timing that sampling, control computation, and actuation must have in the computer implementation.

Next, we discuss the timing assumptions which discrete-time control theory relies on and we show that the timing requirements that the closed-loop implementation must meet (to fulfil the discrete-time control theory timing assumptions) are not realistic for current computing platforms.

3.1.1 Closed-loop timing assumptions

From classical discrete-time control theory, we can derive the following timing assumptions about the three main parts of a closed-loop:

- The *input data collection or sampling* is performed at the *sampling instants,* t_k, i.e., times when the measured signals (*inputs,* $y(t_k)$) are converted to digital form (through the AD converter)
- The *output data transmission or actuation* is performed at the *actuation instants*, i.e., times when the control signals (*outputs*, $u(t_k)$) are converted to a continuous form (through the DA converter)
- *Sampling* is performed at equidistant time instants given by the *sampling period*, h
- The *time delay*, τ, i.e., time elapsed between *related*[1] sampling and actuation instants, is *constant* (either *instantaneous*[2] - or not, depending on how the controller was designed)
- The *control computation*, i.e., the algorithm implementing the controller design, calculates the output as soon as the input (sample) is available.
- The events are *synchronized* by the *real-time clock* of the computer

Although these assumptions are the basis for discrete-time control theory, in practice it is not possible to keep all of them in a computer implementation (as we further discuss in the next Section). For example, it is clear that computations take time, therefore the time delay, which includes the control computation execution, can not be *instantaneous*. The implementation approach behind assuming an *instantaneous* time delay is to build the system in such a way that the time delay is minimized in order to ignore it in the controller design. However, this does not mean that the control computation execution time is zero. It means that the control computation execution time is not relevant to the controlling purposes if compared to the closed-loop system dynamics.

However, due to the fact that computations take time, a more realistic approach is to assume a time delay, $\tau > 0$, which includes the control computation worst-case execution time, which is then taken into account in the controller design. Discrete-time control theory provides several approaches for dealing with this delay. The most common approach is to assume that the delay is constant. However, depending on the control model we assume when designing a discrete-time controller (see [AST97] or [VAC95] for further details) the constant time delay has different semantics.

---

[1] *Related* sampling and actuation instants refers to the sampling and actuation actions performed at each execution of a closed-loop.

[2] Recall that the idea of using digital computers as components in control systems started around 1950 [AST97]. The notion of *instantaneousness* comes from the theory of linear time-invariant continuous-time systems, from which discrete-time systems theory emerged. In continuous-time systems, analogue controllers are instantaneous. As a consequence, some of the models and methods used in discrete-time theory, coming from continuous-time theory, maintain the instantaneous assumption.

In the following, we use time diagrams to discuss and illustrate the exact closed-loop timing assumptions that are implicit in the three major and most commonly used classical discrete-time system models for discrete-time controller design, where h denotes the sampling period and τ a constant delay.

Regularly sampled discrete-time system model

This model is the most commonly used for discrete-time controller design and its timing assumptions emerge from sampling continuous-time systems. The timing assumptions behind this model are:

- sampling is performed at equidistant time instants given by the *sampling period,* h
- time delay is zero (τ=0)
- actuation is performed at the same sampling time instants.

Notice that this is the most unrealistic case. The control model does not take into account any time delay. In practice, this means that the control computation execution time is assumed to be short enough to not affect the controlled system response when sending the actuation after the control computation. In Figure 3.1 we illustrate the timing assumptions of this model. At each closed-loop execution, denoted by k-1, k, k+1, etc, sampling and actuation are performed, theoretically, at the same time instant every sampling period (h).

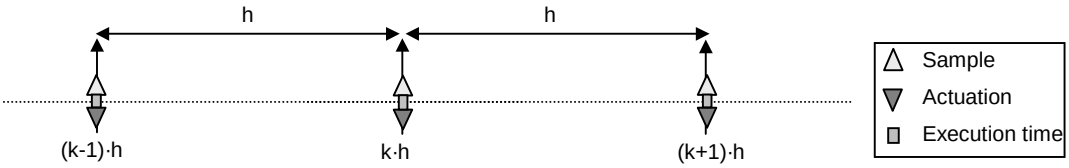


Figure 3.1. Timing assumptions of regularly sampled discrete-time system models

Regularly sampled discrete-time system model with constant time delay

This model is more realistic because it takes into account that the control computation takes time. This is represented by a constant time delay. The timing assumptions behind this model are:

- sampling is performed at equidistant time instants given by the *sampling period,* h
- time delay is constant and less than the sampling period ($0<\tau<h$)
- actuation is performed immediately after the time delay.

Notice that this model is the closest to reality. The control model accounts for a constant time delay, which is assumed to be equal to the control computation execution time. In Figure 3.2 we illustrate the timing assumptions of this model. At each closed-loop execution, denoted by k-1, k, k+1, etc, sampling and actuation are performed at the same time instants given by the sampling period (h), with an elapsed time between them equal to the control computation execution time given by the time delay (τ).

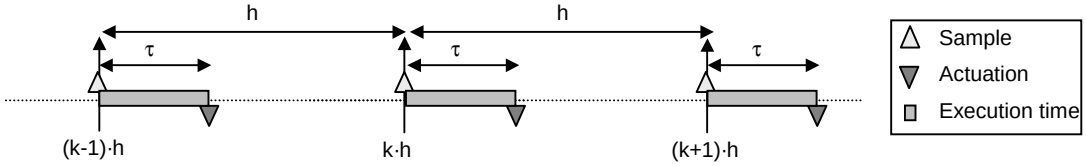


Figure 3.2. Timing assumptions of regularly sampled discrete-time system models with constant time delays

Notice that this model imposes a strict timing requirement on the implementation with respect to the control computation execution: at each closed-loop execution, the control computation execution time must always be the same and also be equal to the time delay (τ) assumed by the control model.

Regularly sampled discrete-time system model with actuation at the next sampling instant

This model assumes that the actuation instant of the k^{th} closed-loop execution occurs at the sampling instant of the $k+1^{th}$ closed-loop execution. The timing assumptions behind this model are:

- sampling is performed at equidistant time instants given by the *sampling period,* h
- time delay is constant and equal to the sampling period (τ=h)
- actuation is performed immediately after the time delay, that is, at the next sampling instant.

In Figure 3.3 we illustrate the timing assumptions of this model. At each closed-loop execution, denoted by k-1, k, k+1, etc, sampling and actuation are performed at the same time instants given by the sampling period h, with an elapsed time between them equal to the sampling period (τ=h).

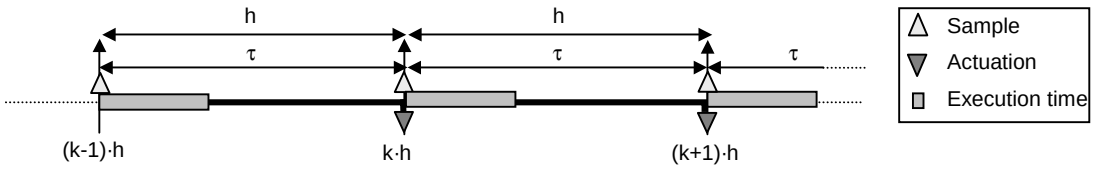


Figure 3.3. Timing assumptions of regularly sampled discrete-time system models with actuation at the next sampling instant

Summary

Taking into account the general description of computer-controlled system we gave in Section 2.2.2, Figure 3.4 shows the three different models in terms of the relation between controller inputs $y(t_k)$, which are the measured variables, controller outputs $u(t_k)$, which are the control signals, and sampling instants t_k:

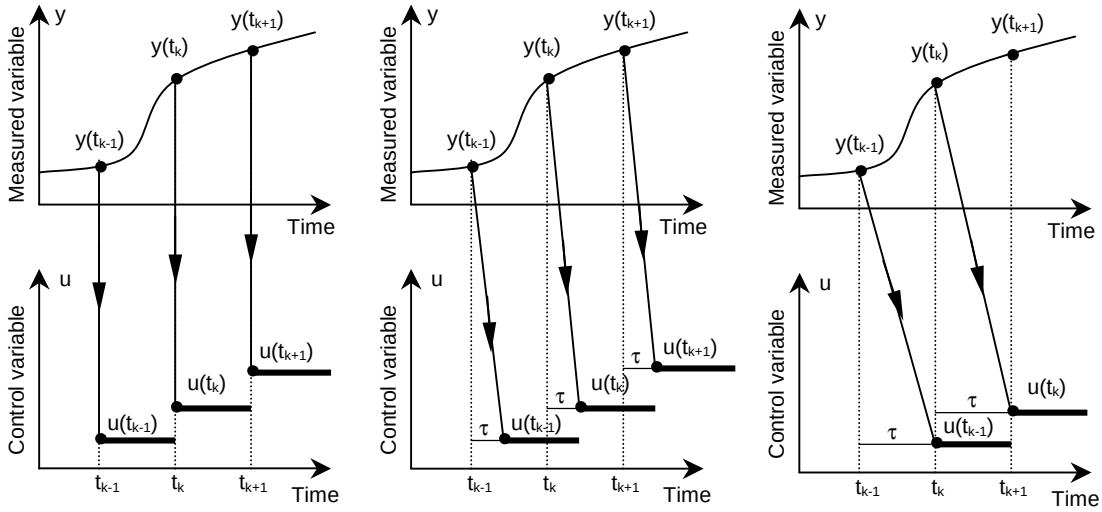


Figure 3.4. Input-output synchronization depending on the time delay: (left) $\tau = 0$, (middle) $0 < \tau < h$ and (right) $\tau = h$ (Notice that $t_{k+1}-t_k = h$)

These three models we analysed are the most common for the analysis and design of discrete-time control systems. From a control theory point of view, the main difference between them is given by the assumptions made on the time delay. It is known ([AST97] or [CER99]) that the shorter the delay, the better the control. However, depending on the system to be controlled and on the digital implementation platform characteristics, one model can be more suitable than another.

Looking at the system itself, if its dynamics and inertias are slow, any of the three models can be successfully applied, with no major difference. However, if the system is fast, the timing given by each model can be of importance, in terms of achieving the desired closed-loop system specifications with different levels of performance.

Looking at the three models from an implementation point of view, the following considerations must be evaluated. Although the disadvantage of the third model (Figure 3.4 right) is that control signals $u(t_k)$ are delayed unnecessarily, its implementation can be successfully achieved by hardware interrupts. The advantage of the second model (Figure 3.4 middle) is that it has a shorter time delay than the third model (Figure 3.4 right). However, due to the fact that its computer implementation is not usually done through hardware interrupts, depending on the computer implementation (depending on the programming and on the processing node mechanisms, e.g., scheduling), the assumed constant time delay can actually vary. The advantage of the first model (Figure 3.4 left) is that there is “no delay”. However, it is not realistic because there will always be a time delay when a control law is implemented using a computer.

Apart from the three models we analysed, there are other possible models that can include constant time delays such as system with time delays longer than the sampling period or systems with internal time delays apart from those with input (or output) delays. For further reading, see [AST97].

In the end, although different control models give different semantics to the time delay, the control timing assumptions behind discrete-time control system theory can be summarized as follows:

a) *Sampling* occurs at equidistant time instants given by the *sampling period*, h

b) Time delay, τ, is *constant*

c) *Actuation* occurs at the time delay completion

Note that from a control point of view, *sampling period* and *time delay* are the timing parameters of interest in the analysis and design of discrete-time controllers. This means that the existing well-known control models and methods used in discrete-time control theory are for systems with a constant sampling period (h) and a constant time delay (τ). That is, the (realistic) timing requirements posed by discrete-time control theory timing assumptions that a computer implementation must meet are:

1. Constant sampling period
2. Constant time delay

3.1.2 From theoretical timing to applied timing

In the previous section we have explained the timing assumptions for closed-loop systems that are implicit in discrete-time control theory and we have derived the timing requirements that an implementation must meet. However, we have not discussed the influence of any specific closed-loop architecture on the timing behaviour of an implementation. In this section, we give an overview of the real expected timing behaviour of closed-loops systems that are either implemented locally or distributed. Afterwards, we point out the problem that arise when mapping the theoretical timing assumed by discrete-time control theory to the real timing of the implementations.

The two main architectures for implementing closed-loops are the local closed-loop architecture and the distributed closed-loop architecture:

- *Local closed-loop*: the three main parts of a closed-loop (sampling, control computation and actuation) take place in the same processing node.
- *Distributed closed-loop:* the three main parts of a closed-loop (sampling, control computation and actuation) take place in different processing nodes and a communication network is used for transmission of signals between nodes.

Figure 3.5 shows the two different architectures. For the *local closed-loop* (Figure 3.5 left), the timing parameters that characterize the functionality of the system are the sampling period (h) and the time delay (τ_c), understood to be the control computation execution time. We regard the conversion operation delay for analogue-to-digital (AD) and digital-to-analogue (DA) to be negligible. If these operations are too long, they can be included for the timing analysis in the time delay τ_c. For the distributed closed-loop, the timing parameters that characterize the functionality of the system are the sampling period (h), the time delay (τ_c) understood to be the control computation execution time, and the network-induced delays, that is, the sampler to controller delay (τ_{sc}) and the controller to actuator delay (τ_{ca}).

As before, we regard the conversion operations to be negligible. If these operations are too long, they can be included for the timing analysis in the time delay τ_c. In Figure 3.5, the measured variables and the control signals are marked by y(t) and u(t) respectively.

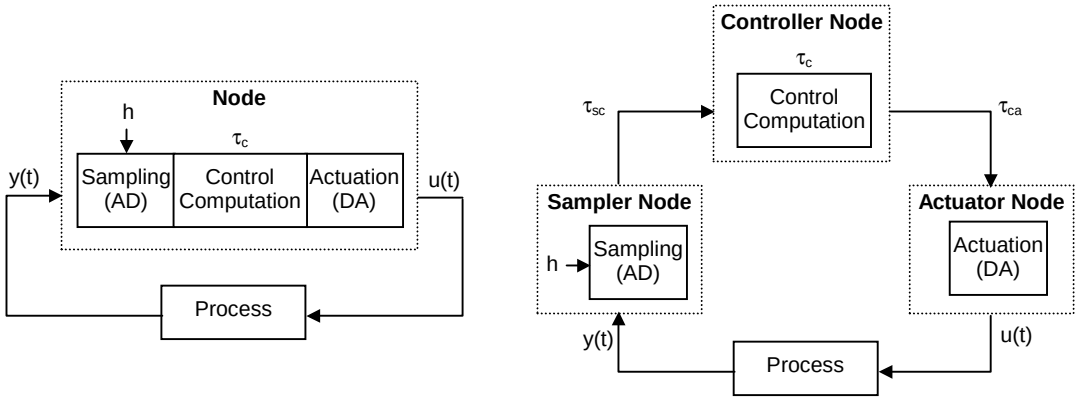


Figure 3.5. Local (left) vs. distributed (right) closed-loop system

If we take a closer look at the significant timing behaviour behind each architecture (see Figure 3.5), we conclude that the same timing problems arise when mapping the timing assumptions that discrete-time control theory requires in an implementation with the timing behaviour of each architecture.

Focusing on the local closed-loop architecture, the following functionality is expected (see [MAR00b] for further reading):

- The sampler samples the process to be controlled with a given sampling period h. The time of the k^{th} sampling is given by (3.1)

$$t_k = t_{k-1} + h \tag{3.1}$$

- The k^{th} control computation execution start time is given the sampling time t_k. The control computation introduces a delay (τ_c) in order to calculate the actuation signal(s). Notice that the delay τ_c may be different at each closed-loop execution. Finally the actuator performs the actuation at time given by (3.2)

$$a_k = t_k + \tau_c \tag{3.2}$$

Consequently, for these local closed-loop architectures, the time delay is given by τ_c.

Focusing on the distributed closed-loop architecture, the following functionality is expected (see [MAR01c] for further reading):

- The sampler samples the process to be controlled with a given sampling period h. The time of the k^{th} sampling is given by (3.3)

$$t_k = t_{k-1} + h \tag{3.3}$$

- When the data has been collected by the sampler it is forwarded to the controller, introducing a communication delay τ_{sc}. Notice that the delay τ_{sc} may be different at

each closed-loop execution. The control computation introduces a time delay (τ_c) in order to derive the actuation signal(s). The controller forwards the actuation signal(s) to the actuator, introducing another communication delay τ_{ca}. Notice that delays τ_c and τ_{ca} may be different at each closed-loop execution. Finally the actuator performs the actuation at the time given by (3.4)

$$a_k = t_k + \tau_{sc} + \tau_c + \tau_{ca} \quad (3.4)$$

Consequently, for these distributed closed-loop architectures, the time delay is given by $\tau_{sc} + \tau_c + \tau_{ca}$.

Summary

In both cases, the constant time delay τ (time elapsed between related sampling and actuation instants) assumed for discrete-time control theory requires the different time delays that appear in both architectures (τ_c for the local closed-loop and $\tau_{sc} + \tau_c + \tau_{ca}$ for the distributed closed-loop) to be constant for each closed-loop execution. However, neither τ_c nor $\tau_{sc} + \tau_c + \tau_{ca}$ can be considered to be constant. Network-induced delays (τ_{sc} and τ_{ca}) may vary depending on the network traffic, medium access protocol, etc. Computation-induced delays (τ_c) may vary depending on the processing node load, programming, scheduling, etc.

Therefore, we conclude that *the timing requirements derived from the timing assumptions behind discrete-time control theory models and methods are not realistic for actual computer implementations*.

3.2 Control systems schedulability

In the previous section we have derived the timing requirements that control systems impose on a computer implementation; namely a constant sampling period (h) and a constant time delay (τ). Real-time implementations of control applications must take these timing requirements into account in order to achieve the expected behaviour of the control system. That is, real-time scheduling of control tasks must be able to preserve the inherent timing behaviour that control models imply. In this section we firstly discuss how real-time tasks models can be used to meet the timing requirements imposed by discrete-time control theory. Afterwards, we point out the implications of expressing the control timing requirements with traditional real-time task models timing constraints on the whole system schedulability.

3.2.1 Mapping control timing requirements to real-time task timing constraints

As we explained in Section 2.1.1, depending on the regularity of task activation, a task can be defined as periodic or aperiodic. For closed-loop systems designed using discrete-time control theory, it is standard practice that control activities, due to their periodic nature, are mapped into periodic tasks.

A periodic task is characterized by several attributes (also called *task timing constraints*), such as period and deadline, and properties, such as worst-case execution time. Common

practice is that for control tasks, the *period* of the task $task_i$, T_i, is given by the *sampling period*, h, used in the controller design and an estimation of the maximum time required by the node to execute the control algorithm is assigned as a worst-case execution time, C_i.

However, the deadline assignment is not clear [RAM96]. In [LIU73], the deadline for periodic tasks is equal to the task period. This relies on the idea that each task must be completed before the next request occurs. Later research has extended the schedulability analysis to handle deadlines shorter [TIN94] or longer than the tasks period [STA98]. However, the origin of deadlines and their relation to control theory is rarely mentioned, except in [SHI85] and [SHI92], where deadlines are derived according to the allowed maximum time delay considering different regions of the state space and stability. In the examples they show, deadlines are typically found to be several times longer than the sampling period. This is due to the fact that the sampling period of a controller is not only chosen to satisfy Shannon sampling theorem but also to achieve the desired controlled system performance (see Section 2.2).

From a control perspective, deadlines must primarily be used to bound the time delay of controllers. We have seen in Section 3.1.1 that discrete-time control theory requires a constant time delay in an implementation. It has been also argued in Section 3.1.1 that the shorter the delay, the better the control. Therefore, the deadline for control task is given by the time delay.

Notice that this deduction may lead to an unrealistic situation. We have explained in Section 3.1.1 that the most commonly used discrete-time control model assumes the time delay to be zero, although the control computation execution time is not zero. Therefore, for these cases, if we assign the deadline equal to the time delay (which is zero), each control task instance execution will miss its deadline. Therefore, for these cases, the realistic approach is to assign the deadline equal to the worst-case execution time of the closed-loop[3] although in the controller design no delay has been accounted for.

In summary, the deadline for control tasks can be specified as follows:

a) the time delay assumed in the controller design if $\tau > 0$, which must include the worst-case execution time of the closed-loop.

b) the worst-case execution time of the closed-loop, if the time delay was assumed zero in the controller design.

Henceforth, we will use the term 'time delay' for both cases. Notice that the following relation will hold: the time delay τ will be always equal to or greater than the worst-case execution time of the closed-loop.

3.2.2 Real-time implementation of closed-loops

As mentioned earlier in Section 3.1, the three main parts of a closed-loop are sampling, control computation and actuation. There are two possible ways of implementing a closed-

[3] The worst-case execution time of a closed-loop is an upper bound estimation of the maximum time required by the processor (if local closed-loop) or processors and networks (if distributed closed-loop) for executing the three main parts of a closed-loop (sampling, control computation and actuation)

loop in real-time systems using periodic tasks: the single task approach or the multiple task approach.

- *Single task approach:* in this case, the closed-loop is implemented as a single periodic control task $task_{ci}$. That is, the different activities of a closed-loop (sampling, control computation and actuation) are implemented sequentially within a single periodic task. In this case, sampling (the AD conversion) takes place when each control task instance starts its execution, and actuation (the DA conversion) takes place when the control task instance completes its execution. Therefore, $t_k=s(task_{ci,k})$ and $a_k=f(task_{ci,k})$.
- *Multiple task approach:* in this case, the three different activities of a closed-loop are implemented in separate periodic tasks. Typically one task performs the sampling ($task_{si}$), another the control computation ($task_{cci}$) and another the actuation ($task_{ai}$). The three tasks must execute according to the logic sequence *sampling/control computation/actuation* (for each closed-loop), that is, $s(task_{ai,k})\geq f(task_{cci,k})$ and $s(task_{cci,k})\geq f(task_{si,k})$[4]. Therefore, the sampling will take place when the sampling task starts its execution ($t_k=s(task_{si,k})$), and the actuation will take place when the actuation task completes its execution ($a_k=f(task_{ai,k})$).

Note that what is important in order to meet the control timing requirements derived from the discrete-time control theory assumptions is to guarantee a constant sampling period and a constant time delay for each closed-loop implementation. To do so, for each closed-loop system, the following relations - expressed by standard real-time task timing constraints (periods and deadlines) and properties (worst-case execution time) - must hold:

- Each task (in the single task approach) or set of tasks (in the multiple task approach) in charge of controlling a physical system (or plant) has a period given by the sampling period used in the controller design stage.
- Each task (in single task approach) or set of tasks (if multiple task approach) in charge of controlling a physical system (or plant) has a deadline given by the time delay used in the controller design (which should coincide - as a best approximation - with the worst-case execution time of the task or task set)

By holding these relations, at each closed-loop execution, sampling will occur at the *start of each period*, thus keeping the constant sampling period assumption. In addition, actuation will occur at some time instant between *the start of each period* and *the start of the period plus the worst-case execution time*, which is the best approximation to the constant time delay assumption that can be achieved using real-time task models. Note that the smaller the difference between the worst-case execution time and the actual execution time, the better the approximation will be. In fact, as we introduced in Section 2.1.2 (and we further discuss in Section 8.1.1), if we assume that each task (in the single task approach) or set of task (in the multiple task approach) is characterized by its exact execution time (rather than its worst-case execution time) and the deadline is equal to the exact execution time, the

[4] The start time of the k^{th} instance of the control computation task ($task_{cci}$) must occur after the finishing time of the related k^{th} instance of the sampling task ($task_{si}$) and the start time of the k^{th} instance of the actuation task ($task_{ai}$) must occur after the finishing time of the related k^{th} instance of the control computation task ($task_{cci}$).

actuation will occur after a constant time interval after the start of each period, thus keeping the constant time delay assumption.

These timing requirements translated into task timing constraints for the single and multiple task can be summarized as follows (note that if we assume the exact - c_i - rather than the worst - C_i - execution time, all the following still holds), where h and τ are the sampling period and time delay used in the controller design stage:

- In the single task approach, for a periodic task $task_{ci}$ characterized by (T_{ci},C_{ci},D_{ci}) where *i* relates the task to a specific closed-loop, the following assignation must hold:

  $$T_{ci} = h,\ C_{ci} = \tau_i, \text{ and } D_{ci} = C_{ci}$$

  Figure 3.6 portrays a sequence of closed-loops executions implemented using this single task approach (shaded boxes mark executions of instances of the control task $task_{ci}$. Each instance of task $task_{ci}$, $task_{ci,k}$, periodically performs sampling (s), control computation (c) and actuation (a) sequentially).

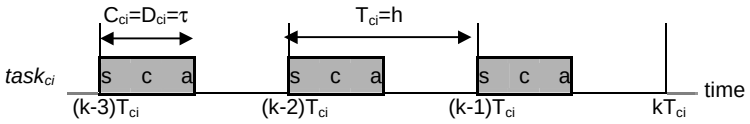


Figure 3.6. Closed-loop implemented using a single periodic task

- In the multiple task approach, for the three periodic tasks $task_{si}$, $task_{cci}$, $task_{ai}$ (each characterized by (T_{si},C_{si},D_{si}), (T_{cci},C_{cci},D_{cci}) and (T_{ai},C_{ai},D_{ai})), where *i* relates the three tasks to a specific closed-loop, the following relations must hold:

  $$T_{si}=T_{cci}=T_{ai}=h,\ C_{si} + C_{cci} + C_{ai} = \tau \text{ and } D_{ai} = C_{si} + C_{cci} + C_{ai}$$

  Note that the previous relations must hold for each three tasks implementing a closed-loop. This can easily be fulfilled by specifying offsets[5] as follows: $O_{cci} = O_{si} + C_{si}$ and $O_{ai} = O_{cci} + C_{cci}$ and deadlines as follows: $D_{si} = C_{si} + O_{si}$, $D_{cci} = C_{cci} + O_{cci}$ and $D_{ai} = C_{ai} + O_{ai}$.

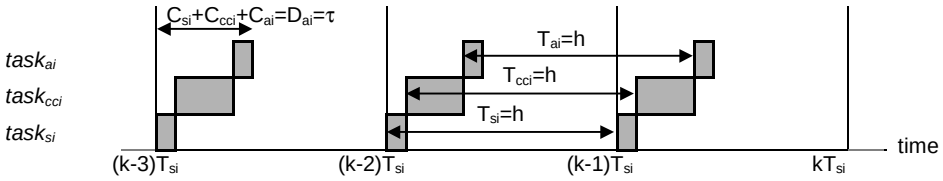


Figure 3.7. Closed-loop implemented using multiple periodic tasks

In Figure 3.7 we show a scheme of a closed-loop implemented using the multiple task approach where the previous relations hold. For the sake of clarity, we have omitted all the offsets. Shaded boxes mark executions of instances of the three tasks $task_{si}$, $task_{cci}$ and $task_{ai}$.

[5] An offset or phase for a periodic task [BUT97], O_i, specifies the activation time of each instance relative to the start of its period. If O_i is the offset of a periodic task $task_i$, the activation time of the k^{th} instance is given by $O_i+(k-1)T_i$.

Summary

Henceforth, if we don't specify the task approach used for implementing a closed-loop, we assume the following: when we use the term "*control task*", we refer to a closed-loop system where a physical process (or plant) is controlled using either the single task approach or multiple task approach. We do that because the results we present in this thesis hold for both approaches. However, when necessary, we will distinguish between these two approaches.

Note that in order to fulfil the discrete-time control theory timing assumptions, if the deadline assignment is given by the WCET (or by the exact execution time, as we assume for control tasks, see Section 8.1.1. for further discussion) no major difference in terms of flexibility exists on implementing a closed-loop using either of the two approaches. This is due to the fact that this deadline assignment over-constrains system schedulability, as we explain in the next section. However, if this deadline assignment is relaxed to $D_i > C_i$ (with $D_i \leq h$), the multiple task approach provides more flexibility and, as can be seen in [CER99] and [CRE99], from a schedulability and control point of view may be advantageous. However, as we explain in Chapter 4, this can lead to other types of disadvantages (i.e., controlled systems performance degradation).

3.2.3 Limits of control task scheduling

Setting deadlines equal to WCET[6] for control tasks (characterized with standard timing constraints) over-constrain the schedulability of the system because it implies that each time a control task instance is released, it has to start its execution immediately. That means that whatever schedule policy we use to schedule a set of tasks that includes control and non-control tasks, to find feasible schedules, in the general case, is not possible. However, we must consider the following scenarios:

- *One control task:* if our system only has one control task, obviously, to schedule this task is straightforward. As an example, if we have a control task $task_i$ with the following characterization: (T_i=6, C_i=1, D_i=1), a feasible offline schedule is shown in Figure 3.8, where shaded boxes mark tasks instances executions.

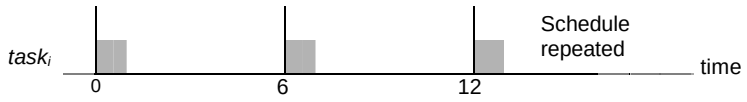


Figure 3.8. Feasible offline schedule

- *Two or more control tasks:* if our system has more than one control task, to find a feasible schedule that meets the control timing requirements is generally not possible. As an example, if we have a set of two control tasks characterized as indicated in Table 3.1, we show that to find a feasible schedule is not possible.

[6] If deadlines are equal to exact execution times, the schedulability limitations we present in this secction are exactly the same. Therefore, here we keep the WCET property for control tasks.

	T_i	C_i	D_i
$task_1$	2	1	1
$task_2$	3	1	1

Table 3.1. Task set

If we explore over the task periods LCM (6 time units) all possible task instance orderings, none of the combinations meets the constant sampling period requirement. For the first task, $task_1$, the two possible instances orderings meeting the constant sampling period requirement is shown in Figure 3.9, where boxes mark period intervals and shaded areas instance executions.

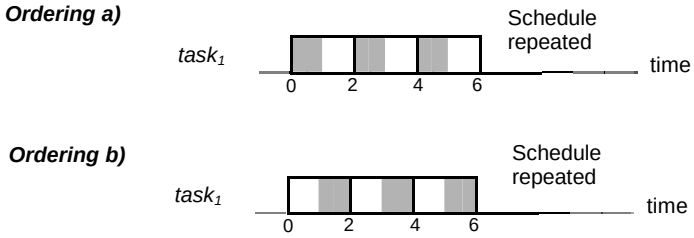


Figure 3.9. Possible orderings for $task_1$

If on each possible instance ordering for $task_1$ we try to schedule the second control task $task_2$, we can not find any feasible ordering for instances of $task_2$ that meet the control timing requirement of having a constant sampling period, as we show in Figure 3.10 (where boxes mark period intervals and shaded areas instance executions).

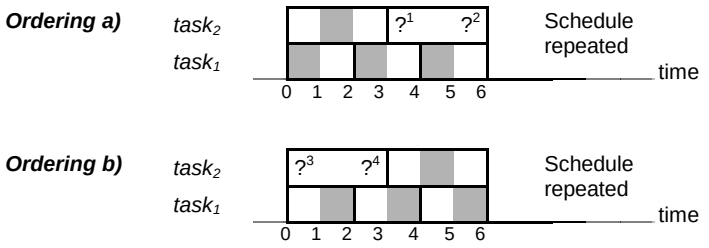


Figure 3.10. Possible orderings for $task_2$

In *Ordering a)* (Figure 3.10) we can see that after placing the first instance of $task_2$ in the only possible position, the only allowed positions to place the second instance are $?^1$ or $?^2$. However, none of these two positions meet the constant sampling period requirement. A similar situation happens in *Ordering b).*

The only case in which it may be possible to find a feasible schedule is when the control task periods are harmonics (see [TOR97] for further reading). In this situation we can distinguish two scenarios:

1) Given a set of control tasks with periods $\{T_1, T_2, T_3, \ldots, T_n\}$ (each T_i corresponding to the specific sampling period used in each controller at the design stage), we define that periods T_i have a *strict harmonic* relation if they are related as follows: $T_{i+1}=n*T_i$, where $GCD(T_1, T_2, T_3, \ldots, T_n)=T_1$, $LCM(T_1, T_2, T_3, \ldots, T_n)=T_n$, periods $\{T_1, T_2, T_3, \ldots, T_n\}$ in increasing order and n is an integer. Such period relations simplify control systems analysis and scheduling. As an example, having a set of two control tasks characterized as indicated in Table 3.2 with *strict harmonic* periods, we show that a feasible schedule exists.

	T_i	C_i	D_i
task₁	3	1	1
task₂	6	2	2

Table 3.2. Task set with two control tasks with *strict harmonic* period relation

In Figure 3.11 we show a feasible schedule (over two times the task periods LCM) for the two control tasks (where boxes mark task periods intervals and shaded areas mark task instances executions).

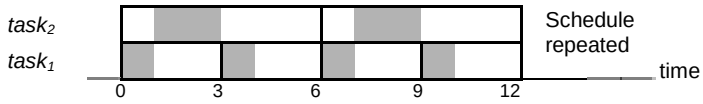


Figure 3.11. Feasible schedule of two control tasks with *strict harmonic* period relation

Notice however that even with strict harmonic periods, it is not always possible to find a feasible schedule. For example, in the previous task set, if *task₂* has $C_i=3$ (and $D_i=C_i$ as mandated for the control timing requirements), it is not possible to find a feasible schedule: either $task_1$ or $task_2$ will miss its deadline or the constant sampling period requirement for both control tasks will be not met.

2) A less strict harmonic relation between periods is the following: given a set of control tasks with periods $\{T_1, T_2, T_3, \ldots, T_n\}$ (each T_i corresponding to the specific sampling period used in each controller at the design stage), we define that periods T_i have a *soft harmonic* relation if they are related as follows: $T_n=n_n*T_0$, where $GCD(T_1, T_2, T_3, \ldots, T_n)=T_0$, periods $\{T_1, T_2, T_3, \ldots, T_n\}$ in increasing order and $T_1=n_1*T_0$, $T_2=n_2*T_0$, …, , where n_i's are integers. Such period relations also simplify control systems analysis and scheduling. As an example, having a set of two control tasks characterized by Table 3.3 with *soft harmonic* periods, we show that a feasible schedule exists.

	T_i	C_i	D_i
task₁	6	1	1
task₂	4	1	1

Table 3.3. Task set with two control tasks with *soft harmonic* period relation

In Figure 3.12 we show a feasible schedule (over the task periods LCM) for the two control tasks (where boxes mark task period intervals and shaded areas mark task instances executions).

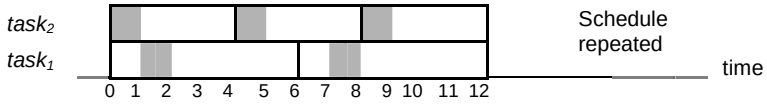


Figure 3.12. Feasible schedule of two control tasks with *soft harmonic* period relation

Notice however that as before in situation 1), even with soft harmonic periods, it is not always possible to find a feasible schedule. For example, in the previous task set, if $task_2$ has C_i=2 (and D_i=C_i as mandated for the control timing requirements), it is not possible to find a feasible schedule: either $task_1$ or $task_2$ will not meet the constant sampling period requirement.

As we have seen above, setting deadlines equal to WCET for control tasks to meet the control timing requirements (constant sampling period and constant time delay) mandated for discrete-time control theory over-constrain the schedulability of the task set that includes these control tasks.

The determinism and complex constraints fulfilment that control timing requirements impose on control tasks execution explains why real-time control systems have traditionally been implemented using offline schedules. Although the advantages of using such offline schedules for computer-controlled systems have been proven (run time determinism and ability to fulfil complex timing constraints, see Section 2.1.2), some real limitations have also been identified (fragility to changes, null overrun handling, lack of run time flexibility).

A solution to overcome this problem is to use scheduling approaches that combine offline scheduling and online scheduling as in [FOH94]. In this approach, the offline part can be used to keep complex timing constraints, like the ones imposed by control timing requirements on control tasks. The online part can be in charge of meeting the constraints of the other tasks, achieving a good schedulability of the whole task set.

3.3 Summary

In this chapter we have analysed the control impact in schedulability. First of all, we have argued that discrete-time control models and methods have an inherent timing (timing assumptions) that imposes unrealistic restrictions on an implementation. That is, the control timing requirements posed on an implementation that can be derived from control assumptions are not realistic regarding the timing behaviour of possible implementations.

Moreover, we have shown that to use traditional task models and constraints (attributes) for control tasks scheduling to express discrete-time control theory timing requirements results in poor system schedulability, because schedules become over-constrained (regardless of assuming control tasks characterized by worst-case execution time or exact execution time as we do).

Chapter 4

Schedulability impact on control

In this chapter, we discuss the timing effects that standard real-time scheduling algorithms have on task instance executions. Since the early work on real-time scheduling presented by [LIU73], real-time tasks can be scheduled using a wide variety of scheduling algorithms. As we show in this chapter, one common feature of scheduling algorithms is that they introduce different forms of *jitter* in task instance execution, i.e., *scheduling inherent jitters*. We characterize these jitters in terms of control theory when the scheduled tasks are control tasks. Finally, we give examples to show the effects of control tasks subject to jitters in the controlled systems performance.

In the previous chapter we concluded that applying standard timing constraints for control tasks to express the timing requirements (constant sampling period and constant time delay) imposed by discrete-time control theory results in poor system schedulability. This is due to the fact that deadlines for control tasks have to be assigned equal to their worst-case execution time (or to be more accurate, to their exact execution time, see Section 3.2.2), which implies that each time a control task instance is released, it has to start its execution immediately. One way to solve this problem is to relax the strict deadline assignment for control tasks (as we pointed out in Section 3.2.3), in such a way that system schedulability can be improved. However, when the scheduled tasks are control tasks with relaxed deadlines (that is, deadlines equal to periods, for example), we show in this chapter that scheduling inherent jitters for control tasks have an undesirable effect on the closed-loop system: they produce a degradation of the controlled system response, and may even cause a critical failure, i.e., instability.

The work explained in this chapter has been partially presented in [MAR01a].

4.1 Real-time scheduling timing analysis

Real-time theory has provided scheduling algorithms that use task models characterized by fixed timing constraints (or attributes) such as periods and deadlines to express the application timing requirements. In Section 2.1 we described the task models that are usually used in real-time scheduling, and we gave a brief review of different types of scheduling policies. In this section, we investigate the timing effects of such scheduling algorithms in task instance executions.

4.1.1 Jitters characterization

In real-time scheduling, a task can be seen as a successive execution of instances (also called jobs). As we pointed out in Section 2.1, the k^{th} instance of a feasible periodic task

task$_i$, *task*$_{i,k}$, fulfils the following constraints: it has to execute within its period, which starts at $(k-1)T_i$ and finishes at kT_i (where T_i is its period), and has to complete before or at time $(k-1)T_i+D_i$, where D_i is its relative deadline (provided $C_i \leq D_i \leq T_i$, where C_i is the task worst-case execution time). This means that each instance must start and finish its execution within an interval of D_i time units.

Note that in the case of characterizing the task *task*$_i$ by its exact execution time c_i and assigning its deadline as $D_i=c_i$, the exact start and completion time of each k^{th} instance execution will be given by kT_i and kT_i+c_i, respectively.

However, in the general case, no matter which deadline assignment we have for a given periodic real-time task, if deadline complies with $C_i \leq D_i \leq T_i$, no assurances can be made on the exact start and completion time of each task instance execution.

This variability in task instance executions, i.e. *jitters*, is an inherent timing property of real-time scheduling policies. That is, jitters in each task instance execution are allowed as long as the schedulability constraints are preserved. Recall that Figure 2.2 already showed this property. In general, periodic tasks can be subject to jitters in their instance executions due to the following reasons:

- After an instance of a periodic task has been released, its execution start time is delayed because other instances of other tasks are executing.
- After an instance of a task has started its execution, it can be interrupted by the execution of other instances of other tasks or can be blocked when trying to access shared resources other than the processor.

These two reasons imply that:

a) The start times of successive instances of a periodic task are not equidistant (successive instances do not start at the same time instant within every period). As a consequence, the time intervals between successive task instance start times are not constant.

b) The finishing times (or completion times) of successive instances of a periodic task are not equidistant (successive instances do not finish at the same time instant within every period). As a consequence, the time elapsed between the start and finishing time of an instance of a periodic task will vary from instance execution to instance execution.

These jitters on task instance executions are illustrated in Figure 4.1, where we portray three instances of a periodic task *task*$_i$ characterized by C_i (marked with shaded areas), T_i, D_i.

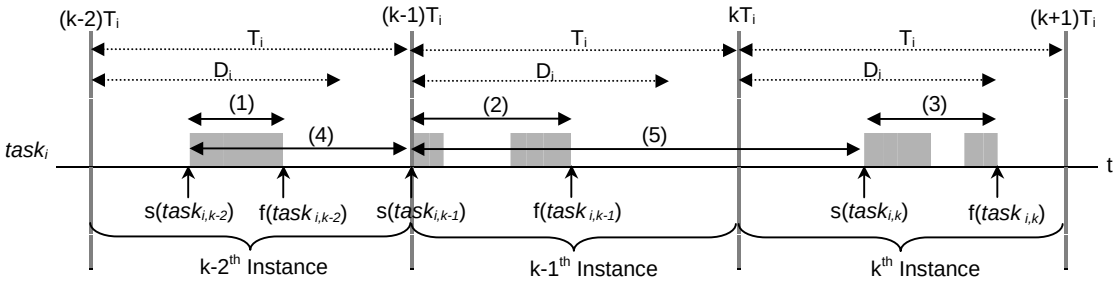


Figure 4.1. Jitters in task instances

Figure 4.1 shows that that the time interval between the k-2th task instance start time and the k-1th task instance start time (4) is shorter than the time interval between the k-1th task instance start time and the kth task instance start time (5). This is because the start times of those instances occur at different time instants within every period. It can also be seen in Figure 4.1 that the time elapsed between the start and finishing time of each instance, denoted by (1), (2) and (3), varies.

In the following, using an example, we illustrate the jitter property. Let us schedule the task set described in Table 4.1 using the Rate Monotonic (RM) algorithm presented in [Liu73] (where deadlines are equal to periods).

	T_i	C_i
$task_2$	7	2
$task_1$	4	2

Table 4.1. Task set

The set of tasks (as shown in equation 4.1) complies the RM schedulability test (see equation 2.2 in Section 2.1.2):

$$U = \sum_{i=1}^{2} \frac{C_i}{T_i} = \frac{2}{4} + \frac{2}{7} \approx 0.78 < 2(2^{1/2} - 1) \approx 0.83 \qquad (4.1)$$

The resulting schedule, over the LCM of the task periods (28 = LCM (7,4)), can be seen in Figure 4.2, where boxes mark period and shaded areas mark instance executions (on the basis of the worst-case execution time). In this case, although all instances of task $task_1$ are not subject to jitters, for instances of task $task_2$, the presence of jitters is clear.

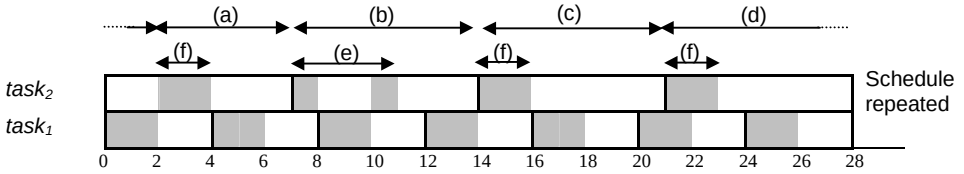


Figure 4.2. Partial schedule corresponding to RM

Note that some instances of task $task_2$ start their execution later than their release time. Therefore, successive instances start times are not equidistant. For example

- the first instance start time and second instance start time differs by 5 time units (a),
- the second instance start time and third instance start time differs by 7 time units (b),
- the third instance start time and fourth instance start time differs by 7 time units (c), and finally,
- the fourth instance start time and fifth instance start time (that corresponds to the first one, due to the cyclic nature of the schedule) differs by 9 time units (d).

Also, the time elapsed between the start and finishing time of each instance of a task $task_2$ is not always constant. For example

- the second instance takes 4 time units to complete from its start time (e) while
- the remaining instances take 2 time units to complete from their start time (f).

The immediate consequence of the two previous properties is that successive instances finishing times are not equidistant.

Remark

Note that the worst-case execution time assumption for periodic tasks increases the run time task instance execution timing variability, thus increasing the jitter problem. This is due to the fact that at run time, task instances will probably execute for less that the assumed worst-case execution time (see Section 2.1.2).

4.1.2 Effects of scheduling inherent jitters on control tasks

In this section we categorize from a control standpoint the effects that scheduling inherent jitters have on the timing of closed-loops when the controlling activities are implemented by real-time periodic tasks (as we explained in Section 3.2.2) with relaxed deadlines (deadlines equal to periods) in a multitasking-computing platform. Recall that whatever task approach (single or multiple task approach) we use to implement the three main parts of a closed loop (sampling, control computation and actuation), sampling takes place at the beginning of each closed-loop execution and actuation takes place at the end of each closed-loop execution.

In the previous section, we identified two major jitter consequences on task instance executions. If these tasks are control tasks following the models we explained in Section 3.2.2, from a control point of view, scheduling inherent jitters in control tasks have the following effects[1]:

- ***Sampling jitter:*** Having non-equidistant control task instance start times implies having irregular sampling. That is, time intervals between consecutive sampling instants are not constant; they vary from one control task instance execution to another. We call this variability *sampling jitter*.

  Given a control task $task_i$,

  - we call ***sampling interval*** each specific time interval between two successive sampling instants
  - we denote each *sampling interval* by $\mathbf{h}(task_{i,k})$, where i and k refer to the k^{th} instance of the control task $task_i$, which is given by (4.2).

$$h(task_{i,k}) = s(task_{i,k+1}) - s(task_{i,k}) \qquad (4.2)$$

[1] For the sake of simplicity, the definitions are given in terms of the *single task approach*. However, they can be easily extended to cope with the *multiple task approach*: sampling jitter is the variability on start times of successive sampling tasks, and sampling-actuation jitter is the variability between related sampling task start time and actuation task completion time.

- ***Sampling-actuation jitter:*** To have a varying elapsed time between the start and finishing time of each instance of a control task implies that the separation between sampling instants and actuation instants vary between control task instance executions. That is, time intervals between related sampling and actuation instants are not constant. We call this variability *sampling-actuation jitter*.

  Given a control task $task_i$,

  - we call ***sampling-actuation delay*** each time interval between related sampling and actuation instants
  - we denote each *sampling-actuation delay* by $\boldsymbol{\tau(task_{i,k})}$, where *i* and *k* refer to the k^{th} instance of the control task $task_i$, which is given by (4.3).

$$\tau(task_{i,k}) = f(task_{i,k}) - s(task_{i,k}) \quad (4.3)$$

If we do not need to identify *the specific* control task because it is clear from the context, in the following, we will omit all *i-subscripts*. In this case, for a given control task, $h(task_k)$ and $\tau(task_k)$ will denote the sampling interval and sampling-actuation delay of the k^{th} instance of the control task. An immediate consequence of the two previous jitters is that the time interval between consecutive actuation instants is not constant, which means irregular actuation.

Using the following examples we show different scheduling situations where control tasks are subject to sampling jitter, sampling-actuation jitter or both. In all the examples, for all control tasks, we assume deadlines equal to periods because we are now interested in relaxing the deadline assignment (which we discussed in Section 3.2.2) for control tasks, in order to characterize scheduling inherent jitters for control tasks.

Sampling jitter example

Let us suppose we have two control tasks (each one in charge of controlling an independent process/plant), characterized as shown in Table 4.2 (deadlines equal to periods), that have to share a processor.

	T_i	C_i
$task_2$	5	1
$task_1$	4	2

Table 4.2. Task set

Notice that each control task period T_i is given by the sampling period h used in the controller design stage (T_i=h) (see Section 3.2.2). That is, each control task with period T_i is implementing a control law that has been designed assuming a constant sampling period h in the implementation. For this task set, which is schedulable by RM [LIU73], we show in Figure 4.3 the partial schedule over the LCM of the tasks periods that we obtain using the RM algorithm, where boxes mark task periods and shaded areas execution times.

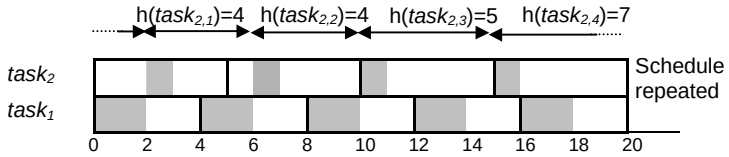


Figure 4.3. Sampling jitter: partial schedule

Despite the fact that control task $task_1$ is not subject to any jitters, control task $task_2$ is subject to sampling jitter. That is, the time intervals between successive sampling instants are not constant. Over the task periods LCM, sampling jitters for control task $task_2$ originate the following sampling intervals sequence: 4, 4, 5 and 7. Note that according to the resulting schedule, $task_2$ will execute with varying sampling intervals instead of having a constant sampling period (equal to 5), as it was supposed to have in the implementation from the controller design stage.

Sampling-actuation jitter example

Let us suppose we have two control tasks to execute in a single processor (each one in charge of controlling an independent process/plant), characterized as shown in Table 4.3 (we assume deadlines equal to period), with the following fixed task priority assignment: $task_1$ has higher priority than $task_2$.

	T_i	C_i	O_i	P_i
$task_2$	6	2	1	2
$task_1$	4	1	0	1

Table 4.3. Task set

Notice that in this case we are also relaxing the deadline assignment for control tasks. The deadline D_i for each control task should coincide with the worst-case (if not exact) execution time of the control task C_i, which is given by each time delay τ assumed in each controller design (see Section 3.2.2). However, although each task deadline is now equal to the task period ($D_i=T_i$), each control task is implementing a control law that has been designed assuming a constant time delay τ (that for illustrative purpose now coincides with C_i) in the implementation.

For this task set, which is schedulable by FPS [TIN94], we show in Figure 4.4 the partial schedule over the LCM of the tasks periods that we obtain using FPS, where boxes mark task periods and shaded areas execution times.

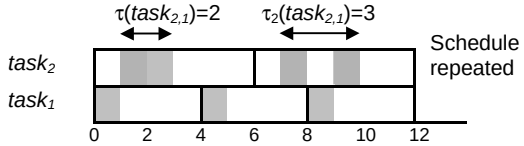


Figure 4.4. Sampling-actuation jitter: partial schedule

Despite the fact that control task $task_1$ is not subject to any jitters, control task $task_2$ is subject to sampling-actuation jitter. That is, the time intervals between related sampling and

actuation instants are not constant. Over the task periods LCM, sampling-actuation jitter for control task $task_2$ produces the following sampling-actuation delay sequence: $\tau(task_{2,1})=2$ and $\tau(task_{2,2})=3$. Note that according to the resulting schedule, $task_2$ will execute with varying sampling-actuation delays instead of having a constant time delay (equal to 2), as was assumed in the controller design stage.

Sampling jitter and sampling-actuation jitter example

Let us suppose we have two control tasks to execute on a single processor (each one in charge of controlling an independent process/plant), with the characterization given by Table 4.4 (we assume deadlines equal to periods).

	T_i	C_i
$task_2$	5	2
$task_1$	4	2

Table 4.4. Task set

As in the previous two cases, each control task is implementing a control law that has been designed assuming a constant sampling period h and a constant time delay τ in the implementation.

For this task set, which is schedulable by RM [LIU73], in Figure 4.5 we show the partial schedule over the LCM of the tasks periods that we obtain using the RM algorithm, where boxes mark task periods and shaded areas execution times.

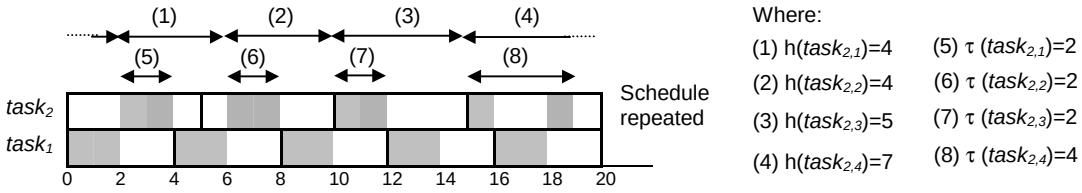


Figure 4.5. Sampling jitter and sampling-actuation jitter: partial schedule

Despite the fact that control task $task_1$ is not subject to any jitters, control task $task_2$ is subject to sampling jitter and sampling-actuation jitter. Over the task periods LCM, we summarize jitters for control task $task_2$ in Table 4.5.

	Instances of $task_2$ ($task_{2,k}$)			
Jitters	$task_{2,1}$	$task_{2,2}$	$task_{2,3}$	$task_{2,4}$
h($task_{2,k}$)	4	4	5	7
τ($task_{2,k}$)	2	2	2	4

Table 4.5. Jitters summary for the partial RM schedule

Note that according to the resulting schedule, $task_2$ will execute with both varying sampling intervals and sampling-actuation delays instead of both having a constant sampling period (equal to 5) and time delay (equal to 2), as was assumed at the controller design stage.

4.1.3 Sampling and sampling-actuation jitters properties

In this section, we present the timing properties of sampling intervals and sampling-actuation delays that can be derived from periodic task scheduling of control tasks. In the previous section we characterized from a control point of view the effects of scheduling inherent jitters for control tasks: sampling jitter and sampling-actuation jitter. These jitters imply that at run time, for each closed-loop system (for each task – if the closed loop is implemented using the single task approach – or for each set of tasks – if the closed loop is implemented using the multiple task approach, see Section 3.2.2), varying sampling intervals and varying sampling-actuation delays apply. In this section, for each closed-loop, we characterize this variability, taking into account that control tasks are feasibly scheduled by a periodic scheduling policy.

For each closed loop system, generically denoted by $task_i$,[2]

- implemented by one or more periodic control tasks (single or multiple task approach, Section 3.2.2),
- in charge of controlling a physical system or plant (see Section 1.3)
- implementing a control law assuming a constant sampling period (h) and constant time delay (τ, where $0 \leq \tau \leq h$),
- that is executing in one or multiple nodes (locally or distributed, Section 3.1.2)
- characterized by a relative deadline equal to or less than the task period ($D_i \leq T_i$),
- characterized by an exact execution time c_i
- belonging to a set of feasible tasks (each task instance $task_{i,k}$ must start and finish its execution within D_i, see Section 4.1.1)

we characterize

1. the maximum and minimum sampling intervals that appear at run time for all control task instances $task_{i,k}$ as follows (we illustrate this property in Figure 4.6.):
    - maximum sampling interval: $\forall k$, $\max(h(task_{i,k})) = T_i + D_i - c_i$
    - minimum sampling interval: $\forall k$, $\min(h(task_{i,k})) = c_i + (T_i - D_i)$

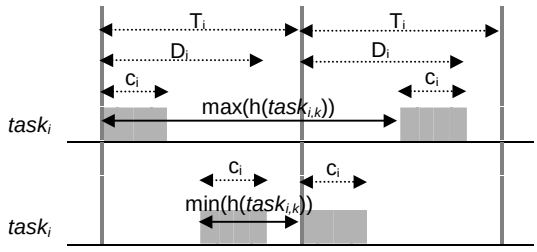


Figure 4.6. Maximum and minimum sampling intervals

[2] Note that for a given task $task_i$ executing in our system, T_i, D_i, c_i are multiples of the basic time granularity length *g* that we specified in our system model (section 1.3)

2. the maximum and minimum sampling-actuation delays that can appear at run time for all control task instances $task_{i,k}$ as follows (we illustrate this property in Figure 4.7):
    - maximum sampling-actuation delay: $\forall k, \max(\tau(task_{i,k})) = D_i$
    - minimum sampling-actuation delay: $\forall k, \min(\tau(task_{i,k})) = c_i$

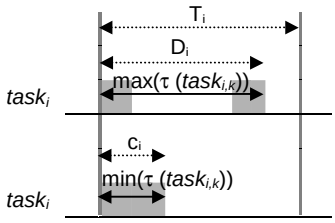


Figure 4.7. Maximum and minimum sampling-actuation delays

In summary, for a given feasible periodic control task $task_i$,[3]

1. each sampling interval that occurs at run time
    - belongs to a bounded interval $[c_i+(T_i-D_i), T_i+D_i-c_i]$
    - and is a multiple of the basic time granularity length, *g* (Section 1.3).
2. each sampling-actuation delay that occurs at run time
    - belongs to a bounded interval $[c_i, D_i]$.
    - and is a multiple of the basic time granularity length, *g* (Section 1.3).
3. all possible sampling intervals and sampling-actuation delays caused by jitters will be given by the two sets specified by (4.4) and (4.5) respectively.

$$\left\{ c_i + (T_i - D_i) + m \cdot g \;\middle|\; m = 0, 1, .., \frac{2 \cdot (D_i - c_i)}{g} \right\} \tag{4.4}$$

$$\left\{ c_i + n \cdot g \;\middle|\; n = 0, 1, .., \frac{D_i - c_i}{g} \right\} \tag{4.5}$$

For example, for a feasible periodic control task with period 80ms, deadline equal to 50ms and exact execution time of 20ms (deadline equal to period), if the granularity length g is 10ms, all possible sampling intervals and sampling-actuation delays due to jitters that may apply at run time are given by sets {50, 60, .., 110} and {20,30,40,50} respectively.

To conclude, the sampling intervals and sampling-actuation delays that can appear at run time for a given control task are a finite number, that is, they belong to a set of finite numbers of sampling intervals and to a set of finite numbers of sampling-actuation delays respectively. Note that prior to run time, all possible sampling intervals and sampling-actuation delays that may apply at run time are known for each control task.

---

[3] The following conditions could also be formally summarized as follows:
$\forall task_{i,k}$: $h(task_{i,k})=m*g \in [c_i+(T_i-D_i), T_i+D_i-c_i]$ and $\tau(task_{i,k})=n*g \in [c_i, D_i]$, $m,n \in N$

4.2 Jitter impact on control

In this section we show and explain the impact on the controlled system performance when control tasks are subject to the previously identified scheduling inherent jitters. We show through two examples and by using simulations (obtained using the simulator presented by [EKE99]) that the presence of jitters in control task instance executions degrades the performance of the control system, even causing a critical failure (instability).

4.2.1 Illustrative examples

The two examples we use are the DC servo problem, controlled by a discretization of a continuous-time designed PID controller, and the inverted pendulum problem, controlled by a discrete state feedback controller obtained using pole placement observer design (see Section 2.2.2)

Specifically, for illustrating the degrading effects of sampling jitter and sampling-actuation jitter on the controlled systems performance, we use both examples to show that the degradation appears whatever the control problem we are dealing with or whatever controller design approach we are using.

Example 1

In the servo problem, the major goal is to follow the command (reference) signal. To study the scheduling effects on the control performance of the DC servo closed-loop, we used the task $task_{PID}$ we presented in Section 2.2.2. However, in this case, it has to share the processor with other tasks. For illustrative purposes, we use RM as a scheduling algorithm (tasks are assigned rate monotonic priorities, i.e., the task with the shortest period gets the highest priority). It is easy to verify that the task set (Table 4.6), with task deadlines equal to periods, is schedulable [LIU73][4].

	T_i	c_i
$task_{PID}$	2 ms	0.2 ms
$task_1$	1.3 ms	0.5 ms

Table 4.6. Task set

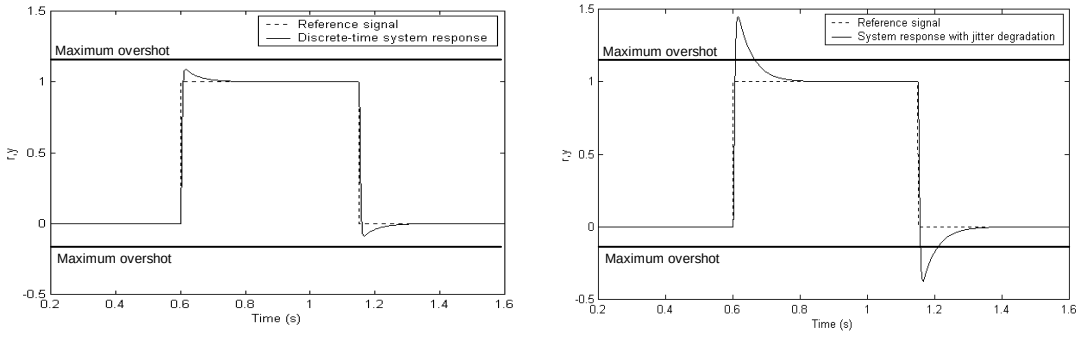


Figure 4.8. DC servo response (left - expected response) with jitter degradation (right)

[4] Note that $task_{PID}$ is characterized in Table 4.6 by its exact execution time. The same effects (or even worse) in the system response would occur if the task would have been characterized by the worst-case execution time.

Figure 4.8 (right) shows the effects of scheduling on the DC servo response performance when the $task_{PID}$ is scheduled along with other tasks. The system response, which was to track the square pulse input, suffers important degradation, which drives the system response out of the specified requirements: the response goes beyond the maximum allowed overshoot. In comparison, Figure 4.8 (left) also shows the response we obtained when the task $task_{PID}$ was executed in isolation.

This degradation is caused by the inherent jitters that RM scheduling policy introduces in task instance execution. Specifically, $task_{PID}$ is subject to sampling jitter and sampling-actuation jitter, as seen in Figure 4.9, where we show the RM partial schedule over the task periods LCM (0.026 seconds). For each task, high-level marks instance executions, medium-level marks either pre-emption or start time delays at each instance execution, and low-level marks task sleeping. Therefore, looking at $task_{PID}$, its instance executions are subject to sampling jitter (marked, for example, by (a)) and sampling-actuation jitter (marked, for example, by (b)).

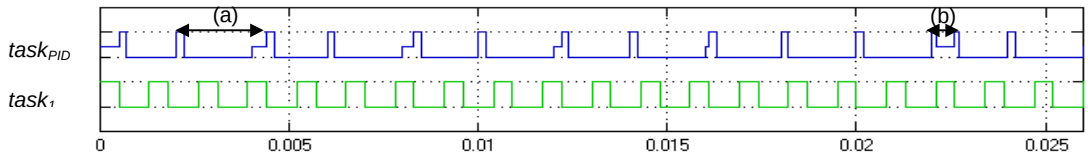


Figure 4.9. RM partial schedule for the task set over the task periods LCM (26ms)

The system response degradation can be explained as follows. Firstly, control actions are calculated using the PID algorithm, where the parameter h is constant (2ms). However, due to sampling jitter, at run time, sampling intervals for each PID control task instance execution are no longer constant. They vary from 1.5ms to 2.5 ms. Secondly, sampling-actuation delays were supposed to be constant (0.2ms). However, at run time, they also vary (from 0.2 ms to 0.7ms).

Example 2

In the inverted pendulum control problem, where the controller has to maintain the desired vertical position of the inverted pendulum at all times, the major goal is to recover from a perturbation in less than two seconds.

To study the scheduling effects on the response performance of the inverted pendulum closed-loop, we use the task $task_{sfc}$ we presented in Section 2.2.2. However, for illustrative purposes, in this case, it has to share the processor with other tasks.

In this case, firstly we isolate each jitter type to see what the effects of sampling jitter or sampling-actuation jitter are on the performance of the inverted pendulum system response. Afterwards, we show the effects when the task $task_{sfc}$ controlling the inverted pendulum is affected by the combination of both jitters. For this example, we use RM and EDF as scheduling algorithms with the set of tasks described in Table 4.7 (where deadlines are equal to periods, and c_i denotes the exact execution time). Before running the system, we have verified for each task set the schedules feasibility in terms of the response time analysis [JOS86].

	T_i	c_i
$task_1$	50ms	10ms
$task_2$	60ms	10ms
$task_{sfc}$	80ms	20ms
$task_3$	100ms	20ms

Table 4.7. Task set

In Figure 4.10 we can see the resulting schedules over the task periods LCM (1.2 seconds) when using RM and EDF. Specifically, in both resulting schedules, $task_{sfc}$ is affected for sampling jitter and sampling-actuation jitter, as seen in Figure 4.10. For example, looking at $task_{sfc}$ in each schedule, $task_{sfc}$ instance executions are subject to sampling jitter (marked, for example, by (a) and (b)) and sampling-actuation jitter (marked, for example, by (c) and (d)).

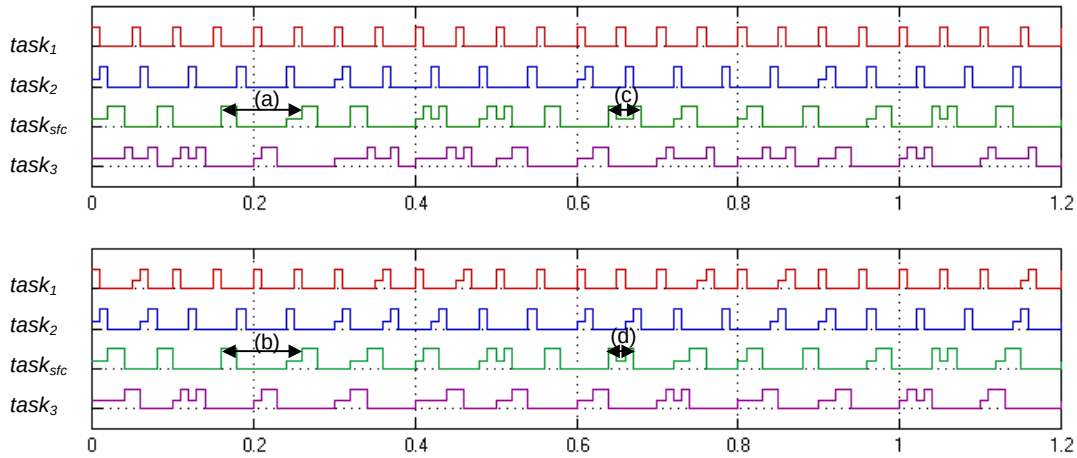


Figure 4.10. RM (top) and EDF (bottom) partial schedule for the task set over the task periods LCM

We first study the response of the inverted pendulum when the task $task_{sfc}$ is only affected by sampling jitter.

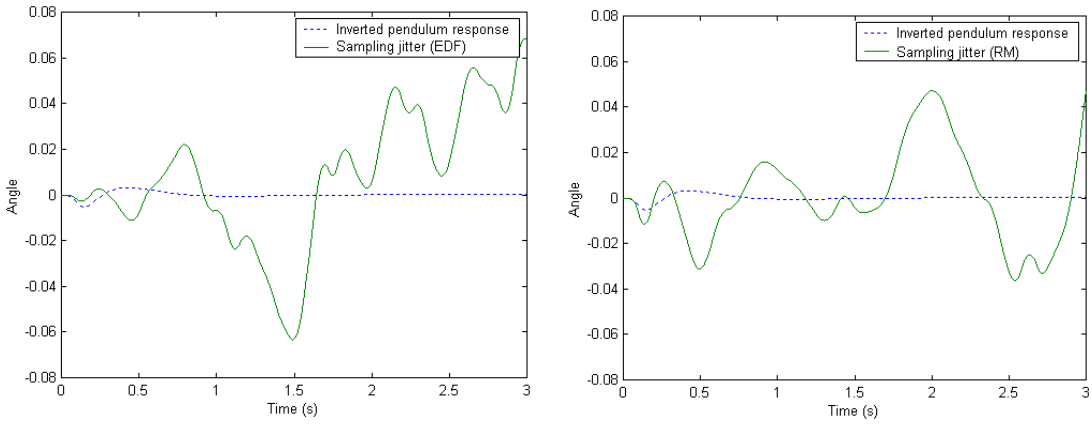


Figure 4.11. EDF (left) and RM (right) sampling jitter degradation

In Figure 4.11 we show the degradation on the inverted pendulum response (solid line) due to sampling jitter, for both RM (right) and EDF (left) scheduling algorithms. As can be seen, the degradation in both cases leads the system to instability (the inverted pendulum falls down). We also plot in Figure 4.11 the inverted pendulum response (dotted line) we obtained earlier (Figure 2.22, when the inverted pendulum was controlled by task $task_{sfc}$ executing in isolation on a single processor), for comparative purposes. Note that from Figure 2.22 to Figure 4.11 we have changed the y-axis scale (angle) in order to allow a better appreciation of the type of degradation introduced by jitters for each scheduling policy.

For both the RM and EDF scheduling algorithms, the response suffers important degradation. The period (T_{sfc}) of task $task_{sfc}$ was set to 80ms, equal to the sampling period h used in the controller design. However, due to the sampling jitter introduced by each schedule, sampling intervals h($task_{sfc,k}$) vary, taking values of 60, 70, 80, 90, or 100ms at each instance execution, following a periodic pattern over the LCM of the task periods for both scheduling algorithms.

However, since RM and EDF give different schedules using the same task set, the sampling interval variability over the task periods LCM is also different. This is why the degradation on the system response also varies if RM or EDF is used. Although in both cases the system response is unstable, when using EDF, the system becomes unstable faster than when using RM.

If we analyse the sampling interval (h($task_{sfc,k}$)) variability over the task periods LCM (1.2s) of the task set, we can see that the periodic sequence of sampling intervals that appears when RM or EDF is used (as stressed in Table 4.8 in *italics*) differs in 6 values. Therefore, the effects on the system responses are rather different.

h($task_{sfc,k}$)	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	k=12	k=13	k=14	k=15
RM	60	80	100	***60***	***90***	80	70	80	***90***	***80***	70	90	70	***80***	***100***
EDF	60	80	100	***80***	***70***	80	70	80	***100***	***70***	70	90	70	***90***	***90***

Table 4.8. Sampling interval (h($task_{sfc,k}$)) variability (in ms)

Note also an interesting property of the sampling interval variability obtained either by RM or EDF. In both cases, the mean variability of the sampling intervals h($task_{sfc,k}$) is 80ms, equal to the theoretical sampling period used at the design stage of the controller.

We secondly study the response of the inverted pendulum when the task $task_{sfc}$ is only affected by sampling-actuation jitter. In Figure 4.12 we show the degradation on the inverted pendulum response (solid line) due to sampling-actuation jitter, for both RM (right) and EDF (left) scheduling algorithms. As can be seen, the degradation in both cases, although present, is not as critical as it was when the task $task_{sfc}$ was subject to sampling jitter. In this case, the system remains stable. We also plot in Figure 4.12 the inverted pendulum response (dotted line) we obtained earlier (Figure 2.22, when the inverted pendulum was controlled by task $task_{sfc}$ executing in isolation on a single processor), for comparative purposes (note that the y-axis scales have changed again from Figure 4.11 to 4.12, in order to make the degrading effects of sampling-actuation jitter more visible in this case).

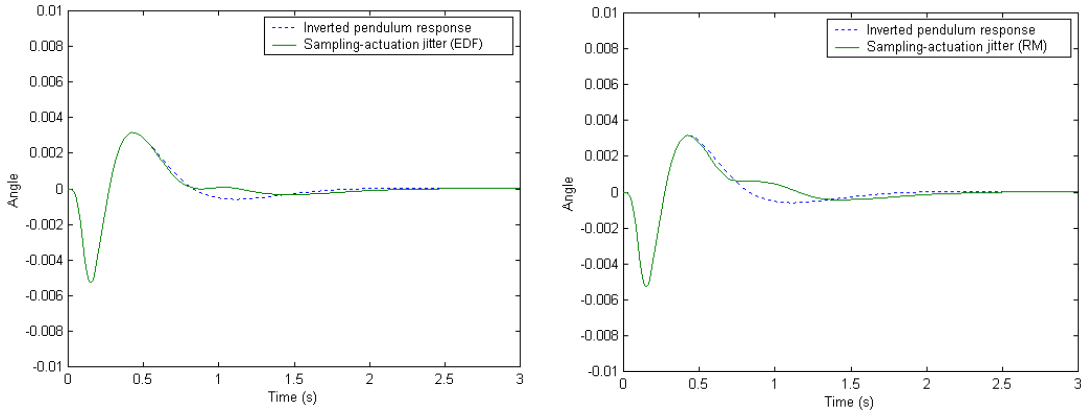


Figure 4.12. EDF (left) and RM (right) sampling-actuation jitter degradation

As before, for either RM or EDF scheduling algorithms, the response suffers degradation. Although the time delay assumed in the controller design was 20ms, at run time, due to the sampling-actuation jitter introduced by each schedule, sampling-actuation delays $\tau(task_{sfc,k})$ vary, taking values of 20, 30, or 40ms in the RM schedule or values of 20 or 30ms in the EDF schedule. This different sampling-actuation delays variability that appears in each schedule explains why the degradation on the system response also varies if RM or EDF is used. Although with RM the system response suffers a more significant degradation, in both cases the system response remains stable. If we analyse the sampling-actuation delay ($\tau(task_{sfc,k})$) variability over the tasks periods LCM (1.2s) of the task set, we can see that the sampling-actuation delays (that follow a periodic sequence) that appear when RM or EDF is used (Table 4.9) differ in 2 values (marked in *italics* in Table 4.9). thus, the effects on the inverted pendulum responses are also different.

$\tau(task_{sfc,k})$	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	k=12	k=13	k=14	k=15
RM	20	20	20	20	20	***30***	30	20	***40***	20	20	20	20	30	20
EDF	20	20	20	20	20	***20***	30	20	***30***	20	20	20	20	30	20

Table 4.9. Sampling-actuation delay ($\tau(task_{sfc,k})$) variability (in ms)

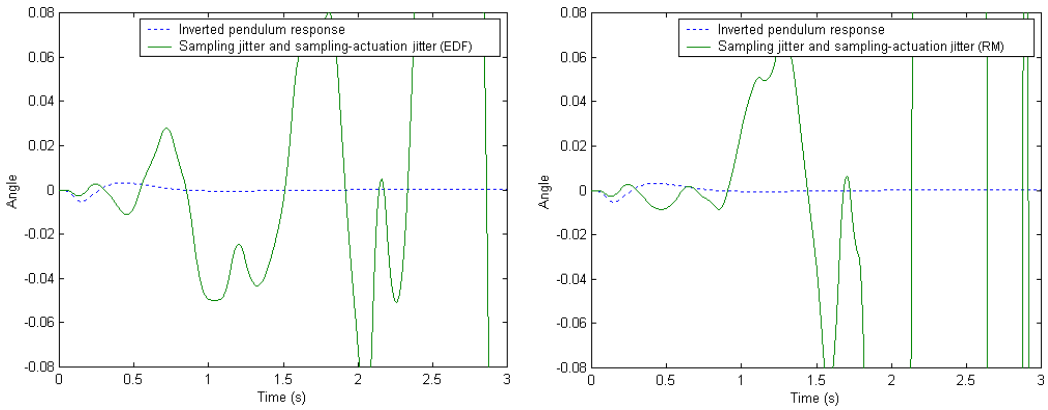


Figure 4.13. EDF (left) and RM (right) sampling jitter and sampling-actuation jitter effects on the inverted pendulum response

Finally, Figure 4.13 shows the inverted pendulum response when the task $task_{sfc}$ is subject to both sampling jitter and sampling-actuation jitter due to EDF (left) or RM (right) scheduling algorithms (the y-axis scale again coincides with the one used in Figure 4.11). In each case, the system response, which should keep the inverted pendulum in the vertical position, leads to instability; that is, it falls. This degradation is again due to sampling jitter and sampling-actuation delays.

4.2.2 Impact explanation

In this section, we explain the degrading effects that scheduling inherent jitters (sampling jitter and sampling-actuation jitter) in control task instance execution have in the controlled systems response. This degradation in the controlled systems response is due to two main reasons:

a) The control (measured) variables (from which the controller calculates the control signals) are taken at the wrong time instants (thus acquiring wrong values) and the control signals are also transmitted at wrong time instants

b) Control signals are calculated according to a constant sampling period (h) and a constant time delay (τ) while at run time, sampling intervals and sampling-actuation delays vary at each control task instance execution.

To illustrate these reasons with numbers let us focus on the DC servo example (note that a similar reasoning would also apply to the inverted pendulum example). After tuning the PID controller, in the discretization, we use a constant sampling period h. That is, we assume that at run time, samples are taken at equidistant times, with a constant interval of h between sampling times. However, since at run time task $task_{PID}$ suffers sampling jitter, the time elapsed between successive sampling times is not constant. Therefore, the samples we are acquiring and thus the successive errors $e(t_k)$ from which we calculate each control signal are not correct. In addition the integration and differentiation parts are calculated assuming a constant sampling period when what we have are different sampling intervals applying at each task instance execution. As a result, the numerical value of the output (control signal) is inaccurate.

Note also that when we designed the controller, we assumed *zero* time delay, as is usually done at the design stage (see Section 3.1.1 for further discussion). However, when we implemented the controller in task $task_{PID}$, we assigned 0.2ms of exact execution time (because execution takes time). As a result, at run time, the outputs we are sending are not performed at the actuation times we expected. In addition, due to the fact that task $task_{PID}$ suffers sampling-actuation delays, even designing the PID controller with a 0.2ms of time delay, the outputs will still not be performed at the expected times.

In summary, at run time, due to sampling jitter, we are reading wrong values and thus calculating wrong outputs, which in turn are sent at wrong times due to sampling-actuation jitter. That's why at run time, the DC servo response is not as good as we expected from the PID controller design. In Table 4.11 we show, over the task periods LCM of the set of tasks we used in the DC servo example, when the reference signal (that has to be tracked) changes from 0 to 1, what the values of the correct controller should be, and what the values of the

task $task_{PID}$ are. Note the difference between the times in which the correct sampling should occur (and the corresponding value of the measured variable), and the times (in bold in Table 4.10) in which task $task_{PID}$ performs its sampling (and the corresponding value of the measured variable). Note also the difference between the correct outputs and the actuation times at which these are sent, and the wrong outputs calculated from wrong samples, which are sent at wrong actuation times (in bold in Table 4.10). It is also easy to follow in Table 4.10 that sampling intervals (looking at column "Wrong sampling – time (s)") and sampling-actuation delays (looking at columns "Wrong sampling – time (s)" and "Wrong output – time (s)") for task $task_{PID}$ vary instead of being constant. Observe that none of the values of both the correct and wrong sampling and actuation (output) coincide, except for the first one.

Correct sampling		**Wrong sampling**		**Correct output**		**Wrong output**	
Time (s)	Value	Time (s)	Value	Time (s)	Value	Time (s)	Value
0.6020	0	**0.6024**	0	0.6022	46.8001	**0.6026**	46.8001
0.6040	0.1687	0.6040	0.2262	0.6042	-6.0965	0.6042	-8.7850
0.6060	0.5193	**0.6063**	0.5652	0.6062	-14.9082	**0.6065**	-14.4747
0.6080	0.7885	0.6080	0.9502	0.6082	-11.7335	0.6082	-17.2345
0.6100	0.9491	**0.6102**	1.1294	0.6102	-7.1371	**0.6104**	-8.2939
0.6120	1.0320	0.6120	1.2486	0.6122	-3.7861	0.6122	-5.8118
0.6140	1.0696	**0.6141**	1.2922	0.6142	-1.8163	**0.6143**	-2.4879
0.6160	1.0839	0.6160	1.3072	0.6162	-0.7930	0.6162	-1.2296
0.6180	1.0873	0.6180	1.3054	0.6182	-0.3092	0.6182	-0.4684
0.6200	1.0860	0.6200	1.2964	0.6202	-0.0995	**0.6206**	-0.1259
0.6220	1.0831	0.6220	1.2847	0.6222	-0.0172	0.6222	0.0132
0.6240	1.0797	**0.6245**	1.2723	0.6242	0.0109	**0.6247**	0.0687
0.6260	1.0762	0.6260	1.2572	0.6262	0.0181	0.6262	0.2163

Table 4.10. PID correct values vs. $task_{PID}$ wrong values

For illustrative purposes, observe that a correct sample should have been taken at time 0.6060s, expecting a value of 0. 0.5193, while the actual sampling due to sampling jitter, occurs at time 0.6063s, with a measured value of 0.5652 (the sampling interval in this instance execution has been 0.6063-0.6040=0.0023s, instead of the assumed constant sampling period of 0.0020s). In the same way, observe that a correct output should have been sent at time 0.6202s, expecting a value of -0.0995, while the actual actuation, due to sampling-actuation jitter, occurs at time 0.6206s, with a calculated value of -0.1259 (the sampling-actuation delay in this instance execution has been 0.6206-0.6200= 0.0006s, instead of the assumed constant time delay of 0.0002s).

This explanation for the DC servo response degradation can be also extended to the inverted pendulum example. Recall that in the state feedback controller we designed for the inverted pendulum, although we were more realistic in the sense of not assuming zero time delay, we have exactly the same problems as in the DC servo. Due to sampling jitter, we are reading inputs at wrong sampling instants, thus obtaining wrong values. As a consequence we are calculating wrong output values, which are sent at wrong actuation instants due to sampling-actuation jitters.

4.3 Summary

In the implementation of real-time control systems, real-time scheduling algorithms and classical control theory cannot be developed separately because unexpected controlled systems performance may occur. In both examples presented, we have followed the standard design procedure. That is, we have first designed the controller according to the specified system requirements, assuming a constant sampling period and a constant time delay. Once the controller has been designed, its computer implementation has been carried out. In the implementation, we have basically mapped the sampling period used in the control design to the period of the periodic control task implementing the control law. We also assigned to each control task an exact execution time. However, to overcome the poor schedulability problem detected in Chapter 3, we have set each control task deadline equal to its period, instead of setting the deadline equal to its exact execution time. By doing this, the system becomes less constrained, allowing better schedulability. However, by doing this, we also allow each control task instance to take the whole period to complete, causing effects like sampling jitter and sampling-actuation jitter. We have shown that jitter effects on control task instance execution damage the controlled system response, even causing a critical failure in the system, i.e., instability.

Chapter 5

Integrated scheduling and control co-design

In this chapter we provide an image of the approach to scheduling and control co-design we present in order to build effective real-time control systems is such a way that

a) we solve the problems identified in Chapter 3, *poor system schedulability*, and Chapter 4, *controlled systems response degradation*, and

b) we improve the control and scheduling performance of real-time control systems compared to classic control and scheduling approaches.

In the two previous chapters we explained the control impact on schedulability and the schedulability impact on control. On the one hand, in Chapter 3 we have seen that when standard task timing constraints are used to express the timing requirements derived from discrete-time control models and methods, specifications become over-constrained, thus resulting in poor system schedulability. On the other hand, in Chapter 4 we have seen that general purpose real-time scheduling algorithms introduces different form of jitters in task instance execution. These jitters in control tasks produce unexpected controlled systems performance degradation.

Henceforth, we will show how these problems can be addressed using a combination of control and real-time scheduling principles so that control systems can exploit new (and more flexible) scheduling approaches and scheduling approaches can take advantage of properties of new (and more flexible) control design approaches.

In this chapter we firstly identify the problems we wish to solve and what we need to solve them, followed by an overview of the solutions we present. Finally, we introduce the main applications of our approach to build effective real-time control systems in such a way that the identified problems are solved.

5.1 Motivation

In the design and implementation of real-time control systems, we have clearly identified two main problems: poor system schedulability and controlled systems performance degradation. These two problems arise when such systems are designed in a traditional way, that is, differentiating two separate stages: first, control design and then its computer implementation. This separate design procedure has made it difficult to establish a clear flow of information between the activities of the control and the real-time communities, thus losing the possibility of sharing critical parameters of the computer implementation (e.g., scheduling inherent jitters) that may be considered in the control design stage and vice versa, sharing critical parameters of the control design (e.g., constant sampling period and constant time delay) that the computer platform could have taken into account in the

implementation. This lack of connection between the two communities results in system designs that do not fulfil the expected performance, from a schedulability and control performance point of view, as we have seen in the two previous chapters.

Specifically, when facing the two problems (poor real-time system schedulability and controlled systems response degradation), we encounter two major difficulties:

- Discrete-time control models and methods consider implementation issues to a very little extent. This is due to the fact that in the control design stage, controllers are assumed to execute in dedicated processors and these processors are assumed to be fast and deterministic enough not to worry about the timing that the controlling activities may have in the implementation. This also explains terms like *instantaneous execution*, which are hard to understand (if not unrealistic) when talking about execution of control algorithms. However, as we have shown (Chapter 4), when resources (e.g., processors) are limited, the timing variations in the execution of control algorithms (e.g., scheduling inherent jitters) do affect the controlled systems performance, introducing degradation and even causing instability (this is due to the fact that jitters violate the timing assumptions that are classically undertaken in discrete-time controller design). It is worth mentioning that control theory offers no advice on how to account for implementation issues (e.g., execution timing variations) in control designs.
- Real-time scheduling policies are based on standard timing constraints for periodic tasks (such as period and deadlines) that are used to express the application timing requirements. Those timing constraints are assumed to be constant for all task instances. However, this does not necessary imply that each task instance execution will have the same timing. In real-time, timing variation in instance executions (i.e., jitters) is allowed as long as the schedulability constraints are preserved. Because control designs regard the computer platform as deterministic, this variation for control task instance executions has to be eliminated (otherwise, the controlled systems performance degrades, see Chapter 4). Using standard timing constraints, this can be achieved by assigning the deadline for control tasks equal to their worst-case execution time. However, by doing this, the control task specification becomes over-constrained, resulting in poor system schedulability (Chapter 3). It is worth mentioning that real-time theory has no task models and constraints that can be used to ensure a periodic task execution free of jitters without over-constraining system schedulability.

Therefore, to build effective real-time control systems, we need a more integrated analysis and design approach, bringing together control and real-time community activities in such a way that control design and computer implementation are jointly considered. Specifically, to solve the problems we outlined and to overcome the difficulties posed by discrete-time control theory and real-time task timing constraints for control tasks, we need

1) *more flexible control design approaches*: classic discrete-time control design approaches are based on the assumption of equidistant sampling and actuation, requirements that over-constrain system schedulability if those assumptions are expressed by standard real-time task timing constraints. However, if we use a more flexible control design approach based on the assumption of non-equidistant sampling and non-equidistant actuation, we will be providing more flexible requirements to the

timing of control activities, thus relaxing control task specifications that may lead to better system schedulability.

2) *more flexible timing constraints for control tasks*: standard task timing constraints for periodic tasks, although being constant for all task instances, allow jitter in task instance execution, which for control tasks implies degradation in the controlled systems response. However, if we use more flexible timing constraints that do not introduce unexpected jitters while expressing the control timing requirements, the control performance requirements will be met.

In this thesis we provide both a new discrete-time controller design approach based on irregular sampling and irregular actuation that we call the *compensation approach*, and a set of new *flexible timing constraints* for control task scheduling which are able to express the control timing requirements derived from the compensation approach. In the next two sections, we give an overview of each main contribution.

5.2 Flexible control design

The controller design method we present, called compensation approach, is based on the assumption of non-equidistant sampling and non-equidistant actuation. The compensation approach controller design method is based on the same classic controller design methods (such as pole placement or optimization approaches) used in discrete-time control theory (that is for systems with constant sampling period (h) and constant time delay (τ, where $0 \leq \tau \leq h$)). However, instead of specifying a single value for the sampling period and a single value for the time delay at the design stage, we specify several values. Then, at each controller execution, we allow a run time controller parameters adjustment according to a specific setting for the sampling period and time delay from the set of values we specified at the design stage. That is, at run time, at each control task instance execution, specific values for both the sampling period and the time delay will apply (Chapter 6). To implement a controller designed using the compensation approach, the code resulting from implementing a control law designed using traditional discrete-time controller design methodologies (assuming constant sampling period and constant time delay) is slightly modified in order to allow the run time parameters adjustment (Chapter 7).

In summary, we present a control approach to deal with irregularly sampled discrete-time systems with varying time delays. This approach is based on more flexible timing assumptions than the rigid assumptions (constant sampling period and constant time delay) assumed by classic discrete-time control theory. Controllers obtained using the compensation approach depend on a finite set of values for both the sampling period and the time delay. This relaxes the timing requirements that an implementation must guarantee for control tasks instance executions. Variation of control task instance executions is allowed as long as this variability conforms to the different values for the sampling period and time delay assumed in the controller design stage.

It is important to point out that the compensation approach can be used to design controllers that can deal with the scheduling inherent jitters, solving the degradation that would otherwise occur, as we have shown in Chapter 4. We investigate these issues in Chapter 8.

5.3 Flexible timing constraints for control tasks

The compensation approach controller design method permits us to derive more flexible timing constraints for control tasks scheduling, as we show in Chapter 8. It allows us to design controllers that depend on a finite set of sampling period values and on a finite set of time delays values. From the timing assumptions behind the compensation approach, we define new flexible timing constraints for control tasks. The idea of flexible timing constraints for task scheduling was presented in [FOH97]. The flexible timing constraints we present are new specific constraints for control tasks, which express the timing assumptions of the compensation approach.

Standard (also called *traditional* or *fixed*) timing constraints for periodic tasks are constant for all task instances. That is, a single value holds for a constraint (e.g., period or deadline) for all task instances. In contrast, our flexible timing constraints for control tasks do not set specific values. Rather, they provide ranges and combinations to choose from (at each control task instance execution), taking into account, for example, schedulability of other tasks or controlled systems performance (Chapter 9).

In addition, feasible periodic tasks characterized by standard timing constraints suffer jitter in their instance executions, which is an undesired scheduling property if these tasks implement classically designed discrete-time controllers. Our timing constraints eliminate the jitter phenomena by allowing control task instances to start and complete their execution at specific time instants that belong to a set of pre-determined time instants. Although control tasks instance execution can have different timings, these specific timings are known *a priori*, and are thus analysable offline. Note that this allows feasible control tasks to execute with more flexibility while keeping the timing requirements posed by the compensation approach, thus meeting the control systems performance specifications.

In is important to point out that these new timing constraints for control tasks allow us to obtain feasible schedules from task sets that are not feasible when scheduled using traditional fixed timing constraints. In addition, by associating control performance information with flexible timing constraints, we show how scheduling decisions can also be taken to improve the quality of the controlled systems responses. In Chapter 9 we explore these issues.

5.4 Applications

The integrated approach we present in this thesis combines both control and scheduling principles. It combines a new flexible controller design method, the compensation approach, and more flexible timing constraints for control task scheduling. This combination allows us to apply our approach to the analysis and design of real-time control systems in different ways. In the following, we point out the three most relevant applications that we present in this thesis.

Eliminating control system response degradation caused by scheduling inherent jitters

The objective of this application is to use the compensation approach design method to solve the problems posed by jitters in control tasks that are scheduled by real-time

scheduling algorithms. The main idea is to design flexible controllers that take into account the scheduling inherent jitters that control tasks (characterized by fixed timing constraints) are subject to.

This design method is based on a two-step procedure. We assume that we have a) a set of tasks, which includes control and non-control tasks, all of them characterized by traditional timing constraints such as periods and deadlines, and b) the scheduling algorithm that we will use. For each control task (or set of control tasks) in charge of controlling a plant, we do:

1. *Offline jitter analysis:* we analyse the jitters that can appear at run time. Depending on the scheduling policy, this analysis is done in different ways:
   - *Offline scheduling and a run time dispatcher that guarantees that at run time, task instance start and completion times coincide with the task instance start and completion times specified in the offline schedule*: by analysing the offline schedule, we obtain the exact sampling intervals and sampling-actuation delays that will apply at run time.
   - *Offline scheduling and a run time dispatcher that does not guarantee that at run time, task instance start and completion times coincide with the task instance start and completion times specified in the offline schedule, or online schedule*: before run time we don't know the exact task instance timing (start and completion times) that will apply at run time. However, if the task set is feasible, we can derive all possible sampling intervals and sampling-actuation delays that will apply at run time (see Section 4.1.3)

   At the end, we group all the specific values of sampling intervals and sampling-actuation delays that will apply at run time into two sets.
2. *Offline control analysis:* Given the two sets of sampling intervals and sampling actuation delays derived from the *offline jitter analysis*, we design each controller to be implemented in each periodic control task (specified with fixed timing constraints such as period and deadline) in such a way that these two sets are included in the sampling period values and time delay values on which we base the compensation approach controller design method.

Consequently, at run time, the analysed control tasks, still characterized by fixed timing constraints and subject to jitters, by executing a slightly modified code (which we will explain in Section 7.2.2), will be readjusting their controllers parameters according to the run time jitters. These run time jitters for each control task match the jitters a) obtained in the first step (*offline jitter analysis*) and b) used in the design of the controller (that each control task will implement) in the second step (*offline control analysis*). In this way, control tasks designed under the compensation approach and still characterized by the traditional timing constraints such as periods and deadlines, will be accepting the scheduling inherent jitters that they are subject to. That is, the degradation that these jitters imply in the controlled systems response will be eliminated because the controller design that control tasks are executing has been designed precisely to take these jitters into account, thus again meeting the control performance specifications (stability and response characteristics).

In summary, using this procedure, the compensation approach controller design method can be used for control tasks in standard scheduling policies to *compensate* for the degradation that scheduling inherent jitters for traditional designed control tasks would otherwise introduce in the controlled systems response (see Chapter 8 for more details).

Transforming unfeasible schedules into feasible schedules and stable control systems

The objective of this application is to use the flexible timing constraints we derive from the compensation approach in order to solve the problems posed by applying standard timing constraints for control tasks (i.e., poor system schedulability). The main idea is to use flexible timing constraints for control tasks scheduling in such a way that, while meeting the control demands, we obtain feasible schedules of tasks sets that are not feasible when scheduled using traditional timing constraints.

Assuming that we have a) a set of tasks, that includes control and non-control tasks, all of them characterized by traditional fixed timing constraints such as periods and deadlines, and b) the scheduling algorithm that we will use, the method to be applied in this case is divided into the following steps:

1. We look for the *scheduling conflicting situations*[1] involving control tasks that impair the feasibility of the task set that is scheduled using fixed timing constraints.
2. From the conflicting scheduling situations we derive feasible values for the control tasks timing constraints (in terms of different values for each control task period and deadline, which will result in new timing constraints for such control tasks) in such a way that the system becomes feasible. These different values are grouped into two sets.
3. Given these two sets, we design the controller implemented by each of these control tasks using the compensation approach controller design method, as we explain in Chapters 6 and 7. For each control task, we use the two sets of feasible values for the period and deadline as the different values for sampling periods and time delays needed to apply the compensation approach. The resulting controllers a) keep the system stable and b) fulfil the performance requirements in terms of the controlled system response, whatever settings of the period and deadline (of each control task) applies at run time, provided they belong to the feasible values obtained in step 2.
4. Given the control tasks characterized by the new timing constraints we derived in step 2, we schedule them jointly with the remaining tasks (which are characterized by the traditional timing constraints) is such a way that
    a. non-control tasks meet their original constraints
    b. control tasks meet the new timing constraints
    c. the controlled system meets the performance requirements due to the fact that control tasks will be readjusting their controllers parameters according to the two sets of feasible values (step 2) used in the controller design stage (step 3).

[1] By *scheduling conflicting situations* we mean scheduling scenarios where two or more instances of different tasks need to be executed in order to meet their constraints. However, not all them can be feasibly accommodated, which imply that some of them will not meet their constraints.

In this way, we obtain feasible schedules that meet the scheduling and control demands of task sets that were not feasible when scheduled using fixed timing constraints (see Chapter 9 for more details).

Scheduling control tasks to improve the quality of the controlled systems response

The objective of this procedure is, when taking scheduling decisions, to use the control performance information that the flexible timing constraints for control tasks implicitly have in order to obtain feasible schedules that improve the performance of the controlled systems response. Flexible timing constraints are derived from the control timing assumptions which the compensation approach controller design method relies on. This means that these flexible timing constraints, apart from providing temporal information, incorporate control information. By making this control information explicit and available to the scheduling policy, control tasks can be scheduled in such a way that the performance of the controlled systems can be improved.

Assuming that we have control tasks characterized by the new flexible timing constraints and implementing controllers designed using the compensation approach, the method to be applied in this case can be divided into the following steps:

1. We associate with each flexible timing constraint (which characterizes each control task) value, a new value (attribute) that expresses control performance information in terms of the characteristics of the controlled system response resulting from the use of that timing constraint.
2. Given this new characterization of control task, we have to provide the mechanisms for taking scheduling decisions based on this control information for each control task invocation.

In this way, although the responsibility of meeting each closed-loop performance requirements still falls on the controller, the possibility of dynamically improving the performance of each closed-loop (in certain identified situations, e.g., perturbation arrivals) will depend on the specific run time timing of each control task, which is given by the scheduling policy. In Chapter 9 we investigate these issues.

5.5 Summary

In the analysis and design of real-time control systems we identified two main problems in Chapters 3 and 4, namely poor system schedulability and controlled systems response degradation. We have discussed why it is difficult to solve these problems and we have identified that we need more flexible controller design methods and more flexible timing constraints for control tasks in order to deal with these problems. As a solution to these problems, we have introduced a new control and scheduling co-design approach based on a) the compensation approach controller design method and b) flexible control timing constraints for control tasks. Finally, we have presented the main applications of our novel approach. In the next chapters, we investigate all of the concepts and methods we have introduced here in detail.

Chapter 6

Adapting control algorithms to implementation constraints

In this chapter we present the compensation approach controller design method. Controllers designed following classic discrete-time controller design methods [AST97] depend on a constant sampling period (h) and a constant time delay (τ); timing requirements that have proven difficult to maintain in complex computer implementations (Chapters 3 and 4). With the compensation approach, we want to be able to design controllers based on timing assumptions which are more realistic for complex computer implementations: the timing requirements that we impose on controllers designed using the compensation approach are to depend on a set of feasible values for both the sampling period and the time delay. After defining the compensation approach in terms of these new timing requirements, and before formulating the controller design problem that the compensation approach implies, we show that the definition of the compensation approach covers all possible implementation cases. We do so by discussing the effects that implementations of closed-loops have on the timing parameters of classically designed controllers (sampling period and time delay). We categorize them in six cases, which in turn, without losing generality from a control point of view, we reduce to three cases for the control formulation and analysis. We then show how the new timing parameters of the controllers obtained using the compensation approach can cope with the three main effects that possible implementations have on the timing of closed-loops systems. Finally, we formulate the compensation approach controller design method problem based on state space models to deal with closed-loops characterized by these new timing assumptions (set of feasible values for both the sampling period and the time delay).

The work explained in this chapter has been partially presented in [MAR01d] and [MAR01e].

6.1 Control algorithms design adjustment

In this section we define the compensation approach in terms of new timing parameters that are based on more appropriate timing assumptions for actual computer implementations. We also prove that the definition is complete, from a control point of view, in the sense of covering all the possible implementations.

6.1.1 Problem definition

The compensation approach controller design method is based on the same classic controller design methods (such as pole placement or optimization approaches [AST97]) used in discrete-time control theory. However, the main difference is that a controller designed

using the compensation approach does not depend on a single value for the constant sampling period h and constant time delay τ, but rather on a set of several different values for the sampling period and time delay that we specify at the design stage. Specifically, given a control task $task_i$ (implemented either using the single or multiple task approach, Section 3.2.2), we define these two sets at the design stage as (where *g* is the granularity length, see Section 1.3):

- Set of ***feasible sampling intervals*** **(FH_i):**

  $FH_i = \{h_{i,j} |$ where *i* refers to the control task $task_i$, j=1…m, $j \in N$, and each feasible sampling interval $h_{i,j}$ is a multiple of *g*$\}$

- Set of ***feasible sampling-actuation delays*** **(FT_i):**

  $FT_i = \{\tau_{i,j} |$ where *i* refers to the control task $task_i$, j=1…n, $j \in N$, and each feasible sampling-actuation delay $\tau_{i,j}$ is a multiple of *g*$\}$

For the sake of clarity, if we don't need to specify *the* control task that we are characterizing with these two sets (because it is clear from the context), we will omit the i-subscript (which identifies each task) in all the related symbols[1]. Note that each set contains a finite number of values, that is, a finite set of feasible sampling intervals and a finite set of feasible sampling-actuation delays. Note also that *feasible* means meeting the control performance specifications of each specific closed-loop, an issue that we address in Section 7.1 where we explain the compensation approach controller design method. These feasible sampling intervals and feasible sampling-actuation delays are the different values of the sampling period and the time delay, specified at the controller design stage[2].

Then, at run time, the controller parameters, which depend on the constant sampling period and the constant time delay assumed in classic control design methodology, are updated in each control task $task_i$ code according to specific values of the sampling period, feasible sampling intervals $h_{i,j} \in FH_i$, and according to specific values of the time delay, feasible sampling-actuation delays $\tau_{i,j} \in FT_i$. Therefore, using the compensation approach, we design controllers assuming *irregular sampling* and *varying time delays*. At run time, at each controller execution, different feasible sampling intervals and different feasible sampling-actuation delays will apply. In summary, each controller obtained using the compensation approach depends on a finite set of feasible sampling intervals (FH_i) and on a finite set of feasible sampling-actuation delays (FT_i).

The compensation approach is based on the notion of compensations wherein controller parameters are adjusted at run time for the presence of jitters. This technique was originally suggested as an *ad hoc* technique for PID controller design in [WIT80], [ALB90] and [ÅRZ00] in order to compensate for the degradation of the controlled system response due to variations from sample to sample, that is, due to sampling jitter. We not only extend the

[1] For example, instead of writing FT_i, we will write FT or instead of writing $\tau_{i,j}$, we will write τ_j.

[2] Note that $h_{i,j}$ and $\tau_{i,j}$, although having a similar name like the effects of jitters for control tasks (sampling intervals, $h(task_{i,k})$ due to sampling jitter, and sampling-actuation delays, $\tau(task_{i,k})$ due to sampling-actuation jitter, see Section 4.1.3), denote the timing parameters specified at the controller design stage for controllers designed using the compensation approach.

applicability of the compensation technique to deal with both varying sampling intervals and varying sampling-actuation delays but also provide a complete control analysis of the compensation approach as a discrete-time controller design method based on state space models. Apart from introducing the new controller design problem formulation in a state space form (Section 6.2), we present the controller design method, which includes new stability and response analysis (Section 7.1). Note that with the compensation approach, we jump from strict, regularly sampled discrete-time systems with constant time delays to irregularly sampled discrete-time systems with varying time delays. Control theory provides well-known methods of analysing and designing regularly sampled discrete-time systems with constant time delays [AST97]. However, for the analysis and design of irregularly sampled discrete-time systems with varying time delays, no formal approach based on state space models has been presented (as far as we know). Although particular solutions have been presented to deal with irregularly sampled discrete-time systems [WIT80], [ALB90], [ÅRZ00] and [SCH01] or systems with varying time delays [CHA95] [SHI96], [NIL98] and [WIT98] no integrated controller design method has provided a solution to both problems.

6.1.2 Closed-loop implementation effects on the controller timing parameters

In this section we categorize the effects that closed-loop implementations have on the timing of the control activities. A classically designed discrete-time controller depends on a sampling period (h) and time delay (τ, where $0 \leq \tau \leq h$), which are supposed to be kept at a constant value in the controller computer implementation (see Section 3.1.1). However, depending on the assurances that the implementation platform can give us in terms of the sampling type (*regular* or *irregular* sampling) and time delay type (*instantaneous*, *constant* or *varying* time delay), these discrete-time control theory assumptions may no longer hold. We show that in some cases, the constant sampling period and constant time delay assumption is not valid because varying values of the sampling period and/or varying values of the time delay appear in the implementation. Depending on the sampling and time delay type, we have to consider the following six cases.

1. Equidistant sampling instants, with insignificant[3] time delays.

The closed-loop implementation strategy guarantees that samples are taken periodically, i.e., at equidistant sampling instants (h, sampling period). In addition, the time delay (between when a sample is taken and when the corresponding actuation is completed) does not have to be accounted for in the controller design because the implementation guarantees a negligible controller execution with respect to the closed-loop system dynamics (see Figure 6.1 for the timing scheme of such an implementation). Control theory provides well-known methods for dealing with this case, based on regularly sampled discrete-time systems [AST97].

[3] Insignificant sampling-actuation delays refers to the control assumption of instantaneous execution time (see Section 3.1.1): sampling and actuation are regarded as occurring at the sampling instants, if perfect synchronization is assumed [AST97] and the execution time of the controller is negligible with respect to the closed-loop dynamics.

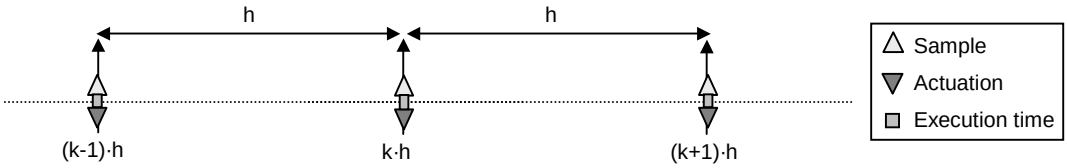


Figure 6.1. Implementation guaranteeing equidistant sampling

2. Equidistant sampling instants, with constant time delays.

The closed-loop implementation strategy guarantees that samples are taken periodically, i.e., at equidistant sampling instants (h, sampling interval), and that there is a constant time delay (τ) between when a sample is taken and the actuation is completed. As before, control theory also provides well-known methods, based on regularly sampled discrete-time systems with constant time delays, for dealing with this case [AST97]. Comparing this case with the previous one, what is important is to have a constant time delay, regardless of whether

- the execution of the control computation is assumed to be instantaneous (Figure 6.2),

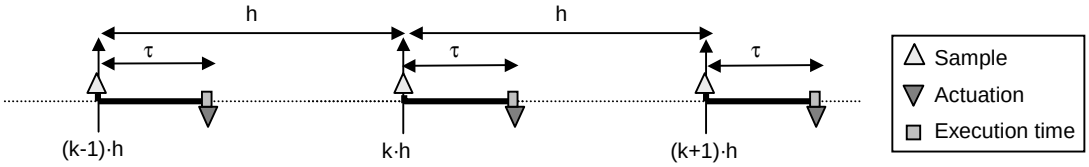


Figure 6.2. Implementation guaranteeing equidistant sampling and constant time delay (1)

- the execution of the control computation is constant (Figure 6.3),

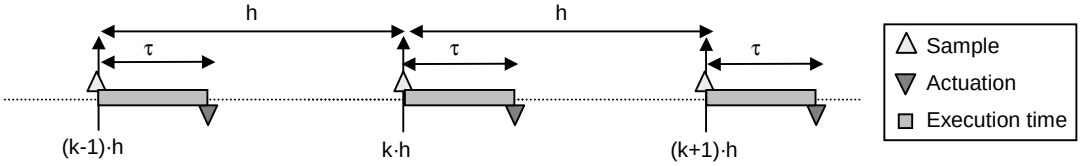


Figure 6.3. Implementation guaranteeing equidistant sampling and constant time delay (2)

- the execution of the control computation constant but with a delay in its execution start time due to the execution of other instances of higher priority tasks (Figure 6.4),

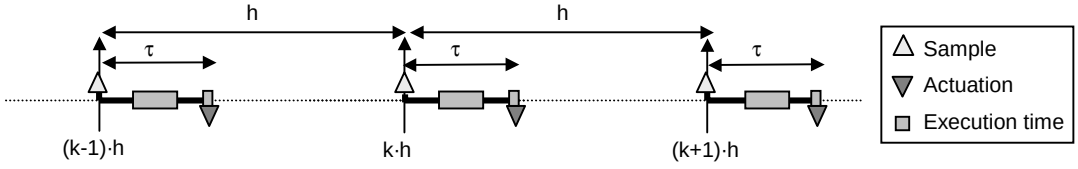


Figure 6.4. Implementation guaranteeing equidistant sampling and constant time delay (3)

- the execution of the control computation varies due to varying execution times and/or pre-emptions (Figure 6.5) or

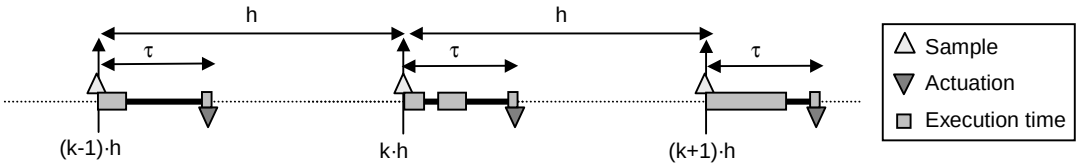


Figure 6.5. Implementation guaranteeing equidistant sampling and constant time delay (4)

- the execution of the control computation varies due to varying execution times and pre-emptions, and with a delay in its execution start time (Figure 6.6).

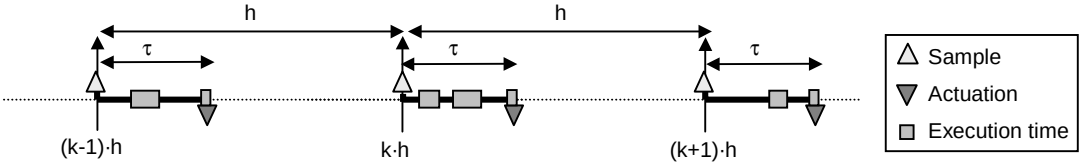


Figure 6.6. Implementation guaranteeing equidistant sampling and constant time delay (5)

A similar case occurs when the implementation allows the actuation to be completed at the beginning of the next sampling. This latter case is a special case, when the constant sampling-actuation delay is equal to the sampling period (τ = h). Control theory also provides well-established methods (regularly sampled discrete-time systems with actuation in the next sample) for dealing with closed-loops implementations that guarantee the actuation in the next sampling instant [VAC95]. As before, for this case, the assumptions that can be made on the control computation execution time are not important (as long as the implementation guarantees that the execution takes places within the sampling period).

3. Equidistant sampling instants, with varying time delays.

The closed-loop implementation guarantees that samples are taken periodically, i.e., at equidistant sampling instants (h, sampling period). However, a constant time delay is not guaranteed: different values for the time delay (τ) will appear. As we have seen in Section 4.1.2, sampling-actuation jitter for control task instance executions produces this situation. In this case, although the closed-loop implementation strategy can guarantee a constant sampling interval, no assurances are given about the time delays that each control task instance execution may be subject to. For this case, control theory has recently provided methods for dealing with this problem (see for example [NIL98]). Although we have regular sampling, varying time delays apply at run time. This problem is due to start time delays in the control computations, regardless of whether

- the execution time of the control computation is assumed to be instantaneous (Fig. 6.7)

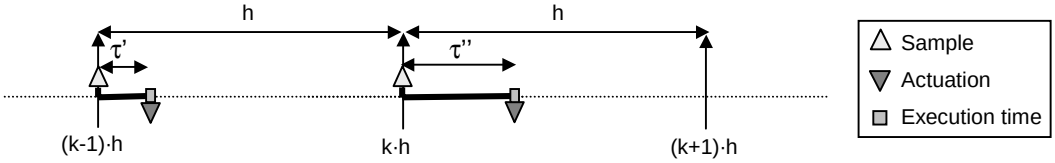


Figure 6.7. Implementation guaranteeing equidistant sampling but with different values (τ',τ'') for the time delay (1)

- the execution time of the control computation is constant (Figure 6.8)

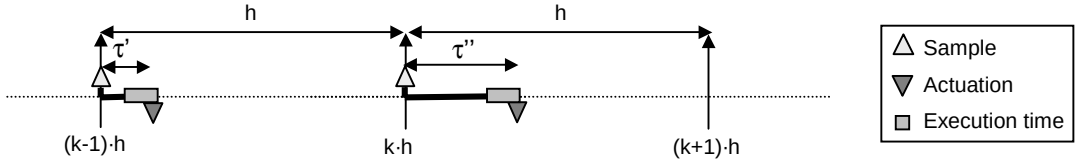


Figure 6.8. Implementation guaranteeing equidistant sampling but with different values (τ',τ'') for the time delay (2)

- the execution time of the control computation varies due to varying execution times and/or pre-emptions (regardless of whether execution start times of task instances are delayed) (Figure 6.9)

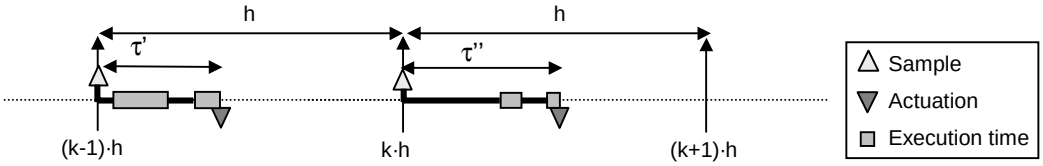


Figure 6.9. Implementation guaranteeing equidistant sampling but with different values (τ',τ'') for the time delay (3)

- the execution time of the control computation varies due to varying execution times and/or pre-emptions (regardless of whether execution start times of task instances are delayed) (Figure 6.10)

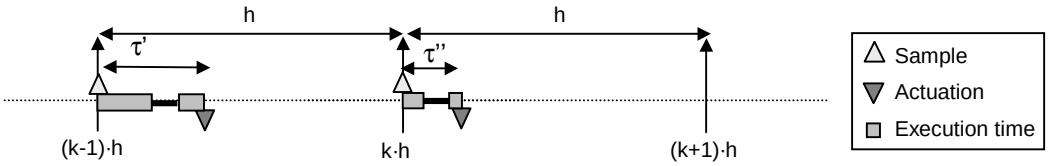


Figure 6.10. Implementation guaranteeing equidistant sampling but with different values (τ',τ'') for the time delay (4)

4. Non-equidistant sampling instants, with insignificant time delays

The closed-loop implementation cannot guarantee equidistant sampling instants (as can be seen in Figure 6.11, where h' and h'' are different values that apply as a sampling period) for the sampling period h. As we have seen in Section 4.1.2, sampling jitter for control task instance executions introduces this problem. As a consequence, time intervals between consecutive sampling instants are not constant. Note that not having equidistant sampling instants and assuming insignificant time delays implies not having equidistant actuation. But from the point of view of the controller timing parameters, this non-equidistant actuation is irrelevant, since there is no time delay between sampling instants and actuations instants. For this case, (irregularly sampled discrete-time systems), we cannot apply classic discrete-time control theory [AST97] because the assumption of equidistant sampling is not met.

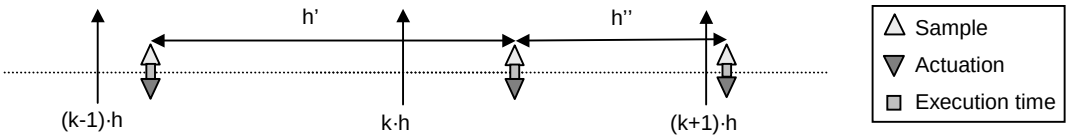


Figure 6.11. Implementation that does not guarantee equidistant sampling

5. Non-equidistant sampling instants, with constant time delays.

The closed-loop implementation does not guarantee equidistant sampling instants (as supposed by the sampling period h), but it does guarantee a constant time delay (τ) between when a sample is taken and when the actuation is completed. As in the previous case, this situation may appear due to sampling jitter in control task instances executions. Having irregular sampling in a closed-loop implementation requires a new theory of control system analysis and design. However, as far as the closed-loop implementation ensures a constant time delay for each control task instance execution, control theory provides well-known methods to deal with this delay. However, since we have non-equidistant sampling, classic discrete-time control theory [AST97] cannot be applied because it relies on the assumption of equidistant sampling instants. Note that in this case, apart from having irregular sampling, what is important is to have a constant time delay, regardless of whether

- the execution of the control computation is assumed to be instantaneous (Figure 6.12),

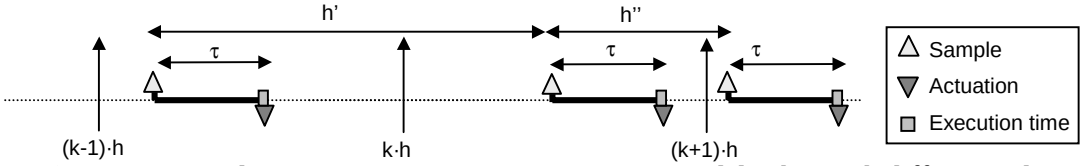


Figure 6.12. Implementation guaranteeing constant time delay but with different values (h',h'') for the sampling period (1)

- the execution of the control computation is constant (Figure 6.13),

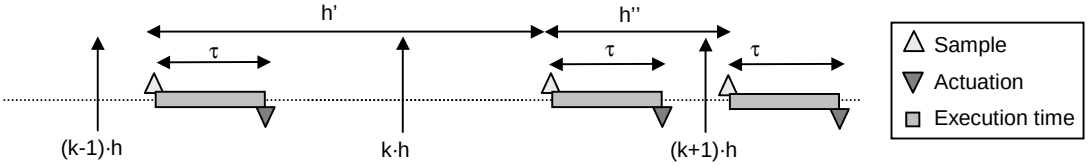


Figure 6.13. Implementation guaranteeing constant time delay but with different values (h',h'') for the sampling period (2)

- the execution of the control computation is constant but with a delay in its execution start time due to execution of other instances of higher priority tasks (Figure 6.14),

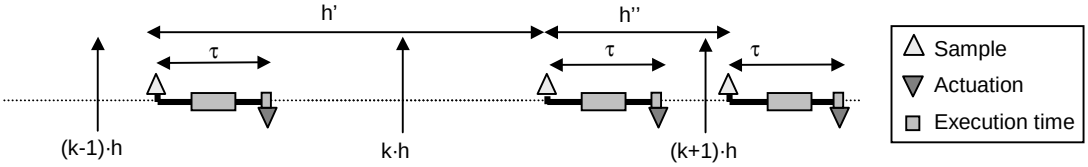


Figure 6.14. Implementation guaranteeing constant time delay but with different values (h',h'') for the sampling period (3)

- the execution of the control computation varies due to varying execution times and/or pre-emptions (Figure 6.15)

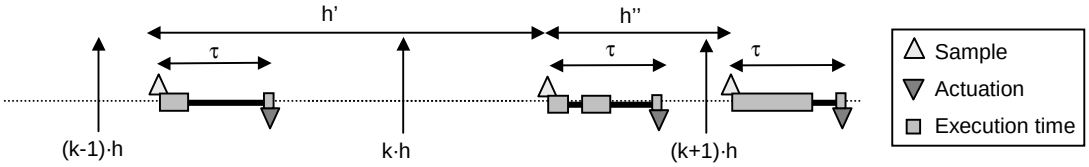


Figure 6.15. Implementation guaranteeing constant time delay but with different values (h',h'') for the sampling period (4)

- the execution of the control computation varies due to varying execution times and pre-emptions, and with a delay in the execution start time (Figure 6.16).

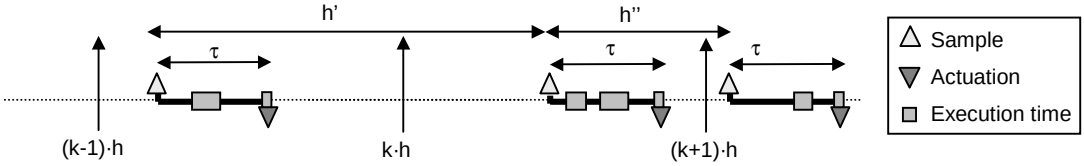


Figure 6.16. Implementation guaranteeing constant time delay but with different values (h',h'') for the sampling period (5)

6. Non-equidistant sampling instants, with varying time delays.

The closed-loop implementation can guarantee neither equidistant sampling instants (although it is supposed by the sampling period h) nor a constant value for the time delay τ. As we have seen in Section 4.1.2, both sampling jitter and sampling-actuation jitter for control task instance executions produce this situation. For this case, as we are facing irregularly sampled discrete-time systems with varying time delays, classic discrete-time control theory [AST97] cannot be applied because it is based on the assumption of equidistant sampling instants and constant time delays. In this case, the different values that apply for the time delay appear to be due to start time delays in the control computations, regardless of whether

- the execution time of the control computation is assumed to be instantaneous (Figure 6.17)

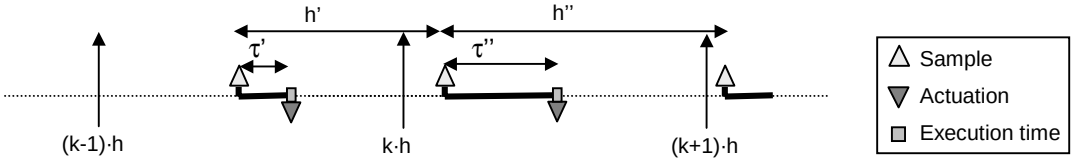


Figure 6.17. Implementation with different values for the sampling period (h', h'') and time delay (τ', τ'') (1)

- the execution time of the control computation is constant (6.18),

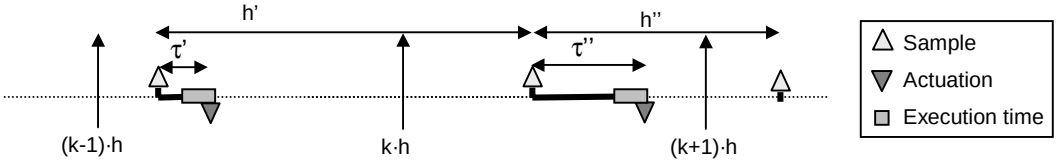


Figure 6.18. Implementation with different values for the sampling period (h', h'') and time delay (τ', τ'') (2)

- the execution time of the control computation varies due to varying execution times and pre-emptions (regardless of whether task instances suffer start time delays) (Figure 6.19)

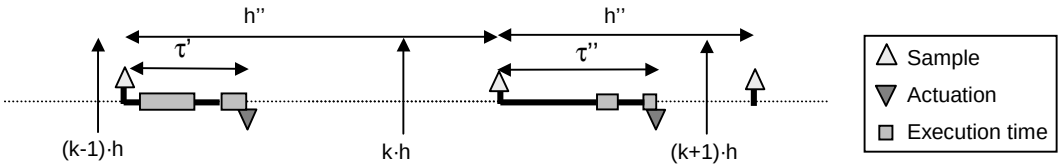


Figure 6.19. Implementation with different values for the sampling period (h', h'') and time delay (τ', τ'') (3)

- the execution time of the control computation varies due to varying execution times and pre-emptions (regardless of whether task instances do not suffer start time delays) (Figure 6.20).

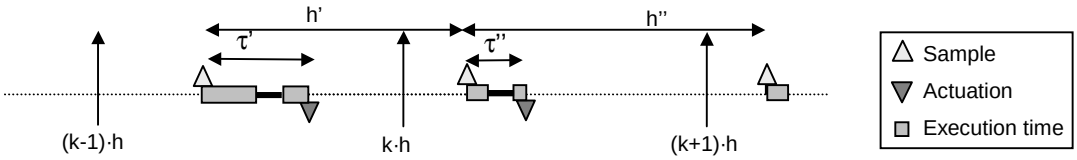


Figure 6.20. Implementation with different values for the sampling period (h', h'') and time delay (τ', τ'') (4)

6.1.3 Summary of the variation in the controller timing parameters

Depending on the guarantees the closed-loop implementation provides, we have analysed all possible implementation effects on the controller timing parameters. Looking at the controller timing parameters, the next list summarizes all the cases:

- **Case 1:** constant sampling period
- **Case 2:** constant sampling period and constant time delay
- **Case 3:** constant sampling interval and different values for the time delay
- **Case 4:** different values for the sampling period
- **Case 5:** different values for the sampling period and constant time delay
- **Case 6:** different values for the sampling period and for the time delay

Case reduction

Given the implementation case study of Section 6.1.2, in the rest of this chapter, without losing generality, we will focus on cases 3, 4 and 6 for the control analysis. Note that we do not lose generality because from a control analysis point of view, the remaining cases (cases 1, 2 and 5) are already included in the three cases 3, 4, and 6 or classic discrete-time control theory provides the required methods for analysing them. This is detailed next.

We keep for the analysis:

- *Case 6:* non-equidistant sampling and different values for the time delay occur at run time, violating the discrete-time control assumptions of a) equidistant sampling instants and b) actuations which have to be performed at fixed time instants after the sampling instants. In consequence we are in *irregularly sampled discrete-time systems with varying time delays*.
- *Case 4:* non-equidistant sampling occurs at run time, violating the discrete-time control assumption of equidistant sampling instants. In consequence we are in *irregularly sampled discrete-time systems with constant time delays*.
- *Case 3:* different values for the time delay occur at run time, violating the discrete-time control assumptions that actuations have to be performed at fixed instants after the sampling instants. Consequently we are in *regularly sampled discrete-time systems with varying time delays*.

Note that *case 6*, which requires new control analysis (classic discrete-time control theory does not provide integrated models and methods for the analysis and design of *irregularly sampled discrete-time systems with varying time delays*, as we pointed out in Section 6.1.1), already includes case 4 (if all values for the time delay in case 6 are equal to 0) and 3 (if all values for the sampling period in case 6 are equal, that is, to have a single value for h). However, we keep cases 4 and 3 to clarify the control analysis we present in this chapter. We are not keeping:

- *Cases 1 and 2*: control theory provides well-known methods of dealing with controllers with constant sampling period h and constant (or zero) time delay τ, that is, for regularly sampled discrete-time systems (case 1 and 2) with constant time delays (case 2). Note also that in keeping case 4 for the analysis, we are implicitly analysing case 1, which is a particular situation of case 4 (when all values for the sampling period in case 4 are equal to h of case 1). Similarly, in keeping case 6 for the analysis, we are implicitly keeping case 2, which is a particular situation of case 4 (when all values for the sampling period and time delay in case 6 are equal to h and τ of case 2 respectively)
- *Case 5:* different values of the sampling period occur at run time. However, from a control point of view, it can be modelled similar to case 6, which is already included for analysis. Note that in keeping case 6 for the analysis, we are addressing case 5, which is a particular situation of case 6 (when all values for the time delay in case 6 are equal to τ of case 5). Note also that since discrete-time theory provides well-known methods of dealing with constant time delays, as we have in this case, the delay in case 5 is not relevant in terms of needing new control analysis; what is relevant is the non-equidistant sampling, which is analysed in case 4.

In summary, for the three cases 3, 4, and 6 that we have kept, we will provide new control analysis, starting with the compensation approach controller design problem formulation we present in Section 6.2.

6.1.4 Completeness of the compensation approach

In this section we show the completeness of the definition of the compensation approach (Section 6.1.1) in terms of being an approach to discrete-time controller design able to cope with all the timing effects that closed-loop implementations may have in the controller timing parameters. A controller (to be implemented by a control task $task_i$) designed using the compensation approach depends (see Section 6.1.1) on a finite set of feasible sampling intervals (FH_i) and on a finite set of feasible sampling-actuation delays (FT_i).

To demonstrate the completeness of the compensation approach in terms of covering all the implementation cases, we have to show that all the different values for the sampling period and time delay that occur at run time for *cases 3*, *4* and *6* are a finite number. If all the possible values that occur at run time for the sampling period and all the possible values for the time delay

1. are a finite number and
2. can be known before run time,

by including them at the design stage into sets FH_i and FT_i respectively, a controller designed with the compensation approach with sets FH_i and FT_i will be able to cope with the timing variations that occur for cases 3, 4 and 6, thus covering all the implementation cases.

Note that as we explained in cases 3, 4, and 6 (Section 6.1.2), all the different values of the sampling period and time delay that apply at run time can be caused, for example, by sampling jitter and/or sampling-actuation jitter in control task instance executions. As we concluded in Section 4.1.3, the sampling intervals and sampling-actuation delays that can appear at run time for a given control task subject to sampling jitter and/or sampling actuation jitter are a finite number. In addition, in Section 4.1.3, we explained how these sampling intervals and sampling-actuation delays (due to jitters) could be known before run time (that is, by analysing their maximum variability while taking into account the discrete-time model that we presented in Section 1.3).

Consequently, all the possible values that occur at run time for the sampling period and time delay for cases 3, 4 and 6 are finite numbers and can be known before run time. Therefore, if sets FH_i and FT_i (which the compensation approach relies on) include them, for all the cases, we have demonstrated that the compensation approach is complete in the sense of covering all possible timing variations that closed-loop implementations may introduce in the timing of the control activities.

6.2 Controller design problem formulation

In this section, we first describe the process of designing a classic discrete-time controller and its consequences on the closed-loop system evolution. Afterwards, by imposing the

compensation approach requirements on discrete-time controllers (to depend on a set of feasible sampling intervals and on a set of feasible sampling-actuation delays), we raise the new controller design problem.

Recall that with the compensation approach, feasible sampling intervals and feasible sampling-actuation delays will vary between closed-loop executions. Therefore the system evolution (and the controller implemented in a control task) depends on varying values for the sampling period and time delay, rather than on a single value for the sampling period h and the time delay τ.

In order to formalize the control problem introduced by the compensation approach in this design procedure, we proceed in the following way: in the first approximation, we assume that the controller we are designing depends on a set of feasible sampling intervals (which is case 4 in Section 6.1.2), and that no time delay is accounted for. After formulating this first approximation in what we call an *irregularly sampled discrete-time system model*; we analyse the case with feasible sampling-actuation delays. That is, the controller to be designed depends on a set of feasible sampling-actuation delays, in what we call a *discrete-time system model with varying time delays* (which is case 3 in Section 6.1.2). Finally, both models are combined in what we call an *irregularly sampled discrete-time system model with varying time delays* (case 6 in Section 6.1.2).

In Figure 6.21 we show an overview of the control implementation problem formulation we explain in the following subsections.

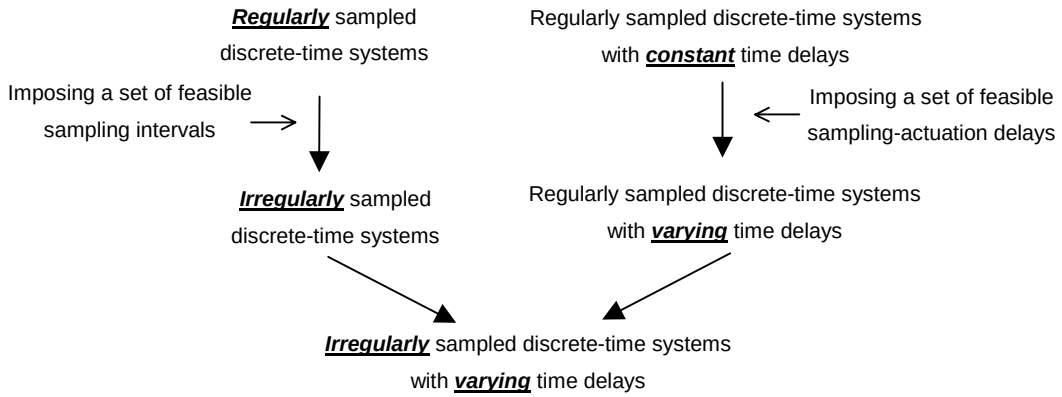


Figure 6.21. Control implementation problem formulation scheme

6.2.1 Irregularly sampled discrete-time system model

A continuous linear time-invariant process is modelled by Equations (6.1) and (6.2) [AST97].

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t) \tag{6.1}$$

$$y(t) = Cx(t) + Du(t) \tag{6.2}$$

In (6.1) and (6.2) A is the system matrix, x(t) is the state vector, B is the input matrix, u(t) is the control input, y(t) is the system output, C is the output matrix and D is the direct matrix,

all matrices of suitable dimension. Equation (6.2) is the output equation. For periodic sampling with constant sampling period h (t_k=kh), the discrete-time system can be described by (6.3) and (6.4), where Φ(h) and Γ(h) are obtained from (6.1) and (6.2) as detailed in (6.5) and (6.6) [AST97].

$$x(kh+h) = \Phi(h)x(kh) + \Gamma(h)u(kh) \tag{6.3}$$

$$y(kh) = Cx(kh) + Du(kh) \tag{6.4}$$

$$\Phi(h) = e^{Ah} \tag{6.5}$$

$$\Gamma(h) = \int_0^h e^{As} ds B \tag{6.6}$$

To meet the closed-loop requirements, the system specified by (6.3) and (6.4) is controlled using state feedback (6.7), where gain matrix (or state feedback matrix) L can be obtained by a design method such as pole placement or optimization approach [AST97].

$$u(kh) = -L(h)x(kh) \tag{6.7}$$

At the end, the closed-loop time-invariant system is characterized by Equations (6.3), (6.4), and (6.7). Equation (6.3) can be rewritten in terms of (6.7) as in (6.8).

$$x(kh+h) = (\Phi(h) - \Gamma(h)L(h))x(kh) \tag{6.8}$$

The closed-loop matrix (described previously in Equation (6.8)), where Φ(h), L(h), and Γ(h) are constant matrices in terms of a constant sampling period h, is given by (6.9).

$$\Phi_{cl} = \Phi(h) - \Gamma(h)L(h) \tag{6.9}$$

For discrete-time systems with a closed-loop matrix specified by (6.7), we can describe the closed-loop system evolution by (6.10)

$$\begin{array}{ll}
\text{k=1} & x(h) = \Phi_{cl}x(0) \\
\text{k=2} & x(h+h) = \Phi_{cl}x(h) = \Phi_{cl}\Phi_{cl}x(0) = \Phi_{cl}{}^2x(0) \\
\text{k=3} & x(2h+h) = \Phi_{cl}x(2h) = \Phi_{cl}\Phi_{cl}{}^2x(0) = \Phi_{cl}{}^3x(0) \\
\text{k=4} & x(3h+h) = \Phi_{cl}x(3h) = \Phi_{cl}\Phi_{cl}{}^3x(0) = \Phi_{cl}{}^4x(0) \\
\ldots. & \\
\text{k} & x(kh+h) = \Phi_{cl}x(kh) = \Phi_{cl}\Phi_{cl}{}^{k-1}x(0) = \Phi_{cl}{}^k x(0)
\end{array} \tag{6.10}$$

Note that in this first approximation we want the system evolution and thus the controller (to be implemented in a control task) to depend on a set of feasible sampling intervals $(FH)^4$ rather than on a single value for the sampling period h.

[4] Note that we omit the i-subscript (which relates each set of feasible sampling intervals to a control task) because it is clear in the context that we don't have to identify the control task that will implement the controller. Consequently, feasible sampling intervals belonging to FH will not have the i-subscript either.

Specifically, in this case, feasible sampling intervals will vary from one closed-loop execution to another. That is, at each closed-loop execution the controller parameters are adjusted according to each specific feasible sampling interval h_j^2 belonging to FH. Consequently, the discrete-time system is no longer time-invariant. If we denote the actual feasible sampling interval of each k^{th} closed-loop execution by h_j, we model systems described by Equations (6.3), (6.4) and (6.7) but with irregular sampling by Equations (6.11), (6.12), (6.13) and (6.14).

$$x(t_{k+1}) = \Phi(h_j)x(t_k) + \Gamma(h_j)u(t_k) \tag{6.11}$$

$$y(t_k) = Cx(t_k) + Du(t_k) \tag{6.12}$$

$$u(t_k) = -L(h_j)x(t_k) \tag{6.13}$$

$$t_k = sum\ of\ all\ h_j\ already\ applied \tag{6.14}$$

where matrices $\Phi(h_j)$, $\Gamma(h_j)$ are obtained from (6.5) and (6.6) at each controller execution for each specific feasible sampling interval (h_j, $h_j \in$ FH). Similarly, the state feedback controller $L(h_j)$ is obtained at each controller execution using the same control design approach as in (6.7) for the same specific feasible sampling intervals.

For such systems, the closed-loop time-variant system is characterized by Equations (6.11), (6.12), (6.13) and (6.14). Therefore, the closed-loop matrix that applies at the k^{th} closed-loop execution specified in (6.15) depends on $\Phi(h_j)$, $\Gamma(h_j)$, and $L(h_j)$, which are varying matrices in terms of each feasible sampling interval h_j, $h_j \in$ FH.

$$\Phi_{cl\,k} = \Phi(h_j) - \Gamma(h_j)L(h_j) \tag{6.15}$$

For discrete-time systems with a closed-loop matrix specified by (6.15), we can describe the closed-loop system evolution by (6.16), where t_k is given by (6.14).

$$\begin{aligned}
&\text{k=1} \quad x(t_1) = \Phi_{cl_1}x(0)\\
&\text{k=2} \quad x(t_2) = \Phi_{cl_2}x(t_1) = \Phi_{cl_2}\Phi_{cl_1}x(0)\\
&\text{k=3} \quad x(t_3) = \Phi_{cl_3}x(t_2) = \Phi_{cl_3}\Phi_{cl_2}\Phi_{cl_1}x(0)\\
&\ldots.\\
&\text{k} \quad x(t_k) = \Phi_{cl_k}x(t_{k-1}) = \Phi_{cl_k}\Phi_{cl_{k-1}}\ldots\Phi_{cl_2}\Phi_{cl_1}x(0)
\end{aligned} \tag{6.16}$$

The closed-loop system evolution (6.16) will depend on a product-sequence of closed-loop matrices Φ_{cl_k} (as in (6.15)), each one depending on each feasible sampling interval h_j, as indicated in (6.17):

$$\left\{\prod_{k=1}^{\infty}\Phi_{cl_k}\right\} \tag{6.17}$$

Since the closed-loop system evolution when having varying feasible sampling intervals is different (6.16) from that of classic discrete-time systems (6.10), in the controller design method we have to analyse the stability and response of these new systems (see Chapter 7).

6.2.2 Discrete-time system model with varying time delays

A continuous linear time-invariant process with a constant time delay τ is modelled by (6.18) and (6.19) [AST97].

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t-\tau) \tag{6.18}$$

$$y(t) = Cx(t) + Du(t) \tag{6.19}$$

In (6.13) τ is assumed to be less than or equal to the sampling period h [AST97]. For periodic sampling with constant sampling period h (t_k=kh), the discrete-time system can be described by (6.20) and (6.21) [AST97].

$$x(kh+h) = \Phi(h)x(kh) + \Gamma_0(h,\tau)u(kh) + \Gamma_1(h,\tau)u(kh-h) \tag{6.20}$$

$$y(kh) = Cx(kh) + Du(kh) \tag{6.21}$$

Matrices $\Phi(h,\tau)$, $\Gamma_0(h,\tau)$ and $\Gamma_1(h,\tau)$ are obtained from (6.18) and (6.19) as detailed in Equations (6.22), (6.23) and (6.24) [AST97].

$$\Phi(h) = e^{Ah} \tag{6.22}$$

$$\Gamma_0(h,\tau) = \int_0^{h-\tau} e^{As}\,ds B \tag{6.23}$$

$$\Gamma_1(h,\tau) = e^{A(h-\tau)} \int_0^{\tau} e^{As}\,ds B \tag{6.24}$$

A state space model of (6.20) and (6.21) is given by (6.25)

$$\begin{bmatrix} x(kh+h) \\ u(kh) \end{bmatrix} = \begin{bmatrix} \Phi(h) & \Gamma_1(h,\tau) \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x(kh) \\ u(kh-h) \end{bmatrix} - \begin{bmatrix} \Gamma_0(h,\tau) \\ I \end{bmatrix} \cdot u(kh) \tag{6.25}$$

Notice that *r* extra state variables u(kh-h), which represent the past values of the control signal, are introduced.

As in the previous section, to meet the closed-loop systems requirements, the system specified by (6.20) and (6.21) is controlled using state feedback (6.26) where the gain matrix *L* can be obtained using the same methods.

$$u(kh) = -L(h,\tau)\begin{bmatrix} x(kh) \\ u(kh-h) \end{bmatrix} \tag{6.26}$$

Equation (6.25) can be rewritten in terms of (6.26) as in (6.27).

$$\begin{bmatrix} x(kh+h) \\ u(kh) \end{bmatrix} = \left(\begin{bmatrix} \Phi(h) & \Gamma_1(h,\tau) \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} \Gamma_0(h,\tau) \\ I \end{bmatrix} \cdot L(h,\tau) \right) \cdot \begin{bmatrix} x(kh) \\ u(kh-h) \end{bmatrix} \tag{6.27}$$

At the end, the closed-loop time-invariant system is characterized by Equations (6.20), (6.21), and (6.26). Therefore, the closed-loop matrix (6.28) described previously in Equation (6.27) depends on $\Phi(h)$, $\Gamma_0(h,\tau)$, $\Gamma_1(h,\tau)$ and $L(h,\tau)$, which are constant matrices in terms of a constant sampling period h and a constant time delay τ.

$$\Phi_{cl} = \begin{bmatrix} \Phi(h) & \Gamma_1(h,\tau) \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} \Gamma_0(h,\tau) \\ I \end{bmatrix} \cdot L(h,\tau) \tag{6.28}$$

For discrete-time systems with a closed-loop matrix specified by (6.28), we can describe the closed-loop system evolution by (6.29)

$$\begin{array}{ll} \text{k=1} & x(h) = \Phi_{cl}\, x(0) \\ \text{k=2} & x(h+h) = \Phi_{cl}\, x(h) = \Phi_{cl}\Phi_{cl}\, x(0) = \Phi_{cl}{}^{2} x(0) \\ \text{k=3} & x(2h+h) = \Phi_{cl}\, x(2h) = \Phi_{cl}\Phi_{cl}{}^{2} x(0) = \Phi_{cl}{}^{3} x(0) \\ \text{k=4} & x(3h+h) = \Phi_{cl}\, x(3h) = \Phi_{cl}\Phi_{cl}{}^{3} x(0) = \Phi_{cl}{}^{4} x(0) \\ \ldots. & \\ \text{k} & x(kh+h) = \Phi_{cl}\, x(kh) = \Phi_{cl}\Phi_{cl}{}^{k-1} x(0) = \Phi_{cl}{}^{k} x(0) \end{array} \tag{6.29}$$

Note that in this second approximation we want the system evolution and thus the controller (to be implemented in a control task) to depend on a set of feasible sampling-actuation delays (FT)[5] rather than on a single value for the time delay (τ).

Specifically, in this case, feasible sampling-actuation delays will vary between one closed-loop execution and another. That is, at each closed-loop execution the controller parameters are adjusted according to each specific feasible sampling-actuation delay τ_j belonging to FT. Consequently, the discrete-time system is no longer time-invariant. If we denote the actual feasible sampling-actuation delay of each k^{th} closed-loop execution by $\tau_j{}^3$, we model systems described by Equations (6.20), (6.21), and (6.26) but with varying sampling-actuation delays by Equations (6.30), (6.31) and (6.32).

$$x(kh+h) = \Phi(h)x(kh) + \Gamma_0(h,\tau_j)u(kh) + \Gamma_1(h,\tau_j)u(kh-h) \tag{6.30}$$

$$y(kh) = Cx(kh) + Du(kh) \tag{6.31}$$

$$u(kh) = -L(h,\tau_j)x(kh) \tag{6.32}$$

Matrices $\Phi(h)$, $\Gamma_0(h, \tau_j)$, $\Gamma_1(h, \tau_j)$ and $L(h, \tau_j)$ are obtained from (6.22), (6.23) and (6.24) at each controller execution for each specific feasible sampling-actuation delay (τ_j, $\tau_j \in$ FT). Similarly, the state feedback controller $L(h, \tau_j)$ is obtained at each controller execution using the same control design approach as in (6.26) for the same specific feasible sampling-actuation delays.

For such systems, the closed-loop time-variant system is characterized by Equations (6.30), (6.31) and (6.32). Therefore, the closed-loop matrix that applies at the k^{th} closed-loop execution specified in (6.33) depends on $\Phi(h)$, $\Gamma_0(h, \tau_j)$, $\Gamma_1(h, \tau_j)$ and $L(h, \tau_j)$, which are varying matrices in terms of each feasible sampling-actuation delay τ_j, $\tau_j \in$ FT.

[5] Note that, as we did before with FH, here we also omit the i-subscript (which relates each set of feasible sampling-actuation delays to a control task) because it is clear in the context that we do not have to identify the control task that will implement the controller. Consequently, feasible sampling-actuation delays belonging to FT will not have the i-subscript either.

$$\Phi_{cl\,k} = \begin{bmatrix} \Phi(h) & \Gamma_1(h,\tau_j) \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} \Gamma_0(h,\tau_j) \\ I \end{bmatrix} \cdot L(h,\tau_j) \tag{6.33}$$

For discrete-time systems with a closed-loop matrix specified by (6.33), we can describe the closed-loop system evolution by (6.34)

$$\begin{aligned}
&\text{k=1} \quad x(h) = \Phi_{cl_1} x(0) \\
&\text{k=2} \quad x(h+h) = \Phi_{cl_2} x(h) = \Phi_{cl_2}\Phi_{cl_1} x(0) \\
&\text{k=3} \quad x(2h+h) = \Phi_{cl_3} x(2h) = \Phi_{cl_3}\Phi_{cl_2}\Phi_{cl_1} x(0) \\
&\ldots. \\
&\text{k} \quad x(kh+h) = \Phi_{cl_k} x(kh) = \Phi_{cl_k}\Phi_{cl_{k-1}} \ldots \Phi_{cl_2}\Phi_{cl_1} x(0)
\end{aligned} \tag{6.34}$$

The closed-loop system evolution (6.34) will depend on a product-sequence of closed-loop matrices Φ_{cl_k} (as in (6.33)), each one depending on each sampling-actuation delay τ_j, as indicated in (6.35):

$$\left\{ \prod_{k=1}^{\infty} \Phi_{cl_k} \right\} \tag{6.35}$$

Since the closed-loop system evolution when having varying feasible sampling-actuation delays (6.34) is different from that of classic discrete-time systems with constant time delay (6.29), in the controller design method we have to analyse the stability and response of these new systems (see Chapter 7).

6.2.3 Irregularly sampled discrete-time system model with varying time delays

By combining the two models described in the two previous subsections appropriately, we impose the compensation approach timing requirements, that is, to support controllers that depend on a set of feasible sampling intervals (FH) and on a set of feasible sampling-actuation delays (FT)[6]. If we denote the actual sampling interval and sampling-actuation delay of each k^{th} closed-loop execution by h_j and τ_j[4], we model systems described by Equations (6.20), (6.21), and (6.26) but with irregular sampling and varying time delays by Equations (6.36), (6.37), (6.38) and (6.39).

$$x(t_{k+1}) = \Phi(h_j)x(t_k) + \Gamma_0(h_j,\tau_j)u(t_k) + \Gamma_1(h_j,\tau_j)u(t_{k-1}) \tag{6.36}$$

$$y(t_k) = Cx(t_k) + Du(t_k) \tag{6.37}$$

$$u(t_k) = -L(h_j,\tau_j)x(t_k) \tag{6.38}$$

$$t_k = sum\ of\ all\ h_j\ already\ \ applied \tag{6.39}$$

[6] Note that here again we also omit the i-subscript because it is clear in the context that we do not have to identify the control task that will implement the controller. Consequently, feasible sampling intervals belonging to FH and feasible sampling-actuation delays belonging to FT will not have the i-subscript either.

Matrices $\Phi(h_j)$, $\Gamma_0(h_j, \tau_j)$, $\Gamma_1(h_j, \tau_j)$ and $L(h_j, \tau_j)$ are obtained from (6.22), (6.23) and (6.24), at each closed-loop execution for each specific feasible sampling interval (h_j, $h_j \in FH$) and feasible sampling-actuation delay (τ_j, $\tau_j \in FT$). Similarly, the state feedback controller $L(h_j, \tau_j)$ is obtained at each closed-loop execution using a classic discrete-time controller design method such as pole placement or optimization approach for the same specific feasible sampling intervals and feasible sampling-actuation delays (as was used in (6.26)).

For such systems, the closed-loop time-variant system is characterized by Equations (6.36), (6.37), (6.38) and (6.39). Therefore, the closed-loop matrix that applies at the k^{th} closed-loop execution specified in (6.40) depends on $\Phi(h_j)$, $\Gamma_0(h_j, \tau_j)$, $\Gamma_1(h_j, \tau_j)$ and $L(h_j, \tau_j)$, which are varying matrices in terms of each feasible sampling interval h_j, $h_j \in FH$ and each feasible sampling-actuation delay τ_j, $\tau_j \in FT$.

$$\Phi_{cl\,k} = \begin{bmatrix} \Phi(h_j) & \Gamma_1(h_j, \tau_j) \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} \Gamma_0(h_j, \tau_j) \\ I \end{bmatrix} \cdot L(h_j, \tau_j) \tag{6.40}$$

For discrete-time systems with a closed-loop matrix specified by (6.40), we can describe the closed-loop system evolution by (6.41), where t_k is given by (6.39).

$$\begin{array}{ll} \text{k=1} & x(t_1) = \Phi_{cl_1} x(0) \\ \text{k=2} & x(t_2) = \Phi_{cl_2} x(t_1) = \Phi_{cl_2} \Phi_{cl_1} x(0) \\ \text{k=3} & x(t_3) = \Phi_{cl_3} x(t_2) = \Phi_{cl_3} \Phi_{cl_2} \Phi_{cl_1} x(0) \\ .... & \\ \text{k} & x(t_k) = \Phi_{cl_k} x(t_{k-1}) = \Phi_{cl_k} \Phi_{cl_{k-1}} ... \Phi_{cl_2} \Phi_{cl_1} x(0) \end{array} \tag{6.41}$$

The closed-loop system evolution (6.41) will depend on a product-sequence of closed-loop matrices Φ_{cl_k} (as in (6.40)), each one depending on each feasible sampling interval h_j and on each feasible sampling-actuation delay τ_j, as indicated in (6.42).

$$\left\{ \prod_{k=1}^{\infty} \Phi_{cl_k} \right\} \tag{6.42}$$

Since the closed-loop system evolution with varying sampling-actuation delays is different (6.42) from that of classic discrete-time systems (6.29), in the controller design method we have to analyse the stability and response of these new systems (see Chapter 7).

6.2.4 Sequences of feasible sampling intervals and sampling-actuation delays

We have seen, therefore, that for each system (irregularly sampled discrete-time systems, discrete-time systems with varying time delays and irregularly sampled discrete-time systems with varying time delays), the closed-loop system evolution depends on a product-sequence of matrices. We can describe the system evolution at the k^{th} closed-loop execution by (6.43), which depends on a product of closed-loop matrices Φ_{cl_k}, each one depending on each specific h_j and/or τ_j.

$$x(t_k) = \Phi_{cl_j} x(t_k) = \Phi_{cl_k} \Phi_{cl_{k-1}} ... \Phi_{cl_2} \Phi_{cl_1} x(0) \tag{6.43}$$

Depending on the closed-loop implementation, feasible sampling intervals and feasible sampling-actuation delays will follow different sequences at run time. That is, the feasible

sampling intervals and feasible sampling-actuation delays sequences that will apply are combinations of values taken from sets FH and FT.

From all possible combinations, depending on the closed-loop implementation, we distinguish three types of sequences that can apply at run time:

1. a known constant feasible sampling interval (h_j, $h_j \in$ FH) with a known constant feasible sampling-actuation delay (τ_j, $\tau_j \in$ FT).
2. a finite sequence of pairs of known feasible sampling intervals taken from FH and known feasible sampling-actuation delays taken from FT, ($<(h_1,\tau_1), (h_2,\tau_2), \ldots, (h_n,\tau_n)>$, $h_j \in$ FH ,$\tau_j \in$ FT), that repeats periodically.
3. an infinite sequence of feasible sampling intervals taken randomly from FH ($<h_1, h_2, \ldots>$, $h_j \in$ FH) with an infinite sequence of feasible sampling-actuation delays taken randomly from FT ($<\tau_1, \tau_2, ...>$, $\tau_j \in$ FT).

Observe that although the three different types of sequences are specific combinations of all possible combinations of values of sets FH and FT, they cover all feasible sampling intervals and feasible sampling-actuation delays types of sequences that can appear in a closed-loop implementation.

The closed-loop system evolution when using the compensation approach depends on a product of closed-loop matrices Φ_{cl_k} (6.43), which in turn depends on the specific feasible sampling intervals and feasible sampling-actuation delays that will apply at run time. Since we distinguished three different types of feasible sampling intervals and feasible sampling-actuation delays sequences that possible closed-loop implementation produces, three different types of product-sequences of closed-loop matrices will have to be taken into account in the analysis of the compensation approach controller design method we explain in Chapter 7.

6.3 Summary

In this chapter we have defined the compensation approach as a discrete-time controller design method of designing controllers that depend on a set of a finite number of feasible sampling intervals and on a set of finite number of feasible sampling-actuation delays. The compensation approach, as we shown in this chapter, can be used to handle all possible types of closed-loop implementations regarding the guarantees that the implementation gives in terms of the sampling type (*regular* or *irregular* sampling) and time delay type (*instantaneous*, *constant* or *varying* time delay) that will apply at run time.

By imposing the compensation approach requirements on the implementation of discrete-time controllers, we have formulated the new controller design problem, for which we provide a solution in the next chapter.

Remark

Note that the case study (Section 6.1.2) we presented to characterize the closed-loop implementation effects on the controller timing parameters includes local or distributed implementations (see Section 3.1.2). For example, the different values applying for the time

delay at each closed-loop execution can be produced by network-induced delays in a distributed closed-loop implementation or by scheduling inherent jitters for a control task in a local closed-loop implementation. Therefore, the results we present in the next chapter (stability and response analysis of closed-loops systems with controllers designed with the compensation approach), which solve the control problems posed by timing variations in the control activities, can be applied for controllers implemented locally or distributed. Although in chapters 8 and 9 we apply the compensation approach control results focusing on closed-loops implemented in a node level and focusing on real-time scheduling, their applicability has also been successfully tested when closed-loops are closed over communication networks (see [MAR01c] and [YEP02] for further details).

Chapter 7

Flexible discrete-time controller design

In this chapter we address the theoretical and practical aspects of the compensation approach. As we explained in Section 6.1.1, the compensation approach is a flexible control design approach that allows us to design controllers which surpass both the equidistant sampling and actuation instants assumptions (see Section 3.1.1) on which classic discrete-time control theory (see for example [AST97] or [PHI95]) is based.

Firstly, we explain the controller design method that includes response and stability analysis, the latter focusing on these three cases. The stability analysis includes a new sufficient stability condition for irregularly sampled discrete-time systems with varying time delays in a state space formulation based on matrix algebra. Afterwards, we show some of the implications that affect the controlled system response when a process is controlled by a control tasks that adjusts its controller parameters at each controller execution according to different feasible sampling intervals and feasible sampling-actuation delays (specified at the design stage). Finally, we discuss the practical aspects of the application of the compensation approach in terms of code implementation details, computational and memory costs.

The work explained in this chapter has been partially presented in [MAR01d], [MAR01e] and [MAR02c].

7.1 Compensation approach controller design method

The compensation approach controller design method is based on the same classic controller design methods (such as pole placement or optimization approaches [AST97]) used in discrete-time control theory. However, the main difference is that in the controller design stage, for a given controller, we neither select a single (therefore constant) value (constant) for the sampling period (h) nor specify a single (therefore constant) value for the time delay (τ). We specify a finite set of feasible sampling intervals (FH) and a finite set of feasible sampling-actuation delays (FT)[1] (see Section 6.1.1). Then, at run time, the controllers parameters of the task implementing the controller are adjusted according to the different specific pairs of feasible sampling intervals and feasible sampling-actuation delays that apply at each controller execution, (h_j,τ_j). Therefore, using the compensation approach, we design controllers assuming varying sampling feasible intervals (*irregular sampling*) and varying feasible sampling-actuation delays (*varying time delays*).

[1] Note that we omit the i-subscript of FH and FT because it is clear in the context that we don't have to identify the control task that will implement the controller.

In the analysis and design of discrete-time controllers supporting a set of a finite number of feasible sampling intervals and a set of finite number of feasible sampling-actuation delays, we have to proceed in two steps: given

- the process (system to be controlled)
- the set of feasible sampling intervals (FH) and feasible sampling-actuation delays (FT)
- the type of sequence of feasible sampling intervals and feasible sampling-actuation delays guaranteed by the implementation
- the closed-loop performance specifications

we have to analyse whether

1. the closed-loop system response will meet the performance specifications and
2. the closed-loop system will be stable.

7.1.1 Closed-loop system response analysis

First of all, we have to analyse for each pair (h_j,τ_j), with $h_j \in$ FH and $\tau_j \in$ FT (pairs of feasible sampling intervals and feasible sampling-actuation) the closed-loop system response resulting from its use. That is, by using a particular classic discrete-time controller design methodology (specified in terms of the sampling period and time delay), we have to study whether we can locate the closed-loop poles in such a way that the different system responses (resulting from the use of each pair for the former sampling period and time delay) meet the closed-loop performance specifications.

Note that in the compensation approach, although we are able to use the same controller design methodologies as classic discrete-time control theory (such as pole placement or optimization approach), we have to be aware that for each feasible sampling interval, when updating the controller parameters at each controller execution (which results in a new controller $L(h_j,\tau_j)$ at each controller execution, see Section 6.2.3), we have different closed-loop pole locations. This implies that for each closed-loop poles location, we have different closed-loop system responses, which means being able to meet different performance specifications. Moreover, for each feasible sampling-actuation delay we have a different delay in the closed-loop system response.

Extensive simulations have shown that if all the system responses we obtain for all possible combinations of specific h_j and τ_j are good enough (in terms of meeting the set of closed-loop performance specifications), the closed-loop system response we will obtain from the run time controller parameter adjustment will still fulfil these requirements.

Although it is beyond the scope of this work to characterise the closed-loop system response mathematically when the controller parameters are adjusted at each closed-loop execution, a short qualitative analysis is given next. Given the different closed-loop system responses corresponding to all possible combinations of specific feasible sampling intervals and specific feasible sampling-actuation delays, if we change the controller parameters at each execution according to any (h_j,τ_j), what we are doing is driving the closed-loop system according to each particular controller $L(h_j,\tau_j)$ at each closed-loop execution. Therefore, as a

result, the final closed-loop system response we obtain is a mixture of all the closed-loop system responses we obtained for each pair of all possible combinations of feasible sampling intervals and feasible sampling-actuation delays (belonging to sets FH and FT).

7.1.2 Stability analysis

Using the compensation approach, looking at the system evolution (Equation 6.43), the stability of the system depends on a product of closed-loop matrices Φ_{cl_k}. In Section 6.2.4 we identified three different types of feasible sampling intervals and feasible sampling-actuation delays sequences that can appear in a closed-loop implementation, which produces three different types of product-sequences of closed-loop matrices. For the stability analysis, we specify three cases depending on the type of sequence that the closed-loop implementation originates.

For systems following sequence 1 (in Section 6.2.4), the following *Case 1* applies.

***Case 1*:** the closed-loop system will be characterized by only one closed-loop matrix, Φ_{cl_k} (which depends on a specific single h_j and τ_j). This is the classic case for discrete-time control systems. That is, for systems with a constant sampling period h (which in this case is the specific h_j) and a constant time delay τ (which in this case is specific τ_j). In this case, the system evolution at the k^{th} closed-loop execution specified in (6.43) can be re-specified as follows (7.1), where Φ_{cl} is the closed-loop matrix we specified in (6.28):

$$x(kh+h) = \Phi_{cl}\, x(kh) = \Phi_{cl}\, \Phi_{cl}^{\,k-1} x(0) = \Phi_{cl}^{\,k} x(0) \tag{7.1}$$

As explained in [AST97], if it is possible to diagonalize Φ_{cl}, then the solution of $x(kh+h) = \Phi_{cl}^{k} x(0)$ is a combination of terms λ_i^k, where λ_i, i=1, ..., n are the eigenvalues of Φ_{cl}. In the general case, when Φ_{cl} can not be diagonalized, the solution is instead a linear combination of the terms $p_i(k)\lambda_i^k$, where $p_i(k)$ are polynomials in k of order one less than the multiplicity of the corresponding eigenvalue. To have asymptotic stability, all solutions must go to zero as k increases to infinity. The eigenvalues of Φ_{cl} then have the property (7.2):

$$|\lambda_i| < 1 \quad i=1,\ldots, n \tag{7.2}$$

Consequently, in this case, the stability condition can be formulated as follows: the system is stable iff the spectral radius of the closed-loop matrix Φ_{cl} is less than one (7.3).

$$\text{Stable} \Leftrightarrow \rho(\Phi_{cl}) < 1 \tag{7.3}$$

where the spectral radius ρ of a matrix A is defined as in (7.4).

$$\rho(A) = \max\{|\lambda| \mid \lambda \text{ is an eigenvalue of } A\} \tag{7.4}$$

For systems following sequences 2 or 3 (in Section 6.2.4) (note that these systems are no longer time-invariant and Equations (6.36), (6.37), (6.38) and (6.39) can describe them), we can apply *Case 2 and Case 3*.

***Case 2*:** in this case, the closed-loop system will be characterized by a known, finite set of matrices that repeats periodically ($<\Phi_{cl_1}, \Phi_{cl_2}, ..., \Phi_{cl_n}>$, Φ_{cl_k} is the closed-loop matrix that depends on each specific pair (h_j,τ_j), $h_j \in FH$, $\tau_j \in FT$ that will appear at each controller execution). Therefore, a known repeating sequence of known matrices will apply. In this case, the stability test can be performed by checking the stability of the product of the repeating sequence of matrices (7.5), as presented in [DOG95].

$$\text{Stable} \Leftrightarrow \rho(\Phi_{cl_1} \cdot \Phi_{cl_2} \cdot ... \cdot \Phi_{cl_n}) < 1 \qquad (7.5)$$

***Case 3*:** in this case, the closed-loop system will be characterized by a product of an infinite number of matrices ($<\Phi_{cl_1}, \Phi_{cl_2}, ...>$, Φ_{cl_k} is the closed-loop matrix that depends on each specific pair (h_j,τ_j), $h_j \in FH$, $\tau_j \in FT$ that will appear at run time) taken randomly from a finite set of matrices, which we specify in (7.6).

$$\Omega = \{\Phi_{cl_k} \mid \Phi_{cl_k} \text{ is the } k^{th} \text{ closed-loop matrix that depends on } (h_j,\tau_j),\ h_j \in FH,\ \tau_j \in FT, \text{ for all possible combinations of } (h_j,\tau_j)\} \qquad (7.6)$$

In this case, the stability test (7.5) cannot be used. However, [DOG95], in corollary 2, also gives the necessary and sufficient stability condition (where inequalities are in the sense of positive or negative definiteness [STR80]), as in (7.7)

$$\Omega \text{ asymptotically stable} \Leftrightarrow \exists\ P>0:\ \Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0,\ \forall \Phi_{cl_k} \in \Omega^k,\ k \geq 1 \qquad (7.7)$$

Explanation of (7.7): all power sets of the closed-loop matrix set Ω (combining them two by two, three by three, and so on) should be checked to see whether there is a positive definite matrix P in such a way that satisfies all closed-loop matrices, $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$.

Note that the application of this condition is not easy in terms of computability. The use of a linear matrix inequalities (LMI) solver [GAH95] can be helpful. However, to make it easier to analyse the stability of systems that fall into *Case 3*, in the following we list two sufficient although not necessary stability conditions that can also be applied. The first (7.8) was presented in [DOG95], corollary 1, and also needs the use of a LMI solver. The second (7.9) was presented in [MAR01b] and does not require the use of a LMI solver. However, it is more conservative than the previous one:

$$\text{If } \exists\ P>0:\ \forall\ \Phi_{cl_k} \in \Omega,\ \Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0 \Rightarrow \Omega \text{ asymptotically stable} \qquad (7.8)$$

Explanation of (7.8): if there is a positive definite matrix P>0 such that it verifies that $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$ for all closed-loop matrices $\Phi_{cl_k} \in \Omega$, then, the set of matrices Ω is asymptotically stable, which means that the discrete-time systems will be stable.

$$\text{If } \forall \Phi_{cl_k} \in \Omega,\ \Phi^T_{cl_k} \cdot \Phi_{cl_k} - I < 0 \Rightarrow \Omega \text{ asymptotically stable} \qquad (7.9)$$

Explanation of (7.9): if each matrix Φ_{cl_k} satisfies that all eigenvalues of $(\Phi^T_{cl_k} \cdot \Phi_{cl_k} - I)$ are less than zero (each $(\Phi^T_{cl_k} \cdot \Phi_{cl_k} - I)$ is negative definite), each matrix Φ_{cl_k} will guarantee immediate decay

(so that $\|x_{k+1}\| = \|\Phi_{cl_k} x_k\| < \|x_k\|$ for all non-zero starting vectors x_k). In this case, any product of an infinite number of such matrices Φ_{cl_k} will guarantee stability in the system.

Proof. of (7.9): If $eig(\Phi^T_{cl_k} \cdot \Phi_{cl_k} - I) < 0$ (that is, $\Phi^T_{cl_k} \cdot \Phi_{cl_k} - I$ is negative definite) then, for negative definite matrix definition, for all x_k non-zero vectors, it holds that $x_k^T(\Phi^T_{cl_k} \cdot \Phi_{cl_k} - I)\, x_k < 0$.

From $x_k^T(\Phi^T_{cl_k} \cdot \Phi_{cl_k} - I)\, x_k < 0$ and performing some operations:

$$x_k^T(\Phi^T_{cl_k} \cdot \Phi_{cl_k} - I)x_k < 0$$

$$(x_k^T\Phi^T_{cl_k} \cdot \Phi_{cl_k} - x_k^T)x_k < 0$$

$$x_k^T\Phi^T_{cl_k} \cdot \Phi_{cl_k} x_k - x_k^T x_k < 0$$

we obtain

$$x_k^T\Phi^T_{cl_k} \cdot \Phi_{cl_k} x_k < x_k^T x_k$$

Using the Euclidean norm definition ($\|x_k\| = (x_k^T x_k)^{1/2}$), the following holds:

$$\|\Phi_{cl_k} x_k\|^2 = (\Phi_{cl_k} x_k)^T \Phi_{cl_k} x_k = x_k^T\Phi^T_{cl_k} \cdot \Phi_{cl_k} x_k < x_k^T x_k = \|x_k\|^2$$

Therefore, $\|\Phi_{cl_k} x_k\| < \|x_k\|$

Observe that systems described by Equations (6.11), (6.12), (6.13) and (6.14), that is, systems with varying feasible sampling intervals and zero time delay, are a specific case of systems described by (6.36), (6.37), (6.38) and (6.39), that is, systems with varying feasible sampling intervals and varying feasible sampling-actuation delays (when all $\tau_j \in FT$ are equal to zero). Therefore, the stability analysis of *Cases 2 and 3* also applies to them.

Similarly, systems described by Equations (6.30), (6.31) and (6.32), that is, systems with constant sampling period but with varying feasible sampling-actuation delays, are specific cases of systems described by (6.36), (6.37), (6.38) and (6.39), that is, systems with varying feasible sampling intervals and varying feasible sampling-actuation delays (when all $h_j \in FH$ are equal to the specific value for the sampling period). Therefore, the stability analysis of *Cases 2 and 3* also applies to them.

The presented stability analysis has covered three types of product-sequences of matrices that may appear if controller parameters are adjusted at each closed-loop execution according to the specifics pairs of feasible sampling intervals and feasible sampling-actuation delays belonging to sets FH and FT. Since the three types of sequences cover all possible combinations of feasible sampling intervals and feasible sampling-actuation delays that can appear at run time, the three *cases* provide a complete stability analysis.

7.1.3 Summary

If all the system responses we obtain for the use of each pair of feasible sampling intervals and feasible sampling-actuation delays (as we explained in Section 7.1.1) fulfil the closed-

loop performance requirements and the system is stable (as we explained in Section 7.1.2), we have found the controller we were looking for. Note that in this case, the controller obtained depends on a finite set of feasible sampling intervals (FH) and on a finite set of feasible sampling-actuation delays (FT). If we do not meet the closed-loop requirements in terms of stability or closed-loop system response, we can either change the specifications, the design methodology, sets FH and/or FT, or re-specify the whole control problem.

7.1.4 Example

In this section we use an example to illustrate the application of the compensation approach design method. The system to control is the inverted pendulum (Section 2.2.2). Recall that the goal of our controller is to maintain the desired vertical position of the inverted pendulum at all times. For this example, the performance specification is to recover from a perturbation (modelled by a pulse) in less than one second. That is, the settling time is 1s. Let us assume that the set of feasible sampling intervals and set of feasible sampling-actuation delays that we specify at the design stage are FH={50,60} (in ms) and FT={10,20} (in ms) respectively.

Design

First of all, we analyse whether we can locate the closed-loop poles in such a way that we obtain different system responses for all possible combinations of feasible sampling intervals and feasible sampling-actuation delays that meet the closed-loop performance requirements (as explained in Section 7.1.1). By setting ω_n=8rad/s and ζ =0.6 (see Sections 2.2.1 and 2.2.2), and using pole placement observer design [AST97], we obtain a location for the closed-loop poles for each feasible sampling interval of FH in such a way that, taking into account each feasible sampling-actuation delay of FT, we meet the specified performance requirements.

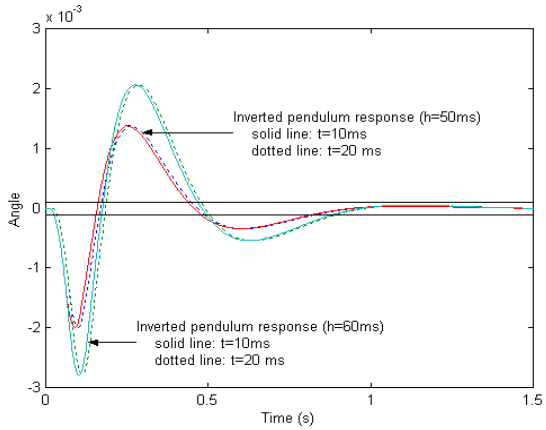


Figure 7.1. Inverted pendulum responses if the controller is characterized each time for one of all possible combinations of feasible sampling intervals and feasible sampling-actuation delays.

In Figure 7.1 we show the four closed-loop system responses we have obtained if the controller (which has been classically designed through pole placement with observer, see Section A3 in Appendix A for further details)) is characterized each time for one of all possible pairs (four pairs) of combinations of feasible sampling intervals and feasible sampling-actuation delays taken from sets FH and FT ((50,10), (50,20), (60,10) and (60,20)). All responses meet the performance specifications, that is, the inverted pendulum recovers from the perturbation in less than 1 second.

For the stability analysis, for illustrative purposes, we assume that at run time, we will have two different implementations, which give two types of sequences of feasible sampling intervals and feasible sampling-actuation delays (Section 6.2.4). In this way, we can apply two different stability *cases* of Section 7.1.2, *Case 2* and *Case 3* (note that in this example we are not interested in *case 1*, because it corresponds to the classic stability case, given by Equation 7.3).

Example 1: Let us assume that the implementation we have gives a finite sequence of pairs of known feasible sampling intervals and feasible sampling-actuation delays taken from FH and FT, which is of sequence type *2* (Section 6.2.4). Specifically, the periodic sequence we have is: (<(60,20), (50,10), (50,20), (60,10)>, in ms).

Therefore, we have to apply the stability analysis given in *Case 2* (Section 7.1.2). The closed-loop system is characterized by a known, finite set of matrices that repeats periodically: $<\Phi_{cl_1}, \Phi_{cl_2}, \Phi_{cl_3}, \Phi_{cl_4}>$ where Φ_{cl_k} is the closed-loop matrix that depends on each pair (h_j,τ_j), $h_j \in FH$,$\tau_j \in FT$ we listed before. The stability test is performed by checking the stability of the product of the repeating sequence of matrices (Equation 7.5, that is: Stable $\Leftrightarrow \rho(\Phi_{cl_1} \cdot \Phi_{cl_2} \cdot \Phi_{cl_3} \cdot \Phi_{cl_4}) < 1$).

Applying this test, we conclude that the system is stable because $\rho(\Phi_{cl_1} \cdot \Phi_{cl_2} \cdot \Phi_{cl_3} \cdot \Phi_{cl_4}) = 0.349 < 1$ (see Section B1 in Appendix B for further details).

Example 2: The implementation we have gives an infinite sequence of feasible sampling intervals and feasible sampling-actuation delays taken randomly from FH and FT, which is of sequence type *3* (Section 6.2.4).

Therefore, we have to apply *case 3* for the stability analysis (Section 7.1.2). The closed-loop system is characterized by a product of an infinite number of matrices ($<\Phi_{cl_1}, \Phi_{cl_2}$, ...>, where Φ_{cl_k} is the closed-loop matrix that depends on each pair (h_j,τ_j), $h_j \in FH$,$\tau_j \in FT$) taken randomly from a set of finite number of matrices, specified in (7.5). Set Ω will have the four closed-loop matrices that we had before in the previous example (note that in this case, all the possible combinations that characterize each closed-loop matrix coincide with the combinations we had in the previous example): $\Omega=\{\Phi_{cl}(50,10), \Phi_{cl}(50,20), \Phi_{cl}(60,10), \Phi_{cl}(60,20)\}$.

In this case, applying the sufficient stability condition given in (7.9) (that is: if $\exists\ P>0$: $\forall\ \Phi_{cl_k} \in \Omega$, $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0 \Rightarrow$ Stable), we conclude that the

set of matrices Ω fulfils the condition. That is, we obtain a matrix P>0 such as for all four closed-loop matrices Φ_{cl_k} of Ω, $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$ (see Section B2 in Appendix B for further details):

$$P' = \begin{bmatrix} 2.3229543 & 0.3526428 & 3.0679626 & 0.6544445 & -0.0004749 \\ 0.3526428 & 0.1037981 & 0.4604631 & 0.1920243 & -0.0000580 \\ 3.0679626 & 0.4604631 & 4.7176875 & 0.8626853 & -0.0002955 \\ 0.6544445 & 0.1920243 & 0.8626853 & 0.3585137 & -0.0001056 \\ -0.0004749 & -0.0000580 & -0.0002955 & -0.0001056 & 0.0000008 \end{bmatrix}$$

$$P=P'\cdot 1.0e+004$$

Therefore, no matter which of the four matrices apply at each controller execution, the closed-loop system that includes the inverted pendulum and the controller designed using the compensation approach for sets FH={50,60} (in ms) and FT={10,20} (in ms) will be stable.

In summary, we have obtained the controller we were looking for because all the system responses we obtained fulfil the closed-loop performance requirements and the system, for both implementation case examples, is stable.

Implementation

Finally, we show what kind of response we obtain by doing the run time parameters adjustment. The actual response we obtain when controlling the inverted pendulum with this control task performing the run time parameters adjustment can be seen in Figure 7.2.

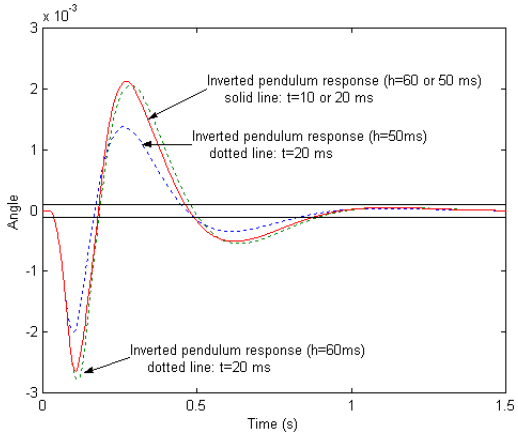


Figure 7.2. Two of all the possible responses of Figure 7.1 (dotted) and the compensated (solid curve)

It can be seen in Figure 7.2 that the thick-solid line is the actual response we obtain with the control task performing the run time parameters adjustment. It is a combination of the four possible ones we gave as acceptable in the control design. It can also be seen that the response obtained by the compensation approach still meets the control performance requirements.

For this case, at each controller execution, the feasible sampling interval and feasible sampling-actuation delay that applied was chosen randomly from sets FH and FT. To implement a controller designed under the compensation approach, the code resulting from implementing a control law designed using traditional discrete-time controller design methodologies (assuming constant sampling period and constant time delay) is slightly modified in order to allow the run time parameters adjustment (see Section 7.2.2 for an overview and Section A4 in Appendix A for a more detailed code). That is, at run time, at each control task instance execution, different feasible sampling intervals (FH={60, 70, 80, 90, 100}), and different feasible sampling-actuation delays (FT={20,30}) apply and are used to update the controller parameters.

7.2 Practical implementation considerations

In this section we give details on how to implement the control computation in order to allow the run time parameters adjustment. First of all, we discuss when feasible sampling intervals and feasible sampling-actuation delays are needed by control tasks for the run time parameters adjustment on which the compensation approach is based. Additionally we show how the code of the control task should be implemented to account for these varying feasible sampling intervals and feasible sampling-actuation delays.

In addition, since the parameters adjustment is performed at run time, we investigate the implementation cost. This leads us to assess what the computational overhead and memory requirements are in order to apply the run time controller parameters adjustment. With respect to the implementation cost, at each control task instance execution the controller parameters must be updated according to each specific feasible sampling interval and feasible sampling-actuation delay. Two strategies apply: run time or offline calculations.

If the controller parameter adjustment is performed by online extra calculations, the introduced computational overhead will depend on the process (the system to be controlled), the control design method and controller design strategy being used. If the computational overhead is not negligible, the controller parameter adjustment can be performed online by accessing offline pre-calculated look-up tables. These tables will contain the necessary parameters to allow the control computation parameters to be adjusted according to the different feasible sampling intervals and/or feasible sampling-actuation delays that apply at run time. In this case, the memory required to store these tables must be assessed. We will estimate the size of the tables, which depends on both the design method and the controller design strategy.

7.2.1 Temporal information required for the controller parameters adjustment

At the implementation level, when applying the compensation approach, control tasks require that at run time we know the *controller timing parameters*, i.e., feasible sampling intervals and feasible sampling-actuation delays, for the parameters adjustment. As we have seen in Section 6.2.3, a generic state feedback controller designed using the compensation approach to be implemented in a control task (for irregularly sampled discrete-time systems

with varying time delays) depends on each particular feasible sampling interval and feasible sampling-actuation delay that appears at run time, $L(h_j, \tau_j)$ (see Equation 6.38 in Section 6.2.3). This means that the implementation of the control strategies used in the compensation approach requires each h_j and/or τ_j to be known at the beginning of each control task instance execution.

Consequently, in this section, we firstly address which type of controller timing parameters are needed for the run time parameters adjustment, which depends on the implementation guarantees on the controller timing parameters (see Section 6.1.2). Secondly, we identify who, at run time, is in charge of providing or obtaining each value of the required controller timing parameters, and finally we discuss what requirements/restrictions the application of the compensation approach imposes on the implementation.

In Section 6.1.2, we reviewed the six different cases we can have in the implementation of closed-loop systems, which we reduced in Section 6.1.3 to three cases without losing generality. In the following, for each of these three cases, we analyse which controllers timing parameters the control tasks need in order to readjust their controller parameters at run time[2]. Note that since the three cases cover all the others (as we discussed in Section 6.1.3), the following study is complete:

- In Case 4 (Section 6.1.2), the closed-loop implementation cannot guarantee a constant sampling period, thus preventing the application of classic discrete-time control theory. For this case, a control computation implementing a design obtained using the compensation approach (based on the irregularly sampled discrete-time system model specified in Section 6.2.1) can solve the problem. By specifying at the design stage a set FH including all possible values that can apply at run time for the sampling period, at each instance execution, the control computation can adjust its controller parameters according to each specific h_j that applies. In this case, control computations only need to know each different h_j at their execution start times.
- In Case 3 (Section 6.1.2), the closed-loop implementation cannot guarantee a constant time delay, thus preventing the application of classic discrete-time control theory. For this case, a control computation implementing a design obtained using the compensation approach (based on the regularly sampled discrete-time system model with varying time delays specified in Section 6.2.2) can solve the problem. By specifying at the design stage a set FT including all possible values that can apply at run time for the time delay, at each instance execution, the control computation can adjust its controller parameters according to each specific τ_j that applies. In this case, control computations only need to know each different τ_j at their execution start times.
- In Case 6 (Section 6.1.2), which is a combination of case 3 and 4, the implementation cannot guarantee a constant sampling period and constant time delay, thus preventing the application of classic discrete-time control theory. For this case, a control

[2] For a generic controller obtained through the compensation approach, the *controller timing parameters* are the feasible sampling intervals and feasible sampling-actuation delays (h_j,τ_j) while the *controller parameters* are all the other parameters that depend on h_j, and τ_j. If we denote the controller (as in Equation 6.38 in Section 6.2.3) by $L(h_j,\tau_j)$, L itself, the gain matrix, is the controller parameters. For each pair (h_j,τ_j), L will have different values.

computation implementing a design obtained using the compensation approach (based on the irregularly sampled discrete-time system model with varying time delays specified in Section 6.2.3) can solve the problem. By specifying at the design stage sets FH in FT including all possible values that can apply at run time for the sampling period and time delay, at each instance execution, the control computation can adjust its controller parameters according to each specific pair (h_j,τ_j) that applies. In this case, control computations need to know each pair (h_j,τ_j) at their execution start times.

In summary, the type of timing parameter (feasible sampling interval and/or feasible sampling-actuation delay) that the control computation requires for the controller parameters adjustment varies, depending on the closed-loop implementation. Next, we discuss for each type of timing parameter, the different implementation possibilities we have in order to obtain their precise value at each control task instance start time execution.

Feasible sampling interval: each h_j that applies at run time can be easily obtained by online time measurements carried out by the control task at each sampling instant. That is, at the beginning of the current instance, at the sampling time instant, we have to measure the time elapsed from the previous sampling instant. This is illustrated in Figure 7.3, where the current instance (the k^{th} instance), at its execution start time (that may or not correspond to the sampling time instant), uses the measured h_j.

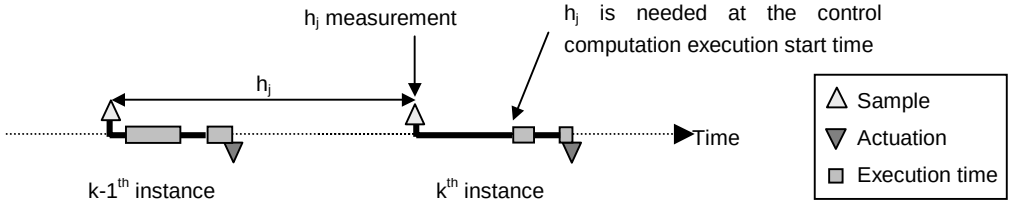


Figure 7.3. Feasible sampling interval measurement

Therefore, in this case, the application of the compensation approach does not impose any specific restriction on the implementation apart from guaranteeing that the time intervals between consecutive sampling instants will occur at run time according to the feasible sampling intervals specified at the design stage in FH.

Feasible sampling-actuation delay: each τ_j, in the general case, cannot be obtained by online time measurements. As we pointed out in the previous section, for the parameters recalculation, the control computation needs to know -at its execution start time- for the parameters recalculation, the elapsed time from its sampling instant to its actuation completion time instant.

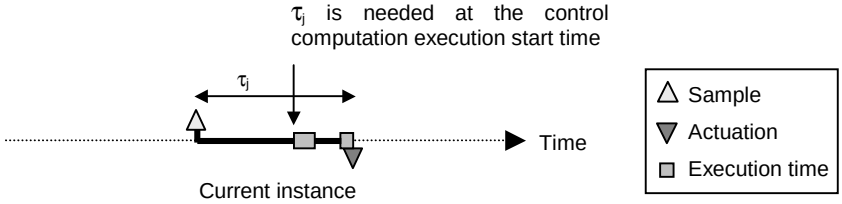


Figure 7.4. Feasible sampling-actuation delay measurement

However, in the general case, at the sampling instant, it is physically impossible to measure the time elapsed from sampling to actuation. Figure 7.4 illustrates this problem. For each current instance, the sampling-actuation delay (τ_j) must be known by the control computation at its execution start time. But, since the actuation completion still has to occur, the time elapsed from sampling to actuation cannot be measured.

Note that we have said that in the general case, the sampling-actuation delay cannot be measured at run time. There is a specific case in which the sampling-actuation delay can indeed be measured online by the control computation: i.e., if we consider at the controller design stage that the control computation execution time (and the actuation) is not significant in terms of time delay (see Section 3.11). In this case, the sampling-actuation delay is either zero (see Figures 6.1 and 6.11 in Section 6.1.2) if the sampling and actuation are assumed to execute at the same time instant, or it can be obtained through online time measurements by the control computation (see Figures 6.7 and 6.17 in Section 6.1.2). In the latter case, since control computation execution time and actuation are not significant in terms of time delay, the elapsed time from the sampling instant to the actuation instant can be measured online and used for the control computation, thus not imposing any tight restriction on the implementation apart from guaranteeing that the time intervals between pairs of sampling instant/actuation instants of the same control task instance will occur at run time according to the feasible sampling-actuation delays specified at the design stage in FT.

However, apart from the previous specific scenario, in general, since the time elapsed from related sampling instants and actuation instants can not be measured online by the control task at each instance execution and thus used at the control computation execution start time, the application of the compensation approach requires other implementation strategies which go beyond time measurements. This can be achieved by the application of scheduling techniques (as we discuss in Chapter 8), and making the scheduler responsible for providing at each control task instance execution the exact time that will elapse from the current sampling to actuation. Note that this transfers to the scheduling approach the responsibility of guaranteeing that the time intervals between related sampling and actuation time instants will occur at run time according to the feasible sampling-actuation delays specified at the design stage in FT.

7.2.2 Controller parameters adjustment code implementation details

In this section, we will show how the control computation implements a controller obtained through the compensation approach controller design method (Section 7.1). Recall that the compensation approach controller design method is based on the same design methods (such as pole placement or optimization approaches [AST97]) used in classic discrete-time controller design with the particularity that the controller parameters are adjusted at each control task instance execution according to specific pairs of feasible sampling intervals and feasible sampling-actuation delays (belonging to sets FH and FT specified in the controller design stage). The *controller parameters* [2] refer to all parameters that influence the calculation of the control signal.

For classic computer control, the calculation of the control signal given by the controller $L(h,\tau)$ (see Equation 6.26 in Section 6.2.2) depends on the sampling period and time delay assumed in the controller design stage. Using the compensation approach, the control signal given by the controller $L(h_j,\tau_j)$ (see Equation (6.38) in Section 6.2.3) depends on each specific feasible sampling interval and feasible sampling-actuation delay that apply at run time at each controller execution. Therefore, the code of a control task implementing a design obtained through the compensation approach must allow the controller to be updated at each control task instance execution.

Figure 7.5 (left) gives the generic code of a controller obtained using classic discrete-time controller design methods, and Figure 7.5 (right) gives the generic code obtained using the compensation approach controller design method. In Figure 7.5, $y(t_k)$ and $r(t_k)$ are, respectively, the output of a controlled process and reference signals at the sampling instants, from which the error ($e(t_k)$) is calculated. Observe however that in the calculation of the control signal, $u(t_k)$, in the *Generic controller*, the gain matrix $L(h,\tau)$ is constant, while in the *Compensation approach controller*, the gain matrix $L(h_j,\tau_j)$ has to be updated according to the specific pair (h_j,τ_j) that applies at each control task instance execution and that has to be adequately obtained (Section 7.2.1). Recall that the h_j and τ_j that apply at run time belong to the set of feasible sampling intervals (FH) and to the set of feasible sampling-actuation delays (FT) specified at the design stage. Finally, in both codes, the actuation takes place (represented by *write_output* ($u(t_k)$)).

```
Generic controller
{     read_inputs (y(tk), r(tk));
      e(tk) = r(tk) – y(tk);
      u(tk) = calculate_output (L(h,τ),e(tk));
      write_output (u(tk));
}
```

```
Compensation approach controller
{     read_inputs (y(tk), r(tk));
      obtain (hj, τj);
      e(tk) = r(tk) – y(tk);
      obtain (L(hj, τj))
      u(tk) = calculate_output (L(hj, τj),e(tk));
      write_output (u(tk));
}
```

Figure 7.5. Left - generic controller code. Right - compensation approach controller code

Although the gain matrix L is the generic *controller parameter* that has to be updated for the application of the compensation approach, depending on (see Section 2.2)

- the system to be controlled (process),
- the control design method (e.g., PID or pole placement controller design methods) and
- the controller design strategy that is being used (*discrete-time controller* design or *discretization of continuous time controller* design),

the actual controller parameters to be updated appear in many different ways. Therefore, in the following, using the two examples we introduced in Section 2.2.2, we show the specific code modification required for the application of the compensation approach. We first

consider the usual control computation and the computation adjusting for feasible sampling intervals and/or feasible sampling-actuation delays (using online calculations, see Section 7.2.3 for further discussion).

Example 1

The first example (Section 2.2.2, Example 1) is for a discrete PID controller (obtained through a discretization of a continuous time design), now compensating (only) for varying feasible sampling intervals. Note that in Section 2.2.2, no time delay was taken into account in the discretization of the continuous time designed PID controller. The reason is that in practice, PID controllers are designed without time delays due to the fact that their execution time is not considered relevant for controlling purposes.

As we explained in Example 1 in Section 2.2.2, for a discretization of a continuous time designed PID controller, at each execution of a task $task_{PID}$ (Figure 7.6 left) the usual computations involve the calculation of the three actions (p_k, i_k, d_k) according to the current error ($e(t_k)$) in order to obtain the control signal ($u(t_k)$). Note that the integral (i_k) and the derivative (d_k) actions depend on the sampling period h. The reference signal is $r(t_k)$ and the controlled variable is $y(t_k)$.

```
task_PID
{   read_inputs (y(t_k), r(t_k));
    e(t_k) = r(t_k) – y(t_k);
    u(t_k) = calculate_output (p_k(e(t_k)), i_k(h,e(t_k)), d_k(h,e(t_k)));
    write_output (u(t_k));
}
```

```
task_CPID
{   read_inputs (y(t_k), r(t_k));
    e(t_k) = r(t_k) – y(t_k);
    obtain(h_j);
    u(t_k) = calculate_output (p_k(e(t_k)),i_k(h_j,e(t_k)),d_k(h_j,e(t_k)));
    write_output (u(t_k));
}
```

Figure 7.6. Top - Classic PID code. Bottom - Compensated PID code

However, if the same task executing this code has to update its controller parameters according to each specific feasible sampling interval h_j as imposed by the compensation approach, the required code modification is shown in Figure 7.6 right (named $task_{CPID}$). The only difference between the task codes in Figure 7.6 is that in the modified code (marked in bold) h_j has to be obtained at each instance execution because this value is used in the rest of the calculations. However, regardless of whether it is h_j (in Figure 7.6 right) or h (in Figure 7.6 left), the amount of computations to obtain the control signal ($u(t_k)$) is the same. For a more detailed code, see Section A1 and A2 in Appendix A.

Example 2

The second example (Section 2.2.2, Example 2) is the state feedback controller (where the gain matrix is obtained by state feedback using pole placement observer discrete design), now compensating for both varying feasible sampling intervals and varying feasible sampling-actuation delays. For the sake of clarity, we omit the observer part. However, for a more detailed code including the observer part, see Sections A3 and A4 in Appendix A.

As we explained in Example 2 in Section 2.2.2, in this case, at each control task instance execution implementing a state feedback controller obtained through pole placement design (Figure 7.7 top, named task $task_{sfc}$), the usual computation involves the calculation of the control signal ($u(t_k)$) according to the state ($x(t_k)$), reference signal ($r(t_k)$) and gain matrix $L(h,\tau)$, apart from updating the state for the next controller execution according to the current state, output, input ($y(t_k)$), closed-loop matrices ($\Phi(h, \tau)$, $\Gamma(h, \tau)$ and C). Note that the gain matrix and the closed-loop matrices, which depend on the sampling period h and time delay τ, are fixed parameters of the controller algorithm, calculated at the design stage, because h and τ are supposed to remain constant at run time.

```
task_sfc
{   read_input (y(t_k), r(t_k));
    u(t_k) = calculate_output (x(t_k), -L(h,τ), r(t_k));
    write_output (u(t_k));
    x(t_k+1) = update_state (x(t_k), Φ(h, τ), Γ(h, τ), C, u(t_k), y(t_k ));
}
```

```
task_Csfc
{   read_input (y(t_k), r(t_k));
    obtain (h_j , τ_j) ;
    (Φ(h_j, τ_j), Γ(h_j, τ_j)) = system _discretization (A, B, h_j, τ_j)) ;
    L(h_j , τ_j) = Controller_design(pole_placement, Φ(h_j, τ_j), Γ(h_j, τ_j));
    u(t_k) = calculate_output (x(t_k), -L(h_j , τ_j), r(t_k));
    write_output (u(t_k));
    x(t_k+1) = update state (x(t_k), Φ(h_j, τ_j), Γ(h_j, τ_j), C, u(t_k), y(t_k));
}
```

Figure 7.7. Top - Classic state feedback controller code designed using pole placement. Bottom – Compensated state feedback controller code designed using pole placement

However, if the same task executing this code has to update its controller parameters according to each specific feasible sampling interval, h_j, and feasible sampling-actuation delay, τ_j, as imposed by the compensation approach, the required code modification is shown in Figure 7.7 (bottom), named $task_{Csfc}$). We can see that the extra calculations (marked in bold) in the readjusted controller can be important. First of all, we obtain h_j and τ_j. Afterwards, we have to recalculate the discretization of the system model ($\Phi(h_j, \tau_j)$, $\Gamma(h_j,$

τ_j)) because the controller has been designed in the discrete-time domain, that is, from a discretization of the inverted pendulum model. In addition, since the location of the closed-loop poles depends on the sampling period (Section 2.2.2), to have varying feasible sampling intervals implies recalculating the gain matrix using pole placement design ($L(h_j, \tau_j)$) given the same performance specifications. In summary, in this case, the discretization of the system model and the controller design are the main recalculations required in order to apply the compensation approach (see Sections A3 and A4 in Appendix A for a more detailed code and Appendix C for the closed loop matrices that have to be recalculated at each controller execution for the inverted pendulum case)

7.2.3 Computational overhead

In the previous section we have seen that the extra calculations of each controller designed with the compensation approach depend on the controller design method (PID or state feedback pole placement observer design) and the control strategy (discretization of a continuous design or discrete design). For the PID, the computational overhead is insignificant because only the new computation is *obtain(h_j)*. However, for the state feedback controller, the computational overhead may be significant. Therefore, we evaluate it precisely now, applied to the inverted pendulum example (described in Section 2.2.2).

To control the angle of the inverted pendulum, we have obtained a state feedback controller from the discrete-time pole placement observer design approach using Ackerman's formula [AST97]. In this case, we estimate the computational overhead (which includes a recalculation of a matrix inverse and the evaluation of the closed-loop matrix characteristics polynomial on a (*n x n)* matrix) of the control task $task_{Csfc}$ (Section 7.2.2) performing the extra-calculation of the compensation approach to be $O(n^4)$, where *n* is the closed-loop system matrix dimension. For the example of the inverted pendulum model (4x4 matrix) we introduced in Section 2.2.2 and for a simplified model (2x2 matrix) of the same system (see [MAR01a]), Table 7.1 details the approximate numbers of flops, obtained via simulations using Matlab™ for each control task instance execution (executing the code of the task $task_{Csfc}$ in Section A4 of Appendix A), with and without the extra-calculation necessary to update the controller parameters according to feasible sampling intervals and feasible sampling-actuation delays.

	Number of Flops (without extra calculations)	Number of Flops (with extra calculations)
simplified pendulum (*2x2 matrix*)	25	250
example pendulum (*4x4 matrix*)	60	2000

Table 7.1. Computational overhead

Table 7.1 shows that the overhead of extra calculations required to apply the compensation approach with the online recalculations strategy may be too high (sometimes, orders of magnitudes higher) depending, for example, on the processor speed and operating system. In summary, if the computational overhead is insignificant, run time calculation of controller parameters is a feasible approach. However, if the computational overhead is significant, the offline calculation approach, taking advantage of the look-up tables, must be considered. However, we must determine the memory required to store these tables.

7.2.4 Memory requirements

If the overhead of online calculations is significant, the offline calculation approach taking advantage of the look-up tables must be considered. However, we must determine the memory requirements of these tables because they will store the controller parameters. In the following, we derive the size of this table for the three main cases, that is, for control tasks subject to varying feasible sampling intervals, or varying feasible sampling-actuation delays, or both. These tables, where the controller parameters must be stored, will have h_j as an input parameter (if subject to varying feasible sampling intervals), and for each h_j, τ_j as a second input parameter (if subject to varying feasible sampling-actuation delays). The worst-case assumption is to assume that the control task is subject to both varying feasible sampling intervals and varying feasible sampling-actuation delays. Knowing that for each control task $task_i$, the feasible sampling intervals and feasible sampling-actuation delays that appear at run time belong to sets FH_i and FT_i (see Section 6.1.1), the size of the table for the $task_i$ ($table_i$) is specified by (7.10)

$$\text{Size}(table_i) = \text{cardinal}(FH_i) * \text{cardinal}(FT_i) * \text{size}(\text{controller parameters of } task_i) \quad (7.10)$$

The size of the table if the control task is subject only to varying feasible sampling intervals is specified by (7.11)

$$\text{Size}(table_i) = \text{cardinal}(FH_i) * \text{size}(\text{controller parameters of } task_i) \quad (7.11)$$

If the control task is subject only to varying feasible sampling-actuation delays, the table size is specified by (7.12)

$$\text{Size}(table_i) = \text{cardinal}(FT_i) * \text{size}(\text{controller parameters of } task_i) \quad (7.12)$$

In order to give specific numbers, in Table 7.2, we show the size of the look-up table that the control task controlling the inverted pendulum (Section 2.2.2) needs for each of the three cases, if h_j and τ_j are allowed to take 100 different values (taking into account that a float is coded using, for example, 32 bits):

	Size (table)
Feasible sampling intervals	8 Kb
Feasible sampling-actuation delays	8 Kb
Feasible sampling intervals and feasible sampling-actuation delays	64 Kb

Table 7.2. Memory requirements for the inverted pendulum

From this analysis, we can conclude that these tables with the offline calculations are small enough to be stored in any micro-controller's RAM. For the specific values for these tables, see next Chapter, Section 8.1.3.

In Section 7.2.2 we presented the generic code of a control task prepared for the run time parameters adjustment required for the application of the compensation approach (see Figure 7.6 right). This generic code applies for both run time controller parameters adjustment strategies, that is, for the run time recalculations or for the run time look-up table access. However, in the examples we presented in Section 7.2.2, the exact codes for the controllers prepared for the run time parameters adjustment are based on the run time recalculations strategy.

Since we discussed in Section 7.2.3 that for the PID, the computational overhead is insignificant while for the state feedback controller it may be significant, in the following we show the required code modification for the control task implementing the state feedback controller if the run time parameters adjustment is performed by the look-up table access strategy. If the task $task_{sfc}$ (Section 2.2.2) is finally using the look-up table access approach at run time, the pseudo-code that should execute it can be seen in Figure 7.8 (compare to Figure 7.7 bottom in Section 7.2.2), task named $\boldsymbol{task_{CTsfc}}$. For a more detailed code, see Section A4 in Appendix A.

```
task_CTsfc
{  read_input (y(t_k), r(t_k));
   obtain (h_j , τ_j) ;
   (Φ(h_j, τ_j), Γ(h_j, τ_j), L(h_j , τ_j)) = ParametersTableIndexedAccess (Table, h_j, τ_j) ;
   u(t_k) = calculate_output (x(t_k), -L(h_j , τ_j), r(t_k));
   write_output (u(t_k));
   x(t_k+1) = update state (x(t_k), Φ(h_j, τ_j), Γ(h_j, τ_j), C, u(t_k), y(t_k));
}
```

Figure 7.8. Compensated State Feedback Controller code designed through pole placement with access to the controller parameters table

Note that after obtaining h_j and τ_j, instead of calculating the system discretization (matrices $\Phi(h_j,\tau_j)$, $\Gamma(h_j, \tau_j)$) and the controller ($L(h_j , \tau_j)$) at each instance execution (which increases the computational cost), these matrices are obtained accessing the pre-calculated controller parameters table. Therefore, at run time, the required extra computations (marked in bold) are, apart from obtaining the specific feasible sampling interval and feasible sampling-actuation delay, to perform the indexed table access, thus eliminating the computational overhead introduced when calculating all those matrices at each instance execution.

7.3 Summary

In this chapter we have presented the compensation approach as a discrete-time controller design method of designing controllers that depend on a set of a finite number of feasible sampling intervals and on a set of finite number of feasible sampling-actuation delays. For this new approach to controller design, we have provided a new response and stability analysis, illustrating all the controller design steps with examples.

In addition, we have discussed the implementation aspects for the applicability of controllers obtained through the compensation approach controller design method. Afterwards, we have explained, with pseudo code details, how to modify existing controller codes to prepare controllers for the run time controller parameters adjustment that the compensation approach requires. We have also distinguished two strategies for the run time parameters adjustment: the online recalculation approach if this incurs negligible overheads, and the online access to pre-calculated tables. We have provided exact numbers for characterizing when each strategy is feasible.

Chapter 8

Compensation approach in standard real-time scheduling policies

In this chapter we investigate the implications of using the compensation approach controller design method for control tasks scheduling. First of all, we address the practical aspects of the application of the compensation approach when the implementation is based on standard real-time systems technology. Specifically we focus on the aspects that the application of the compensation approach imposes on standard real-time scheduling strategies. We show that the compensation approach can be used to design controllers to be implemented in standard real-time periodic tasks that are scheduled by standard scheduling policies. After analysing the scheduling inherent jitters that control tasks are subject to, the controller is designed in such a way that the control task code can accept these jitters by adjusting its controller parameters at run time while meeting the closed-loop performance specifications, thus solving the degradation that would otherwise occur, as we have explained in Chapter 4.

In addition, we present a control performance study in which we evaluate the compensation approach in terms of closed-loop systems performance when jitters are included in the controller design stage and they are accounted for at each control task instance execution for the run time controller parameters adjustment.

The work explained in this chapter has been partially presented in [MAR01b] and [MAR01e] and [MAR02c].

8.1 Compensation approach as a control-based solution for dealing with jitters

In this section we explain how the compensation approach can be used to design controllers that can accept scheduling inherent jitters (sampling jitter and sampling-actuation jitter, see Section 4.1.2), thus solving the controlled system response degradation that would otherwise occur (as we show in Section 4.2) if real-time scheduled control tasks subject to jitters implement classically designed discrete-time controllers.

Specifically, we present the procedure for designing controllers (to be implemented in control tasks) in such a way that all *sampling intervals* and all *sampling-actuation delays* that apply at run time (for a given control task) due to *sampling jitter* and *sampling-actuation jitter* are grouped into sets FH and FT used in the compensation approach controller design stage (see Section 6.1.1). Consequently, we show that closed-loops

implemented by periodic control tasks (specified with standard fixed timing constraints, such as periods and deadlines, and scheduled by standard real-time scheduling policies) and implementing these new controllers, although subject to scheduling inherent jitters, will meet the control performance requirements.

However, we first concentrate on the requirements that the application of the compensation approach imposes on real-time scheduling (as we introduced in Section 7.2.1) in terms of meeting the timing constraints on which controllers obtained through the compensation approach controller design method are based. Recall that controllers designed using the compensation approach depend on a finite set of feasible sampling intervals (FH) and on a finite set of feasible sampling-actuation delays (FT) (Section 6.1.1).

8.1.1 Requirements of the compensation approach in real-time scheduling

As we discussed in Section 7.2.1, to apply the compensation approach, control tasks need, at each instance execution start time, information about the specific feasible sampling interval and feasible sampling-actuation delay for the run time parameters adjustment (Section 7.2.1). However, while feasible sampling intervals can be obtained by control task instances through online time measurements, in the general case, at run time the scheduler must provide the feasible sampling-actuation delay that will apply at each control task instance execution. This requirement imposes different constraints on real-time scheduling depending on the type of jitter (sampling jitter or sampling-actuation jitter) which is accounted for in the compensation approach controller design stage.

Requirements on feasible sampling intervals

As we discussed in Section 7.2.1, feasible sampling intervals can be obtained by online time measurements carried out by the control task. Therefore, as far as the controller is designed (using the compensation approach) to account for all possible sampling intervals caused by sampling jitter that can apply at run time for a given periodic control task (see Section 4.1.3), the application of the compensation approach does not impose any restriction on the scheduling policy.

Therefore, if for each controller to be implemented in a control task $task_i$ (specified with standard fixed timing constraints) we specify at its design stage the set of feasible sampling intervals, FH_i, as in Equation 4.4 in Section 4.1.3 (containing all possible sampling intervals due to sampling jitter that will appear at run time for the feasible periodic control task), the control task, although subject to sampling jitter, will meet the closed-loop system performance requirements

- by measuring at each instance execution the sampling interval caused by sampling jitter and
- by adjusting its controller parameters at run time according to the measured sampling interval

Note that the controller implemented in the control task has been designed to account for any of the possible sampling intervals that may apply at run time due to sampling jitter.

Therefore, whatever specific sampling interval applies at each instance execution, it has been already taken into account in the compensation approach controller design stage. This implies that the control task will update its controller parameters for that specific measured sampling interval, according to the specifications already used at the design stage.

In summary, standard offline and online scheduling approaches (or combined offline/online approaches) can be used with the compensation approach without any restrictions in terms of meeting the feasible sampling intervals used in the controller design stage.

Preserving feasible sampling-actuation delays

As we discussed in Section 7.2.1, sampling-actuation delays, in the general case, cannot be obtained by online time measurements carried out by the control task. Therefore, in this case, the scheduler is in charge of providing the control task with the specific values for each feasible sampling-actuation delay that applies at each control task instance execution.

In this case, regardless of designing the controller to account for all theoretical sampling-actuation delays caused by sampling-actuation jitter (as in Equation 4.5 in Section 4.1.3) that can apply at run time for a given periodic control, the application of the compensation approach imposes new restrictions on the scheduling policy. This is due to the fact that at each control task instance execution start time, the scheduler must guarantee the time instant at which the corresponding actuation will be completed.

For example, looking at EDF or FPS based scheduling approaches or even at offline scheduling approaches, at each instance execution start time, in the general case, there is no guarantee of when each instance will complete its execution. This may be due, for example, to varying execution times or pre-emption. Therefore, for periodic tasks, there are no guarantees regarding the time intervals that will appear at run time from each instance execution start time to its completion time. However, we are interested in control tasks.

By assuming an exact execution time for control tasks as we introduced in Section 2.1.2 and formally specified in Section 3.2.2, the varying execution times problem disappears. Observe that the assumption of exact execution time for control tasks (which we introduced in Section 2.1.2) is valid. Looking at control tasks, their execution times can be completely assessed. In general, control tasks execute a sequential code, with no conditional or loop sequences. For example, the code of a PID controller or the code of a state feedback controller, whether it is adjusting the controller parameter or not, is completely sequential (see Section 7.2.2). In this case, an exact execution time rather than a worst-case one can be calculated. Therefore, although general real-time task models require us to take into account for task scheduling the worst-case execution time for tasks, which may be too pessimistic [BUT98] and thus waste the processor resource (due to shorter executions than the assumed worst-case), for control task, we can assign the exact execution time[1].

Consequently, assuming an exact execution time for control tasks, offline scheduling can be used at run time to meet the set of feasible sampling-actuation delays used in the controller

[1] Note also that in general, in the hardware used in the implementation of closed-loops systems, features of modern architectures such as cache memories or pipelined processors that can introduce variation in the execution time of tasks are not usual.

design stage and given by the offline schedule sampling and actuation instants (if the offline schedule is enforced at run time, preserving then the offline sampling and actuation instants, see for example [FOH94] or [RAM90] or Section 2.1.2)[2]. However, although offline scheduling provides determinism, as all times for tasks (including control and non-control tasks) executions are determined and known in advance, run time flexibility is lacking. Since we are only interested in enforcing the offline schedule of the control tasks at run time, we can also use scheduling approaches that combine offline and online scheduling (such as [FOH95]), taking advantage of the determinism provided by the offline scheduling for the sampling and actuation instants, and the flexibility provided by the online scheduling for the rest of the tasks.

However, online scheduling approaches such as EDF or FPS still impose problems because pre-emption prevents us from knowing the exact completion time at each control task instance start time. One way of solving this problem and meeting the control task finishing time requirement is by setting the deadline equal to the exact execution time. By doing this, we guarantee that each control task instance will execute without interruption. However, as we discussed in Section 3.2.3, with this deadline assignment, to find feasible schedules, in general, is not possible. To overcome this problem we can use the same scheduling approaches but with non pre-emptive control task execution. That is, to relax the deadline assignment to improve system schedulability while ensuring that each control task instance will execute without any interruption at run time.

Therefore, whatever scheduling approach is used (taking into account the restrictions explained above), the set of feasible sampling-actuation delays, FT_i, (grouping all possible sampling actuation-delays caused by sampling-actuation jitter in a control task $task_i$) used in the compensation approach controller design stage will be adequately preserved at run time. Consequently, at run time, the scheduler will be able to give the specific feasible sampling-actuation delay required for the run time parameters adjustment to each control task instance, at its execution start time.

In summary, we have discussed how control tasks can adjust their controller parameters at run time according to the specific sampling intervals (due to sampling jitter) and sampling-actuation delays (due to sampling-actuation jitter) that apply at run time at each instance execution if controllers designed using the compensation approach are implemented by control tasks characterized by standard timing constraints and scheduled by standard real-time scheduling approaches.

8.1.2 Application of the compensation approach for scheduled control tasks

In this section we explain how to use the compensation approach controller design method to solve the problems imposed by jitters (controlled system response degradation) when

[2] It is assumed that the offline schedule is created in such a way that no pre-emptions will occur at run time. However, if they do occur at run time, the pre-empted control tasks in the offline schedule can be split into new *artifact* tasks (as in [DOB01]). This would solve the problem that the varying execution time of the pre-empting task will introduce in the run time timing of the control task (see example 1 in Section 8.1.3 for further discussion).

control tasks (specified by fixed timing constraints such as periods and deadlines) are scheduled by real-time scheduling algorithms.

As we introduced in Section 5.4 (in *Eliminating control system response degradation caused by scheduling inherent jitters*), this procedure has two steps: *offline jitter analysis* and *offline control analysis*.

The objective of the offline jitter analysis is to obtain for each control task the sampling intervals and sampling actuation delays that will apply at run time if the control task is subject to sampling jitter and sampling-actuation jitter in order to group them into the sets FH and FT (sets of feasible sampling intervals and feasible sampling-actuation delays) that will be used in the controller design stage. As we explained in Section 4.1.3, there is a finite number of all possible sampling intervals and sampling-actuation delays (that appear at run time) caused by sampling jitter and sampling-actuation jitter for each feasible periodic control task and which therefore can be known before run time (see Equations 4.4 and 4.5 in Section 4.1.3). Therefore, before run time, it possible to group them into two finite sets that will act as sets FH and FT. However, depending on the online scheduling policy or offline schedule, not all the possible sampling intervals and sampling-actuation delays caused by jitters and given by Equation 4.4 and 4.5 will apply at run time. Consequently, FH and FT may not have to include all of them, but only a subset.

Therefore, the objective of the offline jitter analysis is, for each control task $task_i$, to group in FH_i and FT_i the exact sampling intervals and sampling actuation delays that will apply at run time due to jitters. Once we have these two sets, we proceed with the offline control analysis, that is, we design each controller to be implemented in each periodic control task (specified with fixed timing constraints such as period and deadline) using the previous sets FH_i and FT_i.

Regarding the offline control analysis, in Section 7.1.2 we explained that for the stability analysis required in the compensation approach controller design method, we had to account for the type of sequences (Section 6.2.4) of feasible sampling intervals and feasible sampling actuation delays that the implementation guarantees for each control task. In the following, we discuss what type of sequences real-time scheduling causes; what we call *jitter patterns*. In this way, we are able to map different types of *jitter patterns* that offline schedules and online scheduling policies cause for periodic tasks with the different types of sequences of feasible sampling intervals and feasible sampling-actuation delays on which the stability analysis (*cases*, in Section 7.1.2) of the compensation approach controller design method is based.

Scheduling jitters patterns vs. stability analysis

As we saw in Section 2.1.2, real-time scheduling policies can be divided into two major categories according to the time when jitters are generated and analysable:

- *Offline*: the whole schedule is constructed before run time (e.g., see [FOH94] or [RAM90]). Therefore, all the sampling intervals and sampling-actuation delays due to jitters are known or could be specified beforehand. Although, theoretically, offline scheduling can construct non-periodic schedules, for practical purposes offline schedules are constructed over the LCM of the task's periods, which implies a LCM

periodic pattern. Depending on how the offline schedule is constructed and looking at the sampling intervals and sampling actuation delays of the control tasks over the LCM, two kinds of jitter periodic patterns apply for each control task:

a) a single value for all sampling intervals with a single value for all sampling-actuation delays through all the executions of a controller task[3] or

b) a finite sequence of pairs of known sampling intervals and known sampling-actuation delays that repeat periodically for each LCM throughout the execution of a controller task.

if the run time execution preserves the offline scheduled control task instances start and finishing times.

- *Online*: task instances are dispatched at run time according to the scheduling algorithm being used. Therefore, before run time we know the task set and the scheduling policy. However, before run time we don't know the exact timing for control tasks instances that will apply at run time. The same situation occurs if the run time dispatcher cannot preserve the start and finishing times of the offline scheduled control tasks. In this case, if the task set is feasible according the given scheduling policy, the only available knowledge is that at run time,

  c) an infinite sequence of unknown sampling intervals will apply with an infinite sequence of unknown sampling-actuation delays throughout the execution life of a controller task.[4]

Note that patterns a), b) and c) characterize the sampling intervals and sampling-actuation delays due to jitter that will apply at run time for a given control task. With these patterns, in the offline control analysis, we simplify the use of the compensation approach controller design method (which we explained in Section 7.1) for designing the controllers implemented in the scheduled control tasks. That is, when designing a controller using the compensation approach that will be implemented in a control task subject to scheduling inherent jitters, for the stability analysis, the following procedure applies: at the controller design stage, given the scheduling specifications, that is:

- offline schedule for patterns a) and b), or
- the task set and scheduling policy for pattern c),

after obtaining in the offline analysis the specific sets FH and FT for the controller design stage, for the stability analysis, we use (see Section 7.1.2 for the stability *cases*):

- case 1 if the scheduling specifications fall into pattern a)
- case 2 if the scheduling specifications fall into pattern b)

---

[3] Note that for this pattern, we don't have to redesign the controller (using the compensation approach) because to have a single value for all sampling intervals with a single value for all sampling-actuation delays is to have a constant sampling period (h) and a constant sampling actuation dealy (τ), which are already the timing assumptions for classic discrete-time controllers (Section 3.1.1).

[4] *Unknown* refers to the fact that, although they will match the values belonging to sets FH and FT, the exact values are not known before run time.

- case 3 if the scheduling specifications fall into pattern c)

Observe that the stability analysis procedure is also applicable to scheduling approaches that combine online and offline scheduling, such as in [FOH95].

8.1.3 Examples

In this section, using two examples, we illustrate all the concepts we discussed in the previous two sections when analysing the requirements and procedures for applying the compensation approach to deal with the scheduling inherent jitters. Recall that as we showed in Section 4.2.1, control tasks, implementing a classic designed controller, and subject to sampling jitter and sampling-actuation jitter, do not meet the control performance requirements, that is, the closed-loop system response undergoes important degradation.

As a controlled plant we use the inverted pendulum presented in Section 2.2.2, which is controlled by control task $task_{sfc}$ that implements a discrete-time state feedback controller designed to meet the performance requirements (to recover from a perturbation in less than two seconds) using pole placement observer design. To produce jitters, we use the set of tasks specified in Section 4.2.1 (see Table 8.1 for each control task characterization), which includes the control task $task_{sfc}$. Recall that for those tasks, deadline is equal to period. In addition, we regard the execution time of the control task to be the exact execution time, rather than the worst-case.

	T_i	C_i
$task_1$	50ms	10ms
$task_2$	60ms	10ms
$task_{sfc}$	80ms	20ms
$task_3$	100ms	20ms

Table 8.1. Task set

In the following, given this task set, for illustrative purposes, we apply offline scheduling (in example 1) and EDF (in example 2).

Example 1: offline scheduling

Given the set of tasks, the first thing we have to do is to obtain a feasible offline schedule with the objective of obtaining the jitter variability for each control task in the offline jitter analysis: in this case, that is, to obtain sets FH_{sfc} and FT_{sfc} for $task_{sfc}$. For example, a feasible offline schedule over the task periods LCM (1.2sec) is shown in Figure 8.1.

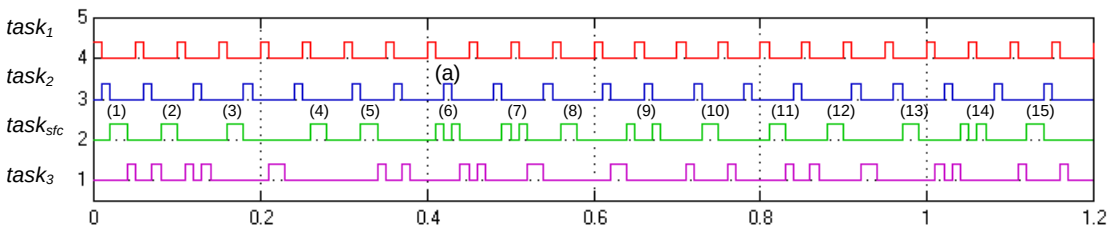


Figure 8.1. Offline schedule

The sequence of sampling intervals, $h(task_{sfc,k})$, and sampling-actuation delays, $\tau(task_{sfc,k})$, caused by sampling jitter and sampling-actuation jitter we obtain from the offline schedule for the control task $task_{sfc}$, a sequence that keeps a periodic pattern (marked with the instance number in Figure 8.1), and which is shown in Table 8.2.

Instance	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
$h(task_{sfc,k})$	60	80	100	60	90	80	70	80	90	80	70	90	70	80	100
$\tau(task_{sfc,k})$	20	20	20	20	20	30	30	20	40	20	20	20	20	30	20

Table 8.2. Sequence of sampling intervals and sampling-actuation delays due to scheduling inherent jitters (in ms)

Note that although this jitter characterization is obtained offline, we have to study if at run time, all these sampling intervals and sampling-actuation delays will also apply. For the sampling intervals variability, all sampling intervals that we obtain from the offline schedule will apply at run time because offline scheduling permits us to enforce the offline schedule instance start times at run time [FOH94]. However, for the sampling-actuation delays variability, this does not always apply. For instances of $task_{sfc}$ that are not interrupted by the execution of other instances of other tasks (in the schedule, instances 1, 2, 3, 4, 5, 8, 10, 11, 12, 13, and 15), the corresponding sampling-actuation delays that we obtain from the offline schedule will apply at run time. Recall that, as we discussed in Section 8.1.1, offline scheduling permits us to enforce at run time the offline schedule instance start and finishing times (recall also that the control task has been scheduled, taking into account its exact execution time). However, for instances of $task_{sfc}$ that are interrupted due to the execution of other instances of other tasks (in the schedule, instances 6, 7, 9, and 14), the corresponding sampling-actuation delays that we obtain from the offline schedule, in the general case, will not apply at run time. This is due to the fact that the interrupting instances (for example, instance (a) of task $task_2$ for the sixth instance of the control task $task_{sfc}$), at run time, may execute sooner than the worst case assumed in the offline schedule, resulting in an earlier execution of the remaining part of the interrupted instance that implies an earlier completion time. However, this problem can be solved by using methods that can enforce – at run time – the equivalent offline starting times of the remaining parts of the interrupted instances (see [DOB01] for further details).At the end of the offline jitter analysis, we obtain the sets FH_{sfc} and FT_{sfc} (grouping the sampling intervals and sampling-actuation delays that can appear at run time due to jitters) for the control task $task_{sfc}$ (required for the offline control analysis), as shown in the following (in ms):

$$FH_{sfc}=\{60,70,80,90,100\}$$

$$FT_{sfc}=\{20,30,40\}$$

Note, for example, that the sampling intervals due to sampling jitter for the first and fourth instances ($h(task_{sfc,1})$ and $h(task_{sfc,4})$) of the sequence of sampling intervals specified in Table 8.2 are equal to the first feasible sampling interval of set FH_{sfc} ($h_{sfc,1}$=60ms). Similarly, the sampling-actuation delays due to sampling-actuation jitter for the sixth, seventh and fourteenth instances ($\tau(task_{sfc,6})$, $\tau(task_{sfc,7})$, $\tau(task_{sfc,14})$) of the sequence of sampling-actuation delays specified in Table 8.2 are equal to the second feasible sampling intervals of set FT_{sfc} ($\tau_{sfc,2}$=30ms).

At this point, we can start with the offline control analysis. That is, we have to design a controller using the compensation approach controller design method using sets FH and FT. First of all, (as we explained in Section 7.1.1) we analyse whether the closed-loop performance specifications (inverted pendulum recovering from a perturbation in less than two seconds) are fulfilled given all combinations of feasible sampling intervals and feasible sampling-actuation delays belonging to FH_{sfc} and FT_{sfc}. However, looking at the sequence specified in Table 8.2, only 8 different combinations will apply at run time. Therefore, we need only look at the different inverted pendulum responses obtained by the control task $task_{sfc}$, each time characterized by one of these 8 different combinations.

In Figure 8.2 we show the inverted pendulum responses we have obtained for the 8 possible pairs of feasible sampling intervals and feasible sampling-actuation delays that will apply at run time: (60,20), (80,20), (100,20), (90,30), (80,30), (70,30), (90,40) and (70,20). To do so, at each simulation, the inverted pendulum is controlled by a task executing in isolation on a processor and implementing a classic controller designed through pole placement with observer (see Section A3 in Appendix A for code details) with each particular pair of values applying for the sampling period and time delay. Although it is difficult to distinguish in Figure 8.2 which curve corresponds to each specific combination of feasible sampling interval and feasible sampling-actuation delay, we can see that all curves meet the performance specifications (the pendulum is brought to angle zero – recovers from the perturbation – in less than 2 seconds).

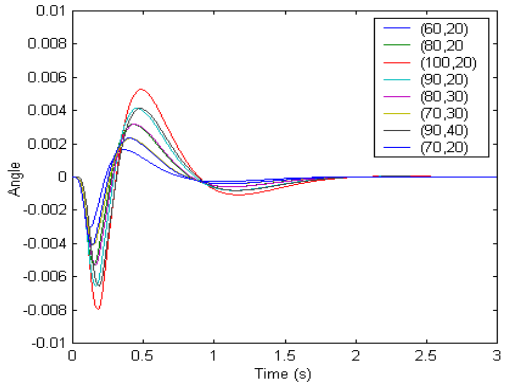


Figure 8.2. Inverted pendulum responses (resulting from the use of each pair – of feasible sampling interval, feasible sampling-actuation delay – generated by the offline schedule).

For the stability analysis, note that the sequence of jitters we have is pattern b) (Section 8.1.2). That is, we have a finite sequence of 15 pairs of known feasible sampling intervals and known feasible sampling-actuation delays that repeats periodically for each LCM. Therefore, for the stability analysis, we have to test *case 2* of Section 7.1.2. The closed-loop system is characterized by a known finite set of 15 closed-loop matrices that repeats periodically: $<\Phi_{cl_1}, \Phi_{cl_2}, \Phi_{cl_3}, \Phi_{cl_4}, \Phi_{cl_5}, \Phi_{cl_6}, \Phi_{cl_7}, , \Phi_{cl_8}, \Phi_{cl_9}, \Phi_{cl_{10}}, \Phi_{cl_1}, \Phi_{cl_{11}}, \Phi_{cl_{12}}, \Phi_{cl_{13}}, \Phi_{cl_{14}}, \Phi_{cl_{15}}>$ where Φ_{cl_k} is the closed-loop matrix that depends on each pair $(h(task_{sfc,k}),\tau(task_{sfc,k}))$ we listed earlier in Table 8.2. The stability test is performed by checking the stability of the product of the repeating sequence of matrices (Stable $\Leftrightarrow$ $\rho(\Phi_{cl_1} \cdot \Phi_{cl_2} \cdot \Phi_{cl_3} \cdot \ldots \cdot \Phi_{cl_{15}}) < 1$). Applying this test, we conclude that the system is stable.

As a consequence, we have obtained the controller we were looking for because all the inverted pendulum responses we obtained for all possible pairs of feasible sampling intervals and feasible sampling-actuation delays that will appear at run time fulfil the closed-loop performance specifications (and the closed-loop system is stable). At this point then, we can run the system, taking into account that the control task will adjust its parameters at each closed-loop execution. That is, the code executed by the control task will be slightly different to the classic one, in order to facilitate the controller parameters adjustment (for this case we use $task_{CTsfc}$, see Section 7.2.2, which performs the run time parameters adjustment by accessing online pre-calculated tables. For more details of the code, see Section A4 in Appendix A). The actual response we obtain when controlling the inverted pendulum with the task $task_{CTsfc}$ performing the run time parameters adjustment according to the sequence of jitters specified in Table 8.2 can be seen in Figure 8.3 left.

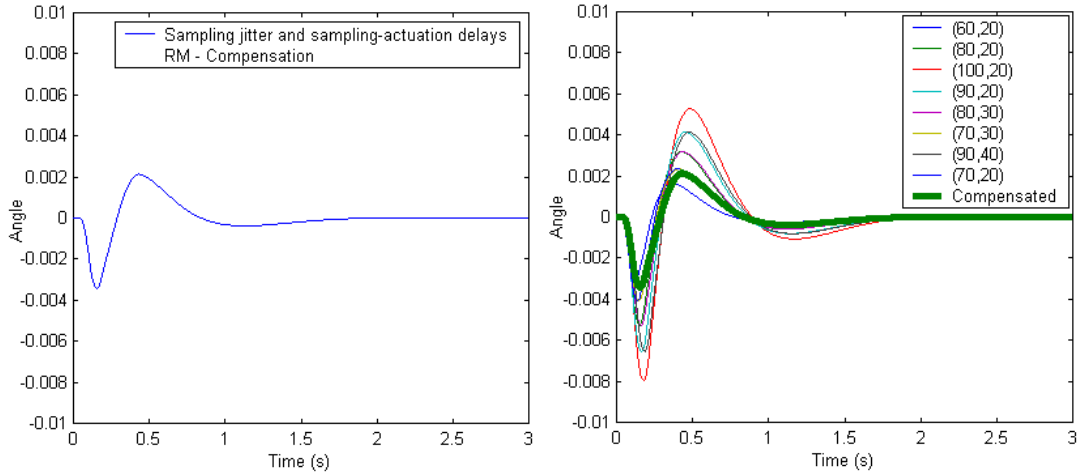


Figure 8.3. System response resulting from the offline schedule. Left- Compensated response. Right-all the possible responses (thin curves) and the compensated (thick curve)

It can be seen in Figure 8.3-right that the actual response we obtain with the control task $task_{CTsfc}$, performing the run time parameters adjustment, is a combination of the possible ones we gave as acceptable in the offline control design, as we explained in Section 7.1.1.

Remember that in the case of not compensating for the scheduling inherent jitters, the system controlled by a control task subject to similar jitters without adjusting the controller parameters leads to instability (Section 4.2.1).

In Table 8.3 we show the look-up table used for task $task_{CTsfc}$.

h_j (ms)	τ_j (ms)	Controller parameters
60	20	$\Phi(60, 20) = \begin{bmatrix} 1.0373 & 0.0607 & 0 & 0 & -0.0010 \\ 1.2514 & 1.0373 & 0 & 0 & -0.0205 \\ -0.0009 & -0.0000 & 1.0000 & 0.0600 & 0.0005 \\ -0.0298 & -0.0009 & 0 & 1.0000 & 0.0100 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ $\Gamma(60, 20) = [-0.0008 \quad -0.0402 \quad 0.0004 \quad 0.0200 \quad 1.0000]$ $L(60, 20) = [-273.5148 \quad -57.8381 \quad -127.0538 \quad -69.4594 \quad 0.4046]$ $K(60, 20) = [1.8871 \quad 15.9001 \quad 0.0310 \quad 0.4441 \quad 0.4527]$

70	20	$\Phi(70, 20) = \begin{pmatrix} 1.0509 & 0.0712 & 0 & 0 & -0.0012 \\ 1.4665 & 1.0509 & 0 & 0 & -0.0208 \\ -0.0012 & -0.0000 & 1.0000 & 0.0700 & 0.0006 \\ -0.0349 & -0.0012 & 0 & 1.0000 & 0.0100 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ $\Gamma(70, 20) = [-0.0013 \quad -0.0504 \quad 0.0006 \quad 0.0250 \quad 1.0000]$ $L(70, 20) = [-221.8512 \quad -47.4701 \quad -94.6528 \quad -54.0628 \quad 0.3605]$ $K(70, 20) = [2.0590 \quad 16.3363 \quad -0.0159 \quad -0.2387 \quad -0.1452]$
	30	$\Phi(70, 20) = \begin{pmatrix} 1.0509 & 0.0712 & 0 & 0 & -0.0017 \\ 1.4665 & 1.0509 & 0 & 0 & -0.0310 \\ -0.0012 & -0.0000 & 1.0000 & 0.0700 & 0.0008 \\ -0.0349 & -0.0012 & 0 & 1.0000 & 0.0150 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ $\Gamma(70, 30) = [-0.0008 \quad -0.0402 \quad 0.0004 \quad 0.0200 \quad 1.0000]$ $L(70, 30) = [-231.5949 \quad -49.7370 \quad -94.6528 \quad -55.0094 \quad 0.5738]$ $K(70, 30) = [2.0767 \quad 16.5885 \quad -0.0228 \quad -0.3341 \quad 0.0747]$
80	20	$\Phi(80, 20) = \begin{pmatrix} 1.0667 & 0.0818 & 0 & 0 & -0.0014 \\ 1.6845 & 1.0667 & 0 & 0 & -0.0210 \\ -0.0016 & -0.0000 & 1.0000 & 0.0800 & 0.0007 \\ -0.0401 & -0.0016 & 0 & 1.0000 & 0.0100 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ $\Gamma(80, 20) = [-0.0018 \quad -0.0607 \quad 0.0009 \quad 0.0300 \quad 1.0000]$ $L(80, 20) = [-186.5671 \quad -40.2808 \quad -73.3730 \quad -43.6299 \quad 0.3280]$ $K(80, 20) = [2.0178 \quad 14.1495 \quad 0.0165 \quad 0.1514 \quad 0.2422]$
	30	$\Phi(80, 30) = \begin{pmatrix} 1.0667 & 0.0818 & 0 & 0 & -0.0020 \\ 1.6845 & 1.0667 & 0 & 0 & -0.0313 \\ -0.0016 & -0.0000 & 1.0000 & 0.0800 & 0.0010 \\ -0.0401 & -0.0016 & 0 & 1.0000 & 0.0150 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ $\Gamma(80, 30) = [-0.0013 \quad -0.0504 \quad 0.0006 \quad 0.0250 \quad 1.0000]$ $L(80, 30) = [-194.8445 \quad -42.1875 \quad -73.3730 \quad -44.3636 \quad 0.5202]$ $K(80, 30) = [2.0403 \quad 14.4212 \quad 0.0073 \quad 0.0474 \quad 0.0570]$
90	20	$\Phi(90, 20) = \begin{pmatrix} 1.0846 & 0.0925 & 0 & 0 & -0.0016 \\ 1.9061 & 1.0846 & 0 & 0 & -0.0213 \\ -0.0020 & -0.0001 & 1.0000 & 0.0900 & 0.0008 \\ -0.0454 & -0.0020 & 0 & 1.0000 & 0.0100 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ $\Gamma(90, 20) = [-0.0025 \quad -0.0712 \quad 0.0012 \quad 0.0350 \quad 1.0000]$ $L(90, 20) = [-161.1375 \quad -35.0308 \quad -58.6093 \quad -36.1711 \quad 0.3029]$ $K(90, 20) = [2.1461 \quad 14.3327 \quad -0.0212 \quad -0.2592 \quad 0.1487]$
	40	$\Phi(90, 40) = \begin{pmatrix} 1.0846 & 0.0925 & 0 & 0 & -0.0029 \\ 1.9061 & 1.0846 & 0 & 0 & -0.0421 \\ -0.0020 & -0.0001 & 1.0000 & 0.0900 & 0.0014 \\ -0.0454 & -0.0020 & 0 & 1.0000 & 0.0200 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ $\Gamma(90, 40) = [-0.0013 \quad -0.0504 \quad 0.0006 \quad 0.0250 \quad 1.0000]$ $L(90, 40) = [-175.8940 \quad -38.3988 \quad -58.6093 \quad -37.3432 \quad 0.6691]$ $K(90, 40) = [2.0281 \quad 13.0412 \quad 0.0746 \quad 0.7462 \quad 0.1075]$
100	20	$\Phi(100, 20) = \begin{pmatrix} 1.1048 & 0.1035 & 0 & 0 & -0.0019 \\ 2.1316 & 1.1048 & 0 & 0 & -0.0217 \\ -0.0025 & -0.0001 & 1.0000 & 0.1000 & 0.0009 \\ -0.0508 & -0.0025 & 0 & 1.0000 & 0.0100 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ $\Gamma(100, 20) = [-0.0032 \quad -0.0818 \quad 0.0016 \quad 0.0400 \quad 1.0000]$ $L(100, 20) = [-142.0324 \quad -31.0413 \quad -47.9226 \quad -30.6136 \quad 0.2829]$ $K(100, 20) = [2.0805 \quad 12.6090 \quad 0.0615 \quad 0.5377 \quad 0.3382]$

Table 8.3. Table for task $task_{CTsfc}$ containing the parameters for the run time adjustment.

The table, indexed by the feasible sampling intervals and feasible sampling actuation delays that will apply at run time (according to the sequence specified in Table 8.2), contain all the

matrices (and vectors) that are required to update the controller parameters. Taking into account that the controller has been designed in the discrete-time domain using pole placement with observer, the required parameters, as we discussed in Section 7.2.4, are the discretization of the system model (matrices Φ and Γ), the gain matrix (L) and the observer matrix (K). For this case, the look-up parameters table should be of 320 floats. Taking into account that a float is coded using, for example, 32 bits, the resulting table would require approximately 10Kb.

Example 2: EDF

If we schedule the tasks set using EDF, the inverted pendulum response becomes instable, as we saw in Section 4.2.1 (the same effect RM had). Therefore, to solve the problem, we can design the controller implemented by the control task $task_{sfc}$ (characterized by standard timing constraints) in charge of controlling the inverted pendulum using the compensation approach controller design method, in order to meet the scheduling inherent jitters criterion.

The first thing we have to do, given the task set and EDF as a scheduling policy, is to obtain the jitter variability (offline jitter analysis). In this case, due to the nature of the scheduling policy, we don't know the exact task timing that will occur at run time. However, what we can obtain given the task set and EDF are the limits of the maximum jitter variability (see Section 4.1.3 for the general case). Given the task set and knowing that the system time granularity length g is 10ms, the jitter variability is, for sampling intervals, $60ms \leq h(task_{sfc,k}) \leq 100ms$, and, for sampling-actuation delays, $20ms \leq \tau(task_{sfc,k}) \leq 30ms$. Therefore, sets FH_{sfc} and FT_{sfc} for the control task $task_{sfc}$ are[5]:

$$FH_{sfc} = \{60, 70, 80, 90, 100\}$$

$$FT_{sfc} = \{20, 30\}$$

At this point, as before, we can start with the offline control analysis. That is, we have to design a controller using the compensation approach controller design method, which uses these FH_{sfc} and FT_{sfc} sets.

First of all, we analyse whether the closed-loop performance requirements (inverted pendulum recovering from a perturbation in less than two seconds) are fulfilled if the inverted pendulum is controlled by a control task specified in terms of each specific combination of feasible sampling interval and feasible sampling-actuation delay (from all combinations of feasible sampling intervals and feasible sampling-actuation delays belonging to FH_{sfc} and FT_{sfc}). In Figure 8.4 we show the inverted pendulum responses we have obtained for all possible pairs of sampling intervals and sampling-actuation delays that will apply at run time. To do so, at each simulation, the inverted pendulum is controlled by a task implementing a classic controller designed through pole placement with observer (see Section A3 in Appendix A for code details) with each particular pair of values applying for the sampling period and time delay.

[5] Note that if this way of obtaining sets FH and FT is not feasible (for example, because simulations of the schedule are not available, as we have in Section 4.2.1), we can always define these sets as in Equations 4.4 and 4.5 in Section 4.1.3

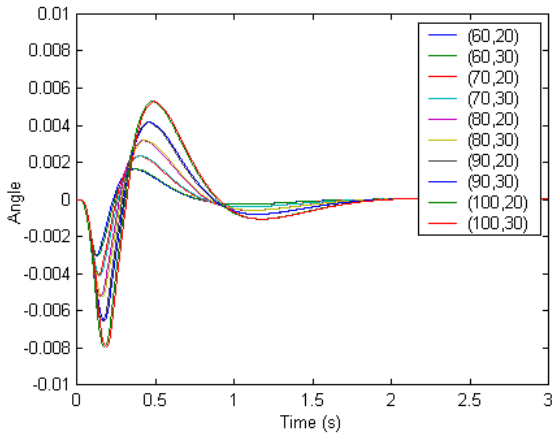


Figure 8.4. Inverted pendulum responses (resulting from the use of each pair of - feasible sampling interval, feasible sampling-actuation delay - that may apply due to EDF scheduling).

Although it is difficult to distinguish in Figure 8.4 which curve corresponds to each specific combination of feasible sampling interval and feasible sampling-actuation delay, we can see that all curves meet the performance requirements (the pendulum is brought to angle zero – i.e. recovers from the perturbation – in less than 2 seconds).

For the stability analysis, the sequence of jitters we have falls into pattern c) of Section 8.1.2. Therefore, for the stability analysis, we have to test *case 3* of Section 7.1.2. The closed-loop system will be characterized by a product of an infinite number of matrices taken randomly from a finite set of matrices Ω specified by all possible combinations of $h_{sfc,j}$ and $\tau_{sfc,j}$ belonging to FH_{sfc} and FT_{sfc}:

$$\Omega = \{\Phi_{cl_1}(60, 20), \Phi_{cl_2}(60, 30), \Phi_{cl_3}(70, 20), \Phi_{cl_4}(70, 30), \Phi_{cl_5}(80, 20), \Phi_{cl_6}(80, 30),$$
$$\Phi_{cl_7}(90, 20), \Phi_{cl_8}(90, 30), \Phi_{cl_9}(100, 20), \Phi_{cl_{10}}(100,30)\}$$

We apply the following stability test:

Ω is asymptotically stable iff there is a positive definite matrix P such that $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$, $\forall \Phi_{cl_k} \in \Omega^k$, for some $k \geq 1$, where the inequality is in the sense of negative definiteness.

Applying this condition to our set of 10 closed-loop matrices, we conclude that the system is stable.

At this point then, we can run the system, taking into account that the control task will adjust its parameters at each closed-loop execution. That is, the code executed by the control task will be slightly different to the classic one in order to facilitate the controller parameters adjustment (for this case we use $task_{Csfc}$ - see Section 7.2.2 -, which performs the run time parameters adjustment through online calculations). The actual response we obtain when controlling the inverted pendulum with the task $task_{Csfc}$ performing the run time parameters adjustment can be seen in Figure 8.5 left.

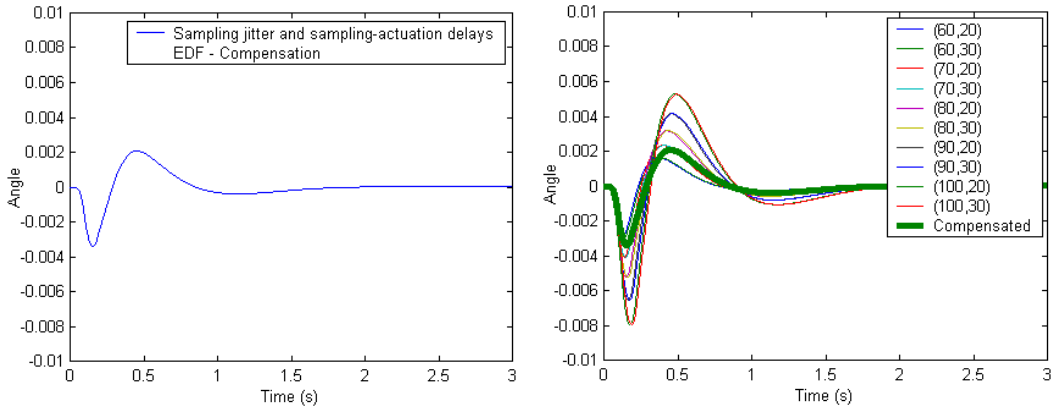


Figure 8.5. System response resulting from the EDF scheduling. Left- Compensated response. Right-all the possible responses (thin curves) and the compensated (thick curve)

It can be seen in Figure 8.5 right that the actual response we obtain with the control task $task_{Csfc}$, performing the run time parameters adjustment, is a combination of the possible ones we gave as acceptable in the offline control design, as we explained in Section 7.1.1.

Remember that in the case of not compensating for the EDF scheduling inherent jitters, the system controlled by the same control task without doing the parameters adjustment becomes unstable (Section 4.2.1, Figure 4.13, left).

8.2 Compensation approach control performance evaluation

The objective of this section is to evaluate, in terms of control performance, the compensation approach as a design methodology to eliminate the degradation that jitters for control task cause in the closed-loop performance. As we have seen in the previous section, the introduced degradation can be completely eliminated. Therefore, the benefits of this design methodology are clear. In this section, we formally specify what performance index can be used for this evaluation, and we present a case study based on the inverted pendulum example (Section 2.2.2) using this performance index.

8.2.1 Performance criterion selection

To evaluate the control performance of the compensation approach as a design methodology to eliminate the degradation that jitters for control task cause in the closed-loop performance we define a performance loss criterion. The applicability of this criterion allows us to compare two different closed-loop system responses.

As introduced in Section 2.2.1, the usual criteria to evaluate the performance of closed-loop systems are based on the integral of some function of the closed-loop system error. Typically, the ISE (Integral of Square Error), ITSE (Integral of Time-weighted Square Error), IAE (Integral of the Absolute Error) or ITAE (Integral of Time-weighted Absolute Error) are used. These criteria give a measure of the error of the closed-loop system response. We want to measure the difference between two closed-loop system responses.

Therefore, using the same idea of these classic performance criteria, but by considering the difference between two closed-loop systems responses as error, we define the following four *performance loss* criteria:

- Square performance loss error $V_q = \int_0^\infty (y_{nom}(t) - y_{act}(t))^2 dt$
- Time-weighted square performance loss error $V_{tq} = \int_0^\infty t(y_{nom}(t) - y_{act}(t))^2 dt$
- Absolute performance loss error $V_a = \int_0^\infty |y_{nom}(t) - y_{act}(t)| dt$
- Time-weighted absolute performance loss error $V_{ta} = \int_0^\infty t|y_{nom}(t) - y_{act}(t)| dt$

where

- y_{nom} is the *nominal system response* (obtained when the system to control is controlled by a task, which we call *nominal task*, executing at a *nominal sampling period* with a *nominal time delay*), which is used as a reference and
- y_{act} is the *actual system response* (obtained when the system to control is controlled by a task that is implementing a controller designed using the compensation approach)

Therefore, the error is understood as the difference between a given reference system response and the actual, that is, the response obtained applying the notions of compensations. Consequently, any of the performance loss criteria defined above can be chosen for our performance evaluation study.However, before choosing one of them, we investigate their sensitivity with respect to different values of the controller timing parameters. Since with the compensation approach controller parameters are adjusted at run time according to varying feasible sampling intervals and varying feasible sampling-actuation delays, we evaluate the relation of the different performance loss criteria with respect to different values for the sampling period and time delay.

To proceed with this performance loss criteria evaluation, we use the inverted pendulum as a system to control and the task $task_{sfc}$ we have been using as a control task (see Section A3 in Appendix A for the code details). In Section 2.2.2 we concluded that for the inverted pendulum problem, several values for the sampling period and time delay fulfilled the closed-loop performance specifications. From these feasible values, we choose a reduced set of feasible sampling intervals ({40, 50, 60, 70, 80, 90, 100, 110 and 120}) and feasible sampling-actuation delays ({20, 40, 50, 60, 70, 80}) because we are interested in the tendency of the values obtained by these performance loss criteria, not in the exact values. We evaluate their influence on each tendency separately. First of all, we have to define the y_{nom}, which will be the response obtained by $task_{sfc}$ with a constant sampling period of 40ms (nominal sampling period) and a time delay of 20ms (nominal time delay)[6]. The nominal response will be compared to the actual response, y_{act}, which is the inverted pendulum response when it is controlled by $task_{sfc}$ implementing a controller designed using the compensation approach with:

[6] Note that we set the nominal period to 40ms because 40ms is, among the selected values for the evaluation, the one that gives (Section 2.2.2) a better system response in terms of the closed-loop system error (Section 2.2.1). For the same reason we choose 20 ms as a nominal time delay.

a) A single value for the feasible sampling interval (from the previous set of values and 20ms as time delay) for all task instance executions, for each simulation.

b) A constant value for the feasible sampling-actuation delay (from the previous set of values and 80ms as a sampling period) for all task instance executions, for each simulation.

The evaluation a) is summarized in Table 8.4. The equivalent curves (which we obtain by joining the actual measurements) can be seen in Figure 8.6. Observe that the scales of the four curves are different because the criteria we evaluate are different, each one characterized by suitable units.

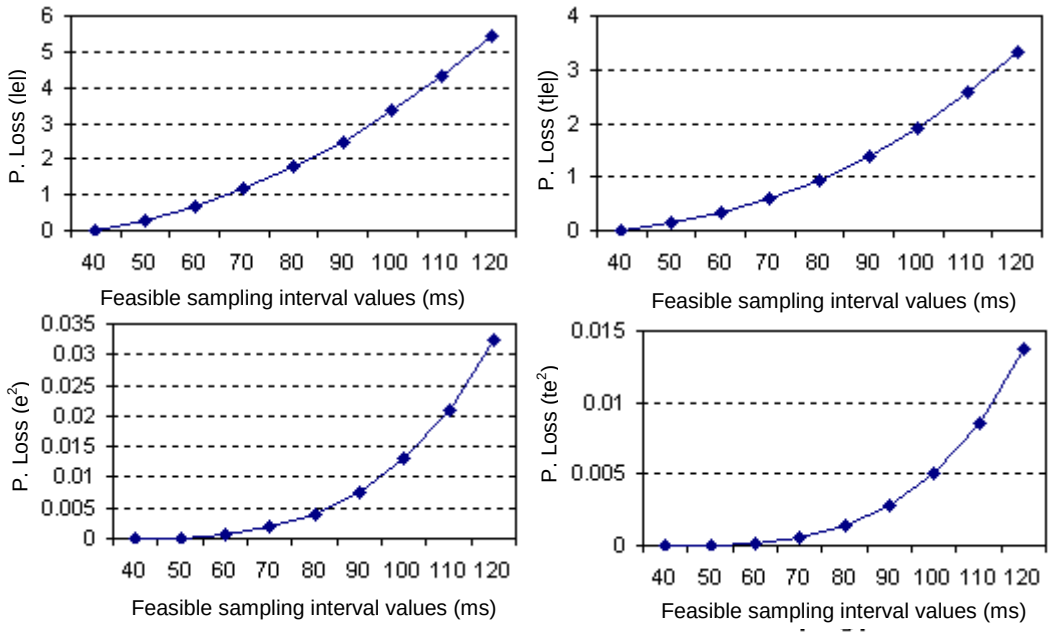


Figure 8.6 Influence of different values for feasible sampling intervals on each performance loss criterion values tendency

h_i (ms)	P. Loss (e^2) * V_q	P. Loss (te^2) * V_{tq}	P. Loss ($\|e\|$) * V_a	P. Loss ($t\|e\|$) * V_{ta}
120	0.03223	0.01371	5.42	3.34
110	0.02108	0.00853	4.31	2.57
100	0.01313	0.00503	3.33	1.92
90	0.00765	0.00276	2.49	1.38
80	0.00406	0.00137	1.77	0.94
70	0.00186	0.00058	1.16	0.59
60	0.00065	0.00019	0.67	0.32
50	0.00012	0.00003	0.28	0.13
40	0	0	0	0

Table 8.4. Influence of different values for feasible sampling intervals on each performance loss criterion values tendency

As can be seen in Figure 8.6, all the curves look like the exponential type. The evaluation b) is summarized in Table 8.5. The equivalent curves (which we obtain by joining the actual

measurements) can be seen in Figure 8.7. Observe that the scales of the four curves are different because the criteria we evaluate are different, each one specified by suitable units.

τ_j (ms)	P. Loss (e^2) * V_q	P. Loss (te^2) * V_{tq}	P. Loss ($\|e\|$) * V_a	P. Loss ($t\|e\|$) * V_{ta}
80	0.00208	0.00049	1.05	0.39
70	0.00150	0.00034	0.88	0.32
60	0.00098	0.00021	0.71	0.25
50	0.00056	0.00012	0.53	0.19
40	0.00025	0.00005	0.36	0.12
30	0.00006	0.00001	0.18	0.06
20	0	0	0	0

Table 8.5. Influence of different values for feasible sampling-actuation delays on each performance loss criterion values tendency

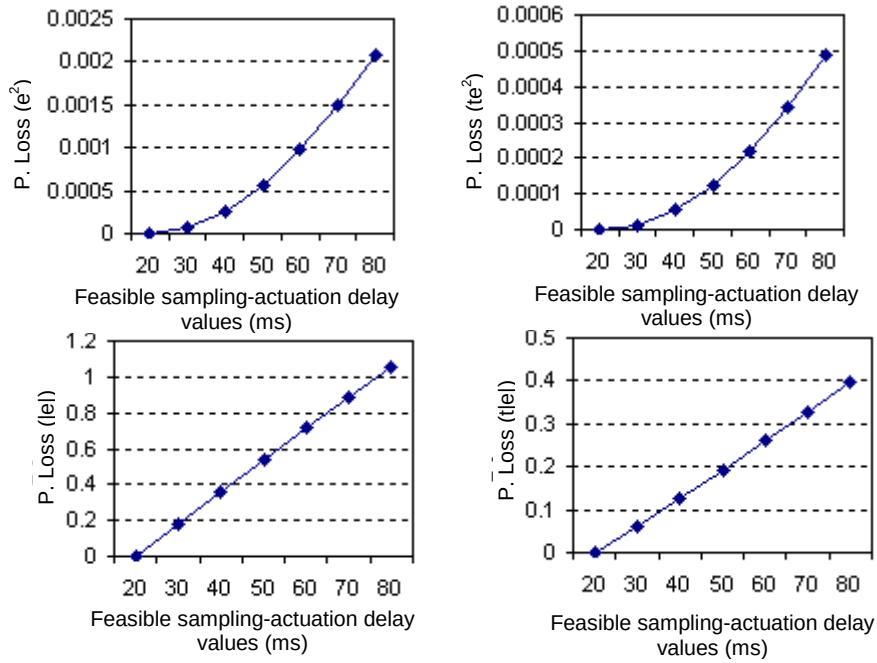


Figure 8.7. Influence of different values for feasible sampling-actuation delays on each performance loss criterion values tendency

Note that in this case (Figure 8.7), not all the curves look like the exponential type. The criteria based on the absolute error (either time-weighted or not), give a linear relation between time delays and errors. Since all the performance loss criteria give similar results, we choose the *Absolute performance loss error* criterion (denoted either by V_a or $|e|$) simply because it gives a better linear relation between timing parameters and errors. In this way, in the compensation approach performance evaluation, using criterion V_a we will have a better discretization between different closed-loop system responses, although any of the performance loss criteria could have been selected. Note also that there is no need to weight the V_a criterion with time because the difference between the nominal response and the actual response will already depend on delays (which are the parameter that usually needs to be weighted by time due to its temporal nature).

8.2.2 Evaluation study

In this section, we evaluate in different scenarios the compensation approach as a design methodology for eliminating the degradation that jitters for control task cause in the closed-loop performance in different situations of interest. This study is based on the inverted pendulum example (Section 2.2.2). The scenarios in which we evaluate the performance of the compensation approach, either comparing closed-loop system responses or using the *Absolute performance loss error* criterion (V_a), are the following:

- A: Control tasks subject to sampling jitter and sampling-actuation jitter.
- B: Compensation approach performance depending on scheduling policies.
- C: Compensation approach performance depending on specific jitter sequences.
- D: Compensation approach performance with fixed actuation instants.

Scenario A: Control tasks subject to sampling jitter and sampling-actuation jitter

As we have seen in Section 8.1.3, the degradation in closed-loop systems caused by jitters in control tasks can be completely eliminated if these tasks use a controller designed using the compensation approach. Then, the benefits of using the compensation approach are clear.

For example, comparing the performance of the closed-loop system when the inverted pendulum was controlled by a control task executing a classic controller code (which is not compensating for scheduling inherent jitters, see Section 4.2.1) and by a control task executing a slightly modified code, adjusting its parameters according to actual jitters (Section 8.1.3), the evaluation is obvious. That is, in the first case the closed-loop system became unstable while in the second case the closed-loop system met the performance requirements. Therefore, in this case, there is no point in using the Absolute performance loss error criterion (V_a) to compare a non-compensated response with the compensated response of the inverted pendulum. However, to compare the improvement obtained by the compensation approach, as a example, in Figure 8.8 we compare (left) the degradation caused by jitters in EDF (corresponding to Figure 4.13 left in Section 4.2.1) with (right) the response obtained when applying the compensation approach in EDF (corresponding to Figure 8.5-left in Section 8.1.3).

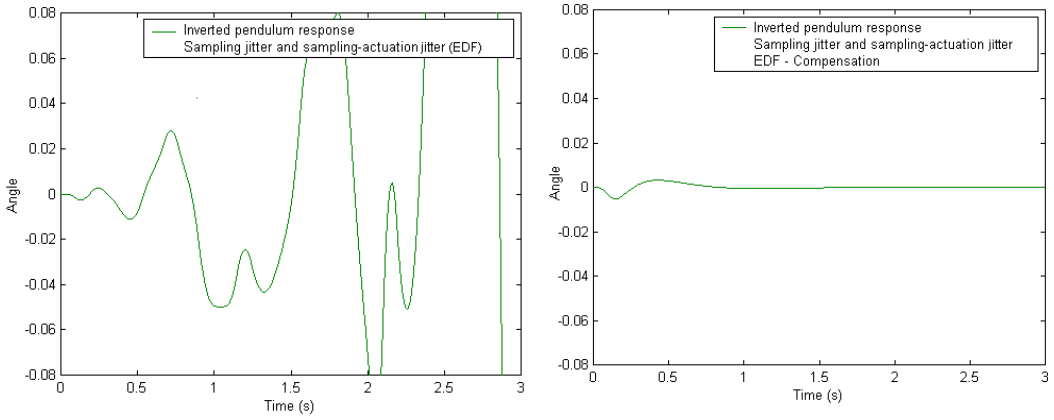


Figure 8.8. Inverted pendulum response. Left - Degradation. Right - Compensation

Scenario B: Compensation approach performance depending on scheduling policies

In this situation we analyse the performance of the compensation approach when the same set of tasks are performed using different scheduling policies. As we have seen in Section 4.2.1, different scheduling policies give different jitter patterns (sequences of sampling intervals and sampling-actuation delays for the same set of tasks) for each control task. Note that the calculation of each control signal at each control task instance execution depends on each specific sampling interval and sampling-actuation delay accounted for by the run time parameters adjustment. Therefore, the response we obtain for the same control task compensating for different jitter patterns will also be different, although all meet the closed-loop performance specifications.

However, if we compare the responses we obtain for the inverted pendulum controlled by a control task adjusting its controller parameters at each instance execution when the set of tasks (specified Table 4.7 in Section 4.2.1) is scheduled by RM or EDF, we observe no significant difference, as shown in Figure 8.9.

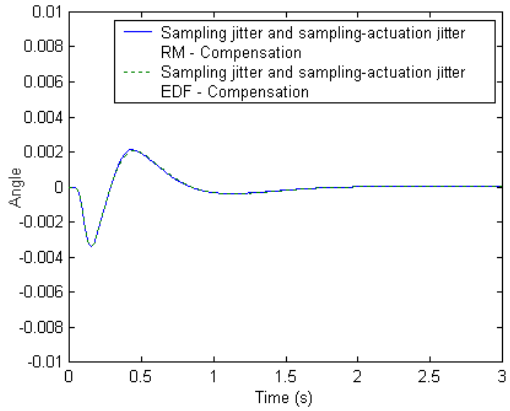


Figure 8.9. Inverted pendulum response. RM vs EDF compensation

However, if we evaluate the two responses with the absolute performance loss error (V_a) criterion by comparing the actual compensated responses with a nominal response obtained by controlling the inverted pendulum by using a control task with a constant sampling period of 40ms and a constant time delay of 20ms, we can see in Table 8.6 that the compensated responses are not equal.

	Absolute performance loss error (V_a)
EDF scheduling	1.156
RM scheduling	1.144

Table 8.6. Absolute performance loss error due to EDF and RM

Therefore, it is clear that for a given control task adjusting its controller parameters at run time according to different jitter patterns produces different closed-loop system responses. That is, the performance of the compensation approach depends on the actual sequence of jitters.

Scenario C: Compensation approach performance depending on specific jitter sequences

Since in the previous situation we have seen that a specific jitter pattern (which a control task is compensating for) influences the performance of the compensation approach, in this situation we investigate if the jitter sequence ordering is important with respect to the performance of the compensation approach. That is, taking a known cyclic jitter sequence, we want to study the response of the inverted pendulum controlled by a control task designed using the compensation approach when the perturbation (modelled as a pulse) arrives at different specific times giving a cyclic schedule.

In Example 1 in Section 8.1.3, we have seen that the actual response of the inverted pendulum we obtain with the control task designed using the compensation approach is a combination of the 8 possible ones we consider acceptable in the control design stage (offline control analysis). However, we showed only one response obtained by the compensation approach, where at each controller execution the sampling interval and sampling-actuation delay that applied followed the jitter sequence specified by the offline schedule (Figure 8.1 in Section 8.1.3). Note that the jitter sequence has 15 pairs (of sampling interval, sampling-actuation delay), as we show again in Table 8.7.

Instance	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
$h(task_k)$	60	80	100	60	90	80	70	80	90	80	70	90	70	80	100
$\tau(task_k)$	20	20	20	20	20	30	30	20	40	20	20	20	20	30	20

Table 8.7. Sequence of sampling intervals and sampling-actuation delays due to scheduling inherent jitters

Therefore, to evaluate the influence of the ordering of the jitters sequence on the inverted pendulum response, we run 15 different simulations. In each simulation, we force the perturbation to arrive at a different task instance execution, which corresponds to a different pair of sampling interval and sampling-actuation delay.

Perturbation arrival	Absolute performance loss error (V_a)
Worst-case (100,40)	3.41
At instance (1): (60,20)	1.14
At instance (2): (80,20)	2.11
At instance (3): (100,20)	2.09
At instance (4): (60,20)	1.12
At instance (5): (90,20)	2.77
At instance (6): (80,30)	1.70
At instance (7): (70,30)	0.88
At instance (8): (80,20)	3.25
At instance (9): (90,40)	1.39
At instance (10): (80,20)	1.68
At instance (11): (70,20)	1.53
At instance (12): (90,20)	1.90
At instance (13): (70,20)	2.05
At instance (14): (80,30)	1.65
At instance (15): (100,20)	2.28

Table 8.8. Performance values depending on the ordering of the jitter sequence

In Table 8.8, we show the performance loss values (with respect to a nominal response obtained for a control task when h=40 ms and τ = 20ms) of the 15 simulations. In addition, we know that the longest sampling interval in the jitter sequence is 100ms (h($task_3$) or h($task_{15}$) in Table 8.7), and the longest sampling-actuation delay is 40ms (h($task_9$) in Table 8.7). In Table 8.8 we also evaluate the performance loss of the response of the inverted pendulum when it is controlled by a control task with a constant sampling period of 100ms and constant computational delay of 40ms, which we call the worst-case response, in order to give a reference value.

In Figure 8.10, we show the performance values for the 15 simulations (series). We also added in Figure 8.10, the mean value of the 15 series we ran (Average).

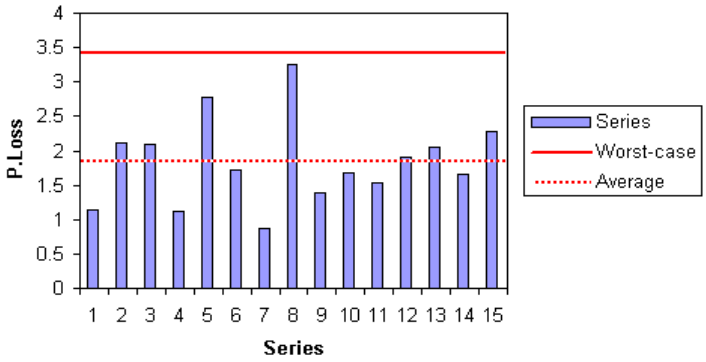


Figure 8.10. Performance values depending on the ordering of the jitter sequence

From Figure 8.10 we can see that the response of the inverted pendulum controlled by a task designed using the compensation approach strongly depends on the jitter ordering. Figure 8.10 suggests an important property: the performance of the compensation approach, at least for this example, is unexpectedly good. That is, any of the inverted pendulum responses obtained by the task adjusting its controller parameters according to jitters is better (in terms of the performance criterion) than the so-called worst-case response. Note the three following facts:

a) the so-called worst-case response is obtained by a control task executing a classic control algorithm executing with a constant sampling period and a constant time delay.

b) in contrast, all of the compensated responses are obtained by a control task that executes the slightly modified code, adjusting its parameters according to different pairs of sampling intervals and sampling-actuation delays.

c) looking at the sampling intervals and sampling actuation delays sequence we are dealing with, it is clear that the specific sampling interval and sampling-actuation delay the control task is compensating for are always equal to or shorter than the constant sampling period and time delay of the task giving the so-called worst-case response.

However, from a theoretical control response analysis point of view (Section 2.2.2), as far as we know, no assurances have been given about the fact that 1) *a controlled system response obtained by a control task adjusting its controller parameters according to different values*

for the sampling period and time delay is better than 2) *the response obtained by a classic controller task designed with values for its constant sampling period and constant time delay that are equal to the longest sampling interval and sampling-actuation delay that appeared in* 1). Consequently, because in this example we have shown this fact, we conclude that the performance of the compensation approach is better than expected.

Scenario D: Compensation approach performance with fixed actuation instants.

In Section 3.2.2, we explained two possible ways of implementing a control loop: the single task and the multiple task approach. Recall that in the multiple task approach, the three main activities of a closed-loop, sampling, control computation and actuation are implemented in separate tasks. In the single task approach, the three main activities of a control loop are implemented within a task. One of the main advantages of the multiple task approach over the single task approach is its ability to provide more flexibility in terms of schedulability (see Section 3.2.2 for further discussion). In addition, the flexibility provided by the multiple task approach could be used to eliminate the actuation instants variability by enforcing (through the scheduling policy) the actuation task execution at a fixed time after the sampling occurs for all control task instances executions, as assumed by classic discrete-time control theory (see Section 3.1.1). As a consequence, rather than compensating for varying feasible sampling-actuation delays, we should compensate for a constant time delay. However, in this case, the constant time delay should be longer than or equal to any of the feasible sampling-actuation delays for which we provide compensations.

Therefore, it is also interesting to evaluate the performance of the compensation approach in this scenario. The easiest way to set up this evaluation is to consider a control task with constant sampling period and either

a) a method of compensating for varying feasible sampling-actuation delays: since $\tau<h$ (as assumed in the compensation approach controller design method, see Chapter 6), if we fix the sampling interval h_j at 80ms (constant sampling period), then we can allow the feasible sampling-actuation delay to go from 20ms to 80 ms (in steps of 10ms), in a randomly generated way at each controller execution.

b) a constant but longer time delay: having a sampling period of 80ms, we choose 80 as a constant sampling-actuation delay, which is the longest sampling-actuation delay that can appear in simulations a).

In all the cases, the nominal response used in the performance loss criterion V_a is obtained by a control task with a constant sampling period of 40 ms and a constant sampling-actuation delay of 20ms.

To simulate a), we run 10 simulations (series). The performance evaluation of the inverted pendulum response we obtained is summarized in Table 8.9, along with the performance evaluation of the response obtained by simulating b). Note that because in each of the 10 simulations of a) the feasible sampling-actuation delays were randomly generated for each controller execution in the single task approach, the performance loss for each inverted pendulum response varies. However, as can also be seen in Figure 8.11, which graphically represents Table 8.9, the performance loss of the system response of the cases with varying sampling-actuation delays is always smaller than the case with constant time delay.

Series	Absolute performance loss error (V_a)
S_0	0.42
S_1	0.27
S_2	0.30
S_3	0.15
S_4	0.38
S_5	0.19
S_6	0.21
S_7	0.32
S_8	0.34
S_9	0.64
Fixed (80,80)	1.05

Table 8.9. Performance study. Varying feasible sampling-actuation delays vs. a fixed time delay

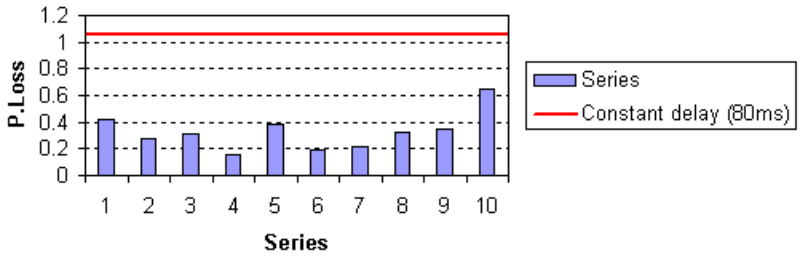


Figure 8.11. Performance study. Varying feasible sampling-actuation delays (series) vs. a constant time delay

From Figure 8.11 we can conclude that, for the inverted pendulum example, compensating for varying values for the time delay is better than having a constant but longer value for the time delay. In summary, the multiple task approach has the potential to use its flexibility to eliminate the variability (caused by sampling-actuation jitter) in the actuation instants. However, if this results in a constant but longer time delay for the controller design, the performance of the closed-loop system decreases.

8.3 Summary

In this chapter we explained how to use standard scheduling approaches to schedule control tasks that implement controllers designed using the compensation approach. In addition, we have shown that using the compensation approach we eliminate the degradation that scheduling inherent jitters for control tasks introduce in the controlled system response.

We also have formally defined a performance criterion to evaluate the compensation approach as a design methodology to eliminate the degradation that jitters for control task cause in the closed-loop performance. Through studying different scenarios given by different scheduling situations, we have concluded that the benefits of using the compensation approach are clear. It is interesting to point out that we have also shown that the performance of the compensation approach depends on the specific sequence of sampling intervals and sampling-actuation delays (caused by jitter) that are used for the run time parameters adjustment. This latter issue is further investigated in the next chapter.

Chapter 9

Flexible control task scheduling

Our compensation approach provides another advantage that leads to better system schedulability. This comes from its ability to derive more flexible timing constraints than just periods and deadlines necessary to apply standard scheduling policies such as EDF and FPS. After defining these new *flexible timing constraints for control tasks*, we demonstrate how to exploit them to improve system schedulability. Specifically, we show that by using flexible timing constraints for control tasks, we obtain feasible schedules for task sets (which include control and non-control tasks) that are not feasible when scheduled using scheduling methods based on fixed timing constraints for periodic tasks.

We also introduce a new approach to control task scheduling called *Quality-of-Control (QoC) scheduling* that takes advantage of the control performance information that these new flexible timing constraints implicitly have. First of all, we define a *Quality-of-Control (QoC) metric* that associates each feasible timing constraint with a quantitative value expressing control performance in terms of the closed-loop system error (see Section 2.2.2) resulting from the use of that timing constraint. This offers the possibility of taking scheduling decisions based on this control information for each control task invocation, rather than using fixed timing constraints with constant periods and deadlines.

This opens up a novel scheduling problem, QoC scheduling, in which the QoC information of control task timing constraints is used to improve the performance of the closed-loop systems in the presence of perturbations. We present the main scheduling issues that underlie this problem and also identify feasible solutions. In sum, we show (a) how the problem of reacting to perturbations during control can be addressed by specifying flexible timing constraints and associated QoC, and (b) how these timing constraints in turn can be satisfied by applying standard scheduling techniques.

The work explained in this chapter has been partially presented in [MAR01e], [MAR02a] and [MAR02b].

9.1 Control task scheduling with flexible timing constraints

In this section we discuss how the temporal constraints imposed by our jitter compensation approach can be exploited to improve real-time system schedulability. First, we address fixed timing constraints as demanded by standard scheduling schemes. Then we show how novel, flexible timing constraints for control tasks can be used to fully exploit the flexibility provided by our approach in order to improve system schedulability. Note that we do not propose a scheduling approach, but rather a new set of flexible timing requirements for control tasks.

9.1.1 Fixed timing constraints for control tasks

Classical discrete-time control theory [AST97] assumes *equidistant sampling* and *actuation* within the closed-loops: the timing requirements that an implementation must guarantee are a constant *sampling period*, h, (between successive sampling instants), and a constant *time delay*, τ, (from related sampling to actuation instants) (see Section 3.1.1). Although several values of the sampling period and time delay guarantee stability and fulfil the control response performance specifications, at the design stage, the control designer is faced with selecting specific values for h and τ.

Standard scheduling schemes are based on fixed timing constraints such as periods and deadlines. Here, *fixed* means that a single value for a constraint holds for all instances of a task. Normally, in real-time control systems, control task periods and deadlines are selected as follows (as explained in Section 3.2.2):

- *Period:* after the control analysis (which includes stability and response analysis), for a classic discrete-time controller design, a sampling period h is chosen. When this controller is implemented in a real-time task, the task period (T) is set equal to the sampling period (h).
- *Deadline*: deadline (D) is set equal to the period (T) or at a relative fixed time with respect to the start of the period (*deadline<period* or *deadline>period*), given by the time delay (τ) assumed in the controller design stage

However, note that to meet the semantics of control models, the exact finishing time is required rather than an upper Limit on the completion time of the control task (see Section 3.2.1 for further discussion)[1]. As stressed in Chapter 3, this imposes additional restrictions on control task scheduling that can impair schedulability, e.g., non pre-emptive execution, or setting a deadline equal to start time plus execution time. This is summarized in Figure 9.1.

Discrete-time control design		Fixed timing constraints
Sampling period, h	⇒	Period, T = h
Time delay, τ		Deadline, D = τ

Figure 9.1. Fixed timing constraints for control tasks

The selection of fixed timing parameters for a task has to be based on worst-case assumptions about load scenarios, task phasing, etc. That is, should the load situation be because of a single instance, the timing constraints to meet this worst-case demand have to be imposed on all instances. This reduces schedulability and may even render the tasks unschedulable although all demands may be met from a control perspective. The approach we describe next enables the derivation and setting of timing constraints on a per-instance basis, adjusting the task instance timing constraints to the situation faced by that instance. Thus, the timing constraints may vary from instance to instance, but the overall goal of control stability is ensured [MAR01e].

[1] Otherwise, scheduling inherent jitters for control tasks will degrade the performance of the closed-loop system (Chapter 4).

9.1.2 Flexible timing constraints for control tasks

The compensation approach (see Section 6.1.1) is based on the assumption of *non-equidistant sampling* and *actuation*. It allows us to design discrete-time controllers that depend on a finite set of *feasible sampling intervals* $h_{i,j}$, and on a finite set of *feasible sampling-actuation delays*, $\tau_{i,j}$, derived during the controller design stage.

From these new timing requirements, for control tasks we define new *flexible timing constraints*. For each control task $task_i$, flexible timing constraints are defined in the form of *instance separations* and *response times*, which are given by the feasible sampling intervals (of set FH_i) and feasible sampling-actuation delays (of FT_i) used in the compensation approach controller design stage (as summarized in Figure 9.2):

- *Instance separation:* the time interval between the start of two consecutive instances of a control task $task_i$, (*instance separation*), is limited by the discrete range of feasible sampling intervals $h_{i,j}$ ($h_{i,j} \in FH_i$) used in the compensation approach controller design stage:

$$\forall\ task_{i,k}, task_{i,k+1} : s(task_{i,k+1}) - s(task_{i,k}) \in FH_i$$

- *Response time:* the time interval from the start time to the finishing time of a task instance, (*response time*), is limited by the discrete range of feasible sampling-actuation delays $\tau_{i,j}$ ($\tau_{i,j} \in FT_i$) used in the compensation approach controller design stage:

$$\forall\ task_{i,k}: f(task_{i,k}) - s(task_{i,k}) \in FT_i$$

  Recall that at run time, the current sampling-actuation delay for a particular instance has to be known at the execution start time of the control computation for the application of the appropriate compensation. That is, the response time constraint, while flexible, has to be kept such that the actuation finishes *at* rather than *before* the specified time.

Compensation approach		**Flexible timing constraints**
Feasible sampling intervals: $h_{i,j} \in FH_i$	$\Rightarrow$	Instance separations: $\forall\ task_{i,k}, task_{i,k+1} : s(task_{i,k+1}) - s(task_{i,k}) \in FH_i$
Feasible sampling-actuation delays: $\tau_{i,j} \in FT_i$		Response times: $\forall\ task_{i,k}: f(task_{i,k}) - s(task_{i,k}) \in FT_i$

Figure 9.2. Flexible timing constraints for control tasks

Our flexible timing method does not set specific values for the timing constraints of control tasks. Rather, it provides ranges and their combinations to choose from, taking into account, for example, the schedulability of other tasks. The application of flexible timing constraints allows us, at run time, to choose different settings (instance separation and response time) for each control task instance execution. Thus, our methods provide more flexibility than can be expressed by fixed timing constraints, as is suggested in [FOH97].

Recall that for each control tasks $task_i$, the different settings (instance separation and response time) have to be chosen from sets FH_i and FT_i. Therefore, instance separations and response times defined as the *flexible timing constraints for the control task* $task_i$ coincide with sets FH_i and FT_i specified in the compensation approach design stage. Therefore, in the rest of this chapter, each $h_{i,j} \in FH_i$ will be referred to either as instance separation or feasible sampling interval and each $\tau_{i,j} \in FT_i$ will be referred to either as response time or feasible

sampling-actuation delay. Note that while similar for subsequent instances, these constraints differ from the fixed ones: using period T_i, the k^{th} instance of a task $task_i$ can start at (k-1)*T_i, whereas with the constraints here it can start at t_k (given by equation (6.39), where t_k is the sum of all $h_{i,j}$ already applied before k^{th} instance, $h_{i,j} \in FH_i$). Since the $h_{i,j}$ can vary, the instance start times will be different.

Figure 9.3 illustrates the flexible timing constraints in comparison with the fixed timing constraints. Assume we have a control task, characterized by fixed timing constraints such as a period (T) of 50ms and the deadline (D) of 10ms. The exact execution time is 10ms. Figure 9.3 first row shows the first four instances of the control task with fixed timing constraints, where boxes mark periods and executions are shaded. As defined by fixed timing constraints, the k^{th} instance of the task will start at (k-1)*50ms. For example, the fourth instance will start at 150ms. Assume now that the same control task is characterized by flexible timing constraints, with 40 or 50ms as possible instance separations. Deadline and exact execution time is 10ms (as before). In this case, the fourth instance of the control task need not start at any fixed time. It can start at any time ranging from 120 to 150ms (in steps of 10ms). Note that for the case of the control task with flexible timing constraints, Figure 9.3 (from the second row) shows all possible instances separation sequences for the first four instances. Our method provides the possible values to be used and ensures control stability by compensation.

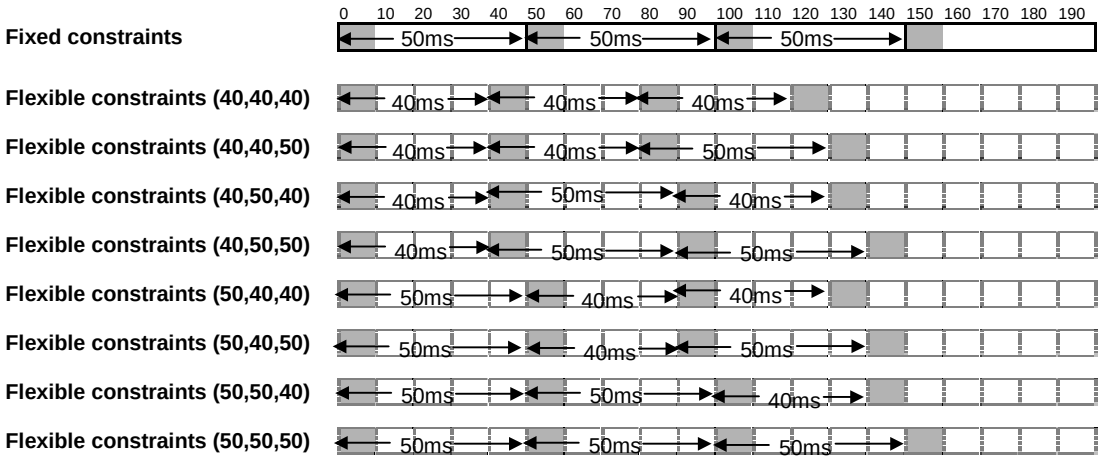


Figure 9.3. Fixed timing constraints vs. flexible timing constraints (times in ms)

Instead of simply trying to guarantee an instance based on these constraints, our compensation approach provides the scheduler with the flexibility to select the constraints as well. If a set of constraints cannot be met given the schedule, the scheduler can pick another set of constraints to find a feasible schedule. Figure 9.3 also illustrates this property. All instances of the control task with fixed timing constraints have to execute according to the predetermined ordering derived from its constraints. Consequently, if any conflicting situation appears when this task is scheduled with other tasks, it will not be possible to find a feasible schedule. However, since the task with flexible timing constraints provides many possible combinations for its four instances (as can be seen in Figure 9.3, executions are shaded), the chances of finding a feasible schedule increases.

9.1.3 Example

In the previous section we have explained that our new timing constraints for control tasks provide more flexibility than can be expressed by fixed timing constraints. In this section, using an offline schedule construction approach, we will show how flexible timing constraints for control tasks can be used to improve system schedulability. First, we present an example of a task set that cannot be scheduled using fixed timing constraints. By exploiting flexible timing constraints, however, we can construct a schedule that meets control demands (as we suggested in Section 5.4, in *Transforming unfeasible schedules into feasible schedules and stable control systems*).

Consider the task set shown in Table 9.1, where task $task_{sfc}$ is a control task implementing a classic state feedback controller designed through pole placement with observer for the inverted pendulum (see Section 2.2 for further details of the control design and Section A3 in Appendix A for the code details). Tasks $task_1$ and $task_2$ were added in order to introduce the jitters into the control task $task_c$.

	T_i	C_i	D_i	O_i
$task_1$	100	60	100	0
$task_2$	200	20	20	0
$task_{sfc}$	$h_{sfc,j}$	20	20	40

Table 9.1. Task set (in ms), where T_i, C_i, D_i and O_i denotes period, computation time, deadline and offset

Let us suppose that after control analysis (see Section 2.2.2), feasible values for the sampling period are 60, 80, or 100ms.

Firstly, consider scheduling using fixed timing constraints. If we schedule using fixed timing constraints, a single value for the period has to be assigned to task $task_{sfc}$. Therefore, for illustrative purposes an $h_{sfc,j}$ of 80ms has been chosen as the period for $task_{sfc}$, as must be done when using fixed timing constraints (see Section 9.1.1). Therefore in this example, $h_{sfc,j}$ for all instances of the control task $task_{sfc}$ is set to 80ms. Note that $task_{sfc}$ has a deadline equal to its computation time (20ms) and an offset of 40ms. It is obvious that both $task_{sfc}$ and $task_2$ need to execute between 200ms and 220ms to meet their respective deadlines, which is not possible, as shown in Figure 9.4. Therefore, a conflicting scheduling situation occurs at time 200ms. Figure 9.4 shows a partial schedule for 400ms, which corresponds to the LCM of the task periods. Note that this schedule is executed repeatedly where boxes mark periods of tasks and executions correspond to shaded areas.

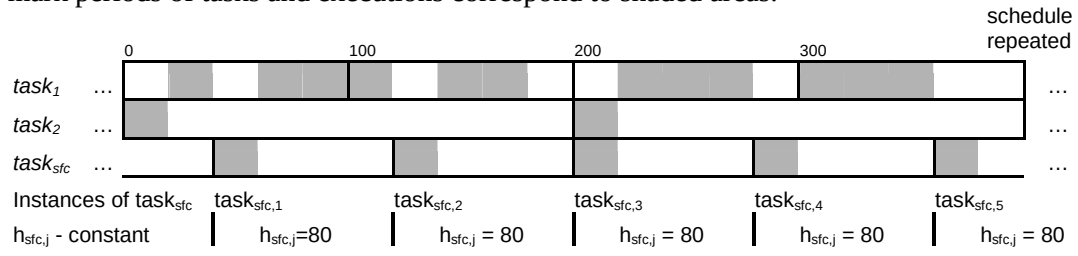


Figure 9.4. Non-feasible schedule (in ms)

Now, consider associating flexible timing constraints with $task_{sfc}$. Instead of selecting only one fixed $h_{sfc,j}$ for $task_{sfc}$, flexible timing constraints allow us to choose a specific $h_{sfc,j}$ for each instance. As we explained in Section 9.1.2, if we group the three feasible values for the sampling period in the set of feasible sampling intervals (FH_{sfc}) and use them to design the controller implemented in the control task using the compensation approach, we will have the three possible values to choose from as instance separations (FH_{sfc}={60,80,100} (in ms)). As in the schedule above, if we set, for instance $task_{sfc,2}$, $h_{sfc,2}$ (80ms), it will cause a scheduling conflict with $task_2$. Instead, we choose $h_{sfc,1}$ (60ms) and so $task_{sfc,3}$ finishes before $task_2$ starts. Then, we choose for $task_{sfc,3}$ $h_{sfc,3}$ (100ms) and $h_{sfc,2}$ (80ms) for the remaining instances. Using these $h_{sfc,j}$ values instead of a fixed period, the task set can be scheduled, as shown in Figure 9.5.

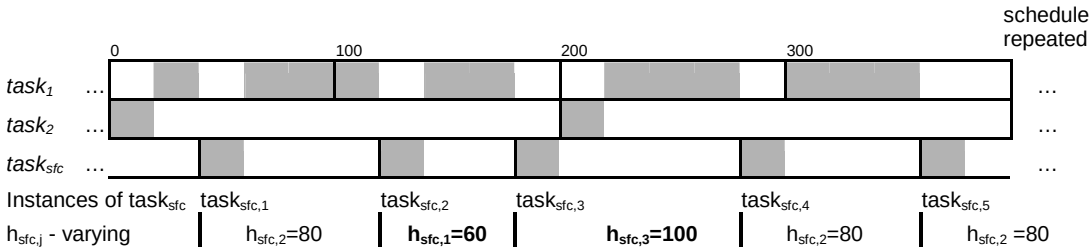


Figure 9.5. Feasible schedule (in ms)

Therefore, by taking advantage of flexible timing constraints, we obtain feasible schedules from non-feasible schedules based on fixed timing constraints.

Still, as Figure 9.6 shows (solid line), because the new controlled implemented now by $task_{sfc}$ has been designed using the compensation approach (like task $task_{Csfc}$ in Section A4 in Appendix code), which takes into account the specific $h_{sfc,j}$ used at each instance execution to update its controller parameters, stability is maintained and the inverted pendulum response meets the performance specifications (see Section 2.2.2) – even though task instances have different timing constraints.

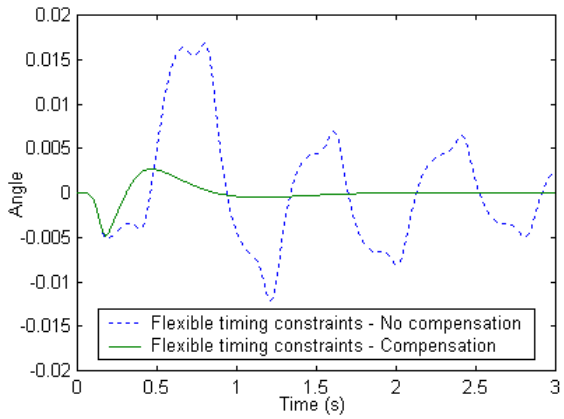


Figure 9.6. Inverted pendulum response of a task scheduled using flexible timing constraints (either using compensations or not), which was not schedulable using fixed timing constraints.

Note that in Figure 9.6 we do not compare this response with the response obtained by fixed timing constraints because the task set is not feasible. The dotted line in Figure 9.6 corresponds to the response if no compensations are used: in this case, although the task set is feasible due to the use of flexible timing constraints, the response is not good because the controllers parameters are not adjusted at run time. Therefore, the control performance requirements (to recover from a perturbation in less than two seconds) are not met, although the task set is feasible.

As the main conclusion, by applying flexible constraints for control tasks, the task set, which is not schedulable using fixed constraints, can be scheduled to meet the control demands, that is, the task set is feasible from both a scheduling and a control perspective.

9.2 Quality-of-Control (QoC) scheduling

Control systems are dynamic systems in which perturbation handling determines their performance, thus challenging the controlling strategy. To dynamically optimize the quality of the closed-loop system response when perturbations occur, we present in this section a new approach to control task scheduling, QoC scheduling, which is able to react to perturbations by tuning the control task execution.

To do so, we first discuss the impact of the different type of flexible timing constraints (instance separation and response time) on the closed-loop system error (Section 2.2.1). We argue and show experimentally that by selecting specific values of instance separations rather than of response times, the effective improvement we obtain on the closed-loop system performance - in terms of minimizing the system error - is considerable. This allows us to define a Quality-of-Control (QoC) metric that we firstly use to associate each instance separation value with a quantitative value expressing control performance in terms of the system error resulting from the use of that timing constraint. Then, using the QoC metric, we investigate the influence of different orderings of instance separation values on the overall QoC. We show how the QoC of the controlled system response can be optimized by appropriately selecting specific values for each control task instance separation constraint.

This offers the possibility of taking scheduling decisions based on the control information for each control task invocation (rather than fixed timing constraints with constant periods and deadlines), thus demanding novel scheduling approaches. We present a new scheduling problem, QoC scheduling, in which the QoC information of task timing constraints can be used to improve the performance of the controlled processes in the presence of perturbations. Throughout the specification of the QoC scheduling problem, we categorize the main scheduling issues and identify feasible solutions. In particular, we show how the problem of reacting to perturbations with control tasks specified with timing constraints expressing QoC can be solved applying standard guarantee techniques.

Note that controller designs attempt to minimize the *system error* for certain anticipated inputs or perturbations, i.e., to bring the actual response closer to the desired one, which also, for example, reduces the control signal magnitude and thus the required energy [AST97]. By defining the QoC metric and providing a solution for the QoC scheduling problem, we associate the responsibility for minimizing the system error with both the controller design and the schedule.

9.2.1 Quality-of-control criterion definition

In order to make the control information that control task flexible timing constraints incorporate explicit and available to the scheduler, we define a Quality-of-Control (QoC) metric that relates the performance of closed-loop systems with the timing of the controlling tasks. Instance separations (time intervals between consecutive samples) and response times (time intervals between related sampling and actuation instants) determine the timing of a controlling task. Consequently, we are interested in the influence of these timing parameters on the performance of the closed-loop system.

Performance of closed-loop systems

As we introduced in Section 2.2.2, the primary evaluation of closed-loop systems concerns meeting *response characteristics* and *stability*. Beyond these requirements, controller designs attempt to minimize the *system error* for certain anticipated inputs or perturbations. Since the compensation approach controller design method already ensures stability and meets the performance requirements (Section 7.1), we define a quality-of-control criterion in terms of controlled system response error, which is the difference between the desired response of the system and the actual response of the system (Section 2.2.1).

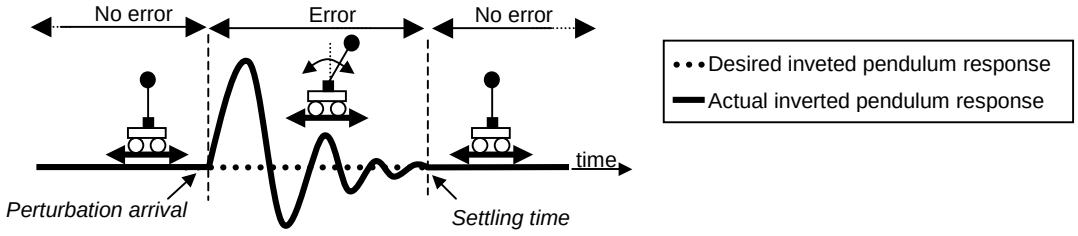


Figure 9.7. Inverted pendulum response error

For example, in the inverted pendulum problem (Section 2.2.2), the desired response is to maintain the vertical position (that is specified with angle zero as a reference) of the inverted pendulum at all times. However, a perturbation may unbalance the pendulum. The controller then is in charge of balancing it again, preventing it from falling. In this case, the system error is the difference between the desired response (zero angle) of the inverted pendulum and the actual response (oscillatory decreasing curve) in the presence of the perturbation, as we show in Figure 9.7.

As we reviewed in Section 2.2.1, classic performance indices used in control that give quantitative measures of closed-loop system responses in terms of errors are ISE (Integral of Square Error), ITSE (Integral of Time-weighted Square Error), IAE (Integral of the Absolute Error) or ITAE (Integral of Time-weighted Absolute Error) [DOR95].

Impact of flexible timing constraints on the closed-loop system error

As we explained in Section 2.2.2, sampling periods affect the location of the closed-loop poles, thus giving different degrees of performance. On the other hand, time delays only affect the controlled system performance in terms of delaying the response. Despite having different responses depending on different values for the sampling period (instance

separations) or time delays (response times), we define the QoC metric in terms of the instance separation constraint because it has a more drastic impact on the control performance than the impact of response times. To confirm this hypothesis, we evaluate separately the influence of different values of the task instance separation constraint and different values of the task response time constraint on the controlled system error.

For this evaluation, the IAE (or ISE) index would give the same evaluation (looking at the system error) for closed-loops designed with or without time delays, because they weight all the errors equally, regardless of the time they occur. Therefore, we use one of the indices that weight errors with time, ITAE or ITSE, thus penalizing delayed responses. We choose the ITAE index[2] (9.1):

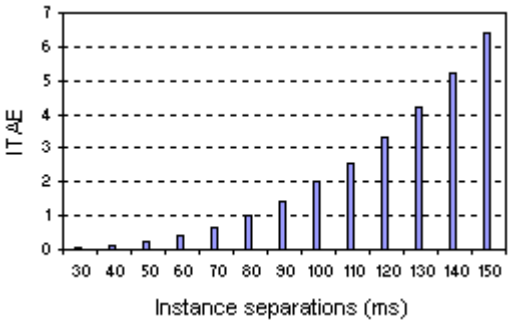
$$ITAE = \int_{t_0}^{t_f} t \cdot \left| y_{des}(t) - y_{act}(t) \right| dt \tag{9.1}$$

where y_{des} is the desired system response, y_{act} is the actual system response and t_0 and t_f are the initial and final times of the evaluation period.

To proceed with this evaluation, we use an inverted pendulum as a controlled process and we introduce a perturbation that causes an imbalance in the pendulum. As we saw in Section 2.2.2, several values for the sampling period (from 30 to 150ms, in steps of 10ms) and time delay (20 to 80ms, in steps of 10ms) guarantee stability and fulfil the control response performance specifications. Figure 9.8 shows the evaluation of the system error using the ITAE criterion (from the perturbation arrival to the settling time) when the inverted pendulum, in the presence of a perturbation, is controlled by a control task executing with a:

- constant value for all instance separations, ranging from 30 to 150 ms (Figure 5 left) and
- constant value for all response times, ranging from 20 to 80ms (Figure 5 right).

Instance separations	ITAE
30	0.053
40	0.127
50	0.249
60	0.430
70	0.680
80	1.008
90	1.425
100	2.007
110	2.562
120	3.306
130	4.183
140	5.207
150	6.393



[2] Note that using the ITSE index the results will be similar. However, the ITAE index, using the absolute error rather than using the squared error, gives a better linear relation between system errors resulting from the use of different instance separations.

Response times	ITAE
20	1.0468
30	1.0670
40	1.0872
50	1.1074
60	1.1276
70	1.1478
80	1.1680

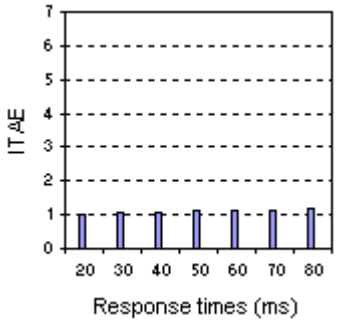


Figure 9.8. ITAE index depending on different instance separations (top) and response times (bottom)

From Figure 9.8 we confirm our hypothesis: *instance separations have stronger effects on the system error than response times*. Although ITAE weights later errors more heavily, thus penalizing longer response times, instance separations still have more influence on determining the system error. This conclusion leads us to decide to focus only on the relation between the values for the instance separation constraint and the system error in defining the QoC metric.

We define the QoC metric for each sequence of instance separation values in terms of the closed-loop system error. Here, we use the IAE criterion (9.2) because now we are interested in weighting all the errors equally. Note that we are defining an absolute metric for measuring the quality of the controlled system response given a control strategy (for example, a specific sequence of instance separations for a control task). Therefore, by weighting all the errors equally, the measured values will not be time-dependent, thus separating the error magnitude from the time it happens.

$$IAE = \int_{t_0}^{t_f} \left| y_{des}(t) - y_{act}(t) \right| dt \tag{9.2}$$

where y_{des} is the desired system response, y_{act} is the actual system response and t_0 and t_f are the initial and final times of the evaluation period.

Since the aim of controllers is to minimize the error, we define that the smaller the error, the better the QoC. That is, the relation between the IAE index and the QoC is inversely proportional. For that reason, we define the QoC metric in terms of the controlled system response error for a control task $task_i$ as follows[3]:

$$QoC(y_{act} : seq < h_{i,j} >, h_{i,j} \in FH_i) = \frac{\frac{1}{IAE(y_{act} : seq < h_{i,j} >)} - \frac{1}{IAE(y_{act} : seq < h_{i,\max} >)}}{\frac{1}{IAE(y_{act} : seq < h_{i,\min} >)} - \frac{1}{IAE(y_{act} : seq < h_{i,\max} >))}} \tag{9.3}$$

- where the *IAE error evaluation time interval* is the time elapsed from the time of occurrence of the perturbation (t_0) to the settling time (t_f). Note that, due to the control

[3] In the following, we will omit in all the symbols the i-subscript index that relates each symbol with the control task $task_i$ if this relation is already clear from the context.

analysis done at the design stage, the closed-loop performance specifications are met by all instance separation values. Consequently, the settling time is the same for all of them.

- where y_{act}:$seq<h_{i,j}>$ denotes that the actual system response has been obtained with a specific sequence of instance separation values for the control task, all belonging to the set given by all feasible sampling intervals (FH_i, Section 6.1.1). Note that y_{act}:$seq<h_{i,min}>$ denotes the actual system response when all instance separation values are equal to the $h_{i,min}$, which is the shortest instance separation of FH_i (similarly is y_{act}:$seq<h_{i,max}>$).

Note that if all instance separation values that apply are the same ($\forall h_{i,p},h_{i,q} \in seq<h_{i,j}> \rightarrow h_{i,p}=h_{i,q}$), the QoC metric allows us to associate each single instance separation value with a QoC measure. Note also that given different values for the instance separations that apply for a control task, the resulting QoC values will fall in the range of *[0,1]* (due to the normalization), where *zero* is equivalent to the worst QoC and *one* is the best QoC. In Figure 9.9 we show, numerically and graphically, the different values expressing control performance in terms of the QoC metric that can be associated with each instance separation value. The inverted pendulum, in the presence of a perturbation, is controlled by a control task executing with a constant value for the instance separation (ranging from 60 to 100 ms, in steps of 10ms) and a constant value of 20ms for the response time.

Instance separation	IAE	QoC
60	0.43	1.00
70	0.68	0.53
80	1.00	0.27
90	1.42	0.11
100	2.00	0.00

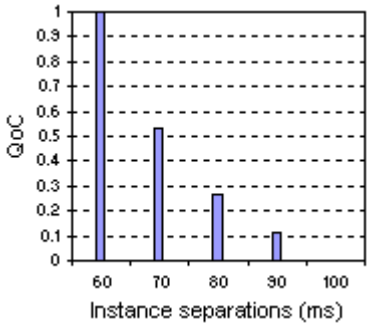


Figure 9.9. QoC of sequences of constant instance separations

In Figure 9.9 it can be clearly seen that a control task running at a constant instance separation of 60ms gives a better QoC than a task running at a constant instance separation of 80ms. Therefore, the main conclusion we draw is that *the shorter instance separation (although constant for all its execution) a control task has, the smaller the system error (the better the QoC).* This corroborates the results from control theory [AST97].

9.2.2 Influence of different instance separation sequences on the QoC

In the previous section we concluded that a control task running with a higher frequency (given by the instance separation) gives better QoC than the same control task running with a lower frequency. Recall that the control task, during each simulation (Figure 9.9), was assigned a constant instance separation. However, since our flexible timing constraints

allow us to choose specific values for the instance separation constraint at each instance execution, we are specifically interested in the influence of different instance separation orderings on the QoC of the controlled system in the presence of perturbations.

First of all, it is important to point out that the time elapsed from the *perturbation arrival* to the *perturbation detection* is important in terms of the *initial error* that the controller will have to account for. That is, the longer it takes to detect the perturbation, the bigger the system response deviation which needs to be controlled. In the following, assuming that the initial error is equal for all the simulations, we divide the study into four cases in order to cover all relevant situations when evaluating the influence of different instance separation orderings on the QoC.

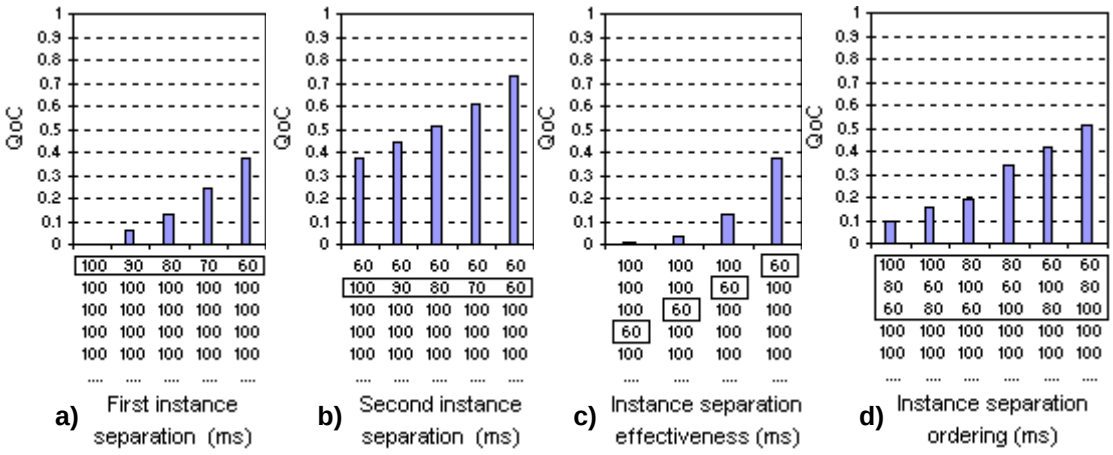


Figure 9.10. Instance separations sequences vs QoC

Figure 9.10 gives representative graphs of these four cases (in each graph, each x-axis column represents a sequence of instance separations, to be read from top to bottom, which has a QoC value associated in the y-axis). Note that we focus our simulations on the instance separations we used in the previous section (FH={60, 70, 80, 90, 100}, in ms). In all the simulation represented in Figure 9.10, the controlling task is executing at the lowest rate (100ms for all its instance separations) before the perturbation arrival. In each case, after the perturbation arrival, we study the effects of:

- First instance separation: Figure 9.10 a) shows that the first instance separation of each sequence has an important influence on the QoC of the inverted pendulum response. If the control task can arbitrarily choose any of the instance separations after the perturbation arrival, the smaller the chosen interval, the better the QoC (recall that a QoC of *1* is the best quality we can obtain and a QoC of *0* the worst, as we explained in Section 9.2.1). For example, if we look at the sequence starting with 60ms and the sequence starting with 80ms, choosing a first instance separation of 60 ms rather than 80 ms gives a better QoC.
- Second (and successive) instance separations: apart from the first instance separation selection, we have studied how different successive instance separations also affect

QoC. Simulations have shown that whatever the first instance separation is, the next instance separation of each sequence also has an influence on the QoC of the system. It can also be observed that the smaller the value chosen for the second instance separation, the better the QoC. Figure 9.10 b) exemplifies this property: given a first instance separation of 60ms, the chosen second instance separation clearly determines the QoC of the system response. The same holds for successive instance separations.

- Instance separation effectiveness: we have seen in the previous two cases that the shorter the instance separation, the better the QoC. Figure 9.10 c) shows the influence of a short instance separation (60ms) on the QoC depending on the time it is applied. It can be seen that the later a short instance separation applies, the less influence it has on improving the QoC. Extensive simulations indicate that short instance separations that apply later than the peak time (when the system error reaches its maximum value, see Section 2.2.1) have no significant effect on the QoC. The time elapsed from the perturbation arrival to the peak time, i.e., *perturbation reaction interval*, is the interval in which a short instance separation significantly improves the QoC of the system response.
- Instance separation ordering: we now focus on the ordering of such instance separations. In Figure 9.10 d) we show the effects of the different orderings of three instance separations (60, 80, and 100ms) on the QoC. The main conclusion we draw from this simulation is that the ordering of different instance separations is important in the sense that the earlier a short instance separation applies, the better the QoC.

To conclude, the influence of different instance separation orderings on the QoC of the controlled system response in the presence of perturbations can be summarized as follows: *the shorter and earlier, although varying, instance separations we have for instances of a control task, the better the QoC.*

9.2.3 Formulation of the QoC scheduling problem

Having described the QoC metric and the impact of instance separations on the QoC, we now formulate the issue of handling perturbations to optimize control response as a real-time scheduling problem. As we want to react to perturbations by executing a sequence of control task instances with short instance separations, the classic real-time approach based on fixed timing constraints with constant task parameter does not suffice.

Scheduling Objective

We mandate the following behaviour for a control task $task_i$ in terms of instance separation sequences:

- During the time the controlled system is in equilibrium (no error area in Figure 9.7), the control task instance separation should have the longest possible value ($h_{i,max}$). This way, the processor demand of the control task will be minimum, allowing an improvement in the schedulability of other tasks.
- Upon detection of a perturbation at the controlled system we want to achieve high QoC to counteract the perturbation.

We can achieve this by assigning shorter values for the control task instance separations (from the set of feasible sampling intervals) until equilibrium is reached again. Figure 9.11 illustrates the scheduling objective.

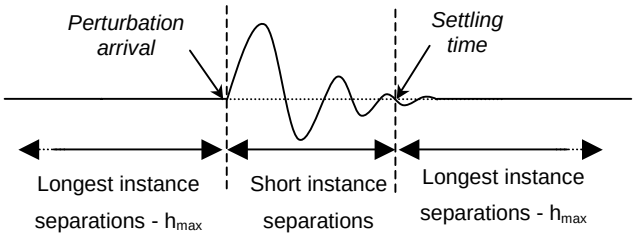


Figure 9.11. Scheduling objective

To be more accurate, since we observed (Section 9.2.2) that short instance separations later than the peak time have an insignificant effect on the QoC, we can reduce the application of short instance separations during the *perturbation reaction interval*. We can achieve these objectives with the following scheduling guidelines for each control task $task_i$:

- Guarantee the execution of each control task with an instance separation of $h_{i,max.}$
- In the *perturbation reaction interval*, schedule the control task with the shortest possible instance separation values $h_{i,j}$ (from the possible values given by set FH_i) based on schedulability of all tasks. In the worst case, we might fall back on a sequence of guaranteed $h_{i,max}$ separations – this ensures stability while providing the ability to improve the control response.[4]

Two issues arise from a scheduling perspective:

- At the beginning of the perturbation reaction interval, we will not execute the control task periodically with an instance separation of $h_{i,max}$ but have to abruptly change to the shortest possible instance separation value (*de-phasing).*
- Once the system is in equilibrium again, we want the control task to execute again with an instance separation of $h_{i,max}$, such that it conforms with the phasing before the perturbation to meet schedulability assumptions (*re-phasing)*. If the control was executing at times $t+p\times h_{i,max,}$ we want it to execute at times $t+p\times h_{i,max}$ after the perturbation reaction interval. As the $h_{i,j}$ values used in between will, in general, not be integer divisors of $h_{i,max}$, we have to construct a specific sequence of $h_{i,j}$ values to achieve the original phasing $t+p\times h_{i,max}$ again.

Note that the de-phasing/re-phasing problem becomes more difficult since for each control task $task_i$ we do not have a continuous range of $h_{i,j}$ values, but only a *finite set of values*. Thus, while being similar in objective to the period adjustment methods of the elastic task model ([BUT98] and [BUT02]), there is an important difference. The compression and decompression mechanisms in the elastic model regard the actual period of a task to be in a

[4] Obviously, shorter instance separations with better control performance can be guaranteed by scheudling assuming a value of h_j that is smaller than h_{max} however, this is at the expense of wasted resources when the controlled system is in equilibrium.

range of [T_{min}, T_{max}] and any task can vary its period according to its needs within the specified range. Our de-phasing and re-phasing problem is performed by selecting specific values for the control task instance separation from among the feasible ones given by the set FH_i. In summary, we don't have *continuous time*, as the elastic model requires, we have *discrete* values. Furthermore, the control tasks in our scenario have to complete at an exact point in time, as opposed to simply before deadlines, as in other approaches.

Scheduling strategies for the perturbation reaction interval

We propose the scheduling of the control tasks during the perturbation reaction interval to be done by an ensemble of control task instances with individual timing constraints reflecting the Quality-of-Control demands individually. The algorithm then tries to guarantee the ensemble in the presence of other, non-control tasks in a fashion similar to aperiodic tasks. As a consequence, our method is not bound to a specific scheduling algorithm. Rather it formulates a scheduling problem, which can be solved by a variety of algorithms ([LIU73], RAM89] and [FOH95]).

While the creation and guarantee testing of the task ensemble with appropriate individual timing constraints for the control instances is straightforward, re-phasing poses an additional problem. We have to find a sequence of instances such that the continuity of executing the control task $task_i$ at $t+p\times h_{i,max}$ is assured, while guaranteeing all instances in the presence of the other tasks and trying to minimize the length of the sequence. This is an optimization problem.

Optimum sequence: The construction of an optimum sequence to handle a perturbation can be done offline if the control task is the only task in the system. However, it is not possible in the realistic case of a set of control and non-control tasks, since the arrival time of the perturbation is unpredictable. At run time, on the other hand, limited resources will prevent the application of methods to determine an optimum sequence of instance separations. Consequently, we consider heuristic methods.

Ad-hoc sequence: We propose to create the task instance sequence in an ad-hoc fashion, i.e., from instance to instance. While not being optimum, we have the fallback option of the guaranteed $h_{i,max}$ value, which provides stable control, albeit of lower quality. Thus, when we can see that selecting shorter $h_{i,j}$ values will not re-phase within a reasonable number of instances, we stay with the guaranteed $h_{i,max}$ even after the perturbation.

Scheduling problem formulation

Assume a mixed task set:

- **Task set:** {$task_1$, …, $task_n$, $task_{n+1}$, …, $task_m$ | $task_i$ is a periodic task if $i \leq n$ or a control task if $i > n$, and $0 \leq n < m$, $n,m \in N$}
- **Periodic tasks:** every periodic task $task_i$ is characterized by fixed timing constraints: (T_i, D_i, C_i), where T_i is the period, D_i is the relative deadline and C_i is the worst-case execution time

- **Control tasks:** every control task $task_i$ is characterized by flexible timing constraints: (FH_i, rt_i, pri_i), where FH_i (set of feasible sampling intervals $h_{i,j}$ obtained in the control analysis) give the instance separation values, rt_i is the response time and is equal to the exact control task execution time and pri_i is the perturbation reaction interval.
- **Scheduling goal:** To find a feasible schedule, capable of meeting periodic and control tasks constraints, in such a way that,
    1. before each perturbation reaction interval, each control task $task_i$ is executing at its minimum rate, $h_{i,max}$
    2. during each perturbation reaction interval pri_i for each control task $task_i$, the $\Sigma h_{i,j}$ should be minimized in order to improve the QoC (where $\forall h_{i,p}, h_{i,q} \in pri$, if $\forall h_{i,p} < h_{i,q}$, $h_{i,p}$ precedes $h_{i,q}$).
    3. after each perturbation reaction interval, each control task has to recover its initial rate with the same phasing ($h_{i,max}$)

Moreover, looking at control tasks, recall that we focus on the instance separation constraint rather than the response time constraint. Instead of having a set of response times, we have a single - exact - value for the response time, which is given by the exact execution time of the control task (see Section 8.1.1).

9.2.4 Solution for the QoC scheduling problem

In this section we use an algorithm to illustrate how standard real-time scheduling techniques can be used to solve the QoC scheduling problem we formulated in the previous section. This particular solution is a *best effort* algorithm on a control task instance basis, to be mounted on top of the slot shifting method [FOH95]. However, first we present an example in order to illustrate the benefits of solving the QoC scheduling problem by exploiting flexible timing constraints (Section 9.1.1) and its QoC properties (Section 9.2.1) in such a way that the schedule meets the timing constraints while the QoC is improved.

Example

Let us suppose we have the set of tasks, including one control task, $task_{sfc}$, characterized using fixed timing constraints, as specified in Table 9.2, where the control task $task_{sfc}$ in charge of controlling the inverted pendulum (Section 2.2.2) implements a controller designed using classic pole placement techniques (see Section A3 in Appendix A for the code details). Deadlines are equal to periods and the control task is characterized by its exact execution time (see Section 8.1.1).

	T_i	C_i
$task_1$	40	20
$task_c$	80	20
$task_2$	120	20
$task_3$	240	20

Table 9.2. Task set (ms)

Using offline scheduling (note that the total utilization is 1, see Section 2.1.2), we construct the schedule that can be seen in Figure 9.12 over 240ms, which corresponds to the LCM of the tasks periods. Note that this schedule is executed repeatedly where boxes mark periods of tasks and shaded areas mark executions.

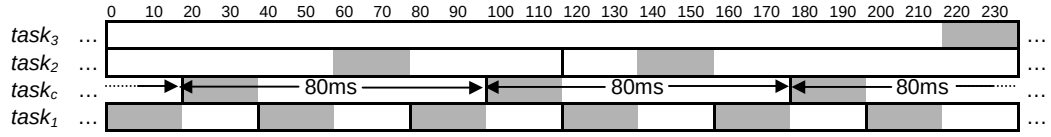


Figure 9.12. Offline construction (schedule repeated)

Due to the harmonic relationship between periods (Section 3.2.3), the control task $task_{sfc}$, in the offline schedule, keeps a constant instance separation (sampling interval) of 80ms. In addition, the offline schedule preserves a constant sampling-actuation delay of 20ms for the control task. Therefore, since it is not suffering any jitter apart from a start time delay of 20ms at each instance execution (which can be enforced by specifying an offset, see Section 2.1.1), the control task can execute a classic control law designed with a sampling period of 80ms and time delay of 20ms; parameters that meet the specifications for the inverted pendulum. As a consequence, the inverted pendulum system response we obtain is as good as can be expected (Figure 9.14, dotted line).

However, it is possible to find a better schedule in terms improving the QoC of the inverted pendulum response. If the control task is executing a code prepared for the run time parameters adjustment (task $task_{Csfc}$ in Section A4 in Appendix A), by taking advantage of the flexible timing constraints and the QoC notions, we can improve the performance of the system response while meeting the periodic task constraints.

Let us suppose that the control task, referred as $task_{Csfc}$, is now defined in terms of flexible timing constraints for the instance separation constraint, that is, it has a set of feasible sampling intervals to choose from, FH={60,80,100,120} (in ms), an exact computation time (c=20ms), and a perturbation reaction interval of 2s. For this case, rather than using an arbitrary offline schedule, we use a hypothetical online scheduling algorithm that solves the QoC scheduling problem. Assuming that the perturbation arrives at time 20ms, the online scheduling algorithm, at run time, obtains a new task ordering over the LCM of the periodic tasks (Figure 9.13) aimed at improving the QoC of the response. Note that in this case we keep the LCM of the periodic tasks as the time interval for which we want to improve the QoC by finding a better ordering of control tasks instances. The reason behind this particular restriction is to maintain a periodic pattern for all tasks instances executions during the LCM.

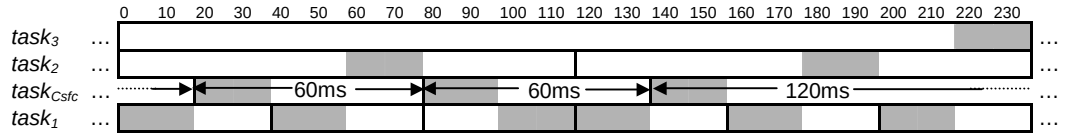


Figure 9.13. Run time schedule (schedule repeated)

In Figure 9.13 it can be seen that after the perturbation arrival time, the instance separations of the two successive control task instances are set to 60ms instead of 80ms as it was before. Note that since the processor utilization of the task set is already 1, the number of control task instances over the LCM cannot be increased. Therefore, in order to improve the QoC of the inverted pendulum response, the only change we can make in the schedule is to shift instances. In order to improve the QoC , it would be desirable, in less constrained schedules (in terms of processor utilization), to add as many control tasks instances as possible, while meeting the control task flexible timing constraints along with the periodic tasks fixed timing constraints.

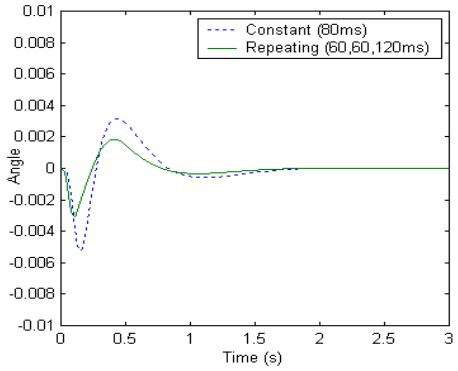


Figure 9.14. Inverted pendulum response obtained by the control task in the offline schedule (dotted) or by the ordering produced by the online algorithm (solid)

However, in this case, due to the actual schedule we have, in order to have two consecutive sampling intervals of 60ms for the control task and to keep the same periodic LCM execution pattern, the third sampling interval has to be set to 120ms, as can be seen in Figure 9.13. As a consequence, although in the first schedule the control task has 80ms as a constant sampling period, the response we obtain from the control task in the second schedule (Figure 9.14, solid line) is better in terms of QoC. Note that in the second schedule, during the perturbation reaction interval, the control task instance separations follow a periodic pattern of 60, 60 and 120 ms over the LCM. But since the first sampling intervals are shorter than 80ms, the QoC of the response is improved.

At this point it is important to stress that without increasing the processor utilization it is possible, given a set of tasks, to order these tasks in such a way that periodic task meets its timing constraints and control tasks improve the QoC of the closed-loop system response.

As we said before, in less constrained schedules (in terms of processor utilization), in order to improve the QoC, it would be desirable to add as many control tasks instances as possible. This is illustrated next.

If in the original task set (Table 9.2) we remove $task_3$, we obtain 20ms of idle time at each LCM. Then, during the perturbation reaction interval, the online algorithm could use the 20 ms to execute an extra instance of the control task $task_{Csfc}$, as we show in Figure 9.15. Observe that in this case, during the perturbation reaction intervals, the sequence of instance separations will occur with the shortest possible value (according to set FH), that is, 60ms. As before, periodic tasks constraints are still met.

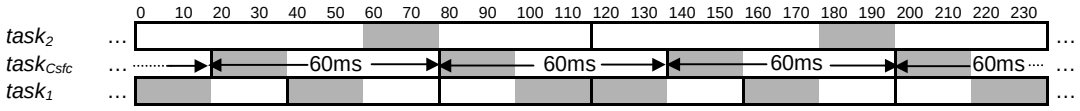


Figure 9.15. New run time schedule (schedule repeated)

In this case, the improvement in the inverted pendulum response is even better than when we had 60, 60 and 120 as a repeating sequence of instances separations during the perturbation reaction interval over the LCM (see Figure 9.16 solid line). However, it has to be pointed out that the price we pay for the trade off is an increase in the processor utilization, since four control task instances (instead of three) are executed over the LCM during the perturbation reaction interval.

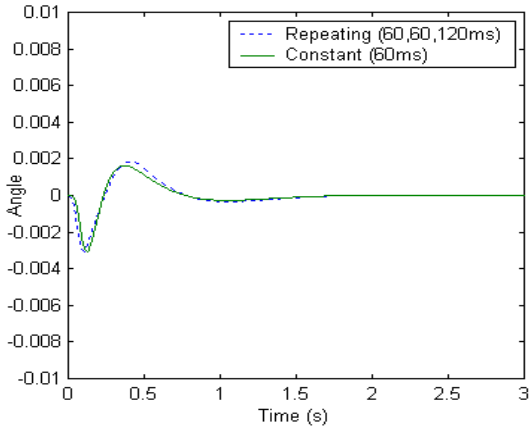


Figure 9.16. Inverted pendulum responses obtained by the control task in the ordering produced by the online algorithm whether new instances are added (solid line) or not (dotted line) during the LCM of the tasks periods

As a summary, the specific QoC obtained by each schedule is summarized in Table 9.3. We distinguish the three cases we studied: the response obtained by an arbitrary offline schedule and the responses obtained by the online algorithm case.

	Sequence of instance separations during the perturbation reaction interval	QoC
Arbitrary offline schedule	always 80ms	0.27
Online algorithm	60, 60,120ms	0.76
	60, 60, 60, 60 ms	1

Table 9.3. Achieved QoC for each scheduling strategy

We have shown that an online algorithm solving the QoC scheduling problem formulated in Section 9.2.3 can improve the QoC of closed-loop systems. With or without increasing the processor load, taking advantage of our flexible timing constraints and the QoC metric we defined, it is possible to find particular schedules for both control and non-control tasks, in such a way that non-control and control task constraints are maintained and the QoC of the closed-loop systems is improved.

Scheduling algorithm - an example

In this section we present a particular solution of an online algorithm that solves the QoC scheduling problem. We assume an offline scheduled task set where each control task $task_i$ has been scheduled according to $h_{i,max}$ and its exact response time (rt_i). In addition, the unused resources, i.e., processor idle time, are determined and grouped in the form of spare capacities for disjoint intervals [FOH95]. This allows us to use the *slot shifting based scheduling algorithm*. Slot shifting provides for the feasible integration of aperiodic tasks into offline schedules by allowing the offline scheduled tasks to be shifted in time while still meeting their original timing constraints.

As the focus of our method is quality-of-control and the construction of instance separation sequences rather than the actual scheduling, we do not detail the exact code of the guarantee algorithm. However, in Figure 9.17, we show its pseudo-code for illustrative purposes. Recall that the algorithm has to be executed upon detection of a perturbation in the process controlled by a control task $task_i$). In fact, the same procedure for sequence construction can be performed by applying a variety of scheduling algorithms.

For k=1:LCM($h_{i,max}$,$h_{i,min}$)/$h_{i,min}$

  $h_{i,j}$= first($h_{i,j} \in FH_i$) *% selects an instance separation value*

  While ($h_{i,j}<h_{i,max}$) and (instance at ($t+h_{i,j}$) $\not\subset$ [I_j | $sc(I_j)$ - $rt_i \geq 0$]) *% checks if **de-phasing** is feasible*

    $h_{i,j}$=next($h_{i,j} \in FH_i$) *% selects an instance separation value*

  End

  If $h_{i,j}<h_{i,max}$ *% Is current instance separation shorter?*

    recall updating: $sc(I_j) = sc(I_j) - rt_i$ *% to update in the offline schedule*

    new instance at ($t+h_{i,j}$)

  Else

    Stop *% test fails: end*

  End

  If $\exists h_{i,j} \in FH_i$: (k* $h_{i,max}$)-($t+h_{i,j}$) $\geq h_{i,j}$ *% is the **re-phasing** is feasible?*

    update and remove (sc, new instances, old instances) *% updates the offline schedule*

    Stop *% test success: end*

  Else

    recall to remove old instance at (k* $h_{i,max}$) *% to remove in the offline schedule*

  End

  $t=t+h_{i,j}$

End

where

- t is the time at which an instance of the control task $task_i$ starts executing, after the perturbation detection.
- $h_{i,max}$ and $h_{i,min}$ are the minimum and maximum instance separation given by FH_i
- I_j are disjoint execution intervals
- $sc(I_j)$ expresses unused resources
- FH_i is the set of feasible instance separations (ordered in increasing order)

Figure 9.17. Best effort algorithm

The algorithm is based on the following idea: at run time, upon perturbation detection, we test whether the control task instance separation can be decreased ($h_{i,j}<h_{i,max}$, better QoC), i.e., *de-phasing*, and smoothly increased again to recover the original rate ($h_{i,max}$), i.e., *re-phasing*. This is done, first of all, by selecting from all possible instance separations (given by set FH_i), a specific (and short) instance separation that allows a feasible execution (in the sense of having enough available processor idle time) of the next instance of the control task. Once this instance separation is found, the algorithm checks if the re-phasing is feasible from the time the new control task instance (given by the instance separation) executes. This is done by selecting again, in a monotonic fashion, new instances separations that give feasible executions of new control task instances until the re-phasing is meet.

If the test fails, no further action is taken. If the test does not fail, the offline scheduled tasks are shifted (without violating their feasibility), thus accommodating the new rate for the control task $task_i$. Note that this algorithm is a particular solution of the QoC scheduling problem. It shows that by using standard scheduling techniques, the QoC problem can be solved.

We illustrate the algorithm with a short example. Let us suppose we have a control task $task_i$ with $FH_i=\{70,90,100\}$. We have an offline schedule based on the longest $h_{i,j}$, which is 100 (first row in Figure 9.18). At run time, the perturbation is detected before the first instance executes. Therefore, at t=0, we invoke the best effort algorithm to check if we can accommodate successive instances in such a way that:

1. the following instance will have an instance separation of $h_i<100$ ms (instead of 100 according to the offline schedule) and
2. the original phasing can be met again.

If this is possible, we update the offline schedule according to the decisions taken by the algorithm.

In Figure 9.18, second row, we show, upon perturbation detection (t=0) a new execution pattern when the instance separation sequence is the optimum one (taking into account that we have a limited set of three instance separations): 70, 70, 70, 90, 100, 100, … ms. In Figure 9.18, third row, we show a new execution pattern (90, 70, 70, 70, 100, 100, … ms) that while not as good as the previous one, also improves the QoC because it applies shorter instance separations than the original offline schedule (100, 100, 100, … ms).

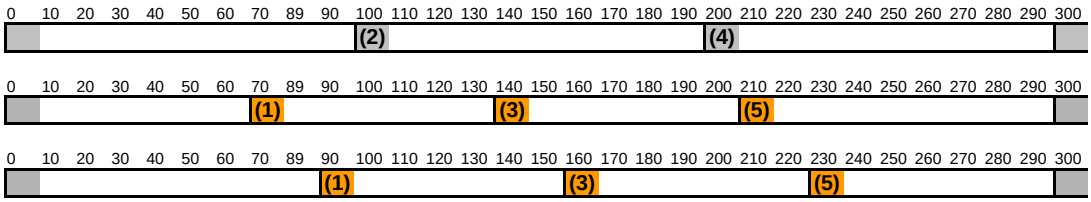


Figure 9.18. Offline schedule (first row), optimum schedule (second row) and sub optimum schedule (third row) de-phasing and re-phasing found by the best effort algorithm.

	Init	De-phasing			Re-phasing		
t time	**$h_{i,j}$** next $h_{i,j} \in FH_i$	**if** Can instance at $(t+h_{i,j})$ be guaranteed?	**Then** Memorize to update new instance $(t+h_{i,j})$	**Else** ***Test fails***	**If** Is the re-phasing feasible from new instance $(t+h_{i,j})$?	**Then** ***Test succeeds***	**Else** Memorize to remove original instance after time $(t+h_{i,j})$ Update time and go **Init**
0	70	Yes	instance (70) **(1)** Go re-phasing		No		remove instance (100) **(2)** $t=t+h_{i,j}$
70	70	Yes	instance (140) **(3)** Go re-phasing		No		remove instance (200) **(4)** $t=t+h_{i,j}$
140	70	Yes	instance (210) **(5)** Go re-phasing		Yes	Update memorizations	

	Init	De-phasing			Re-phasing		
t time	**$h_{i,j}$** next $h_{i,j} \in FH_i$	**if** Can instance at $(t+h_{i,j})$ be guaranteed?	**Then** Memorize to update new instance $(t+h_{i,j})$	**Else** ***Test fails***	**If** Is the re-phasing feasible from new instance $(t+h_{i,j})$?	**Then** ***Test succeeds***	**Else** Memorize to remove original instance after time $(t+h_{i,j})$ Update time and go **Init**
0	70	No Go **Init**					
	90	Yes	instance (90) **(1)** Go re-phasing		No		remove instance (100) **(2)** $t=t+h_{i,j}$
90	70	Yes	instance (160) **(3)** Go re-phasing		No		remove instance (200) **(4)** $t=t+h_{i,j}$
160	70	Yes	instance (230) **(5)** Go re-phasing		Yes	Update memorizations	

Table 9.4. Algorithm execution corresponding to the second row Figure 9.18 (top) and to the third row Figure 9.18 (bottom)

In Table 9.4 we illustrate the step-by-step execution of the best-effort algorithm (we have simplified the steps in the interest of clarity) corresponding to the execution of the second and third row of Figure 9.18 (numbers in bold in Figure 9.18 represent algorithm steps of Table 9.4). It has two main sections: in the first one, it checks whether increasing the task rate (de-phasing) is possible. If it is not possible, the test fails. If it is possible, a shorter instance separation than the one specified in the offline schedule can be accommodated. In this case, in the second section, it checks whether decreasing the task rate (re-phasing) will be possible for that specific new instance separation. The test succeeds when, after finding a sequence of one or more instance separations shorter than the ones specified in the offline schedule, re-phasing is guaranteed. For this simple case, the QoC improvement on the inverted pendulum responses (executing at h_{max} or executing with the second or third execution pattern for the control task $task_i$ implementing a code prepared for the run time parameters adjustment) can be seen in Figure 9.19.

In Figure 9 (where (4) is the desired system response), each of the curves corresponds to the inverted pendulum response if the controlling task, upon perturbation arrival, executes:

- the sequence given by offline schedule (first row Figure 9.18): curve (1) in Figure 9.19
- the optimum sequence corresponding to second row Figure 9.18: curve (3) in Figure 9.19
- the sub optimum sequence corresponding to third row Figure 9.18: curve (2) in Figure 9.19

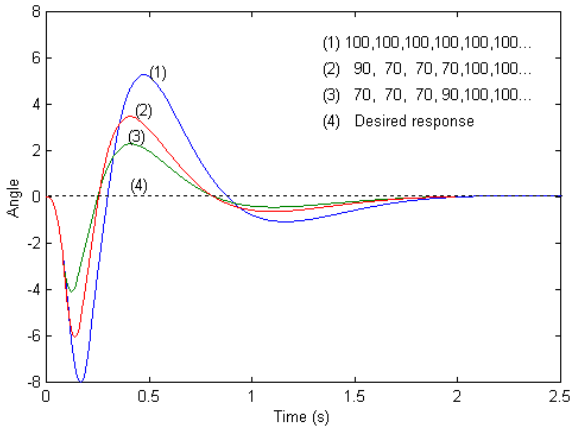


Figure 9.19. QoC improvement

It can be clearly seen in Figure 9.19 that response (2) offers the better QoC because it gives a better minimization of the error (difference between actual response (2) and desired response (4)). It is important to point out that beyond the different levels of QoC, all the responses ((1), (2) and (3)) fulfil the performance specifications (to recover from a perturbation in less than two seconds, see Section 2.2.2).

9.3 Summary

Control applications have timing requirements that cannot be expressed with deadlines and periods or which over-constrain the system strongly if expressed by standard timing constraints. In this chapter we have also defined novel, flexible timing constraints for control task scheduling that meet the control timing requirements better and we have demonstrated how to use them in order to improve system schedulability. Specifically, we have shown that by using these new flexible timing constraints along with the compensations, we obtain stable systems that are not feasible when scheduled using fixed standard timing constraints.

In addition, we have presented the QoC metric that associates control performance information with each feasible instance separation constraint. Through simulation results, we have shown that different orderings of instance separation values give different degrees of control performance. Also, we showed the larger impact that instance separation interval values have compared to response times. This has allowed us to formulate a new scheduling problem, QoC scheduling, where the problem specification considers both flexible timing constraints that incorporate control information as well as perturbation arrivals that occur in the controlled system. We have discussed different scheduling strategies to deal with the QoC scheduling problem (optimum vs. ad-hoc sequences) and we have shown that the problem of reacting to perturbations can be addressed by the application of existing scheduling guarantee techniques. This is an important observation because it shows the generality of previous approaches and also allows us to capitalize on extant algorithms.

Chapter 10

Conclusions

In this thesis we have presented novel methods and tools for the analysis and design of real-time control systems. The topics we covered can be divided into two main parts: one part dealing with the analysis and design of discrete-time controllers for irregularly sampled discrete-time systems with varying time delays and the other part focusing on flexible control task scheduling when control tasks are characterized by varying control timing constraints.

The integration of the two parts provides an adequate framework for the analysis and design of real-time control systems that fulfils the objectives of the thesis: it creates the adequate synergies between the activities of the control and real-time communities in such a way that integrating implementation characteristics into the controller design and control properties into real-time scheduling schemes improves the resulting real-time control systems in terms of both system schedulability and controlled system performance.

First of all, we suggested that the design of real-time control systems, which concerns the design of controllers and its computer implementation by means of scheduling techniques, requires the integration and good understanding of the activities of both control and real-time communities. However, as control theory and real-time scheduling theory have been relatively independent research areas, we started by describing, in a tutorial fashion, key concepts of real-time and control systems that are relevant for understanding the approach we have proposed.

Afterwards, we showed the negative effects of considering the results of control and real-time scheduling theory separately. We showed that the use of real-time standard task timing constraints (periods and deadlines) to express the inherent timing of classic discrete-time control models and methods produces over-constrained specifications for control tasks, which impairs system schedulability. On the other hand, scheduling inherent jitters for less constrained control tasks specifications violate the deterministic timing assumed by classic discrete-time control theory, resulting in a degradation of the controlled systems responses.

In addition, by surveying the state of the art in the field of real-time control systems, we pointed out that control theory offers no advice on how to include, in the design of controllers, the effects that scheduling inherent jitters have on the timing of the control activities. On the other hand, real-time theory lacks task models and timing constraints that can be used to guarantee a periodic task execution free of jitters without over-constraining system schedulability. We concluded that the practical problems posed by the constrained timing of discrete-time control theory in real-time scheduling (poor system schedulability) and by scheduling inherent jitters in control tasks (control performance degradation) have

not been formally addressed, and both control performance and systems schedulability have not been, in any of the previous works, jointly improved.

To solve the problems we outlined, and to overcome the difficulties posed by the application of discrete-time control theory and real-time task timing constraints, we proposed a more integrated approach to the analysis and design of real-time control systems, based on the use of more flexible controller design approaches and more flexible timing constraints for control tasks. Using more flexible controller design approaches based on the assumption of non-equidistant sampling and non-equidistant actuation, we are able to provide more flexible requirements for the timing of control activities, thus relaxing control task specifications and leading to higher levels of system schedulability. In addition, by using more flexible timing constraints for control tasks that do not introduce unexpected jitters while expressing the control timing requirements, the control performance requirements are met.

After analysing the effects that current computing implementations have on the timing parameters (sampling period and time delay) of classically designed controllers, we have formulated in state space models, the realistic timing requirements with which discrete-time controller design methods have to deal. Accordingly, we have presented a new approach to discrete-time controller design, called compensation approach, that takes these new requirements into account. The compensation approach can be used to handle all possible types of closed-loop implementations regarding the guarantees that the implementation gives in terms of the sampling type (*regular* or *irregular* sampling) and time delay type (*instantaneous*, *constant* or *varying* time delay) that will apply at run time.

Consequently, the compensation approach controller design method goes beyond the traditional discrete-time control assumptions presupposed by discrete-time control systems theory. Instead of selecting/specifying a single (and thus constant) value for the sampling period and time delay, we specify two finite sets of feasible values for the sampling period and time delay at the controller design stage. Then, at run time, control tasks adjust their controller parameters according to the specific timing that applies. This implies that closed-loop systems are no longer time invariant. Closed-loop system parameters vary depending on the specific setting for the sampling period and time delay that applies at each controller execution. For such closed-loop systems, we have used state-space notation to provide a complete stability analysis based on linear matrix inequalities algebra and response analysis.

In addition, we have discussed the implementation aspects of the applicability of controllers obtained through the compensation approach controller design method. We have used pseudo code details to explain how to modify existing controller codes to prepare controllers for the run time controller parameters adjustment that is required for the compensation approach. We have also distinguished two strategies for the run time parameters adjustment: the online recalculation approach, if this incurs negligible overheads, and the online access to pre-calculated tables. We have provided exact numbers for characterizing when each strategy is feasible.

Using the compensation approach we have shown how to eliminate the degradation that scheduling inherent jitters introduce in closed-loop systems. First of all, we have explained how to use standard scheduling approaches to schedule control tasks that implement

controllers designed using the compensation approach. In addition, we have shown that by using the compensation approach, we can remove the degradation that scheduling inherent jitters for control tasks introduce in the controlled system response. This is achieved by an offline analysis of all sampling intervals and sampling-actuation delays that apply at run time (for a given control task) due to sampling jitter and sampling-actuation jitter. All these values are then used to design the controller used in the compensation approach. Consequently, we showed that closed-loops implemented by periodic control tasks (specified with standard fixed timing constraints, such as periods and deadlines, and scheduled by standard real-time scheduling policies) and implementing these new controllers, although subject to scheduling inherent jitters, meet the control performance requirements.

We also have formally defined a performance criterion for evaluating the compensation approach as a design methodology for eliminating the degradation that jitters (for control tasks) cause in the closed-loop system performance. Through studying different scenarios given by different scheduling situations, we have stressed the benefits of using the compensation approach.

Our new approach to control task scheduling, which takes control properties into account, relies on the timing assumptions on which the compensation approach is based. From these timing assumptions (to depend on a set of feasible sampling intervals and on a set of feasible sampling-actuation delays), we have defined more flexible timing constraints for control tasks, in terms of two finite sets of values (instead of a single value for the former task period and deadline) of feasible instance separations and response times.

Therefore, we provide a range of values for control task timing constraints that can be selected on a “per task” instance basis by the scheduler to account for schedulability. According to these new control timing constraints, scheduling decisions can be taken to accommodate control task instances more flexibly. We have proved that unfeasible systems (with task sets including control and non-control tasks) characterized by standard timing constraints can be transformed into feasible and stable control systems by applying our flexible timing constraints and the use of the compensation approach.

New control task timing constraints allowed us to specify the scheduling problem, going beyond the classical “meeting deadlines”. Standard timing constraints express temporal requirements. Control task timing constraints, even those expressing temporal requirements, have been shown to also express control performance information.

We have used simulation results to show that different orderings of instance separation values give different degrees of control performance. Also, we showed that instance separation values have a larger impact than response times. Accordingly, we have defined a new metric, QoC metric, that allows us to associate control performance information (in terms of the controlled system error) with each feasible instance separation.

By associating QoC to each feasible instance separation, we have formulated a new scheduling problem, QoC scheduling, where controlled systems performance and schedulability are both of prime concern. We have also shown that standard scheduling techniques can be used to solve the QoC scheduling problem. We have explained how schedules can be adjusted at run time when perturbations are detected in the controlled

systems in order to execute control task instances more appropriately (that is, considering the QoC information) in order to improve the performance of the controlled systems.

In summary, we have proved that by integrating control and scheduling principles, better system schedulability and controlled systems performance improvement can be achieved in real-time control systems.

10.1 Contributions

The main contributions of this thesis, which extends the state of the art with respect to both control theory and real-time scheduling, are:

- To show the effects of degradation on
    a) controlled system performance due to scheduling inherent jitters in control task instance execution, and
    b) system schedulability when timing requirements of classically-designed discrete-time controllers are expressed with fixed timing constraints in periodic control tasks.

  (work partially presented in [MAR00b] and [MAR01a]).
- The state space formulation of closed-loop systems with irregular sampling and/or varying time delays, partially presented in [MAR01d]
- A new approach to discrete-time controller design, called *compensation approach*, based on the assumption of irregular sampling and varying time delays, including new stability and response analysis, partially presented in [MAR01d] and [MAR02c].
- The overhead analysis and implementation strategies for the use of the compensation approach, partially presented in [MAR01e]
- The application of the compensation approach as a control-based solution to eliminate the degradation that scheduling inherent jitters introduce in the controlled system response, partially presented in [MAR01b] and [MAR01e]
- The application of the compensation approach for the analysis and design of networked control systems, partially presented in [MAR01c]
- A performance metric for the evaluation of the compensation approach, partially presented in [MAR02c] and [YEP02]
- New flexible timing constraints for control task scheduling, partially presented in [MAR01e].
- The application of flexible timing constraints for control tasks to obtain feasible schedules of tasks sets that are not feasible if control tasks are characterised by fixed timing constraints, partially presented in [MAR01e]
- A Quality-of-Control (QoC) metric that characterizes task timing constraints in terms of control performance, partially presented in [MAR02a] and [MAR02b].

- The formulation and solution of the QoC scheduling problem in terms of reacting to perturbations with control tasks specified with flexible timing constraints expressing QoC, partially presented in [MAR02b].

10.2 Future work

There are many research topics that could extend the work presented in this thesis, bearing in mind the following points:

- We have shown that flexible timing constraints for control tasks can be used to transform unfeasible schedules into feasible schedules. Note that we have not proposed a scheduling approach, rather a new set of flexible timing requirements. Up to this point, we have used an offline schedule construction approach. An immediate extension of this thesis could be to investigate new scheduling schemes to handle these new types of constraints for control tasks.
- The use of standard scheduling techniques has been shown to solve the QoC scheduling problem we formulated. For example, we proposed a best effort approach to solve the problem. However, it could be a good extension of this thesis to develop new scheduling policies based on this new QoC paradigm.
- Real-time scheduling introduces variability in task instance executions. We have shown that with the use of the compensation approach, control tasks subject to jitters no longer degrade the performance of closed-loop systems. However, an extension of this thesis could be to develop and apply novel control approaches to the analysis and design of time-varying control systems, modelling this variability by uncertainties in the closed-loop system. Robust control techniques could be a starting point.
- In the compensation approach controller design, we did not mathematically formalize the characteristics of the controlled system response obtained with controllers designed using the compensation approach. Another interesting line of research points to the use of interval models based control approaches for the response analysis of control systems subject to time-varying parameters.
- The integration of control and real-time activities has proved to solve the problems that had no solution if the results of the control and real-time communities were applied in isolation. Continuing with this integration, an interesting area of research could be to treat schedules or scheduling policies as the plants to be controlled, and then to apply control theory to them.
- In this thesis we have defined flexible timing constraints for control tasks. One interesting extension could be the generalization of our varying control timing constraints to more flexible timing constraints for any type of task. Timing constraints are primarily used to express the application timing requirements. If the application timing requirements change at run time, timing constraints should be able to express these changes, according to the application dynamics. These would also require new scheduling approaches to deal with these new flexible timing constraints.

References

[ALB90] Albertos, P. and Salt, J. (1990). Digital Regulators Redesign with Irregular Sampling, *11th IFAC World Congress* (Preprints), vol 8, pp 157-161

[ALB00] Albertos, P., Crespo, A., Ripoll, I., Vallés, M. and Balbastre, P. (2000). RT control scheduling to reduce control performance degrading. *39th IEEE Conference on Decision and Control*. Sydney (Australia), December 12-15,

[ALB99] Albertos, P., and Crespo A. (1999). Real-time control of non-uniformly sampled systems. *Control Engineering Practice*, 7 (4) pp. 445-458.

[ARZ99] Årzen, K-E., Bernhardsson, B., Eker, J., Cervin, A., Nilsson, K., Persson, P. and Sha, L. (1999). Integrated Control and Scheduling. Research report ISSN 0820-5316. Dept. Automatic Control, Lund Institute of Technology

[ARZ00] K.-E. Årzen, A. Cervin, J. Eker and L. Sha. (2000). An Introduction to Control and Scheduling Co-Design. *39th IEEE Conference on Decision and Control*, Sydney, Australia, December 12-15

[AST97] Åström, K.J. and Wittenmark, B. (1997). *Computer-Controlled Systems. Theory and Design. Third edition*. Prentice Hall. ISBN 0-13-314899-8

[AUD93] Audsley, N.C., Burns, A., and R.I., Tindell (1993) The end of the Line for Static Cyclic Scheduling? *Fifth Euromicro Workshop on Real-Time Systems*, Oulo, Finland, IEEE Computer Society Press.

[AUD95] Audsley, N.C., Burns, A., Davis, R.I., Tindell, K.W. and Wellings, J. (1995). Fixed Priority Pre-emptive Scheduling: An historical Perspective. *Real-Time Systems, the Int. Journal of Time-Critical Computing Systems*, Vol. 8, N. 2/3,.

[BAL02] Balbastre, P., Ripoll, I. nnd Crespo, A. (2002). Schedulability analysis of window-constrained execution time tasks for real-time control. *14th Euromicro Conference on Real-Time Systems*, Viena, Austria

[BAR97] Baruah, S., Chen, D., and Mok, A. (1997). Jitter concerns in periodic task systems. *IEEE Real-Time Systems Symposium*, pp 68-77, December.

[BAR99] Baruah, S., Buttazzo, G., Gorinsky, S. and Lipari, G. (1999). Scheduling Periodic Tasks Systems to Minimize Output Jitter. *Proc. Int. Conf. on Real-Time Computing Systems and Applications*, IEEE Computer Society Press. pp 62-69

[BUR97] Burns, A. and Wellings, A. J. (1990). *Real-Time Systems and their Programming Languages. Second Edition*. Addison- Wesley.

[BUT97] Buttazzo, G. (1997). *Hard Real-Time Computer Systems. Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers.ISBN 0-7923-9994-3

[BUT98] Buttazzo, G., Lipari, G. and Abeni, L. (1998). Elastic Task Model for Adaptive Rate Control. *IEEE Real-Time Systems Symposium, Madrid*, Spain, December

[BUT02] Buttazzo G. and Abeni, L. (2002). Smooth Rate Adaptation through Impedance Control. *14th Euromicro Conference on Real-Time Systems*, Viena, Austria

[CAC00] Caccamo, M., Buttazzo, G. and Sha, L. (2000). Elastic Feedback Control. *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, pp. 121-128, June

[CER99] Cervin A. (1999) Improved Scheduling of Control Tasks. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, England, June

[CRE99] Crespo, A., Ripoll, I., and Albertos, P (1999). Reducing Delays in RT Control: The Control Action Interval. *IFAC World Congress*, Beijing

[CHA95] Chan, H. and Ozguner, U. (1995). Closed-loop control over a communication network with queues. *Int. Journal of Control*, vol 62, n.3, pp 493-510

[DOB01] Dobrin, R., Fohler,G and Puschner, P. (2001). Translating Off-line Schedules into Task Attributes for Fixed Priority Scheduling. *IEEE Real-Time Systems Symposium*, London, UK , December 2001

[DOG95] Dogruel, M and U. Özgüner (1995). Stability of a Set of Matrices: A Control Theoretic Approach. *Proc. of the 34th Conference on Decision and Control*, New Orleans, USA, September

[DOR95] Dorf, R.C. and Bishop, R.H. (1995). *Modern Control Systems*. Seventh Edition, Addison-Wesley, ISBN 0-201-84559-8

[EKE99] Eker, J. and Cervin, A. (1999). A Matlab Toolbox for Real-Time and Control Systems Co-Design. *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, Hong Kong, China, December.

[FOH94] Fohler, G. (1994). Flexibility in Statically Scheduled Real-Time Systems. Phd Thesis. Technisch-Naturwissenschaftliche Fakultaet, Technische Universitaet Wien, Austria, April

[FOH95] Fohler, G (1995). Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems, *IEEE Real-Time Systems Symposium*, December 1995

[FOH97] Fohler, G. (1997). Dynamic Timing Constraints - Relaxing Overconstraining Specifications of Real-Time Systems. *Proceedings of Work-in-Progress Session, 18th IEEE Real-Time Systems Symposium*. December 1997

[GAH95] Gahinet, P., Nemirovski, A., Laub, A. J., and Chilali, M.(1995). *LMI Control Toolbo*x. The MathWorks Inc, 1995.

[GAI01] Gai, P., Abeni, L., Giorgi, M., and Buttazzo, G. (2001). A New Kernel Approach for Modular Real-Time Systems Development, *IEEE 13th Euromicro Conference on Real-Time Systems*, Delft, The Netherlands, June

[GAR79] Garey, M.R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company

[JEF93] Jeffay; K. and Stone, D.L. (1993). Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. *Proceedings of the 14 th IEEE Real-Time Systems Symposium*, Raleigh-Durham, NC, December, pp. 212-221.

[JOS86] Joseph, M. And Pandya, P. (1986). Finding Response Times in a Real-Time System. *The Computer Journal* (British Computer Society) 29(5):390-395.

[KIM98] Kim, B.K. (1998). Task scheduling with feedback latency for real-time control systems. Proceedings. *Fifth International Conference on Real-Time Computing Systems and Applications*. Page(s): 37 –41

[KOP92] Kopetz, H. (1992). Sparse time versus dense time in distributed real-time systems. *12th Int. Conf. On Distributed Computing Systems*, pages 460-467,Yokohama, Japan, June.

[KOP97] Kopetz, H. (1997). *Real-time systems. Design principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Third Printing 1999. ISBN 0-7923-9894-7

[LIA02] Lian, F., Moyne, J., and Tilbury D. (2002). Network Design Consideration for Distributed Control Systems. *IEEE Trans. on Control Systems Technology*, Vol.10, No.2, March

[LIU73] Liu, C., and Layland, J. (1973). Scheduling Algorithms for multiprogramming in a hard-real-time environment. *J.Association for Computing Machinery*, 20:46-61.

[LOC92] Locke, C.D. (1992). Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives. *Journal of Real-Time Systems*, 4, 37-53, Kluwer Academic Publishers.

[LU99] Lu, C., Stankovic, J., Tao, G. and Son, S. (1999). Design and Evaluation of a Feedback Control EDF Algorithm. *IEEE Real-Time Systems Symposium*, December

[LU00] Lu, C., Stankovic, J., Abdelzaher, T., Tao, G., Son, S., and Marley, M. (2000) Performance Specifications and Metrics for Adaptive Real-Time Systems. IEEE Real-Time Systems Symposium, December

[LUE79] Luenberger, D.G. (1979). *Introduction to Dynamic Systems. Theory, Models and Applications*. John Wiley & Sons, ISBN 0-471-02594-1

[MAR99] Marcos, M., Bass, J.M., Fleming, P. J., and Portillo, J. (1999). An integrated framework for the development of real-time distributed control software based on CAN bus. *Proceedings of the 14th World Congress of IFAC*, pp. 447-452. Beijing, P. R. China (July).

[MAR00a] Marcos, M., Portillo, J., Bass, J.M. (1999). Matlab-Based Real-Time Framework For Distributed Control Systems. *Proceedings IFAC Workshop AARTC'2000*, Palma de Mallorca (Spain), May

[MAR00b] Marti, P., Villà R., Fuertes, J.M, and G. Fohler (2000). Real Time Scheduling Methods Requirements in Distributed Control Systems. *Proceedings of the 5th IFAC Workshop on Real-Time Programming*. Palma de Mallorca. Spain. May. pp 101-108

[MAR01a] Marti, P., Villà R., Fuertes, J.M, and G. Fohler (2001). On Real-Time Control Tasks Schedulability. *Proceedings of European Control Conference*, Porto, Portugal, September 2001, pp 2227-2232

[MAR01b] Marti, P., Fuertes, J.M, and G. Fohler (2001). Sampling Jitter Compensation in Real-Time Control Systems. *Work-in-progress Session, 13th Euromicro Conference on Real-Time Systems*, Delft, The Netherlands, June

[MAR01c] Marti, P., Fuertes, J.M, and G. Fohler (2001). An Integrated Approach to Real-time Distributed Control Systems Over Fieldbuses. *Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation*. Antibes Juan-les-pins. France. October.

[MAR01d] Marti, P., Villà R., Fuertes, J.M, and G. Fohler (2001). Stability of On-line Compensated Real-Time Scheduled Control Tasks. *IFAC Conference on New Technologies for Computer Control*, Hong Kong, November.

[MAR01e] Marti, P., Fohler, G., Ramamritham, K., and Fuertes, J.M. (2001). Jitter Compensation in Real-Time Control Systems. *IEEE Real-Time Systems Symposium*, London, UK, December

[MAR02a] Marti, P., Fuertes, J.M, and G. Fohler (2002). A Control Performance Metric for Real-time Timing Constraints. *Work-in-progress Session, 14th Euromicro Conference on Real-Time Systems*, Viena, Austria, June

[MAR02b] Marti, P., Fohler, G., Ramamritham, K., and Fuertes, J.M. (2002). Control Performance of Flexible Timing Constraints for Quality-of-Control Scheduling. Submitted to *IEEE Real-Time Systems Symposium*, Austin, USA, December.

[MAR02c] Marti, P., Villa, R., Fuertes, J.M. and Fohler, G. (2002). A Controller Design Method to Compensate for Real-time Scheduling Inherent Jitters. Submitted to *41th IEEE Conference on Decision and Control*, Las Vegas, USA, December.

[MIT01] Mitter, S.K. (2001). Control With Limited Information. *European Journal of Control*, 7:122-131, EUCA

[MOT00] Mota, A., Fonseca (2000). Dealing with Jitter in Systems Modelling and Identification, *CONTROLO'2000 - 4th Portuguese Conference on Automatic Control*, Guimarães, Portugal, 4-6 October.

[NIL98a] Nilsson, J. (1998). Some Topics in Real-Time Control. *ACC*, June, 24-26. Philadelphia.

[NIL98b] Nilsson, J.,Bernhardsson, B. and Wittenmark, B. (1998). Stochastic analisys and control of real-time systems with random time delays. *Automatica*, 34:1, p.57-64.

[OGA97] Ogata,K. (1997) *Modern Control Engineering, 3rd ed.* Englewood Cliffs, NJ Prentice-Hall cop. ISBN 0-13-261389-1

[PAL00] Palopoli, L., Abeni, L., Conticelli, F., Di Natale, M., Buttazzo, G. (2000). Real-Time control system analysis: an integrated approach. *Proc. of the Real-Time System Symposium*, Orlando, Florida, November.

[PAL02] Palopoli, L., Pinello, C., Sangiovanni Vincentelli, A., Elghaoui, L. and Bicchi, A. (2002). Synthesis of robust control systems under resource constraints. *In Proc. of Hybrid Sytems: computation and control*, Stanford 25/27, USA, March

[PHI95] Philips, C.L. and Nagle, H.T. (1995) *Digital Control Systems. Analysis and design. Third Edition.* Prentice Hall. ISBN 0-13-309832-X

[PUS89] Puschner, P. and Koza Ch. (1989). Calculating the maximum execution time of real-time programs. *Real-Time Systems*, 1(2):159{176, Sep..

[RAM90] Ramamritham, K. (1990). Allocation and Scheduling of Complex Periodic Tasks. *In 10th Int. Conf. on Distributed Computing Systems*, pages 108--115.

[RAM94] Ramamritham, K. and Stankovic, J.A. (1994). Scheduling Algorithms and Operating Systems Support for Real-Time Systems. *Proc. the IEEE*, Vol. 82, NO.1, January

[RAM96] Ramamritham, K. (1996). Where do time constraints come from and where do they go? *International Journal of Database Management*, 7:2

[REH00] Rehbinder, H. and Sanfridson, M. (2000). Integration of Off-Line Scheduling and Optimal Control. *12th Euromicro Conference on Real-Time Systems*, June

[SCH01] Schinkel, G. and A. Rantzer (2001). Sample Data with Varying Sampling Time. *Proc. European Control Conference*, September, Porto, Portugal, pp. 624-628

[SET96] Seto, D., Lehoczky, J.P., Sha, L. and Shin, D.G. (1996). On Task Schedulability in Real-Time Control Systems. *IEEE Real-Time Systems Symposium* pp. 13-21

[SHI85] Shin, K.G, Krishna, C.M. and Lee, Y-H. (1985). A Unified Method for Evaluating Real-Time Computer Controllers and Its Applications. *IEEE Transactions on Automatic Control*, Vol. AC-30, No.4, April, pp 357-366

[SHI92] Shin, K.G and Kim, H. (1992). Derivation and Application of Hard Deadlines for Real-Time Control Systems. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 22, No.6, November/December, pp 1403-1412

[SHI94] Shin, K.G and Ramanathan, P. (1994). Real-Time computing: A New Discipline of Computer Science and Engineering. *Proceedings of the IEEE*, Vol. 82, NO.1, January 1994.

[SHI96] Shin K., and Cui X. (1996). Computing time delay and its effects on real-time control systems. *IEEE Transactions on Control Systems Technology*. Vol. 3, N. 2, p.218-224

[STA88] Stankovic, J.A. (1988). Misconceptions About Real-Time Computing: A serious Problem for Next Generation Systems. *IEEE Computer*. 21(10):10-19.

[STA90] Stankovic, J.A., and Ramamritham, K. (1990). Editorial: What is predictability for real-time systems? *Real-Time Systems* 2(4):247-254.

[STA94] Stankovic, J.A and DiNatale, M (1994). Dynamic end-to-end guarantees in distributed real-time systems. *Proceedings of the Real-Time Systems Symposium*, San Juan, uerto Rico, December.

[STA95] Stankovic, J.A. et. al. (1995). Implications of classical scheduling results for real-time systems. *IEEE Computer, Vol. 28, No. 6, pp. 16-2*5, June.

[STA98] Stankovic, J.A., Spuri, M., Ramamritham, K and Buttazzo, G. (1998). *Deadline Secheduling for Real-Time Systems. EDF and Related Algorithms.* Kluwer Academic Publishers. ISBN 0-7923-8269-2

[STA99] Stankovic, C. Lu, S. Son, and G. Tao (1999). The Case for Feedback Control Real-Time Scheduling. *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, England, June

[STR80] Strang, G. (1980) *Linear Algebra and its Applications. Second Edition.* MIT. Academic Press, Inc. USA

[TIN94] Tindell, K. W., Burns, A., and Wellings, A. J. (1994). An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems The International Journal of Time-Critical Computing*, 6:133--151

[TÖR95] Törngren M. (1995). Modelling and Design of Distributed Real-Time Control Applications. Ph-D thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Swedwn.

[TÖR97] Törngren, M., Eriksson, C., and Sandström, K. (1997). Deriving Timing Requirements and constraints for Implementation of multirate Control applications. Research Report TRITA-MMK 1997: 1, ISSN 1400-1179, ISRN KTH/MMK/R-97/1-SE. Department of Machine design, The Royal Institute of Technology, S-100 44 Stockholm. Sweden

[TÖR98] Törngren, M. (1998). Fundamentals of implementing real-time control applications in distributed computer systems. *Journal of Real-Time Systems*, 14, 219-250

[VAC95] Vaccaro, R.J. (1995). *Digital control. A state-sapace approach.* McGraw-Hill

[WIT80] Wittenmark, B. and Åström, K.J. (1980). Simple Self-tuning Controllers. *In Unbehauen, Ed. Methods and Applications in Adaptive Control, number 24 in Lecture Notes in Control nd Information Sciences*, pp 21-29. Springer-Verlag, Berlin, FRG

[WIT95] Wittenmark, B., Nilsson J., Törngren M. (1995). Timing Problems in Real-Time Control Systems: Problem Formulation. *Proc. Of the American Control Conference*, Seattle, Washington

[WIT98] Wittenmark, B., Bastian, B. and Nilsson J. (1998). Analysis of Time Delays in sinchronous and Asynchronous Control Loops. *Proc. Of the 37th Conf. On Decision and Control*, Tampa, FL. USA

[WIT01] Wittenmark, B. (2001). Sample-Induced Delays in Synchronous Multirate Systems. *Proc. European Control Conference*, September, Porto, Portugal, pp. 3276-3281

[YEP02] Yepez, J., Marti, P. and Fuertes, J.M. (2002). Control Loop Performance Analysis over Networked Control Systems. *Accepted to 28th Annual Conference of the IEEE Industrial Electronics Society*, Sevilla, Spain, November

[ZHA01] Zhang, W., Branicky, M.S. and Phillips, S.M. (2001) Stability of Networked Control Systems. *IEEE Control Systems Magazin*, pp 84-99, November

Appendix A: Code implementation details

A1. Classic PID controller code

Pseudo-code of a classic designed PID controller, used in the simulations:

```
task_PID()
   {
       r = analogIn(rChan);
       y = analogIn(yChan);
       E = r-y;
       P = Kp·E;
       I = Iold+(Ki·h/2)·(E+Eold);
       D = (Kd/h)·(E-Eold);
       u = P + I + D;
       analogOut(uChan,u);
       Iold = I;
       Eold = E;
    }
```

A2. PID controller code prepared for the run time parameters adjustment

In this case, the only change in the code of the PID controller is to obtain the feasible sampling interval h_k that applies at each controller executions. In this case, there is no difference in adjusting the parameters through online computations or online access to indexed tables (because the overhead of the online computations, compared to the classic code (A1) is negligible).

```
Task_CPID()
   {
       r = analogIn(rChan);
       y = analogIn(yChan);
       E = r-y;
       obtain(h_k);
       P = Kp·E;
       I = Iold+(Ki·h_k/2)·(E+Eold);
       D = (Kd/h_k)·(E-Eold);
       u = P + I + D;
       analogOut(uChan,u);
       Iold = I;
       Eold = E;
    }
```

A3. Classic State feedback controller code based on pole placement observer design

Pseudo-code for a classic state feedback controller, used in the simulation (note that all the matrices are calculated before system run time):

```
Task_sfc()
   {
         r = analogIn(rChan);
         y = analogIn(yChan);
         u = -(L·ob)+r;
         analogOut(uChan,u);
         ob=Φ·ob+Γ·u+K^T·(y-C·ob);
      }
```

A4. Classic State feedback controller code based on pole placement observer design prepared the run time parameters adjustment

In this case, the change in the code of the state feedback controller is made to obtain the feasible sampling interval h_k and feasible sampling actuation-delay τ_k that apply at each controller execution, as well as the system discretization, gain, and observer matrices. Therefore, depending on whether we consider the overhead of the extra computations to be negligible or not, in the following, we show the two possible solutions.

Through online computations:

```
Task_Csfc()
   {
         r = analogIn(rChan);
         y = analogIn(yChan);
         obtain(h_k,τ_k);
         Ad = substitute(Ac,h_k→h,τ_k→τ);
         B0d = evaluate(B0c,h_k→h,τ_k→τ);
         B1d = evaluate(B1c,h_k→h,τ_k→τ);
         Φ = [Ad B1d; 0 0];
         Γ = [Bd0; I];
         dominantpole1 = -2·exp(-ζ·ω·h_k)·cos(ω·h·sqrt(1-ζ^2));
         dominantpole2 = exp(-ζ·ω·h_k);
         desiredpol = [function(dominantpole1,dominantpole2);
         L = poleplacement_akerman(desiredpol,Φ,Γ);
         observerpoles = function(desiredpol);
         K = poleplacement_Akerman(observerpoles,Φ^T,C^T);
         u = -(L·ob)+r;
         analogOut(uChan,u);
         ob=Φ·ob+Γ·u+K^T·(y-C·ob);
      }
```

Through online access to indexed tables:

```
Task_CTsfc()
   {
         r = analogIn(rChan);
         y = analogIn(yChan);
         obtain(h_k,τ_k);
         (Φ,Γ,L,K) = Indexedtableacess(table,h_k,τ_k)
         u = -(L·ob)+r;
         analogOut(uChan,u);
         ob=Φ·ob+Γ·u+K^T·(y-C·ob);
      }
```

Appendix B: Stability analysis

B1. Stability analysis of Example 1 in Section 7.1.4

The periodic sequence of feasible sampling intervals and feasible sampling-actuation delays we have is: (<(60,20),(50,10),(50,20),(60,10)>, in ms).

The stability test is performed by checking the stability of the product of the repeating sequence of closed loop matrices. The four closed loop matrices are

$$\Phi_{cl_1} = \Phi_{cl}(60,20) = \begin{bmatrix} 0.005517 & -0.000348 & -0.004462 & -0.001452 & -0.000006 \\ -0.230940 & -0.037530 & -0.223722 & -0.072839 & -0.000003 \\ 0.002412 & 0.000476 & 0.012225 & 0.001324 & 0.000002 \\ 0.120795 & 0.023817 & 0.111278 & 0.046229 & -0.000000 \\ 6.053064 & 1.191031 & 5.562446 & 1.811019 & -.0005008 \end{bmatrix}$$

$$\Phi_{cl_2} = \Phi_{cl}(50,10) = \begin{bmatrix} 0.004135 & -0.000649 & -0.006253 & -0.001843 & -0.000002 \\ -0.296583 & -0.047607 & -0.313542 & -0.092445 & 0.000002 \\ 0.003047 & 0.000575 & 0.013118 & 0.001419 & 0.000001 \\ 0.152438 & 0.028776 & 0.155953 & 0.055981 & -0.000002 \\ 7.632311 & 1.438742 & 7.795654 & 2.298487 & -0.002605 \end{bmatrix}$$

$$\Phi_{cl_3} = \Phi_{cl}(50,20) = \begin{bmatrix} 0.006686 & -0.000179 & -0.003513 & -0.001071 & -0.000005 \\ -0.228102 & -0.035377 & -0.234592 & -0.071513 & -0.000032 \\ 0.001777 & 0.000341 & 0.011754 & 0.0010347 & 0.000002 \\ 0.118648 & 0.022744 & 0.116952 & 0.0456519 & 0.000014 \\ 7.925202 & 1.516516 & 7.795654 & 2.3764443 & -0.005692 \end{bmatrix}$$

$$\Phi_{cl_4} = \Phi_{cl}(60,10) = \begin{bmatrix} 0.003063 & -0.000813 & -0.006982 & -0.002203 & -0.000002 \\ -0.281121 & -0.046697 & -0.280515 & -0.088525 & 0.000013 \\ 0.003630 & 0.000707 & 0.013477 & 0.0016973 & 0.000001 \\ 0.145326 & 0.028294 & 0.139118 & 0.0539028 & -0.000007 \\ 5.822601 & 1.131663 & 5.562446 & 1.7553952 & -0.002312 \end{bmatrix}$$

Applying this test we are satisfied that the system is stable because $\rho(\Phi_{cl_1} \cdot \Phi_{cl_2} \cdot \Phi_{cl_3} \cdot \Phi_{cl_4}) = 0.349 < 1$, where matrix $\Phi_{cl_1} \cdot \Phi_{cl_2} \cdot \Phi_{cl_3} \cdot \Phi_{cl_4}$ is:

$$\Phi_{cl_1} \cdot \Phi_{cl_2} \cdot \Phi_{cl_3} \cdot \Phi_{cl_4} = \begin{bmatrix} -1.138678 & -0.332271 & -1.181209 & -0.629691 & 0.000165 \\ 13.312320 & 2.498077 & 21.884891 & 4.890800 & -0.000174 \\ 1.034929 & 0.236241 & 1.337305 & 0.449630 & -0.000107 \\ -8.118195 & -1.305419 & -14.058272 & -2.568315 & 0.000023 \\ -82.016508 & -21.186792 & -42.825478 & -37.747148 & 0.028516 \end{bmatrix}$$

B2. Stability analysis of Example 2 in Section 7.1.4

We have an infinite sequence of feasible sampling intervals taken randomly from FH={50,60} with an infinite sequence of feasible sampling-actuation delays taken randomly from FT={10,20} (granularity g=10 ms).

$\Omega = \{\Phi_{cl_k} \mid \Phi_{cl_k}$ is the closed loop matrix that depends on (h_j,τ_j), $h_j \in H$,$\tau_j \in T$, for all possible combinations of pairs $(h_j,\tau_j)\}$.

Possible combinations of pairs (h_j,τ_j) are (50,10), (50,20), (60,10) and (60,20), in ms.

Therefore, set Ω has four matrices (which coincide with the matrices listed in the previous case, B1): $\Omega = \{\Phi_{cl}(50,10), \Phi_{cl}(50,20), \Phi_{cl}(60,10), \Phi_{cl}(60,20)\}$

Applying

$$\text{If } \exists\, P>0: \forall\, \Phi_{cl_k} \in \Omega,\ \Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0 \Rightarrow \text{ Stable}^1$$

we we are satisfied that the set of matrices Ω fulfils the condition. That is, we we are satisfied that a matrix P>0 such as for all four closed loop matrices Φ_{cl_k} of Ω, $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$.

$$P' = \begin{bmatrix} 2.3229543 & 0.3526428 & 3.0679626 & 0.6544445 & -0.0004749 \\ 0.3526428 & 0.1037981 & 0.4604631 & 0.1920243 & -0.0000580 \\ 3.0679626 & 0.4604631 & 4.7176875 & 0.8626853 & -0.0002955 \\ 0.6544445 & 0.1920243 & 0.8626853 & 0.3585137 & -0.0001056 \\ -0.0004749 & -0.0000580 & -0.0002955 & -0.0001056 & 0.0000008 \end{bmatrix}$$

$$P = P' \cdot 1.0e{+}004$$

To solve this linear matrix inequality problem, we use the Linear Matrix Inequality (LMI) solver included in the Matlab LMI toolbox.

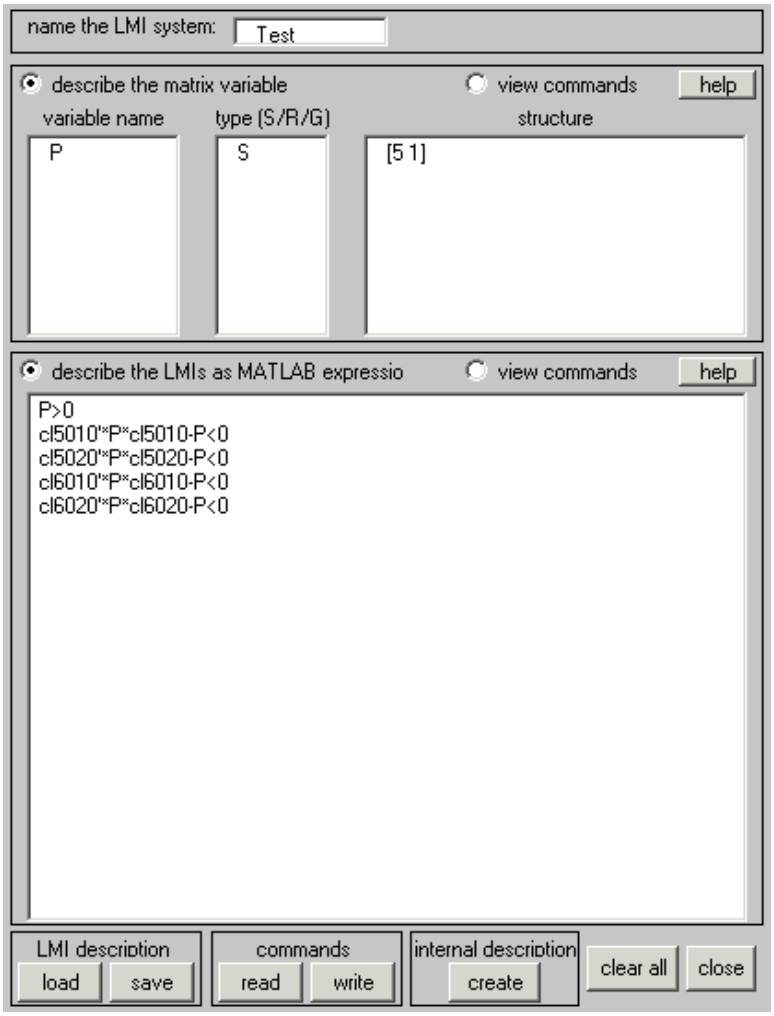
Details of the stability test (in Matlab pseudo code, using Matlab LMI solver):

% we define the four matrices

$\Phi_{cl_1} = \Phi_{cl}(50,10)$ %named cl5010
$\Phi_{cl_2} = \Phi_{cl}(50,20)$ %named cl5020
$\Phi_{cl_3} = \Phi_{cl}(60,10)$ %named cl6010
$\Phi_{cl_4} = \Phi_{cl}(60,20)$ %named cl6020

[1] P>0 in the sense of positive definiteness (eig(P)>0) and $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$ in the sense of negative definiteness ($eig(\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P)<0$).

% using the LMI solve editor of the Matlab LMI toolbox, we define the LMI problem



% the equivalent matlab commands are:

```
setlmis([]);
P=lmivar(1,[5 1]);
lmiterm([-1 1 1 P],1,1);              % LMI #1: P
lmiterm([2 1 1 P],cl5010',cl5010);    % LMI #2: cl5010'*P*cl5010
lmiterm([2 1 1 P],1,-1);              % LMI #2: -P
lmiterm([3 1 1 P],cl5020',cl5020);    % LMI #3: cl5020'*P*cl5020
lmiterm([3 1 1 P],1,-1);              % LMI #3: -P
lmiterm([4 1 1 P],cl6010',cl6010);    % LMI #4: cl6010'*P*cl6010
lmiterm([4 1 1 P],1,-1);              % LMI #4: -P
lmiterm([5 1 1 P],cl6020',cl6020);    % LMI #5: cl6020'*P*cl6020
lmiterm([5 1 1 P],1,-1);              % LMI #5: -P
Test=getlmis;
```

% we evaluate the feasibility of the defined problem

[tmin,Pconstraints]=feasp(Test)

% we are satisfied that there is a matrix P that satisfies the LMI problem

% "Solver for LMI feasibility problems L(x) < R(x)
% This solver minimizes t subject to L(x) < R(x) + t*I
% The best value of t should be negative for feasibility"
% Result: best value of t: -2.421599e-004

% we obtain the specific matrix we need

P=dec2mat(Test,Pconstraints,[1 5])

$$P = 1.0e+004 * \begin{bmatrix} 2.3229543 & 0.3526428 & 3.0679626 & 0.6544445 & -0.0004749 \\ 0.3526428 & 0.1037981 & 0.4604631 & 0.1920243 & -0.0000580 \\ 3.0679626 & 0.4604631 & 4.7176875 & 0.8626853 & -0.0002955 \\ 0.6544445 & 0.1920243 & 0.8626853 & 0.3585137 & -0.0001056 \\ -0.0004749 & -0.0000580 & -0.0002955 & -0.0001056 & 0.0000008 \end{bmatrix}$$

Proof

We have to prove that the obtained matrix fulfils the conditions

$$P>0 \quad \text{and} \quad \forall\, \Phi_{cl_k} \in \Omega,\ \Phi^{T}_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$$

a) P>0

eig(P)= 1.0e+004*0.00071196, 1.0e+004*0.00000062, 1.0e+004*0.16870757,
1.0e+004*0.29115451, 1.0e+004 *7.04237994

Therefore, matrix P is positive definite (all its eigenvalues are positive).

b) Now we check if for all $\Phi_{cl_k} \in \Omega,\ \Phi^{T}_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$

- For $\Phi_{cl_1} = \Phi_{cl}(50,10)$

$$\Phi^{T}_{cl_1} \cdot P \cdot \Phi_{cl_1} - P = 1.0e+003 * \begin{bmatrix} -1.872119 & -0.239306 & -1.015906 & -0.404800 & 0.003028 \\ -0.239306 & -0.199389 & -0.281389 & -0.334920 & 0.000190 \\ -1.015906 & -0.281389 & -1.233114 & -0.459454 & 0.001092 \\ -0.404800 & -0.334920 & -0.459454 & -0.574081 & 0.000385 \\ 0.003028 & 0.000190 & 0.001092 & 0.000385 & 0.000008 \end{bmatrix}$$

eig($\Phi^{T}_{cl_1} \cdot P \cdot \Phi_{cl_1} - P$) = 1.0e+003 *-2.852630, 1.0e+003 *-0.661630,
1.0e+003 *-0.361639, 1.0e+003 *-0.002810,
1.0e+003 *,1.0e+003 *-0.000002

Consequently, $\Phi^{T}_{cl_1} \cdot P \cdot \Phi_{cl_1} - P$ is negative definite (all its eigenvalues are less than zero)

- For $\Phi_{cl_2} = \Phi_{cl}(50,20)$

$$\Phi_{cl_2}^T \cdot P \cdot \Phi_{cl_2} - P = 1.0e+003 * \begin{bmatrix} -2.494002 & -0.137641 & -2.721416 & -0.299674 & 0.000742 \\ -0.137641 & -0.141770 & -0.349731 & -0.250258 & -0.000410 \\ -2.721416 & -0.349731 & -4.358931 & -0.656657 & -0.001134 \\ -0.299674 & -0.250258 & -0.656657 & -0.452806 & 0.000385 \\ 0.000742 & -0.000410 & -0.001134 & 0.000385 & 0.000006 \end{bmatrix}$$

eig($\Phi^T_{cl_2} \cdot P \cdot \Phi_{cl_2} - P$)= 1.0e+003 *-6.412360, 1.0e+003 *-0.683253,1.0e+003 *-0.349660, 1.0e+003 *-0.002240,1.0e+003 *-0.000002

Consequently, $\Phi^T_{cl_2} \cdot P \cdot \Phi_{cl_2} - P$ is negative definite (all its eigenvalues are less than zero)

- For $\Phi_{cl_3} = \Phi_{cl}(60,10)$

$$\Phi_{cl_3}^T \cdot P \cdot \Phi_{cl_3} - P = 1.0e+003 * \begin{bmatrix} -3.744356 & -0.432162 & -3.962794 & -0.770644 & 0.003057 \\ -0.432162 & -0.238891 & -0.644769 & -0.407882 & 0.000212 \\ -3.962794 & -0.644769 & -5.886782 & -1.133297 & 0.001255 \\ -0.770644 & -0.407882 & -1.133297 & -0.710531 & 0.000385 \\ 0.003057 & 0.000212 & 0.001255 & 0.000385 & -0.000008 \end{bmatrix}$$

eig($\Phi^T_{cl_3} \cdot P \cdot \Phi_{cl_3} - P$)= 1.0e+003 *-9.219464, 1.0e+003 *-0.773747, 1.0e+003 *-0.583834, 1.0e+003 *-0.003521, 1.0e+003 *-0.000003,

Consequently, $\Phi^T_{cl_3} \cdot P \cdot \Phi_{cl_3} - P$ is negative definite (all its eigenvalues are less than zero)

- For $\Phi_{cl_4} = \Phi_{cl}(60,20)$

$$\Phi_{cl_4}^T \cdot P \cdot \Phi_{cl_4} - P = 1.0e+003 * \begin{bmatrix} -3.528794 & -0.244081 & -4.534288 & -0.496271 & 0.000851 \\ -0.244081 & -0.176242 & -0.590618 & -0.310879 & -0.000328 \\ -4.534288 & -0.590618 & -7.419128 & -1.090615 & -0.000748 \\ -0.496271 & -0.310879 & -1.090615 & -0.561842 & -0.000483 \\ 0.000851 & -0.000328 & -0.000748 & -0.000483 & -0.000006 \end{bmatrix}$$

eig($\Phi^T_{cl_4} \cdot P \cdot \Phi_{cl_4} - P$)= 1.0e+004*-1.0590080,1.0e+004*-0.0766065,1.0e+004*-0.0327174, 1.0e+004 *-0.0002690, 1.0e+004 *-0.0000004

Consequently, $\Phi^T_{cl_3} \cdot P \cdot \Phi_{cl_3} - P$ is negative definite (all its eigenvalues are less than zero)

Summary

Since we have found a positive definite matrix P that complies with the condition that for all $\Phi_{cl_k} \in \Omega$, $\Phi^T_{cl_k} \cdot P \cdot \Phi_{cl_k} - P < 0$, the system will be stable, whatever closed loop matrix applies at run time.

Appendix C: Closed-loop matrices

A state feedback controller is characterized by its closed loop matrix, which depends on the sampling period h and the time delay τ:

$$\Phi_{cl} = \begin{bmatrix} \Phi(h) & \Gamma_1(h,\tau) \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} \Gamma_0(h,\tau) \\ I \end{bmatrix} \cdot L(h,\tau)$$

where $\Phi(h) = e^{Ah}$, $\Gamma_0(h,\tau) = \int_0^{h-\tau} e^{As} ds B$ and $\Gamma_1(h,\tau) = e^{A(h-\tau)} \int_0^{\tau} e^{As} ds B$ and where L(h, τ) is the state feedback matrix obtained by pole placement observer design (using the Ackerman formula). As can be seen, the closed loop matrix, which characterizes the stability and the performance of the system, depends on h and τ. The equivalent Φ(h), Γ_0(h, τ), Γ_1(h, τ) for the inverted pendulum are:

$$\Phi = \begin{bmatrix} \frac{1}{2}e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{1}{2}e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} & \frac{5}{6867}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{5}{6867}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} & 0 & 0 \\ \frac{3}{200}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{3}{200}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} & \frac{1}{2}e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{1}{2}e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} & 0 & 0 \\ -\frac{1}{84}e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{1}{84}e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{1}{42} & \frac{1}{42}h + \frac{5}{288414}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{5}{288414}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} & 1 & h \\ -\frac{1}{2800}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{1}{2800}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} & -\frac{1}{84}e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{1}{84}e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{1}{42} & 0 & 1 \end{bmatrix}$$

$$\Gamma_0 = \begin{bmatrix} -\frac{500}{20601}e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} - \frac{500}{20601}e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} + \frac{1000}{20601} \\ \frac{5}{6867}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} - \frac{5}{6867}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} \\ -\frac{500}{432621} + \frac{250}{432621}e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} + \frac{250}{432621}e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} - \frac{1}{84}h^2 + \frac{1}{42}h\,\tau - \frac{1}{84}\tau^2 + \frac{1}{4}(h-\tau)^2 \\ -\frac{5}{288414}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} - \frac{10}{21}\tau + \frac{5}{288414}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} + \frac{10}{21}h \end{bmatrix}$$

$$\Gamma_1 = \begin{bmatrix} -\frac{500}{20601}e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{500}{20601}e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} - \frac{500}{20601}e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{500}{20601}e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} \\ -\frac{5}{6867}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{5}{6867}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} + \frac{5}{6867}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{5}{6867}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} \\ \frac{250}{432621}e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{250}{432621}e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{5}{21}\tau^2 + \frac{10}{21}h\,\tau - \frac{250}{432621}e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} - \frac{250}{432621}e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} \\ \frac{5}{288414}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,h\right)} - \frac{5}{288414}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} - \frac{5}{288414}\sqrt{22890}\,e^{\left(-\frac{3}{100}\sqrt{22890}\,h\right)} + \frac{5}{288414}\sqrt{22890}\,e^{\left(\frac{3}{100}\sqrt{22890}\,(-h+\tau)\right)} + \frac{10}{21}\tau \end{bmatrix}$$

and L(h, τ), obtained through the Ackerman formula, takes different values according to each specific h and τ.