KTH Machine Design

Quality of control and real-time scheduling

Allowing for time-variations in computer control systems

MARTIN SANFRIDSON

Doctoral thesis
Department of Machine Design
Royal Institute of Technology
Stockholm, Sweden

TRITA–MMK 2004:7
ISSN 1400-1179
ISRN/KTH/MMK--04/7--SE

TRITA–MMK 2004:7
ISSN 1400-1179
ISRN/KTH/MMK--04/7--SE

Quality of control and real-time scheduling — allowing for time-variations in computer control systems.

Martin Sanfridson

Doctoral thesis

Academic thesis, which with the approval of Kungliga Tekniska Högskolan, will be presented for public review in fulfilment of the requirements for a Doctorate of Engineering in Machine Design. The public review is held at Kungliga Tekniska Högskolan, Valhallavägen 79, Stockholm in Kollegiesalen at 09.00 am on the 4th of June 2004.

Mechatronics Lab
Department of Machine Design
Royal Institute of Technology
S-100 44 Stockholm, Sweden

TRITA-MMK 2004:7
ISSN 1400-1179
ISRN KTH/MMK--04/7--SE

Document type
Doctoral Thesis

Date
2004-05-07

Author(s)
Martin Sanfridson
mis@md.kth.se

Supervisor(s)
Prof. Martin Törngren
Prof. Jan Wikander

Title
Quality of control and real-time scheduling — allowing for time-variations in computer control systems

Sponsor(s)
Vinnova (Nutek)
SSF via Artes

Abstract

The majority of computers around us are embedded in products and dedicated to perform certain tasks. A specific task is the control of a dynamic system. The computers are often interconnected by communication networks forming a distributed system. Vehicles and manufacturing equipment are two types of mechatronic machines which often host dedicated computer control systems. A research problem is how the real-time behaviour of the computer system affects the application, especially the control of the dynamic system.

If the internal or external conditions varies over time, it becomes difficult to assign a fixed resource reservation that will work well in all situations. In general, the more time an application gets of a resource, the better its gauged or perceived quality will be. A strategy is to alter the resource reservation when the condition changes. This can be constructed as a negotiation between competing applications, a method for which the term *quality of control*, QoC, has been coined. Scalability is the ability to change the structure and configuration of a system. It promotes evolving systems and a can help manage a complex product family. An architecture for a QoC middleware on top of a scalable computer system, has been proposed.

As a *quality measure* of a control application, the well-known weighted quadratic loss function used in optimal control, has been revised to encompass a subset of the so called *timing properties*. The timing properties are the periods and the delays in the control loop, including time-varying period and delay. They are the interface between control and computer engineering, from a control engineering viewpoint. The quality measure can be used both off-line and on-line given a model of the sampled-data system and an appropriate description of the timing properties.

In order to use a computer system efficiently and to guarantee its responsiveness, real-time scheduling is a must. In fixed priority scheduling each task arrives periodically and has a fixed priority. A task with a high priority can preempt a low priority task and gain access to the resource. The best-case response time characterizes the delays in the system, which is useful from a control viewpoint. A new algorithm to calculate the *best-case response time* has been derived. It is based on a scheduling scenario which yields a recurrence equation. The model is dual to the well-known worst-case response time analysis.

Besides the dynamic fixed priority scheduling algorithm, optimal control using *static scheduling* has been studied, assuming a limited communication. In the static schedule, which is constructed pre-runtime, each task is assigned a time window within a schedule repeated in eternity. The optimal scheduling sequence is sought by optimizing the overall control performance. An interesting aspect is that the non-specified control period falls out as a result of the *optimal schedule*. The time-varying delay is accounted for in the control design.

Keywords
Real-time scheduling, sampled-data control, performance measure, quality of control, limited communication, time-varying delay, jitter.

Language
English

Preface

Mechatronics is a multi-disciplinary research field. When I started at DAMEK, it was within the Nutek funded project DICOSMOS — a cooperation between DAMEK, Automatic control in Lund and Computer engineering at Chalmers and during the last years of the project also Volvo Technological Development. The acronym was spelled out: distributed control of safety-critical motion systems. This reveals the span of possible research directions that were open to me. I decided to focus on the so called timing properties, which taken together constitute an interface between control and computer engineering, from a control engineering point of view. My first study plan contained the phrase “the controller is not alone in the loop”. This research line is still pursued as quality of control. The work on quality of control is substantially different from the work on the best-case response time, the performance measure or the optimal scheduling. It is a top down approach which never really makes it down to a model of any kind, but to a description and proposal of an architecture.

During the first years, I read a lot of control theory, real-time scheduling and similar subjects. Like many other PhD students I was puzzled by the thought of finding something new, or “unpublished and original” as it is usually expressed in a call for papers. Today, it feels like I can figure out more interesting problems with reasonable technical innovation potential, than there is time to dig into them. The text book and the research papers teach you how to think and how to model a problem. New ideas will come if you stare at the model and equations long enough to start viewing the problem from a different perspective and long enough to start asking the right questions. The great idea will come a gloomy day in October, a rainy day when you don’t feel like riding the bicycle through the city to the office. It will come as a sudden glimpse of clarity the moment you sink down in your sofa.

To borrow terms from the control theory, research is in my opinion observable but not controllable. Hard work and luck are two virtues, and you might very well stumble over something relevant and fruitful. The general research direction should be broad, but the actual work should be restricted and focused. Looking back, it is painful to see how much effort was spent on ideas that were later surrendered. The PhD study is an eduction to take responsibility, find a problem, search data bases, read related literature, dig into the theory, read more, find a better problem and dig deeper. The devil is in the details.

Acknowledgement

First of all, I would like to express my gratitude to my supervisors Professor Martin Törngren and Professor Jan Wikander. Thank you Martin for always being so optimistic and for your golden comments. Thank you Jan for your understanding and supportive guidance through all these years.

The Mechatronics lab — or DAMEK — a division of the department of Machine Design at KTH, is a wonderful place to work and it is full of talented people. It has been fun.

And to my co-authors: Thank you Dr. Ola Redell (DAMEK) and Dr. Henrik Rehbinder (KTH, OptSyst) for good cooperation in the writing of successful papers! I've learned a lot. Spontaneous cooperation deserves much attention.

During my many weeks in Gothenburg for the DICOSMOS case study, I particularly enjoyed working with Dr. Magnus Gäfvert (Automatic control, LTH) and Dr. Vilgot Claesson (Computer engineering, Chalmers), and learned to enjoy their mutual devotion to Thai food lunches and movie nights. I would also like to thank Volvo Technological Development and the group around Dr. Mats Andersson, for their part in the DICOSMOS case study.

I would like to express my gratitude to my fellow colleagues of the ReTiS Lab at Scuola Superiore Sant'Anna, especially Paolo, Luigi and Peppe, for sharing their interests and enthusiasm with me during my stay in Pisa.

And last but not least, to my friends, relatives, parents and sister, who have done nothing to this thesis, but who have endured my many working days — thank you.

Martin Sanfridson

KTH, May 2004

Appended papers

PAPER A

Martin Sanfridson and Ola Redell, “Exact best-case response-time analysis of fixed priority scheduled tasks with arbitrary response times”, submitted to a journal, September 2003.

A paper submitted to a journal, describing in detail how to model and analyse the best-case response time of a fixed priority scheduled uniprocessor with preemption.

PAPER B

Martin Sanfridson, “Discretization of loss function for a control loop having time-varying period and control delay”, Technical report, ISRN KTH/MMK/R--03/2--SE, Mechatronics Lab, KTH, May 2004.

A technical report describing in detail the discretization of a continuous time process and quadratic loss function, in case of time-varying sampling period and control delay.

PAPER C

Martin Sanfridson, Martin Törngren and Jan Wikander, “A quality of control architecture and codesign method“, work in progress conference paper at the Real-Time and Embedded Technology and Applications Symposium, Toronto, May 2004.

An extended version of a conference work in progress paper, describing a framework for on-line adaptation of resources with respect to periods and delays.

PAPER D

Henrik Rehbinder and Martin Sanfridson, “Scheduling of a limited communication channel for optimal control“, Automatica, Vol. 30, No. 3, pp. 491-500, March 2004.

A brief paper published in the control theoretical journal Automatica, describing a method to optimize the static processing schedule of optimal controllers.

Other publications

Chen, D.-J. and Sanfridson M., “Introduction to distributed systems for real-time control“, Technical report, ISRN KTH/MMK/R--98/22--SE, Mechatronics Lab, KTH, November 2000.

Claesson V., Gäfvert M. and Sanfridson M., “Proposal for a distributed computer control system in heavy-duty trucks“, Technical report 00-16, Computer engineering, Chalmers university of technology, 2000.

Gäfvert M., Sanfridson M. and Claesson V., “Truck model for yaw and roll dynamics control“, Technical report, ISRN LUTFD2/TFRT-7588--SE, Automatic control, Lund institute of technology, 2000.

Redell O. and Sanfridson M., “Exact best-case response time analysis of fixed priority scheduled tasks“, Proc. of the 14th Euromicro Workshop on Real-Time Systems, Vienna, 2002.

Rehbinder H. and Sanfridson M., “Integration of off-line scheduling and optimal control“, Proceedings of the 12th European Conference on Real-Time Systems, Euromicro, pp. 137-143, Stockholm, 2000.

Rehbinder H. and Sanfridson M., “Scheduling a limited communication channel for optimal control“, Proceedings of the 39th IEEE Conference of Decision and Control, volume 1, pp. 1001-1016, Sydney, December 2000.

Sanfridson M., “Problem formulation for QoS management in automatic control”, Technical Report, ISRN KTH/MMK/R--00/3--SE, Mechatronics Lab, KTH, March 2000.

Sanfridson M., “Timing problems in distributed real-time computer control systems”, Technical report, ISRN KTH/MMK--00/11--SE, Mechatronics Lab, KTH, May 2000.

Sanfridson M., “Timing problems in distributed control”, Licentiate thesis, ISRN KTH/MMK--00/14--SE, Mechatronics Lab, KTH, May 2000.

Sanfridson M., “An overview of the use of LMI in control to assess robustness and performance”, Technical Report ISRN KTH/MMK--03/08--SE, Mechatronics Lab, KTH, 2003.

Sanfridson M., Claesson V. and Gäfvert, M., “Investigation and requirements of a computer control system in a heavy-duty truck“, Technical report, ISRN KTH/MMK--00/5--SE, Mechatronics Lab, KTH, June 2000.

Törngren M. and Sanfridson M. (ed.), “Research problem formulations in the DICOSMOS project”, Technical report, ISRN KTH/MMK--98/20--SE, Mechatronics Lab, KTH, 1998.

Törngren M., El-khoury J., Sanfridson M., and Redell O., “Modelling and simulation of embedded computer control systems: problem formulation”, Technical report, ISRN KTH/MMK--98/20--SE, Mechatronics Lab, KTH, 2001.

Contents

Wir denken die einzelnen Elemente unseres Gegenstandes,
dann die einzelnen Teile oder Glieder desselben
und zuletzt das Ganze in seinem inneren Zusammenhange zu betrachten,
also vom Einfachen zum Zusammengesetzten fortzuschreiten.
Aber es ist hier mehr als irgendwo nötig,
mit einem Blick auf das Wesen des Ganzen anzufangen,
weil hier mehr als irgendwo
mit dem Teile auch zugleich immer das Ganze
gedacht werden muß.

von Clausewitz, Vom Kriege, 1832

1 Introduction to the thesis

The introduction to the appended papers is organized as follows. After a description of the research problem the aims are stated. The technologies and methods used, together with a brief annotation of delimitations are added to make the picture clearer. Thereafter, a bit of the background in terms of research projects follows. Before the four appended papers are introduced, a brief treatment will introduce codesign of computer control systems, which this thesis very much is about. The introduction is split in three sections: the so called timing properties, the off-line approaches and the on-line approach. This is to give more material to the beginner in this area, but also to help place the appended papers in their proper context. A brief treatment of each appended paper stating its contents, contributions, division of work and related work is given. Some ideas for future research conclude this introduction.

1.1 Problem description

An embedded computer system is physically a set of processing and communication units, which are parts of some machine. The machine is not directly referred to, or thought of, as having a computer system — the embedded computer is a machine element that is dedicated to perform a certain job. The embedded computer system is typical for a mechatronic system, where mechanical and electronic machine elements are integrated.

Vehicles and manufacturing systems are two common examples of mechatronic systems. In many cases, the motions of the mechanical machine is controlled by interconnected microprocessors. The control law, which controls the mechanical system or process, is implemented in software. The control theory is based on the feedback principle: the deviation between a setpoint and a measured value of interest, is used to compute a control law, and the result is then fed back to the physical process via an actuator. Almost from the advent of computers, as computers grew more commercially available, controllers have been implemented in software.

A controller has to react both adequately and timely in relation to its physical environment. The environment is a process which has its own dynamics. This kind of requirement put on a computer system leads to the notion of real-time. The hardware has to sustain a sufficiently high rate of actions, without getting overloaded by the requested utilization, in order for it to respond timely to its environment. A traditional definition of real-time is based on the notion of deadline. In a real-time system, should the processing time of tasks and the transmission time of messages not conform to the deadline specifications, the result could be a system failure. The ability of a computer system to meet deadlines for to the intended rate of actions depends, among other

things, on how fast the computer system is and how well it is programmed. If the timing is predictable, real-time guarantees can be stated.

The timing behaviour of a processing or communication unit can be controlled by scheduling periodically recurring tasks and messages. Problems can often be solved by throwing in more resources or using parts which are fast and state-of-the-art, but such alternatives are usually expensive. The quest for cost-efficiency is an industrial perspective on the research. It is a constraint which calls for an optimization of affordable resources. To achieve cost-efficiency, the use of components-off-the-shelf, such as the hardware, the communication protocols and the operating system, is often proposed instead of a closed proprietary system. The need for speed can to some extent be met by a better real-time scheduling. Although every part of a computer system contributes to the timing behaviour, the scheduling is the most interesting part to study. From a practical point of view, it is desirable that a system designer has a good knowledge of both computer and control engineering. It is vital to understand the consequences of a trade-off in order to make good design decisions. To over-dimension a design without achieving any additional qualities must be considered bad engineering practice.

1.2 Aim of the research

The main objective of the research has been to study the timing aspects of computer control systems from a codesign viewpoint. The stability, robustness and performance of a control loop are affected by the choice of period, the actual delay and any time-variations in both the period and the delay. These timing properties are ultimately the result of design activities such as: the choice of processing and communication hardware, the choice of operating system, the synchronization policy, the scheduling strategy, and the actual execution times of tasks.

The timing properties are useful both at design-time and at run-time. At design-time they help convey vital information between the control and computer engineers. At run-time they can be used as a foundation for optimizing a control related performance measure.

1.3 Research approach

The research described in this thesis is interdisciplinary. The two primary research fields, which are relatively independent, are real-time scheduling theory and feedback control theory. Interdisciplinary combinations are typical for the research in mechatronics. The idea of interdisciplinary is not to unify two fields into one, but to recognize the dependence and combine them both when addressing a problem. However, addressing two disciplines can make the work complex. A traditional approach to master complexity is to break down the system into suitable pieces, analyse the pieces and then study the reassembled system, cf. the quotation on the first page. First, a look at the system as a whole, then each entity apart, and finally all entities should be put together and the entire design reviewed.

An ambition has been to find theoretical models, that describe specific problems and offer some kind of prediction or valuable statement. For example, if the utilization of a fixed priority scheduled uniprocessor is below 0.2, the response time jitter is not likely to be a problem, since the congestion is negligible.

Simulation is a valuable tool when a desired analytical solution is hard to obtain or evaluate. Simulation is also useful for acquiring an understanding of the studied problem and for comparing results obtained by other means. Simulation can be used as a complement to analytical solutions to characterize the average behaviour of a system or a specific scenario, but

it is generally not the right tool when searching for worst-case conditions. As in other modelling approaches, a main difficulty in simulation is to set up a model such that it adequately reflects the phenomenon to be studied. Simulation requires an understanding of the dynamics and in this case also of the timing behaviour. In essence, the problem is to choose the level of details for different parts of the simulation model in order to make good approximations.

Schedule simulation is a pure discrete event simulation, and can be used to evaluate the average performance of a task set, e.g. in terms of latencies. The implementation of the scheduler and dispatcher is preferably done in a high level programming language such as Matlab. The simulation tool Simulink has been used throughout the whole research, since it is well suited for simulation of hybrid systems.

Response time analysis in distributed systems is very complex. Task chains stretch from one node over communication channels into other nodes. It is difficult to obtain the upper and lower limits of the end-to-end response time, which is interesting from control point of view. The end-to-end response time of a controller usually corresponds to the time from the actual sampling to the actual output. Without loss of generality, the end-to-end response time can for convenience be thought of as the response time from the first to the last task in a task chain. It makes sense to start an analysis by studying the response time of a uniprocessor rather than that of a distributed system, since the complexity is lower and the expected impact on the controller is to some extent, via the timing properties, independent of the complexity of the computer system.

1.4 Delimitation of the scope

In order to facilitate an analysis, a useful approximation is to consider linear time invariant (LTI) processes and controllers only. This is a very common delimitation and it is also adopted in this thesis. An LTI can be obtained by linearization of a nonlinear system around an equilibrium point. The analysis of system properties, such as stability, is carried out in the region local to the linearization point. Hybrid systems have mixed continuous and discrete event driven state evolution over time, an evolution which cannot be reduced to either continuous or discrete states only. Non-linear and event-driven systems in combination also belong more to the area of hybrid control. A computer control system is indeed an example of a hybrid system, but by limiting the analysis to LTI systems, the analysis becomes more tractable.

When sampling, anti-aliasing filtering should be applied to the continuous time signal, since frequencies higher than the Nyquist frequency fold and falsely add variance to lower frequencies. However, anti-aliasing filtering is seldom seen in theoretical control papers and is not discussed in this work either. The analog filtering can be treated as dynamics belonging to the process, and the dynamics of the process should be more dominating.

Apart from partitioning and allocation, the implementation details of algorithms have received no attention here. The actual coding with respect to numerical issues must be considered in the codesign. For example, the finite resolution (word length) of a discrete value used to represent a continuous time value, can possess a problem if the sensitivity to quantization effects is high.

A distributed control system typically has control loops with different sampling periods. To start with single rate systems and to extend the study with multirate systems later, is the preferred order. A multi-input-multi-output (MIMO) control system with different delays on every signal, should also be considered for a more general distributed system. A MIMO system is more complex to analyse and can behave quite differently and unexpectedly compared to a single-input-single-output (SISO) system.

1.5 Background, research projects

The DICOSMOS 2 (Distributed Control of Safety Critical Motion Systems) research project was a cooperation between Computer Engineering at Chalmers (CTH), the Mechatronics Lab at the Royal Institute of Technology (KTH), the Department of Automatic Control at Lund Institute of Technology (LTH), and Volvo Technological Development (now called VTEC). The project was funded by VINNOVA (Verket för innovationssystem, the Swedish agency for innovation systems), formerly called NUTEK, and VTEC. The project was a continuation of the first DICOSMOS project, which started in 1993. A case study was a central part of DICOSMOS 2. The aim of the case study was to increase the understanding of a specific safety critical distributed real-time system for control applications: a heavy-duty truck and semi-trailer combination. The case study primarily concerned functions for an electronic braking (ABS) and on top of that, vehicle dynamics control (VDC). The idea was to study and develop interdisciplinary analysis and design methods for this type of system (Törngren et al., 2001).

When the DICOSMOS project ended, the FLEXCON project (Flexible Embedded Control Systems) continued from January 2003. The project is funded by SSF (Stiftelsen för strategisk forskning, the Swedish foundation for strategic research) and runs through 2005. Via ARTES (a national Swedish strategic research initiative in real-time systems) this was a logical transfer. FLEXCON is a cooperation between Automatic control and Computer Science at Lund Institute of Technology, the Mechatronics Lab at KTH, MRTC at Mälardalen University and DRTS at the University of Skövde. The main research direction of the FLEXCON project is the problem of providing flexibility and reliability in embedded control systems implemented with components-off-the-shelf. The key issues are design and implementation techniques that support dynamic run-time flexibility. ARTES/ARTES++ is a network within the real-time research community in Sweden, which among other things provides support for international mobility and arranges an annual summer school. During two months in early 2002 the author had the opportunity to visit the real-time research ReTiS Lab at Scuola Superiore Sant'Anna in Pisa.

2 Preliminaries: the timing properties

A control system is indeed a real-time system. The architecture of a computer control system can be modelled in many different ways, and in this context the timing behaviour is of primary interest. Figure 2.1 shows two views of a computer control system. In the *computer engineering* view to the left, the process P is external to the model of the computer system. In the *control engineering* view to the right hand side of the figure, the dynamics of the computer system are similarly rationalised to allow for a focus on the process P and the dynamics of the closed loop. The input-output (I/O) units constitute the physical interface between the computer and process. At every sampling event, the continuous time signal is sampled by the input unit and after a computation of the control law, the new control signal is held constant, by the output unit, until the next control signal has been calculated. This sample and hold scheme is called zero-order-hold.

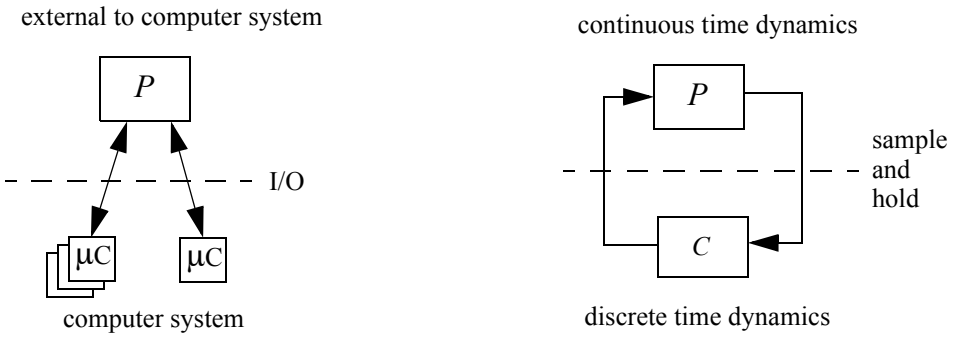


Figure 2.1 Computer engineering view (left) and control engineering view (right). P is the process and C is the controller. µC denotes a processing unit.

The design process is often a team work, but for simplicity of the exposition here the system designer will take on four different roles instead. Each role corresponds to apparent activities in the design process: the control engineer (control analysis and synthesis), the software engineer (architecture design and scheduling), the programmer (implementation) and the service technician (upgrade and repair). The actual design process is always iterative in nature, but this is off the topic.

On the left hand side of Figure 2.2 the architecture of a computer system is sketched. A temporal characterization includes for example the synchronization of tasks and messages, and the scheduling of nodes and communication channels. The temporal characterization is expressed in terms of period, execution time, deadline, response time, transmission delay, target token rotation time, synchronization offset, etc. To the right in Figure 2.2 the control system is represented by a process P and a controller C. The closed loop performance, robustness and stability depend among other things on the timing behaviour of the computer system.

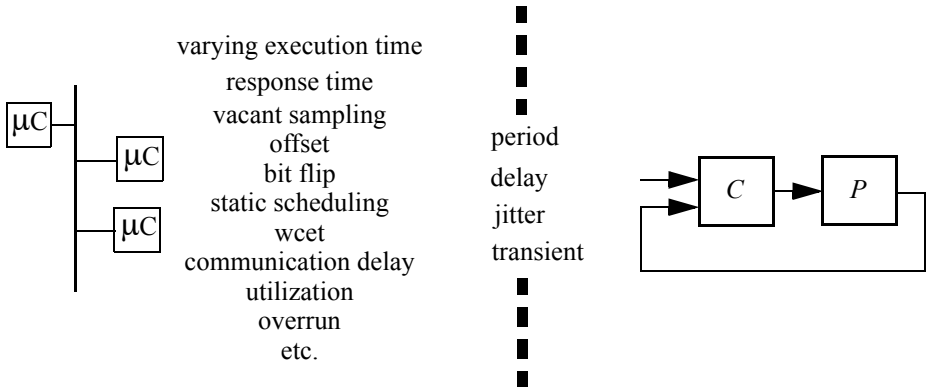


Figure 2.2 The *timing properties* constitute an interface between a model of the timing behaviour of computer control systems (left) and a control theoretical model of the sampled-data system (right).

The timing behaviour of the computer system is described by the so called *timing properties*. This is a temporal characterization of a computer system from a control point of view, see Sanfridson (2000a). These properties must be communicated between the control and software

engineers. The timing properties are:

- *The choice of period*. The periodicity of events in the computer system has a clear correspondence to the theory of discrete control systems. The choice of nominal sampling period is an activity common both for the control design and the implementation with respect to scheduling. The choice of period that the control engineer makes, is often based on some rule-of-thumb regarding the dynamics of the closed loop system. In the traditional view of sampled-data control design, the sampling is uniform, i.e. the sampling period is constant.

- *Constant delay.* A general time-varying delay can be modelled by a constant part plus a time-varying part. A constant delay in an LTI system can conveniently be analysed by familiar control engineering tools. The general rule is that a delay in the closed loop should be as small as possible, because of the decrease in phase margin with increasing delay. The time delay can be inherent in the process, but it can also emerge from the computer system, and thus be called *control delay*. For some control algorithms it is quite easy to compensate for a nominal delay, but allowing for an additional delay brings a cost in terms of control performance.

- *Jitter in delay and period*. Jitter is here defined as the time-variation compared to a nominal constant value. There is a difference between delay jitter and period jitter, since a (continuous) signal (or process) is sampled. When sampling jitter is present, the acquired value depends on the sampling instant. Control delay jitter, on the other hand, causes a delay of the control output to the continuous process. Delay jitter and sampling jitter can of course be present simultaneously. The jitter does not become unintentional because the designer is unable to reduce it. A control designer should of course neglect jitter that is small enough to be of no importance.

- *Transient error.* A transient error is an infrequent change in the normal processing or communication, which causes a long interrupt or delay. The transient error can be the result of a previous fault in the computer system, an example is vacant sampling (or vacant actuation) because of jitter. Transient errors are typically devastating to a system without forgiving dynamics. The transient error can also be deliberate, introduced by skipping tasks or messages to clear the processing unit or communication channel from a temporary overload. A computer system is prone to malfunctions which permanently or temporarily can suspend the normal periodic execution of a control algorithm. A mishap can for example be caused by a memory bit flip, an omitted message or an overrun due to overload. These situations typically occur very seldom, compared to the dynamics of the closed loop, hence the term transient (timing) error. There are many ways to counteract transient errors by redundancy in space and time. An implication of the infrequent occurrence is that a steady state behaviour is less interesting to analyse, compared to the short term effects of a transient error.

The timing properties should not be regarded as problems which need to be solved, but rather as attributes which are important to understand. The period is often chosen deliberately, but the delay and jitter are traditionally something that results from several design choices. Jitter is practically always present in a computer system, but it is often small enough to be neglected. Minor jitter could for example stem from hardware devices such as A/D samplers, pipelines, caches and clock circuits. Larger time-variations, arising from the synchronization and scheduling of tasks and messages are likely to dominate, and they are of course more interesting. Severe jitter might arise when the utilization of a processing unit or communication channel is

high, but this also depends on the scheduling strategy.

Knowing the cause and effect of the timing properties is useful at *design time* to make improvements and trade-offs, and can help the system designer and control engineer to choose a computer architecture or a scheduling strategy from a control point of view. It is obviously relevant to estimate the impact of the timing properties in order to evaluate alternative choices. For example, the most prominent trade-off is to choose sampling period, since it affects both the behaviour of the control loop and the actual utilization of the computer. In design, there is always a multitude of aspects to consider and the aim to optimize a solution is a central theme in any design work.

There is also a *run-time* use of the timing properties, in terms of dynamic scheduling and on-line optimization. The idea is to design an architecture that incorporates some knowledge of the control performance with respect to the timing properties. A dynamic (on-line) approach is more flexible, since decisions are partly made at run-time instead of at design-time. One fundamental difference between run-time and design-time is the availability of information regarding details of the timing behaviour. Execution times, end-to-end latencies and other computer related timing measures can be difficult to estimate pre-runtime. At the best, this information becomes available at a late stage and possibly also becomes more accurate when more design decisions have been settled. This on-line approach is here called quality of control, QoC.

3 Off-line: codesign of computer control systems

This section about codesign of computer control systems at *design time* and the next section devoted to *run-time* issues, serve as an introduction putting the appended papers in their context.

3.1 Partitioning, allocation and scheduling

The goal is to find an architecture for the intended logical and temporal functionalities of the dedicated computer system. Prior to a scheduling of tasks and messages, the software engineer has to both determine the task set, and to do this for each processing, communication or other unit of the distributed system to be scheduled. The first of these activities is called partitioning the latter is called allocation. The partitioning and allocation depend very much on the application, the hardware at hand and on the logical functionality.

In the *partitioning* part of the design process, the software is split into a number of tasks. Their logical and temporal relationship (precedence and exclusion constraints) can be described by constructing a so called task graph (also called directed graph). The partitioning of a control task can be made following a suggestion in Åström and Wittenmark (1997), in order to minimize the latency from sampling to actuation. The idea is to partition updates of the controller's states in a separate task, which can be executed after the actuation instant but before the next sampling. Thus, the partitioning decisions will affect the timing properties.

The *allocation* is of course intimately associated with the topology of the computer system and the task graph. For example, in a distributed control system, a sensor reading and ensuing signal processing can be allocated to a dedicated sensor node, the controller and the actuator can be allocated to other nodes, and their messages allocated to a common bus. Clearly, the allocation decision will affect the timing properties.

The next step — the *scheduling* — can be studied separately from the partitioning and

allocation, if given a set of tasks, a task graph and the allocation structure. An exception is utilization balancing strategies. The scheduling is less application specific than the partitioning and allocation, which makes real-time scheduling stand out as a discipline of its own. Because of different physical prerequisites, communication channels, uniprocessors, multiprocessors and distributed systems require different real-time scheduling strategies. For example, CSMA and Ethernet are fundamentally different and the scheduling strategy should be chosen accordingly. Clearly, the scheduling activities will also affect the timing properties.

3.2 The purpose of a performance measure

Assume that all information concerning partitioning, allocation and scheduling is available to the software engineer. The software engineer is then, to some degree, able to figure out the timing properties, e.g. the end-to-end response time of a control loop's task chain. In order to evaluate and compare solutions, a measure of the relative goodness is needed. This ultimately links the role of the software engineer together with the role of the control engineer. A traditional delimitation imposed to avoid the potential complexity of this relation, is to state that (a) there is no delay jitter, (b) there is no sampling jitter and (c) transients errors do not occur. What remains is the choice of sampling period. With a performance measure, these constraints can be relaxed e.g. by specifying limits of allowable delays and periods, (Törngren, 1995 and Redell et al., 2004).

There are many ways to construct a performance measure. When tuning a controller, the control engineer decides how it will react to disturbances and reference changes. Usually, there is not only one tuning method, or one set of parameters, but many, that will render a good result. The reason is that control performance is to some extent subjective. The situation is similar for the synthesis when selecting the structure of the controller. One of the simplest tuning methods is to look at overshoot and rise time. However simple and effective this might be, it is less applicable to a more automated procedure. Instead, it is natural to apply statistical measures to assess the control performance. A synthesis method frequently encountered in the control research community is optimal control. This method gives the control parameters that will minimize the variance according to a specified objective function, also called loss function. A traditional choice is the integrated square of some signal of the control loop. This measure of energy contents (or variance) is the foundation of a major design theme besides pole placement. The common loss function has the following structure:

$$\bar{J} = E \lim_{T \to \infty} \frac{1}{T} \int_0^T \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix}^T \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix} \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix} dt \qquad (1)$$

The scalar loss $\bar{J}$ depends on the weight matrices $\bar{Q}_{11}$, $\bar{Q}_{12}$ and $\bar{Q}_{22}$, which are the parameters of the loss function defining what is optimal. Note that the symbol T is used both to denote a time period and the matrix transpose, but there should be no risk of confusion. The signals punished are the state vector of the process $x_p(t)$ and control signal vector $u(t)$, coming from the controller to the process. The objective function (1) is often seen, but a loss function can also have a completely different structure and include the weighted variance of other signals. As a simple example, consider the variance of the process output $y(t) = C_p x_p(t)$.

A measure of type (1) is applicable even when the system varies with time because of jitter. However, it is appropriate only to measure the average behaviour. When it comes to infrequent transient errors, it can be an ambiguous measure since a transient effect also can cause a

violation of the allowable state space. This error condition is not necessarily detectable by a measure based on the variance. In robust control, the worst-case gain is typically used instead, or as a complement.

A discrete loss function corresponding to (1) can be written

$$J = E\frac{1}{N}\sum_{k=0}^{N-1}\begin{bmatrix} x_p(t_k) \\ u(t_k) \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix} \begin{bmatrix} x_p(t_k) \\ u(t_k) \end{bmatrix}$$

In case of uniform sampling, it is enough to consider only one sampling period and the average over N periods is superfluous.

3.3 Making trade-off decisions

Here follows a few examples of situations where it can be valuable to make a trade-off in the codesign.

3.3.1 Choice of period vs. utilization

In case the capacity of a processing or communication resource is limited, the choice of period becomes vital. Typically, the performance increases with a decreasing period, but the marginal cost in terms of utilization increases fast for short periods. This trade-off can be cast as an optimization problem if two or more fundamentally similar applications use the same resource.

3.3.2 Delay jitter vs. delay to cancel the jitter

From a control point of view, the performance, robustness and stability are adversely affected by an increased delay in the control loop. At the same time, the control delay jitter is usually also harmful. When implementing the control law, the software engineer (or programmer) should not introduce unnecessary delay. However, an offset between the start of a periodic producer and the start of a periodic consumer can be used for synchronization. The offset should be longer than the worst possible finish time of the producer to avoid a vacant sampling. The drawback is that the offset represents a delay in the task chain and in the loop. The consumer becomes time triggered (TT) by this offset, which can simplify the scheduling algorithm. The jitter is effectively cancelled by this type of buffering. It is clearly of interest to investigate the impact of delay jitter to compare with the cost of an additional offset. A question is how this offset should be selected in an optimal way.

3.3.3 The choice of priority

The actuation should as a rule of thumb follow as soon as possible after a sampling instant. This calls for an event triggered (ET) synchronization, rather than inserting an offset (skew) between the producer and consumer to substitute the precedence and exclusion constraints by a temporal constraint. Consider the previous Section 3.3.2 with an ET consumer. In fixed priority scheduling, a low priority task is prone to jitter because of interference with high priority tasks. Selecting the highest priority will eliminate the problem (but it does not eliminate any blocking). This greedy solution does not work if other tasks need high priority too. Hence, there is a trade-off to be made.

3.3.4 Skipping a task in case of temporary overload

Again, consider the previous Section 3.3.2. The actual delay jitter can be modelled as a stochastic process. Typically, the average delay is considerably smaller than the worst delay — the probability distribution has a long tail — which makes the offset (or deadline) conservative. The violation of the deadline is a sign of temporary overload and a traditional solution is to skip the task. If skips are allowed, the offset can be decreased. Applying a strategy with skips typically decreases control performance and shorter delay increases it, hence there is a trade-off situation.

3.4 Jitter in a static schedule

The trade-off between the timing properties is also relevant for static scheduling. Static scheduling is above all an off-line scheduling method. It is applicable to the scheduling of uniprocessors and communication channels, i.e. to distributed systems. For a broad range of embedded real-time systems, it makes sense to assume that the running software will remain unmodified for long periods of time. The scheduling strategy is quite simple: every task has pre-allocated time slots in a schedule, which is repeated cyclically in eternity. The pattern is constructed pre-runtime and off-line where there is plenty of time to find a feasible or optimal execution pattern. The static schedule can for example be elaborated to encompass mode changes by switching schedules on-line or it can be combined with a dynamic scheduling strategy, to fill non-allocated slots in the, so to speak, background of the primary tasks.

A limited communication channel can be scheduled statically. Many communication protocols are based on a non-preemptive transmission of a predetermined amount of data. In the absence of contention, this gives a possibility to split the transmission into time slots. If a set of periodic messages is to be scheduled, an optimization objective based on for example the latency and the period, can be constructed. Notice that the number of time slots between any two transmissions of the same message do not have to be uniform; jitter can be deliberately built into the schedule. From a control perspective, it is possible to take the known jitter into account in the control synthesis.

In Figure 3.1 an example task set with two messages τ_1 and τ_2 is scheduled. τ_1 is given every second slot to transmit a message, and for τ_2 two alternatives can be compared. τ_2^1 shows an allocation according to a so called harmonic rate pattern. This limits the choice of period, but there will be no jitter. On the other hand, if jitter is tolerable the period can be chosen freely, e.g. as shown by the second alternative τ_2^2. The down pointing arrows denote arrivals, assuming input at the beginning and output at the end of a slot.

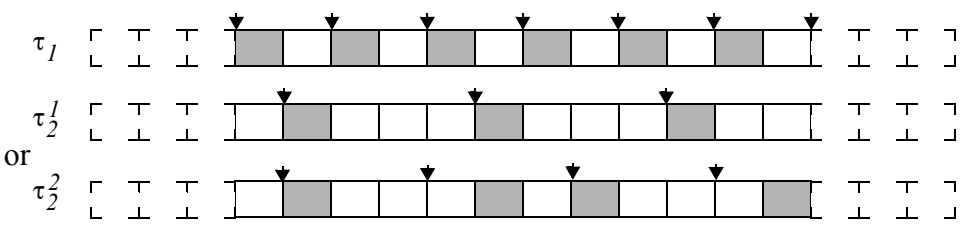


Figure 3.1 A static schedule of two tasks. Top: Every second (2nd) slot is allocated for task τ_1. Middle: Every fourth slot is allocated for τ_2. Bottom: As an alternative to τ_2^1, every third slot is intended to be allocated for τ_2.

The deterministic static schedule brings predictability and with that, a number of advantages. The absence of unknown jitter makes testing easier, since the number of combinations of possible states of the system is kept low compared to a less predictable dynamic scheduling strategy. By the same reasoning, the error detection coverage is likely to be higher and easier to construct. Further, independent tasks are also temporally isolated from each other, such that a misbehaving task cannot affect the timing of the other tasks.

An idea is to search for an optimal or at least a feasible schedule. In the context of this thesis, an objective function is naturally based on control performance, but it could also be a first-hand measure such as the end-to-end delay. There is nearly an infinite amount of time available pre-runtime, to tune the synchronization of tasks in order to meet the objective. Roughly speaking, this means that — given a fixed task set — a dynamic scheduling strategy cannot make a better job than a static scheduling. The optimization problem is in general intractable.

The greatest advantage of the static scheduling strategy is also its major weakness. For a pure static schedule the drawbacks are primarily the inflexibility, but also the potential under-utilization due to periodic scheduling of non-periodic tasks. In this respect, a dynamic scheduling strategy can do better.

3.5 The response time of a fixed priority scheduled task

Fixed priority scheduling (FPS) is above all an on-line strategy, which is dynamic in the sense that tasks are scheduled at runtime. It means that a set of running tasks can be changed on-line. FPS is applicable to the scheduling of uniprocessors and communication channels, i.e. to distributed systems. FPS has been studied for a long time in the real-time scheduling research community. Just to mention, another well-known dynamic scheduling strategy is the earliest deadline first, EDF. There are many more scheduling strategies other than FPS and EDF, see e.g. Krishna and Shin (1997).

During the execution of a task, higher priority tasks can arrive and gain access to (i.e. preempt) the shared resource, making a context switch. This suspends the execution of the preempted task and thus postpones the time instant it will finish. The *response time*, i.e. the time interval between the start and finish of an instance of a task, will in general vary from one instance to the next. Under some general assumptions, it is possible to calculate both a lower and an upper bound of the response time. These bounds, the *best-case response time* and the *worst-case response time*, are both useful when characterizing varying delays when FPS is used to close a control loop. In addition to this, the *expected response time* is valuable. This is in contrast to traditional analysis of FPS systems, where only the worst-case response time is of interest, where a (hard) deadline typically is the sole requirement specification of real-time behaviour.

The scheduling starts by defining the periodic task set. Each task in the set has in its simplest case the following parameters: period, worst-case execution time (WCET) and priority. The period is chosen according to the needs of the application. The execution time depends on both the hardware and the software, and the actual execution time typically varies from one instance of the task to the next. An estimate of the WCET is needed for the worst-case response time analysis. The problem of finding a WCET is a research subject on its own. Suppose that the execution time is the same for each instance of a task. If we take one task instance and delay its current arrival (and the arrival of all its future instances) a short time, the phasing relative to the other tasks changes. This will change the resulting scheduling pattern. As a matter of fact, the act of scheduling a FPS task set is to a large extent deciding upon the relative phasing (or offsets) between tasks. When the phasing is undecided, the task set can be called “unscheduled”. To

achieve hard real-time guarantees, a schedulability test must be performed pre-runtime, i.e. before the task set starts to run. This is often done off-line, since the task set might need to be adjusted manually.

In control, we are not only interested in the extreme values that the response time can take. Transient timing errors should of course be avoided, but the performance of a control loop also depends on the actual behaviour of the delay. The average behaviour over a long time period decides the average control performance. There is a need to analyse the scheduling both from worst-case design and performance optimization viewpoints. Embedded control systems are often distributed. Further, tasks depend logically on each other. These factors complicate the analysis and provide challenges for future work.

It is possible to take on a stochastic view to estimate the number of preemptions made by higher priority tasks. The stochastic approach presented here is not the right way to derive the response time analysis formally, but it can provide some insight. Tasks having a priority lower than the studied will not interfere since they are immediately preempted. Let the time-varying R_i be the response time of the studied low priority task with priority i, and view R_i as a stochastic process. A higher priority task, with priority j, period T_j and constant execution time C_j, will interfere and preempt the low priority task R_i / T_j times on the average. The high priority task is thus expected to execute $(R_i / T_j)C_j$ time units during the interval R_i. The studied task executes C_i time units during R_i. When repeated for all higher priority tasks $hp(i)$ this yields,

$$R_i = C_i + \sum_{\forall hp(i)} \frac{R_i}{T_j} C_j \tag{2}$$

which can be interpreted as the expected-case response time (ECRT). It is tempting to view also the execution time as a stochastic process.

The calculation of the *worst-case response time* (WCRT) of a fixed priority scheduled task is a well-known problem and the simplest case with independent tasks on a uniprocessor has been extended in many directions. The worst-case response time analysis is based on a model called the worst-case phasing scenario, where all higher priority tasks arrive and are released simultaneously, at the so called critical instant of the studied low priority task. Since the phasing of every single task relative to another task is determined in this way, the worst-case response time can be calculated in a straight forward manner by a recurrence equation. A *worst-case scenario* is formed when all higher priority tasks $hp(i)$ arrive and are released simultaneously with the studied task i, at the so called *critical instant*. The WCRT is given by

$$R_i^w = C_i + \sum_{\forall hp(i)} \left\lceil \frac{R_i^w}{T_j} \right\rceil C_j \tag{3}$$

where $\lceil \ \rceil$ denotes the ceiling operator. The equation has to be solved iteratively.

The *best-case response time* (BCRT) analysis is dual to the worst-case analysis. A model of the phasing of tasks called the *best-case phasing scenario*, renders the best-case response time. This fact can be used in the control design. Similar to the previous recurrence equation, the best-case response time can be written,

$$R_i^b = C_i + \sum_{\forall hp(i)} \left\lfloor \frac{R_i^b}{T_j} \right\rfloor C_j \quad (4)$$

where $\lfloor \ \rfloor$ denotes the floor operator. (4) is solved in pretty much the same way as (3). In order to estimate the BCRT, the best-case execution time (BCET) for every task is needed.

The worst-case response time is important in real-time scheduling, because it can be used to check that deadlines are met. The best-case response time is less useful in this respect, but it does give some further information about the delays in the system.

4 On-line: negotiating for resources

This section is a general introduction to quality of control. The motivation of this design architecture starts off-line. An increased number of functionalities in an embedded distributed system threatens to lead to an increased number of nodes. For example, today's mid-range car has several tens of nodes and the number is seemingly increasing for every new model. Such a trend is not sustainable in the long run although many functionalities need dedicated hardware. Today, flexibility in the automotive industry, is apparently upheld by using communication protocols, such as CAN. The protocols hide the implementation of a node, which facilitates modular development of subsystems typically undertaken by subsuppliers. The network acts as an isolator, an indisputable hardware interface, to diverse working groups each with its own node (Electrical Control Unit), programming language and real-time operating system. A node cannot always be equal to a functional module. At some point, cooperating development teams have to abandon the well defined interface typical of a communication protocol. Another set of firm rules, providing isolation between functionalities, is needed.

4.1 The early design and availability of information

The scheduling and response time analysis are typically done at a late design stage. In this context, late means that other activities have to be undertaken first. The timing behaviour depends on many things, such as the choice of micro controller, the communication hardware, the protocols and the operating system. It also depends on the actual task set to be scheduled, the scheduling strategy, the synchronization, the estimation of the execution times, etc. The lack of temporal isolation is annoying, since a even a minor change of one parameter could propagate and affect the timing behaviour of the whole system. It stands clear that an accurate analysis of the response time must necessarily be made at a late stage in the design, when the input to the analysis is more accurate and the design is no longer subject to changes.

The fact that real-time behaviour is hard to predict is a drawback during the whole design, since early design decisions generally are more important than late ones. Decisions taken early are important since they shape the future work, diminishing the solution space. Design decisions are often taken despite a lack of information. This is a well-known problem for every design methodology, and can be illustrated as in Figure 4.1.

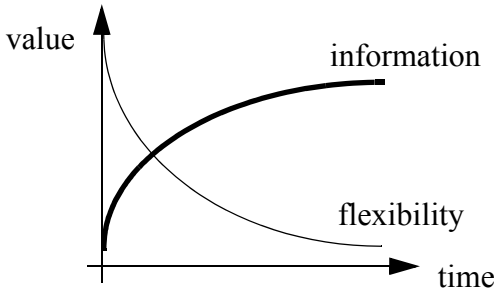


Figure 4.1 Availability of information versus the flexibility to cancel and change decisions made early in a project.

The flexibility to change a decision decreases as time progresses, due to e.g. cost, available man hours and the business aims of a short time-to-market process. To overcome this problem, the available information must be used extensively to predict the outcome of a decision at an early stage. An idea is to compress the development time and collect more information, e.g. by rapid prototyping. From a control engineer's viewpoint, an early estimation of the timing properties in terms of average performance, but also the extreme situations, would be useful.

4.2 Flexibility and modularity

Flexibility is a highly desirable property of an embedded computer control system. Flexibility is often taken for granted in local area networks where bandwidth is more important than guaranteed real-time behaviour. One aspect of flexibility is scalability, e.g. the ability to add and remove hardware and software units without having to redesign the system from scratch.

The flexibility is in many respects more to the customer's or the service technician's advantage rather than to the control or computer engineer's. The modularity concept can be applied to cater for a customer oriented assembly from a family of modules, or it can be useful for upgrades and replacements of parts during the whole life time of a delivered product. A benefit of a modular design is the simplified assembly of customized products, which can reduce the lead time.

There is an initial cost in design effort associated with constructing a modular system. The man hours devoted to redesign and service can be compared against the production cost in terms of processing elements and e.g. memory. As a comparison, consider the use of a high level language or operating system. They both seemingly add to the production cost since more memory and faster processing units are needed compared to assembler on a bare bone system. However, they are highly accepted since they help free the programmer from the complexity of mixing low level implementation details with application specific problems.

4.3 Optimizing the control performance

For many scheduling policies and general task sets, the *expected latency* is considerably shorter than the *worst-case latency*. The expected performance of a control application, i.e. the average performance over an infinitely long time horizon, compared to the dominant dynamics of the closed loop, depends on the expected period, latency and their jitter — and not on the worst-case period or latency (unless these cause instability). However, control applications are often safety-critical and designed to work within a certain state space, and violating specified operating limits is equivalent to system failure. This is the familiar engineering problem of cost efficiency

versus worst-case design. Despite the ability of sustaining a certain amount of timing errors due to the internal memory of a dynamic system, a timing error such as a transient or intermittent vacant sampling/actuation, can be fatal if the allowable state space cannot be upheld. This constraint on the objective of optimizing control performance must be guaranteed. It is typically manifested by the use of hard deadlines. A rational strategy is to minimize the average end-to-end latency or the task period to improve control performance, while still meeting task deadlines. For the same reason, it is necessary to know every task and message in the system, including their worst-case execution times, such that the system is analysable. The worst-case response time needs to be predicted to avoid a deadline miss and the average latency needs to be predicted for the optimization. Unpredictable overload situations cannot be accepted since this could, via excessive response times, lead to an uncontrollable behaviour in the short term.

4.4 Quality of control negotiation

A research hypothesis is that it is advantageous if some work traditionally done at design time could be transferred to take place when the system is running. The abstraction of a quality, as perceived by an application, plays a central role here. Since control applications are the primary design objective, the approach is hereby called Quality of Control, QoC. As the name suggests, ideas have been adopted from the notion of Quality of Service, primarily found in the research areas of multimedia and tele communication. QoC is defined as a method of balancing the demands, primarily of control applications, based on an estimated quality, in order to find a feasible or an optimal use of the available (scarce) hardware resources. The supervisory QoC negotiation is cascaded on top of the applications to negotiate above all period and latency with respect to a measure of performance. The negotiation can be regarded as a feedback loop closed around the controller and its process. The term negotiation is used, rather than optimization, to stress the relative independence of the involved applications.

The benefit of giving one application “more hardware resources” compared to other applications, should be balanced to the needs of all applications. A control application is intrinsically a real-time system, but it is not the only class of application in an embedded system. It is necessary to handle other classes of applications typical for the studied systems, e.g. alarm events, logging and operator interface. An embedded system is rarely altered, i.e. new applications seldom arrive and running ones are seldom removed, compared to the rate of the periodic applications. Frequent changes to a system, with an execution overhead of supervisor logic and optimization, is a minor issue here.

Via the quality measure, an application is to some degree decoupled from the issues of availability and capacity by the ability to accommodate to changes in the processing resources, and in the environment external to the computer system. The decoupling can help facilitate redesign and upgrade during the life time of the product or improve modularity of a product family. The adaptive scheduling procedure is functionally and temporally separated from the applications, and is designed as a negotiation to optimize overall satisfaction.

The negotiation strategy will depend on the scheduling and synchronization policy and on the operating system, but also on the ideas of how the perceived quality of two different applications should be mixed. Two vital parts of the negotiation are admission control and overload handling. The negotiation is a *long term* quality optimization. However, the timing properties being of a *short time* scale, i.e. at the scheduling level, must also be satisfied. In a multi-layer system, layers can have different objectives but they contribute to the same goal. The negotiation is associated with flexibility, and the scheduling with predictability.

5 Summary of the appended papers

The thesis is divided into four papers, Paper A to Paper D, following this introduction. The organization of the dissertation and the main directions of the research are illustrated by the road map in Figure 5.1.

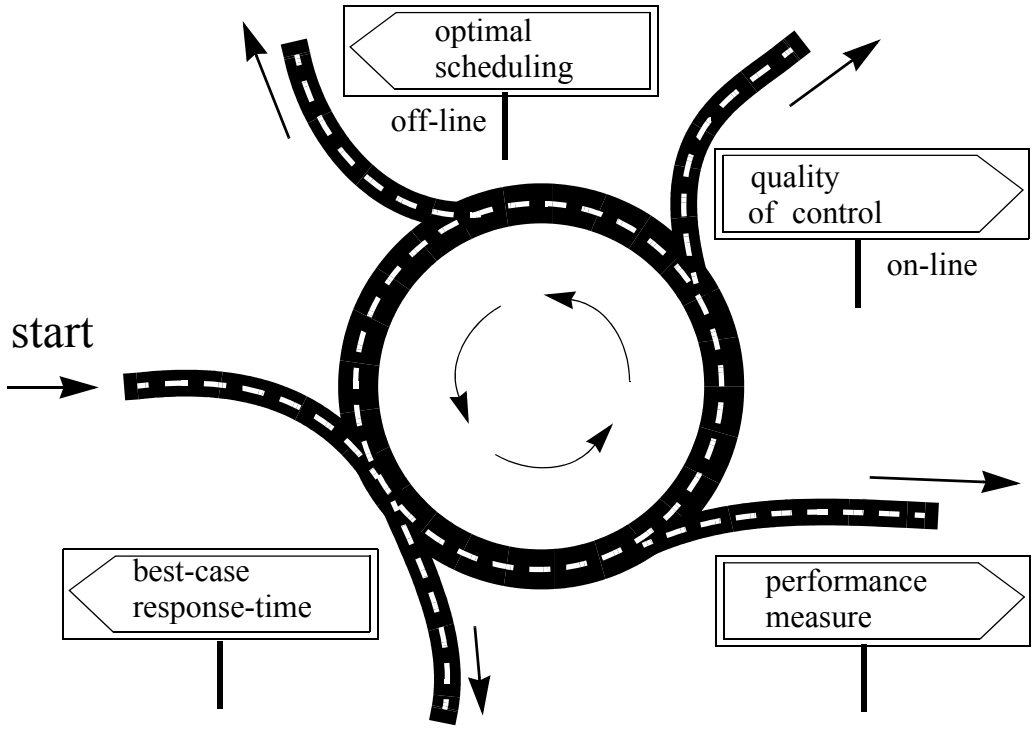


Figure 5.1 Road map to the thesis.

In this section each of the appended papers will be introduced: its contents are outlined, the contributions described and related work done by the author is presented. The division of work between the author and the other contributors is described as well.

5.1 Paper A: The best-case response time

5.1.1 Outline of the paper

This paper has the title “Exact best-case response time analysis of fixed priority scheduled tasks with arbitrary response times” and has been submitted to a journal. The proposed best-case response time (BCRT) calculation is used to obtain a lowest bound on the possible response times of a task within a periodic fixed priority scheduled (FPS) task set. This is useful for synchronization issues, admission control and control design. The response time jitter can be defined as the difference between the worst- and the best-case. It is of interest when it comes to a control theoretical analysis of the end-to-end latencies in a computer control system. The best-case response time has been very little studied, since the application of it is minor compared to the worst-case response time. The analysis and derivation of a recurrence equation to calculate the best-case response time, is based on the here coined *best-case phasing scenario*. The approach is dual to that of finding the worst-case response time using the worst-case scenario,

a well-known piece of theory in the real-time community, extended and refined by many researchers. This BCRT theory now covers the case when released instances of the studied task can interfere with each other within a *busy period*, which is possible if the worst-case response time (WCRT) of a task is greater than its period.

5.1.2 Contributions

The analysis method is by the authors' best knowledge an original piece of theory. The dualism to the worst-case analysis extends the theory of FPS in a beautiful way. The exact best-case response time analysis was previously based on crude and costly searching, and not on a best-case phasing scenario. The idea that the best-case response time could be calculated in pretty much the same way as the familiar worst-case response time, came to the author when working with the problem of stochastic analysis of response times in distributed systems for the DICOSMOS case study. A fresh stochastic view of the traditionally deterministic response time analysis combined with thinking in terms of schedules led to the unexpected finding. Ola Redell immediately grasped the idea and together we quickly formulated a proof and wrote a conference paper. During a two months stay in Pisa, the author had time to review the case when instances of the studied task can interfere with each other — a problem previously set aside. Again, together with Ola the findings were formalised into firm theory.

5.1.3 Related work by the author

The paper is an extension of the earlier conference paper by Redell and Sanfridson (2002). The theory was first presented at the DICOSMOS project's final meeting in December 2001. Theory from the unpublished manuscript of the current paper is also found in Ola Redell's dissertation (Redell, 2003).

5.2 Paper B: A control performance measure

5.2.1 Outline of the paper

The paper has the title "Discretization of loss function for a control loop having time-varying period and control delay". The timing properties is a proposed model of the temporal interface between control and computer engineering. These are the timing matters that control and system engineers need to consider at design time, and the information that is needed to supervise performance at runtime in a QoC architecture. The paper is an attempt to quantify the impact of the timing properties. A continuous time loss function is discretized together with a continuous time process and combined with a discrete time controller. The report contains a very comprehensive theoretical derivation of the measure, including for example the necessary background theory of jump linear systems. It also contains examples of how severe the influence of different timing properties could be, as well as typical and interesting situations where the measure can be applied.

5.2.2 Contributions

A contribution made in the paper is the discretization of a continuous time process for the case of multiple arrivals of a control signal between two sampling instants. A continuous time loss function is also discretized for this case. The discretization method proposed is a natural extension of the traditional text book version, in which a fractional constant delay splits the constant sampling period into two. The jump linear system model is a vital part of the

discretization. The proposed theory enables an analysis of different timing properties simultaneously, including long delays, in a sampled-data system. It also enables the considered jump linear system to be cast as a linear matrix inequality (LMI). The extended discretization of the process with multiple arrivals, combined with a stochastic modelling of delays and periods, is an original approach and the author is not aware of any identical work. The discretization of two loss functions, when there is (time-varying) delay, is also an original work.

5.2.3 Related work by the author

The current paper is the result of a continuing work towards a performance measure, a work started by a paper on QoC that can be found in the licentiate thesis (Sanfridson, 2000a). In that work, a loss function was used as a performance measure in the negotiation of periods and priorities. The current report was also preceded by a self-study course (Sanfridson, 2003b). A brief version of the rather lengthy paper has been submitted to a control conference.

5.3 Paper C: A Quality of Control architecture

5.3.1 Outline of the paper

The paper has the title “A quality of control architecture and codesign method“. Quality of control can be described as a supervisory on-line negotiation. In an embedded system, there are many applications other than controllers. Also, the kind of information available, its freshness and accuracy are typically better at run-time compared to design time, and this fact should be exploited. A dynamic scheduling strategy where it is possible to alter e.g. periods, offsets and priorities, is necessary. The flexibility resulting from a late but dynamic coupling of modules, is primarily intended for the customer, to provide e.g. management of modules within a product family, easier maintenance and upgrade, but also graceful degradation in case of subsystem failure. The proposed architecture aims to organize the work of the control engineer, system engineer and programmer. It rests on a simple hierarchical structure for the applications based on elementary functions and specifications of the applications. The time scale of the negotiation, admission control and optimization routine, is considerably longer than that of the applications, scheduling and overload handling.

5.3.2 Contributions

The analysis work on QoC is more based on conceptions and descriptions on “higher level” than on mathematical models, if compared to the other papers in this dissertation. The contribution is primarily the view of modularity and flexibility in embedded computer control systems, with the need to design an architecture that supports applications evolving during their whole life time. Further, it is necessary to separate the long time-scale optimization for quality from the short time-scale scheduling for deadlines. It is still necessary to make allowance for traditional hard deadline real-time guarantees. In the present work, the co-authors helped refining and expressing the ideas of scalability and quality of control.

5.3.3 Related work by the author

An architecture for QoS in control applications, QoC, has been developed gradually. The first papers are the problem formulation (Sanfridson, 2000b) and a paper found in the author’s licentiate thesis (Sanfridson, 2000a). In the latter paper a CAN bus was shared with three similar control tasks. The bus was simulated on bit level/frame level to get the right timing behaviour.

The resulting delays and periods were used to compute a quadratic performance measure. Two master theses have been initiated and supervised by the author, (Österman, 2001 and Sjunghamn, 2003). These have been carried out in cooperation with the company Enea Real-Time Systems in Stockholm. The target system was a three link robot arm, each link with a micro controller running Enea's operating system OSE Epsilon and communicating by means of a CAN bus.

5.4 Paper D: Optimal scheduling with limited communication

5.4.1 Outline of the paper

The paper has the title "Scheduling of a limited communication channel for optimal control". The paper is based on the so called periodic Riccati equation. The periodic Riccati equation is found by first lifting the system and then solving an ordinary algebraic Riccati equation to get a stationary solution. This also gives a time-varying optimal controller that compensates for the time-varying delay. The periodic Riccati equation is applied in the static scheduling of a scarce hardware unit, e.g. a communication channel. The following problem can be formulated: given a limited hardware unit and a set of control applications, in what sequence should the controllers use the resource in order to optimize the overall benefit? The resulting schedule is optimal in the sense that it minimizes a weighted LQ loss of participating control loops. The loops are independent apart from the sharing of the uniprocessor or communication channel. The control period for each controller is not presumed beforehand. The control signal fed into the actuator may experience delay jitter. The uniform sampling period defines the slot length of the static schedule. Naturally, the computational complexity of the optimization problem is very high.

5.4.2 Contributions

A contribution of this work is the choice of control period. Unlike traditional scheduling and control codesign, the period is not an input parameter but falls out from the optimization. Further, since the actions of one control loop, as governed by the resulting static schedule, not necessarily are equidistant in time, the notion of period is truly an abstraction. It was Henrik Rehbinder's idea to apply this mathematical tool to the scheduling of controllers. Together we settled upon static scheduling, because it is intrinsically periodic and conceptually clear, allowing a focus on the control theoretical parts.

5.4.3 Related work by the author

The present paper is the last of three of papers on the same theme developed in Rehbinder and Sanfridson (2000a) and Rehbinder and Sanfridson (2000b). Different ways to work around the complexity by reducing combinations of sequences that will render the same or similar result have been tried. All papers have been written together with Henrik Rehbinder and a longer version of the present paper is found in his PhD thesis (Rehbinder, 2001).

5.5 A comparison to the licentiate thesis

The licentiate thesis (Sanfridson, 2000a) was finalized in May 2000 during the DICOSMOS project. The contents of the present thesis compared to the licentiate thesis is as follows. The best-case response time analysis, Paper A, is completely new. Instead of a state-of-the-art report on the timing properties in the licentiate thesis, there is Paper B which describes a performance

measure for the timing properties. The QoC architecture in Paper C is a continuation and generalization of a paper in the licentiate thesis, which focused on a specific hardware setup. The optimal scheduling of Paper D is a refinement of the optimization strategy compared to its predecessor found in the licentiate thesis.

6 Future work

In this section a few ideas of possible future work, to be a continuation of the presented thesis, are outlined.

6.1 The worst-case gain of a time-varying sampled-data system

The weighted performance measure in paper B is related to the H_2-norm. The measure can be said to give the average energy contents of the system. This is less applicable when it comes to robustness issues such as coping with transient timing behaviour. In that case, the gain or the H_∞-norm is essential to investigate. This is well-known theory for continuous time multivariable systems (Skogestad and Postlethwaite, 1996) and some work has also been done on discrete time systems (Scherer and Weiland, 2000) and possibly also for multi-rate systems. An open question is if LMIs can be used to study the H_∞-problem, for the case of time-varying sampling and delay. The discretization presented in paper B was developed with this in mind.

6.2 The worst-case jitter in a control loop

Delay jitter is naturally modelled as a stochastic process having some distribution. For example, the delay can vary between a low and high value, it can be modelled as varying according to a Markov chain, or it can be modelled as a uniform random variable. In the latter case, one parameter is enough to describe the jitter and it can easily be plotted in a diagram, which is convenient for the designer. If the distribution can be associated with a single variable, e.g. the worst-case delay, or the delay that gives the worst performance, a simple rule-of-thumb could possibly be constructed. Assume for example that the best-case, average-case and worst-case delays are given from a response time analysis. The actual timing behaviour of the delay jitter is typically time-varying and hard to characterize. Now, if the worst-case kind of jitter is known, the closed loop can be analysed from a worst-case perspective. This motivates a further study of the worst-case probability distribution of jitter.

6.3 Expected case response time for fixed priority scheduling

The control performance, robustness and stability depend on the actual timing behaviour, which can be characterized better by the average behaviour rather than the worst- or best-case. The response time analysis should thus include a notion of the expected-case response time. This analysis is closer to queuing theory, than classical response-time estimation in the real-time community. In queuing theory, jobs arrive at a queue according to some probability distribution. The service time of an element in the queue is also governed by a probability distribution. The total time from arrival to finish includes waiting time in the queue and processing time in the server. Fixed priority scheduling is one example of a scheduling strategy to analyse. There exists theory for priority driven queues including preemption, but the stochastic arrival and service processes do not match the model of a scheduling system. In basic queuing theory, the poison process is used, but for fixed priority scheduling, the arrival is periodic, and the service rate is hardly poison either. The evaluation of an expected-case response time, and possibly any second

moment figure for its accuracy, can be helpful in the design as an approximate indication of a likely timing behaviour.

6.4 On-line schedulability test and adaptive task parameters

A dynamic scheduling policy allows new tasks to be inserted into the task set currently running. In order to guarantee hard real-time, i.e. that every deadline is met, a schedulability test must be undertaken. By changing task parameters such as periods and offsets, a previously unschedulable task set can be made schedulable. This is typically done manually off-line, but in order to achieve flexibility a further automation of the scheduling work could prove useful. The scheduling strategy called earliest deadline first (EDF) has a really easy schedulability test. The response time analysis for fixed priority scheduling (FPS) is also straight forward, at least for a uniprocessor. One problem to study is how to select task parameters rationally in order to satisfy timing constraints (deadlines) during worst-case conditions and avoid transients due to the actual switching of parameters. The change of parameters will also affect the average timing behaviour. An on-line schedulability test made before activation ("pre-runtime") for a new set of tasks, while the old task set is still running, would increase the chances of admitting a new task.

6.5 Safety-critical and testing issues for QoC

Control applications are typically safety-critical. They need to be verified and tested with respect to the temporal behaviour, to assure dependability. Admission control plays a crucial role here. It is often stated that flexibility and dependability are difficult to combine. For example, a statically scheduled system without random jitter is beneficial in order to avoid state explosion during testing, but it is also beneficial at runtime, e.g. to achieve a high error detection coverage. But the static schedule needs to be substituted by a more dynamic scheduling algorithm in order to increase the flexibility. An important step is to assure that the logically verified and tested functions, keep their logical behaviour and output contained within specified boundaries. Another research idea is to make the scalability and negotiation distributed, since this could help increase the dependability.

6.6 Work bench for scalability and QoC

The proposed QoC architecture in paper C needs to be developed and implemented in at least one demonstrator, since this will increase the credibility of the approach. It is likely that new problems and important details will be revealed during this process, which will help impel and refine the architecture. One of the most important parts here, is to achieve scalability. Scalability is the primary feature to get working, since it is important for all classes of applications, not only for controllers. Different negotiation and admission control concepts should be tried out. These are typically specific for the dedicated systems, and it could be advantageous to investigate existing and running prototypes and products. The aim for scalability must be accompanied by a development method, which among other things will give a well-defined interface between modules. A codesign method has already been outlined. A task is to demonstrate that the effort of developing a flexible and modular platform is motivated.

7 References

Krishna C. M. and Shin K. G., “Real-time systems”, ISBN 0-07-114243-6, 1997.

Nilsson J., “Real-time control systems with delays”, Doctoral Thesis, Automatic Control, LTH, 1998.

Redell O., “Response time analysis for implementation of distributed control systems“, Doctoral Thesis, Mechatronics Lab, KTH, ISRN KTH/MMK/R--03/17--SE, 2003.

Redell O., El-khoury, J. and Törngren M., “The AIDA tool-set for design and implementation analysis of distributed real-time control systems”, J. of Microprocessors and Microsystems, Elsevier, Vol 28/4 pp 163-182, 2004.

Redell O. and Sanfridson M., “Exact best-case response time analysis of fixed priority scheduled tasks“, Proc. of the 14th Euromicro Workshop on Real-Time Systems, Vienna, 2002.

Rehbinder H., “State estimation and limited communication control for nonlinear robotic systems“, Doctoral Thesis, TRITA-MAT-01-OS-09, KTH, 2001.

Rehbinder H. and Sanfridson M., “Integration of off-line scheduling and optimal control“, Proceedings of the 12th European Conference on Real-Time Systems, Euromicro, pp. 137-143, Stockholm, 2000.

Rehbinder H. and Sanfridson M., “Scheduling a limited communication channel for optimal control“, Proceedings of the 39th IEEE Conference of Decision and Control, volume 1, pp. 1001-1016, Sydney, December 2000.

Sanfridson M., “Timing problems in distributed control”, Licentiate thesis, Mechatronics Lab, KTH, ISRN KTH/MMK--00/14--SE, May 2000.

Sanfridson M., “Problem formulation for QoS management in automatic control”, Technical Report TRITA-MMK 2000:3, Mechatronics Lab KTH, March 2000.

Sanfridson M., “Discretization of process and loss function for a control loop having time-varying period and control delay”, Technical Report ISRN KTH/MMK--03/02--SE, Mechatronics Lab, KTH, 2003.

Sanfridson M., “An overview of the use of LMI in control to assess robustness and performance”, Technical Report ISRN KTH/MMK--03/08--SE, Mechatronics Lab, KTH, 2003.

Scherer C and Weiland S, “Linear matrix inequalities in control”, version 3.0, October 2000.

Sjunghamn P., “QoS for embedded distributed control systems“, Master thesis, Mitthögskolan, 2003.

Skogestad and Postlethwaite, “Multivariable feedback control - Analysis and design”, ISBN 0-471-94277-4, Wiley, 1996.

Törngren M., “Modelling and design of distributed real-time control applications“, Doctoral thesis, Machine Design KTH, ISRN KTH/MMK--95/7--SE, 1995.

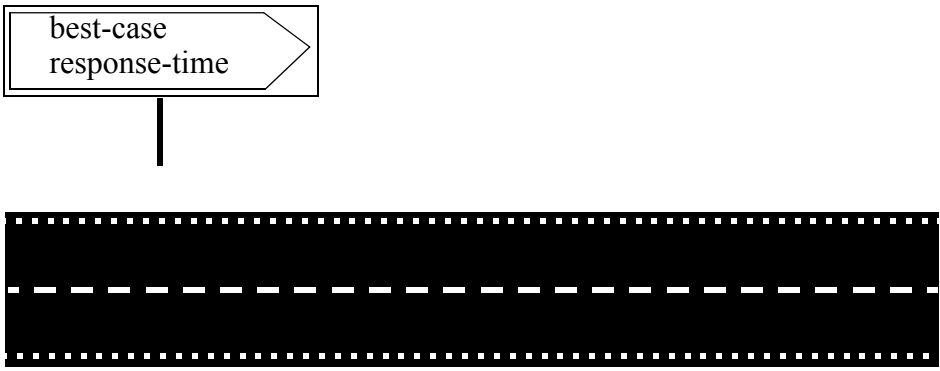
Törngren M., Andersson M., Wittenmark B., Torin J. and Wikander J., “Integrated real-time computer control system architectures — DICOSMOS2 — final report”, Projektnr P11762-1/P11762-2 and Dossie-Diarienr. 1K1P-98-06321/1K1P-99-06187, 2001.

Åström K. J., Wittenmark B., “Computer controlled systems - Theory and design“, 3rd edition, ISBN 0-13-314899-8, Prentice-Hall, 1997.

Österman J., “Scalability and QoS for embedded distributed control systems“, Master thesis, Mechatronics Lab, KTH, MMK 2001:81 MDA181, 2001.

Paper A

Martin Sanfridson and Ola Redell, “Exact best-case response-time analysis of fixed priority scheduled tasks with arbitrary response time”, submitted to a journal, September 2003.



Exact best-case response time analysis of fixed priority scheduled tasks with arbitrary response times

Martin Sanfridson and Ola Redell
{mis, ola}@md.kth.se
Mechatronics Laboratory
Department of Machine Design, KTH, Sweden

Abstract. A solution to the exact best-case response time analysis for a set of independent periodic tasks with fixed priorities is presented here. The analysis is dual to the well-known worst-case response time analysis. The motivation for this work comes primarily from the study of jitter in real-time systems dedicated to feedback control. In the best-case response time analysis, higher priority tasks are first phased relative to the studied low priority task, such that the most favourable interval for execution is established. This enables an expression for the best-case response time. A recurrence equation then yields the best-case response time by converging to a solution from a high initial value. In case the worst-case response time is longer than the period, instances of the studied task can interfere with each other. This is handled by identifying the start of the busy period in which one of the instances experiences the best-case response time.

Keywords: Best-case response time, worst-case response time, fixed priority scheduling, rate monotonic analysis, delay jitter, time-varying delay, response jitter, best-case execution time.

1 Introduction

In this paper we will present an analysis and an accompanying algorithm for calculating the exact best-case response time of a task belonging to a set of periodic and independent tasks. The task set is assumed to be feasibly scheduled on a uniprocessor by fixed priority pre-emptive scheduling. The analysis is based on the introduction of the *best-case phasing* of a low priority task relative to its higher priority tasks, assuming that tasks can be phased arbitrarily. A task has its best-case phasing when it finishes its execution at an instant when all tasks with higher priorities are released, after having experienced their maximum release jitter. We show that a low priority task that has executed according to this best-case phasing scenario has the shortest possible response time.

The analysis of the *best-case response time*, R^b, is in many respects similar to the well established analysis of the *worst-case response time,* R^w. For readers who are acquainted with the worst-case analysis the dualism will become clear, even though the line of reasoning might seem backward initially. A notable duality is the well-known scenario called critical instant, which describes a *worst-case phasing* of a task set. The worst-case phasing is the starting-point when deriving a recurrence equation for calculating the longest possible response time. The equivalent best-case scenario is the *best-case phasing*, which is quite a similar arrangement of the task set. The introduction of the best-case phasing leads in a natural way to an equivalent derivation of a recurrence equation to calculate the shortest possible response time. Also, a sit-

uation where instances of the same task can interfere with each other is handled in a similar way.

One of the most prominent applications of the worst-case response time analysis is to guarantee *schedulability* of a task set, i.e. the case when every task has to meet its hard *deadline*. This is of course the reason why worst-case analysis is important. A prerequisite for the best-case response time analysis imposed here, is that schedulability already has been guaranteed by e.g. a worst-case analysis. In the sequel, the worst-case response time of the studied task is allowed to have an arbitrary length as long as it is bounded, $R^w < \infty$. Thus, an alternative formulation is to define a task set to be schedulable when the worst-case response time for every task is bounded. This is equivalent to assuming a utilization, $U \leq 1$. When the worst-case response time of the studied task is greater than its period, $R^w > T$, an instance studied in the best-case analysis can be interfered by its preceding instances. This is called *self-interference*, and it leads to a postponed start of execution of the studied instance. Therefore, the exact best-case analysis requires a little extra work compared to the case for which $R^w \leq T$ holds, i.e. when there is no self-interference.

Consider the small example task set in Figure 1.1. By a direct inspection it is seen that R^b for task τ_2 is significantly longer than its *best-case execution time*, C^b. It is obvious that the reason is interference from τ_1, which has a higher priority. The simple conjecture to define R^b equal to C^b as a lower bound on the response time, is clearly not exact even though it is correct as a bound. The example in the figure justifies the need for a theory to improve the calculation of a best-case bound for the response time.

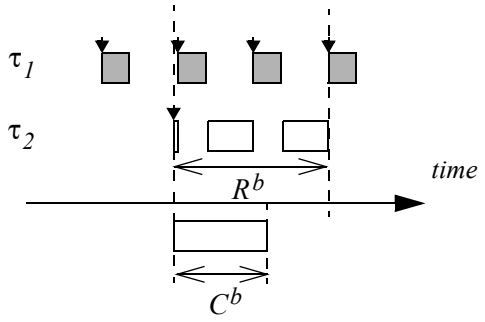


Figure 1.1 The execution time C^b is an ambiguous estimation of the best possible response time R_b. A down pointing arrow means arrival time. A non-preempted execution of τ_2 has been drawn below the time line for comparison.

Why is the best-case response time relevant to study? There are several reasons to this. We are interested in a class of real-time systems which is sensitive to variations in the response time, and this problem can be reduced if the timely behaviour is known — not only the worst-case. Delays in a feedback control system, e.g. as a result of the response time of a task, contribute to the phase lag in the closed loop dynamics, see Törngren (1998) for a discussion of delays in real-time computer control systems. It is well-known from control theory that the phase lag leads to a deterioration of the performance and the stability margin. The main goal in control design is to meet the specifications for performance and stability, which among other things depend on the total picture of delays in the system. By knowing the best-case response time, a more complete picture of the delays can help improve controller synthesis for time-varying delays.

Video transmission over a channel is another class of real-time system susceptible to time-varying delay. For unidirectional transmission of frames, the total delay is typically less important for the quality as perceived by a spectator. When the total delay is important and buffering to eliminate jitter is not an option, e.g. for a teleoperated robot, knowing the best-case delay could help tune the system to improve overall quality.

Another possible use of the best-case response time analysis is in admission control for on-line scheduling. The purpose of admission control is to regulate the utilization of a resource, by allowing or disallowing tasks to commence execution. The admission control can for example let a task execute if it stands some chance to finish within its deadline D, even if $R^w > D$. On the other hand, if there is no chance at all, i.e. $R^b > D$, processing time can be saved by skipping the task. Here, the deadline is compared to R^b rather than R^w because the objective is opposite to the common use of deadlines.

Yet another answer to the question why best-case analysis of response times is relevant, is event synchronization. For example, an event has to be triggered before a specific task finishes. Consider a buffer with one memory position and a global clock, but with no other means of communication between the producer and consumer residing at different processing units. The best-case response analysis enables the consumer to predict the latest time instant it is safe to read the buffer before a new value is written to it.

The analysis presented here is said to be *exact*, which means that it gives the maximum *lower bound* on the response time for the given premises, and not only an approximated lower bound. Assume that the maximum lower bound R^b and the minimum upper bound R^w are known, see Figure 1.2. The time window for a response of τ_2 is clearly bounded by $R \in [R^b, R^w]$. A bound in the shaded section $A^b \in [0, R^b)$ to the left on the time line is a correct but conservative lower bound of the response time. An example of a simple lower bound is the best-case execution time, C^b. Similarly, a correct but conservative upper bound falls within the shaded section $A^w \in (R^w, \infty)$ to the right on the time line.

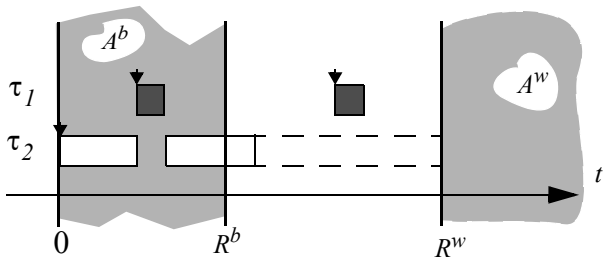


Figure 1.2 The low priority task τ_2 can only finish within a time window $[R^b, R^w]$ between the shaded time intervals A^b and A^w.

The analysis of fixed priority pre-emptive tasks, known as rate monotonic analysis (RMA), was introduced by Liu and Layland (1973). With Joseph and Pandya (1986) the interest also turned to response time analysis of fixed priority systems (FPS) for the design of real-time systems capable of tolerating worst-case conditions. The set of analysis methods now handles a wide range of scenarios for which the original restricting assumptions have been relaxed successively. Extensions made by among others Lehoczky (1990) and Audsley et al. (1993) encompass systems with task synchronisation, precedence and exclusion constraints, periodic tasks mixed with aperiodic tasks, tasks with arbitrary deadlines, etc. Klein et al. (1993) detail a number of different scenarios. The analysis was initially formulated for single processor sys-

tems, but has also been extended to holistic scheduling of systems with multiple processors. A method proposed by Tindell and Clark (1994) gives the worst-case response time of precedence related tasks in distributed systems. The method is based on local analysis of each node with the *response jitter*, which is defined as the difference between the worst- and best-case response times, of one task or message being the release jitter of the next task in the sequence. The conservative bounds of the response jitter propagate in this method to pessimistic bounds of response times for subsequent tasks in the chain; consequently, it is important to find tight bounds on these response times. The method by Tindell and Clark (1994) has been further developed by Palencia et al. (1998) with a tighter estimate of the best-case response time. They find a lower bound by assuming that all higher priority tasks finish at the instant the analysed low priority task is released, after having experienced their best-case response times. This results in a lower bound for the best-case response time, but it is not exact. Henderson et al. (2001) try to develop Palencia's ideas further and do find the exact best-case response time. However, their solution leads to a numerically intractable search through all possible orderings of higher priority task executions, prior to the release of the studied low priority task. In Redell and Sanfridson (2002) the best-case phasing scenario is introduced, which leads to the formulation of a recurrence equation similar to that of the worst-case response calculation. The algorithm converges to an exact solution and has the same complexity as the corresponding worst-case response calculation. This best-case analysis is here extended to encompass the case for which self-interference is present.

The presentation has the following outline. In 2 the task model is described in detail, and the problem of finding the best-case response time is formally stated when the central terminology has been clarified. In 3 the particular phasing of tasks that enables an estimation of the best-case response time is proposed and proved. In 4, the recurrence equation is derived. This recurrence equation is not satisfactory when two instances of the same task can interfere with each other. Thus, in 5, the case of arbitrary response times is dealt with. In 6 we show how the calculations can be performed by giving pieces of pseudo code and an example. It is also investigated, by randomly generated tasks sets, how frequent the theory covered here applies in different situations. A final discussion in Section 7 covers some issues concerning delimitations, the validity of the task model and possible extensions. Finally, we conclude the article in Section 8 by recollecting the most important pieces of the presentation.

2 The task model

The starting-point of the analysis is a model of the real-time computer system. This is accompanied by a problem statement and a recapitulation of the key results from the worst-case analysis.

A set of m periodic tasks τ_k, with $k = 1, 2, \ldots, m$, runs on a uniprocessor. Each task is characterized by the following parameters:

- A fixed and unique *priority*. The index k of τ_k determines the priority order, with $k = 1$ meaning the highest priority, such that the task set is conveniently priority ordered without loss of generality.
- A constant *period time*, T_k.
- Two bounds on the *execution time*: a best-case C_k^b and a worst-case C_k^w execution time. The actual execution time is allowed to change from one instance of a task to the next, but it is lower and upper bounded.

- The *release jitter*, which can change from one instance of a task to the next, but is bounded by $[0, J_k]$, where $J_k \geq 0$ is the maximum jitter.

The task set is assumed to be feasibly scheduled, i.e. the response time requirement that all *deadlines* are met is satisfied, and $R_k^w < \infty$ holds for every k. This implies that the *utilization* is bounded too, i.e. $U \leq 1$.

Index i refers to a studied low priority task τ_i, and index j refers to a higher priority task τ_j, whereas index k is used to denote a general task, τ_k. The formulation $\tau_j \in hp(\tau_i)$ or $j \in hp(i)$ is equivalent to $j < i$, which follows from the priority order of the task set. A second subscribed index is occasionally needed to refer to a particular instance of a task: τ_{kp} means some instance $p = 1, 2, \ldots$ of τ_k, where p is in chronological order. Indices q and s are reserved to denote task instances with specific properties; these will be defined later.

A task τ_k *arrives* periodically from a scheduler with period T_k, i.e. a new instance of τ_k is allowed to run. After its arrival at time a_k, it is put in the ready queue of the dispatcher. This *release* at r_k may be delayed by an amount of time called *release jitter*, which equals to $r_k - a_k$. After its release, τ_k starts executing (*start time*) as soon as it has the highest priority of all the tasks in the ready queue and when all previous instances of τ_k are *finished* (completed). During execution, the task can be *preempted* by any task with a higher priority. The execution resumes when the task regains the highest priority of all tasks in the ready queue. When the execution need has been satisfied, execution ends at the *finish time* f_k, and the task is cleared from the ready queue. The tasks are independent, i.e. it is assumed that there are no *exclusion* or *precedence* constraints that can make a lower priority task *block* the execution of a higher priority task. The *phase* is a relation between arrival times of two different tasks. The *phasing* of a task set defines the phase of every task relative to another task in the set. The phasing and the release jitter are arbitrary and they take on values that benefit the formation of a best-case phasing scenario.

The *response time* R_k is defined to be the interval between arrival and finish times $R_k = f_k - a_k \geq 0$, and it is bounded $[R_k^b, R_k^w]$. The problem we will study is how to calculate the best-case response times R_k^b for every task τ_k in the task set, or equivalently to find $R_i^b = \min f_i - a_i$ for the low priority task τ_i. A direct consequence of the task model, is that every task τ_k in the original task set with a lower priority than that of the studied task, i.e. $k > i$, can be disregarded in the analysis. The only sources of variation in the response time of a task are release jitter, *interference* from higher priority tasks and *self-interference* from previously released but not yet finished instances of the task itself.

As a reference for the analysis described in this paper, we state the corresponding result for the *worst-case response calculation* here, see for example Audsley et al. (1993) and Tindell and Clark (1994). The worst-case response time R_i^w of a task τ_i is expressed by:

$$R_i^w = \max_{q = 0, 1, \ldots} w_i(q) - qT_i + J_i \tag{1}$$

where the workload is

$$w_i(q) = (q+1)C_i^w + \sum_{j \in hp(i)} \left\lceil \frac{w_i(q) + J_j}{T_j} \right\rceil \cdot C_j^w \tag{2}$$

Since equation (2) cannot be solved by isolating $w_i(q)$, it has to be solved by iteration starting with e.g. $w_i(q) = (q+1)C_i^w$ for some $q \in \{0, 1, \ldots\}$. When the iteration has converged, equation (1) is used to calculate the worst-case response of τ_i. When $qT_i > w_i(q)$ the size of the busy period, containing the instance with the worst-case response time, is covered and index q

needs not be increased further. If it is known that $R_i^w \leq T_i$ then letting $q = 0$ gives a correct result.

3 Best-case phasing

In this section the *best-case phasing* will be derived. It is the starting-point for calculating the best-case response time. We introduce the concept by an example and leave the details to Theorem 1, where the best-case phasing is formally stated and proved based on the task model in Section 2.

In Figure 3.1, three tasks without release jitter form a *schedule* when they execute on a uni-processor. An instance of the low priority task τ_3 arrives at a_3 and finishes at f_3. Assume that a_3 is fixed in time and that the arrival of τ_1 also is left untouched. Then, if the arrivals of all instances of τ_2 are *shifted left* (i.e. brought backward in time) by for example a time interval equal to d, the finish time f_3 will necessarily be shifted left. This action yields a shorter response time $R_3 = a_3 - f_3$. The left shift can be repeated for τ_1 and τ_2 until f_3 no longer can be forced to decrease further. The idea is to move as many instances of τ_1 and τ_2 out of the interval $[a_3, f_3]$ as possible. An alternative is to conduct a reasoning where f_3 is held fixed in time and a_3 is floating. Further, release jitter can be handled by similar arguments.

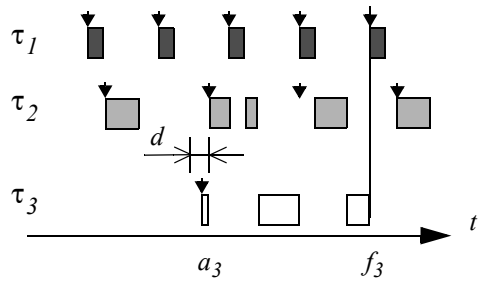


Figure 3.1 An example with three tasks. The studied low priority task τ_3 arrives at a_3 and finishes at f_3. Imagine that the intermediate priority task τ_2 is shifted left by a distance d.

The best-case phasing will render the most favourable time interval for τ_3 to execute within. It turns out that τ_3 finishes at an instant when all its higher priority tasks start executing. We call this instant the *favourable instant* to underline the dualism between the worst-case and best-case analysis. The worst-case analysis is based on a *worst-case phasing* where all tasks, including the low priority task, start at the *critical instant*, as defined by Liu and Layland (1973).

For the derivation of the best-case phasing we adopt the concept of busy period, which was introduced by Lehoczky (1990). A level-*i* busy period is a time interval $[t_a, t_b]$ during which the processor is occupied executing tasks with priority *i* or higher, but no such task is executed during the intervals $(t_a - \varepsilon, t_a)$ and $(t_b, t_b + \varepsilon)$ for some sufficiently small $\varepsilon > 0$. Since the busy period will be used exclusively for τ_i the statement of priority level will be omitted in the sequel.

The relation between release and start of the studied τ_i within a busy period that leads to the best-case response time, is given by Lemma 1. The result will later be used to state what the phasing must be between τ_i and every higher priority task τ_j.

Lemma 1. *Immediate start at release within a busy period leading to* R_i^b. A task instance τ_{iq} that experiences its best-case response time R_i^b, executes in a busy period that includes at least one instance τ_{ip}, with $p \le q$, for which the release and start times coincide.

Proof. This will be proved by a contradiction. Assume that τ_{iq} experiences the minimum response time for τ_i. Further assume that all preceding instances τ_{ip}, $p \le q$, belonging to the same busy period as τ_{iq} are released when either earlier instances of τ_i or higher priority tasks are executing. Instances succeeding τ_{iq} are obviously not of interest. Under these conditions, it is possible to right shift (delay) the arrival times — and thereby the release times — of the instances of τ_i without affecting the start or completion time of τ_{iq}. Since the response time of τ_{iq} will be reduced by such a delay, this contradicts the assumption that τ_{iq} experiences the minimum response time for τ_i. The arrival times of τ_i can be delayed until the release and start time of at least one τ_{ip} coincide.

■

It is quite clear that earlier instances of τ_i will be less likely to interfere with the execution of τ_{iq} if their releases are not delayed by any release jitter. This is true whether the preceding instances are part of the same busy period or not; an earlier instance must be or become a part of the busy period in order to cause interference with τ_{iq}. Furthermore, the instance τ_{iq} itself should not be delayed by release jitter. This is stated in the following lemma.

Lemma 2. *Release jitter of* τ_i. The phasing of the instances of τ_i that allows τ_{iq} to experience its minimum response time is such that all instances τ_{ip} with $p \le q$, should experience zero release jitter.

Proof. The fact that the release jitter of τ_{iq} should be zero in the best-case is obvious given that the response time of τ_{iq} is the interval from arrival to completion, and that any non-zero release jitter can only serve to make this interval longer. Similarly, the interference from earlier instances with the execution of τ_{iq} will be reduced or stay the same when the release jitter of any earlier instance is reduced. On the other hand, if the release jitter is increased, the interference with τ_{iq} will either increase or stay the same. The interference from earlier instances adds directly to the response time of τ_{iq} and hence the best-case is achieved when no release of any earlier instance of τ_i is delayed by release jitter.

■

Given the two lemmas above, we conclude that under the best-case phasing of τ_{iq} there is at least one instance of τ_i, either τ_{iq} or an earlier instance in the same busy period, that has coinciding arrival, release and start times. Below in Theorem 1 we first prove the best-case phasing between every τ_j, and then their phasing in relation to τ_i. This is done in the sense that we uniquely define the relations between all release times of all tasks. The resulting phasing is a sufficient condition for τ_{iq} to experience the minimum response time of τ_i.

Theorem 1. *The best-case phasing.* The best-case response time of a task τ_i is achieved for an instance τ_{iq} that finishes simultaneously with the release of instances of all higher priority tasks, when these have experienced their maximum release jitter. Every instance of a higher priority task that is released before this completion time f_{iq}, should be released without any release jitter. The instance τ_{iq} and all earlier instances of task τ_i should be released without any release jitter and at least one of these instances, that belongs to the same busy period as τ_{iq}, should start its execution immediately on arrival.

Proof. Pick an arbitrary instance τ_{iq} in a feasible schedule, and make its release jitter and the release jitter of every earlier instance of τ_i, equal to zero. According to Lemma 2 this can only lead to a reduction in the response time of τ_{iq}.

We will now apply an iterative procedure that will successively and monotonically reduce the response time R_{iq}. Eventually, the release times of all instances of all tasks that may affect R_{iq} will be specified; or equivalently, every arrival time and release jitter will be specified. Therefore, since the procedure can be applied to reduce the response time of an arbitrary instance τ_{iq} in a schedule, it leads to the best-case phasing for τ_{iq}, which results in the minimum response time of τ_i.

First initialize a set of higher priority tasks P as the empty set, $P = \varnothing$. The iterative procedure is started by picking some higher priority task τ_j that is not yet in P. Given the current phasing of τ_{iq} and all higher priority tasks, let τ_{jp} be the first instance of the task τ_j to be released at or after the completion of τ_{iq}, i.e. the favourable instant f_{iq}. This means that $r_{jp} \geq f_{iq}$ and $r_{j(p-1)} < f_{iq}$. Task τ_{jp} arrives at an instant a_{jp} and the inter arrival time of τ_j is exact periodic with T_k. Figure 3.2 illustrates a busy period including the analysed task instance τ_{iq} when executed together with a single higher priority task τ_j.

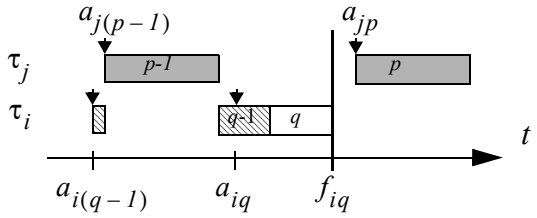


Figure 3.2 Illustration of a busy period including the analysed instance τ_{iq} immediately preceding $\tau_{i(q-1)}$ and one instance of a higher priority task, $\tau_{j(p-1)}$. The task instances are tagged with their instance indices. A down pointing arrow means arrival. No instance experiences release jitter.

In the sequel we make the assumption that τ_{jp} must stay the first instance of τ_j to be released at or after the finish time f_{iq} no matter how f_{iq} changes due to re-phasing of higher priority tasks. Hence, no phasing of the task set that will make this assumption be violated is allowed. The assumption does not restrict the generality of the theorem since there has to be a first instance of τ_j after f_{iq} and it may just as well be τ_{jp}.

We next study the contribution from the execution of instances of τ_j to the response time of τ_{iq}. Shifting the arrival times of the instances of τ_j to the right (increasing a_{jp}) will not decrease the response time of τ_{iq}. The completion time f_{iq} and therefore also the response time R_{iq} will either stay the same or increase by such a shift. On the other hand, shifting the arrivals of instances of τ_j left (decreasing a_{jp}) will not increase the response time of τ_{iq}; f_{iq} will either decrease or stay the same by such a shift. Therefore a_{jp} is decreased until $r_{jp} = f_{iq}$. During this shift of a_{jp}, we simultaneously update the arrival times of the tasks in P (i.e. left shift the arrivals if necessary) such that even though f_{iq} decreases, $r_{mp} = f_{iq}$ is always satisfied for each $\tau_m \in P$.

When the left shifting process terminates because the limit $r_{jp} = f_{iq}$ is reached, we note that if τ_{jp} does not experience its maximum release jitter it is possible to further reduce a_{jp}. Hence if the release jitter of τ_{jp} is not equal to J_j, we successively increase the release jitter while reducing a_{jp} such that $r_{jp} = f_{iq}$ at all times. This will potentially make f_{iq} decrease further (while

decreasing the release jitter will not). The arrivals of all tasks in P are again updated continuously (left shifted) as needed. While increasing the release jitter of τ_{jp}, we must also make sure that no later instance of τ_j, e.g. $\tau_{j(p+1)}$, will get released before f_{iq}. An easy way to assure this is to set the release jitter of all later instances of τ_j to their maximum values. When the release jitter of τ_{jp} has reached its maximum value J_j, the shifting of arrival times is terminated. At this point τ_j is added to the set P: $P = P \cup \tau_j$.

The next step in the process is to tune the release jitter for all earlier instances of τ_j. It is obvious that f_{iq} cannot decrease by an increase in the release jitter of some τ_{js}, $s < p$. On the other hand, f_{iq} will either decrease or stay the same when the release jitter of τ_{js} is decreased. Hence all instances τ_{js}, $s < p$, should experience their minimum release jitter. We reduce any release jitter of all early instances of τ_j while simultaneously updating the arrival (and release) times of all tasks in P as needed, just like before.

During the process of updating the arrival and release times of τ_j and the tasks in P, the busy period that contains the execution of τ_{iq} may have changed and the completion time f_{iq} of τ_{iq} may have been reduced. Since the arrival time of τ_{iq} has not changed, the response time of τ_{iq} has either been reduced or has not been changed.

The above procedure is repeated for every higher priority task τ_j, $j = 1, 2, \ldots, i-1$, successively reducing the response time of τ_{iq}. When the iterations are over and all higher priority tasks are included in P, we need to study the arrival times of τ_i. If at this time τ_{iq} does not satisfy Lemma 1, the arrival times of task τ_i are delayed up to the point when either τ_{iq} or an earlier instance of τ_i belonging to the same busy period arrives at the instant when it starts its execution. Finally, the release times of all task instances that may affect R_{iq}^b have been uniquely specified.

■

Some observations can be emphasized at this point. Because of self-interference, the equality $r_{iq} = a_{iq}$ does not always hold. Nevertheless, if it can be guaranteed in some way that self-interference cannot possible be present, this would simplify the calculations. The busy period does not necessarily have to start by an instance of τ_i and the busy period does not end at the favourable instant. The interpretation of Theorem 1 is that instance τ_{iq}, which finishes at the favourable instant, is the shortest one in the busy period (from the start of the busy period up to the favourable instant). It should also be clear that the best-case response must be shorter than the task period $R_i^b \leq T_i$.

To proceed, we need a concept to capture the contiguous execution of tasks beginning by the arrival and simultaneous start of some instance τ_{ip} and ending in the favourable instant by τ_{iq}.

Definition 1. *Virtual task.* A virtual task τ_{vp} of an original task τ_i has the priority i and the best-case execution time $C_{vp}^b = pC_i^b$ with $p \in \{1, 2, \ldots\}$. τ_{vp} does not experience self-interference.

■

The virtual task is used to substitute the studied task τ_i in a part of the response time analysis. In the task set, τ_i is removed and replaced by τ_{vp}. All τ_j are left unchanged and they do not have virtual tasks. The second subindex refers to the number of substituted instances. The period for τ_{vp} is lacking in Definition 1 and in agreement to this, there is by the definition no self-interference. If there is only one instance of a task (the period can be thought of as infinitely long) then self-interference becomes impossible. The period of the studied task is not necessary for the response time calculations presented in the following 4.

It is important to note that the schedule under the best-case phasing of τ_{vp}, and the original schedule under the best-case phasing of τ_i, are identical for all tasks with priorities higher than *i*. In other words, the execution pattern of the higher priority tasks is the same in the two schedules and the only difference is the execution of τ_{vp} and τ_i. The best-case response time of a virtual task R^b_{vp} is a time interval (within a busy period) between the point in time when the virtual task arrives at some a_{ip}, and the point in time when it ends, f_{iq}. The following Section 4 treats the response time calculation when instances do not interfere with each other. This will later be used in Section 5, which treats the general case of an arbitrary worst-case response time R^w_i.

4 The response time calculation

In this section, a recurrence equation is first derived to calculate the best-case response time of a task which does not experience self-interference. This could be for example a virtual task or a task where $R^w_i \le T_i$ is guaranteed to hold. It is then proved that the derived equation converges to the desired result.

Theorem 2. *Best-case response of a task without self-interference.* Let a schedulable set of independent fixed priority tasks be given. The best-case response time R^b_i of a task τ_i, which does not experience self-interference, satisfies the following equality:

$$R^b_i = C^b_i + \sum_{j \in hp(i)} \left\lceil \frac{R^b_i - J_j - T_j}{T_j} \right\rceil_0 \cdot C^b_j \tag{3}$$

where the ceiling operator has been extended to $\lceil x \rceil_0 = \max(0, \lceil x \rceil)$.

Proof. Assume a best-case phasing of τ_i according to Theorem 1 and a timely coincidence of the arrival, release and start times of τ_i as stated in Lemma 1. We define the phase from the release of τ_i to the first following release of a higher priority τ_j to be ϕ_j. Figure 4.1 shows the best-case phasing of a low priority task τ_3 when executed together with the two higher priority tasks τ_1 and τ_2. These are released at $\{..., f_3\text{-}2T_1\text{-}J_1, f_3\text{-}T_1\text{-}J_1, f_3, ...\}$ and $\{..., f_3\text{-}2T_2\text{-}J_2, f_3\text{-}T_2\text{-}J_2, f_3, ...\}$ respectively. τ_3 arrives at times $\{..., r_3\text{-}T_3, r_3, ...\}$. Task instances released at and after f_3 do not need to be considered since they have no effect on the studied response time.

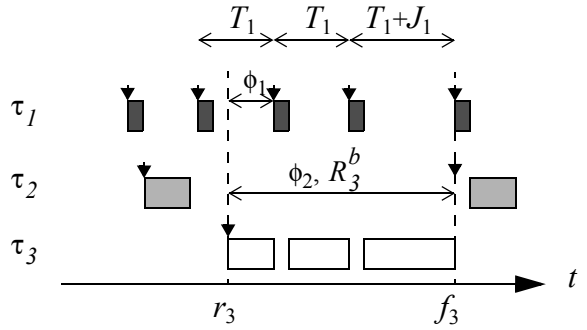


Figure 4.1 The best-case phasing of τ_3 with release time r_3 and finish time f_3.

The response time of τ_i can be expressed as the sum of its execution time and interference by all higher priority tasks:

$$R_i^b = C_i^b + \sum_{j \in hp(i)} I_i^b(j) \tag{4}$$

Here $I_i^b(j)$ is the interference of the execution of τ_i by τ_j when the tasks are phased according to the best-case phasing for τ_i. Hence, in order to find the best-case response time of τ_i, we need to find the amount of interference due to all higher priority tasks. By studying Figure 4.1 we find that there are two cases to be considered to find an expression for the interference of a higher priority τ_j:

$$\text{Case 1: } R_i^b < T_j + J_j$$
$$\text{Case 2: } R_i^b > T_j + J_j$$

whereas from Lemma 1 we see that

$$R_i^b \neq k \cdot T_j + J_j \text{ for } k = 1, 2, \ldots \tag{5}$$

which means that $(R_i^b - J_j) / T_j$ cannot be an integer value. Case 1 is exemplified by τ_2 which has $T_2 + J_2$ large enough not to interfere with the execution of τ_3 at all. This case is obviously simple to handle. Case 2 is exemplified by τ_1, which does interfere with τ_3. It follows from Figure 4.1 that R_3^b can be expressed as $R_3^b = \phi_1 + 2 \cdot T_1 + J_1$, while a general relation between the best-case response time of a low priority task τ_i and any $j \in hp(i)$ that belongs to case 2 is

$$R_i^b = f_i - r_i = \phi_j + n_j \cdot T_j + J_j \tag{6}$$

where $n_j \in \{1, 2, \ldots\}$ is the number of instances of τ_j that interfere with τ_i. As a result of Lemma 1 the phase is bounded by

$$0 < \phi_j < T_j \tag{7}$$

which together with (6) becomes $0 < R_i^b - n_j \cdot T_j - J_j < T_j$. Rewriting this expression results in:

$$\frac{R_i^b - J_j}{T_j} > n_j > \frac{R_i^b - J_j}{T_j} - 1 \tag{8}$$

The bound on n_j in (8) can be rewritten into two equally valid expressions:

$$n_j = \left\lfloor \frac{R_i^b - J_j}{T_j} \right\rfloor \tag{9}$$

$$n_j = \left\lceil \frac{R_i^b - J_j}{T_j} \right\rceil - 1 = \left\lceil \frac{R_i^b - J_j - T_j}{T_j} \right\rceil \tag{10}$$

Any of (9) and (10) can be used to express the number of instances of τ_j that interfere with τ_i during its best-case phasing. We will henceforth use (10) since it has characteristics better suited for finding the best-case response time of τ_i. Notice that $\lfloor a \rfloor = \lceil a \rceil - 1$ for all real-valued positive a except when a is an integer.

By combining the results for the two cases of interference, we find that the number of interfering instances of τ_j can be expressed as:

$$n_j = \begin{cases} 0 & \text{if } R_i^b < T_j + J_j \\ \left\lceil \dfrac{R_i^b - J_j - T_j}{T_j} \right\rceil & \text{if } R_i^b > T_j + J_j \end{cases} \tag{11}$$

which is equivalent to

$$n_j = \left\lceil \frac{R_i^b - J_j - T_j}{T_j} \right\rceil_0 \tag{12}$$

It is now possible to state an expression for the interference of τ_i by a higher priority task τ_j, when τ_i is phased according to the best-case phasing defined in Theorem 1. The interference is:

$$I_i^b(j) = \left\lceil \frac{R_i^b - J_j - T_j}{T_j} \right\rceil_0 \cdot C_j^b \tag{13}$$

Adding up the interference of all tasks in $hp(i)$ and the execution time of τ_i itself, as given by (4), this proves expression (3) to be the best-case response time of τ_i.

■

At this point we have an expression in (3) that the best-case response time of i must satisfy. However, in analogy with the worst-case response time calculations, this equation cannot be solved by isolating the desired variable. Therefore, the minimum response time has to be found through iteration. We now need to show that this is possible, i.e. that the iteration converges to a unique solution. We restate equation (3) as a recurrence equation:

$$R^{n+1} = C_i^b + \sum_{j \in hp(i)} \left\lceil \frac{R^n - J_j - T_j}{T_j} \right\rceil_0 \cdot C_j^b \tag{14}$$

over $n = 0, 1, \ldots$ and note that the right hand side corresponds to the complete workload due to high priority tasks released in an open interval $(f_i - R^n, f_i)$ plus the workload of τ_i itself. Figure 4.2 shows how R^n should be interpreted.

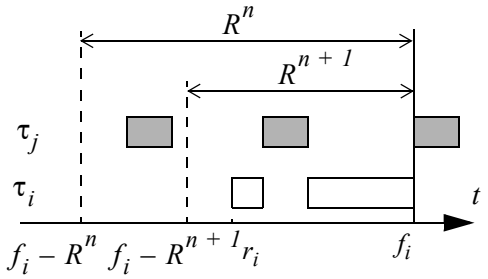


Figure 4.2 Interpretation of R^n during recursion.

We first need to show that a recursion over (14) will converge. Then we need to show that this value is the correct minimum response time and not one of the other possible solutions to (14) (there is usually more than one, see Figure 4.3).

Note that there must exist a finite R^0 such that the interval $(f_i - R^0, f_i)$ includes some interval during which the processor is either idle, executes tasks with a priority lower than *i*, or executes a previous instance of τ_i (whose workload is not included in (14)). If the iteration is started with such an R^0 we know that $R^1 < R^0$ and we call this a valid R^0.

Lemma 3. *Convergence.* The iteration over R^n in (14) starting with a valid R^0, will converge to the largest possible solution of (14).

Proof. By observing that the right hand side of (14) is strictly monotonically decreasing with decreasing R^n, i.e. $R^{n+1} \leq R^n$ with equality only at convergence, and since the number of possible values produced by (14) is finite, the iteration will converge. It is obvious that this solution is the largest solution to (14).

■

In the following, R^* denotes the largest solution of (14). Note that any R^0 larger than or equal to R^* will make the iteration converge to R^*. It now remains to show that R^* is in fact R_i^b and the following Lemma 4 addresses this issue. Figure 4.3 depicts the difference between the two possible solutions R and R^* of (14), but R cannot be the best-case response time of τ_i since the iteration can not converge from R^0 to R.

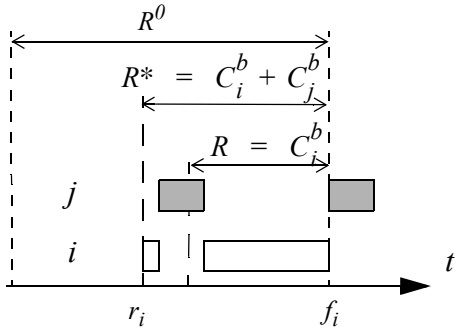


Figure 4.3 R and R^* are two solutions to (14).

Lemma 4. *Uniqeness.* The instant $r_i = f_i - R^*$ must correspond to the release and start of τ_i.

Proof. We show this by making two statements about R^* that can easily be verified through observation of (14).

First, it is clear that R^* corresponds to an interval $[f_i - R^*, f_i]$ during which the processor is completely occupied executing τ_i or higher priority tasks. Furthermore all these tasks are released within the interval $[f_i - R^*, f_i)$.

Second, it is clear that the workload due to τ_j that is included in (14) must be released within the open interval $(f_i - R^*, f_i)$. This behaviour is a result of choosing expression (10) instead of (9) for the number of interfering instances of a higher priority task.

By combining the two observations, we find that τ_i must be released and start its execution at $f_i - R^*$.

■

We are now ready to sum up the results in Theorem 3 regarding the iterative solution (14) of the expression in (3).

Theorem 3. *Best-case calculation of a task without self-interference.* The best-case response time of a task τ_i, which does not experience self-interference, is found by iterating the recurrence equation (14) starting with the worst-case response time $R^0 = R_i^w$ converging down to the best-case response time $R_i^b = R^n$.

Proof. The worst-case response time of τ_i, R_i^w as derived from (1), is always larger than or equal to R_i^b and hence R^*. Therefore, by Theorem 2, Lemma 3 and Lemma 4, this will yield R_i^b in a finite number of steps.

■

Remark. Another possible initial value R^0 is T_i.

5 The best-case response time

Assume a best-case phasing, as derived in Section 3. When two or more instances of τ_i interfere with each other they appear, roughly speaking, as one instance over a busy period. This prolonged instance — or virtual task, see Definition 1 — can be characterized by one start time, which adheres to Lemma 1, and one finish time. Theorem 2 and the recurrence formula derived in Section 4, provide a tool to find the best-case response time of the virtual task. Theorem 1 states that the instance finishing at the favourable instant, i.e. the last instance of the virtual task, has the best-case response time. The main problem we need to solve now is to find out how many instances of τ_i this virtual task consists of. We cannot know this beforehand, except for a special case such as $R_i^w \le T_i$ where the virtual task consists of one instance only.

It is necessary to check for self-interference caused by a number of previously released instances. When the number of instances involved is known, the best-case response time is straight forward to find. Every arrival of τ_i becomes known because the task is periodic, hence a_{iq} is also known and this last instance finishes at the favourable instant f_{iq}. In essence, although the best-case phasing is given in Theorem 1, it is not fully specified because the number of instances of the studied task is unknown. This is the road map laid out for the remaining of this section, and it needs to be stated and proved more formally in order to reach Theorem 5, which is the highlight of the presentation.

A *hyperperiod* L of a task set is the least common multiple (LCM) of all task periods in the set. For simplicity of notation, it is assumed that the studied task τ_i has the lowest priority in the task set. The LCM is usually defined on the domain of integers, but if T_k is a rational number then L is well-defined for our purposes. In a periodic task set the arrival pattern repeats itself after one hyperperiod, but the pattern of release times does not because of release jitter. The *horizon* $h = L / T_i$, is the number of arrivals of τ_i during a hyperperiod. The horizon relates to how far back in time we must seek and resolve the problem of self-interference. For any given instance τ_{ip}, define

$$\sigma(p) = q - p + 1 \tag{15}$$

to identify the number of instances of τ_i that arrive in the interval $[a_{ip}, f_{iq})$. Note that σ is defined relative to q. For example, starting at $p = 1$ there are $h - 1 + 1$ instances arriving before the favourable instant. In Lemma 5 it is stated that any instance needed to be examined is in the set of *synchronization instances*, which consists of a bounded number of elements.

Lemma 5. *The set of synchronization instances.* Under the best-case phasing of τ_i, there must be at least one instance τ_{is}, a synchronization instance, which belongs to the same busy period as τ_{iq} and starts its execution the moment it arrives. The synchronization instance must belong to the set defined by

$$\Sigma = \{q - h + 1, \ldots, q - 1, q\} = \{\sigma(h), \ldots, \sigma(2), \sigma(1)\} \tag{16}$$

Proof. The existence of a τ_{is} is given directly by Lemma 1 and Lemma 2. The fact that the task set is schedulable means that τ_{is} must be found in the hyperperiod immediately preceding the favourable instant, i.e. $[f_{iq} - L, f_{iq}]$. This is obvious for a strictly periodic task set without release jitter, since the schedule repeats itself after one hyperperiod L. As before, it is only the time prior to f_{iq} that is significant. Under the best-case phasing condition, the release jitter is found to be zero for every instance of every task released prior to f_{iq}. The release of τ_j directly after the f_{iq} should on the other hand have a maximum of jitter, which means that the pattern of release times does not repeat itself, i.e. the schedule does not repeat itself, but the arrival pattern does. However, due to the best-case phasing and the schedulability condition, the execution pattern of the tasks in the interval $[f_{iq} - L, f_{iq}]$ has more idle time than any previous interval $[f_{iq} - kL, f_{iq} - (k - 1)L]$, for some $k = 1, 2, \ldots$. Thus, any instance τ_{is} that can possibly be a synchronization instance must belong to the set, $s \in \Sigma$.

■

The synchronization instance establishes a relation between the virtual task τ_v, cf. Definition 1, and the original task τ_i. The virtual task $\tau_{v\sigma(s)}$ has a priority equal to i, a zero release jitter, and a best-case execution time equal to $C^b_{v\sigma(s)} = \sigma(s)C^b_i$. Its best-case response time is denoted $R^b_{v\sigma(s)}$. The second position in the subindex refers to the number of instances of τ_i and not to a particular instance. Only one instance of a virtual task is needed, and its period is superfluous for the calculations. The concept of virtual task relates to the calculations derived in Section 4, whereas synchronization is related to the phasing of τ_i, cf. Section 3.

The *synchronization point* is the last point in time where it is possible to release a specific number of instances of τ_i to be fully executed before the favourable instant. The synchronization point is defined by

$$S_{\sigma(p)} = f_{iq} - R^b_{v\sigma(p)} \tag{17}$$

for some $p \in \Sigma$. S inherits the reversed time order from $\sigma(p)$. The synchronization point is related to the synchronization instant as Lemma 6 shows. Lemma 6 also gives an important relation between the arrivals of τ_i and the synchronization point.

Lemma 6. *The relation between arrivals and synchronization points.* Let $p, s \in \Sigma$. Under the best-case phasing, the condition $a_{ip} \le S_{\sigma(p)}, \forall p$, must be satisfied. Further, τ_{is} is a synchronization instance if and only if $a_{is} = S_{\sigma(s)}$.

Proof. The best-case phasing uniquely defines the arrival times of all instances τ_{ip}, $p \in \Sigma$.

It follows directly from Definition 1 and the definition of the synchronization point, that if an instance τ_{ip} arrives after its synchronization point $a_{ip} > S_{\sigma(p)}$, the execution of the $\sigma(p)$ instances of τ_i (with a total workload of $\sigma(p)C_i^b$) cannot be completed before f_{iq}. This contradicts the fact that the tasks arrive according to the best-case phasing and therefore $a_{ip} \le S_{\sigma(p)}$ must be true for all p.

If $a_{is} = S_{\sigma(s)}$ for an instance τ_{is} with $s \in \Sigma$, then the only way that τ_{iq} may complete exactly at f_{iq} is if the execution of τ_{is} starts immediately on arrival. According to Lemma 5, τ_{is} must be part of the same busy period as τ_{iq} and τ_{is} must be a synchronization instance. If on the other hand $a_{is} < S_{\sigma(s)}$ then, for τ_{iq} to complete at f_{iq}, τ_{is} must either start later than its arrival (due to interference), or must not belong to the same busy period as τ_{iq}; in neither case, τ_{is} is a synchronization instance.

■

A group of exactly p sequential instances of τ_i has to start executing at the latest at the synchronization point $S_{\sigma(p)}$ in order for the last instance in the group to finish before or at the favourable instant. The synchronisation point gives a bound on the latest time that τ_{is} can arrive for τ_{iq} to complete at f_{iq}. When there is no idle time between $S_{\sigma(p)}$ and the favourable instant f_{iq}, this yields the best-case response time for the virtual task.

For the best-case phasing, the periodic arrivals of the instances τ_{ip}, $\forall p$, are phased (or synchronized) relative to some synchronization point $S_{\sigma(s)}$. For this phasing, at least one synchronization point coincides with the arrival and release of a synchronization instance τ_{is}, i.e. there is a synchronization with the equality $a_{is} = S_{\sigma(s)}$, in agreement with Lemma 6. The arrival time of an arbitrary instance τ_{ip} under the synchronization to such an instance s is denoted $a_{ip}(s)$ and it is given by

$$a_{ip}(s) = S_{\sigma(s)} + (p - s)T_i \tag{18}$$

From this equation, $a_{is} = S_{\sigma(s)}$ results naturally as a special case for $p = s$. In Figure 5.1, an example of the best-case phasing of a task τ_i is shown when interfered by a single higher priority task τ_j. The task parameters are $T_i = 25$, $C_i^b = 11$, $T_j = 40$, $C_j^b = 20$ and $J_i = J_j = 0$. τ_{is} with the value $s = q - 1$, is a synchronization instance; this gives $\sigma(s) = 2$ using (15). The figure shows the resulting best-case response time of τ_i, $R_i^b = 17$, and the best-case response time of the virtual task $\tau_{v\sigma(s)}$, $R_{v\sigma(s)}^b = 42$.

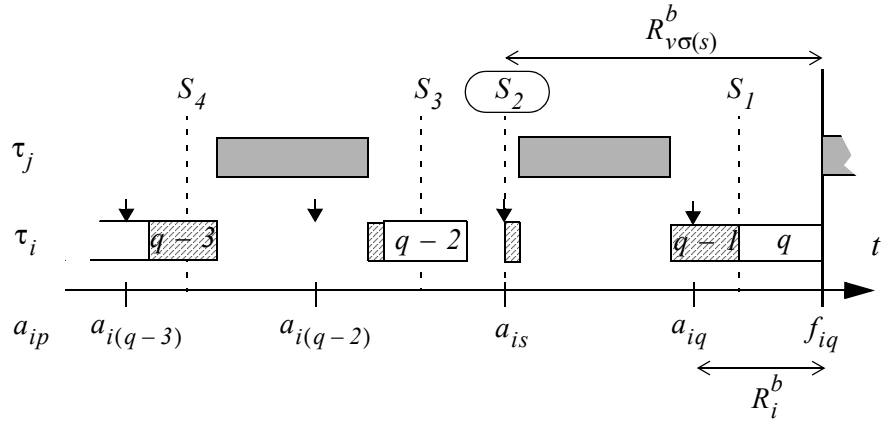


Figure 5.1 The best-case phasing of task τ_i when executed with one higher priority task τ_j. The instances τ_{ip} are tagged with the values $p = \ldots, q-3, q-2, q-1, q$, and every second instance has been marked for the purposes of visualisation only.

We have a best-case phasing scenario as described in Theorem 1, but s — or equivalently, the number of instances of τ_i interfering with each other, $\sigma(s)$ — is yet unknown to us, but we are now ready to describe how τ_i is phased relative to τ_j. Theorem 4 provides this complement to the best-case phasing by giving a sufficient condition on the synchronization.

Theorem 4. *The best-case synchronization.* The instances of τ_i arrive according to the best-case phasing if and only if there is a synchronization instance τ_{is}, synchronized at

$$a_{is} = S_{\sigma(s)} \tag{19}$$

for which a causality condition

$$a_{ip}(s) \le S_{\sigma(p)} \tag{20}$$

is satisfied $\forall p$ with $s, p \in \Sigma$.

Proof. The necessity of the statement follows directly from Lemma 6. It remains to prove that the statement is sufficient to uniquely specify the best-case phasing. We need to prove that if for some τ_{is}, both (19) and (20) are satisfied, then this is the best-case phasing of τ_{iq}. This will be shown by contradiction.

Assume that the phasing synchronized with τ_{is}, i.e. (19) is satisfied, is such that (20) is satisfied too, but this is not the best-case phasing. Then, by Lemma 5, there must be some other phasing of τ_i that is synchronized with some other instance τ_{it}, $t \in \Sigma$, which is a synchronisation instance in the best-case phasing of τ_i.

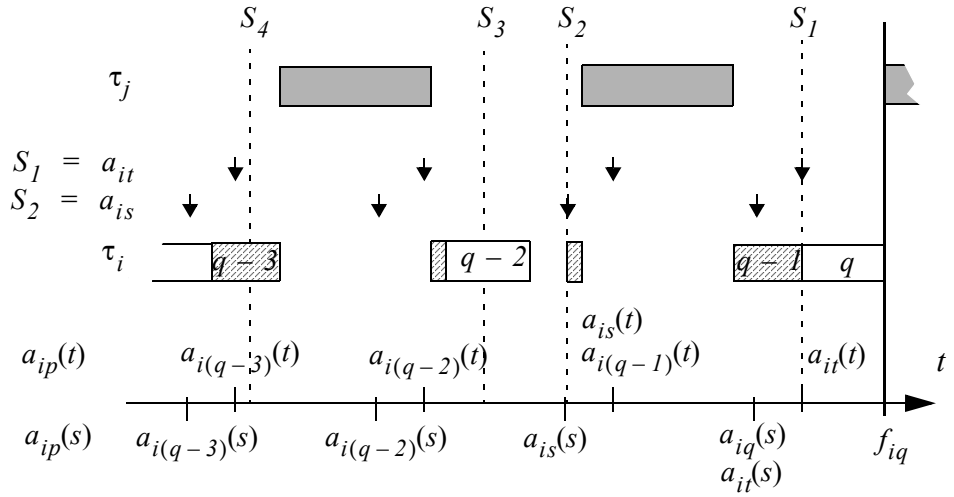


Figure 5.2 Arrival of τ_i synchronized to either $S_1 = a_{it}$ or $S_2 = a_{is}$, see the down pointing arrows. The causality condition $a_{ip}(t) \le S_{\sigma(p)}$ is not satisfied $\forall p$. Again, regard q as a value of index p. The indices of the synchronization instances have the values $s = q - 1$ and $t = q$ for the synchronization S_2 and S_1 respectively.

Since (20) is satisfied when τ_i is synchronized with τ_{is}, but the phasing given with τ_{it} is different, we find that $a_{it}(s) < S_{\sigma(t)}$, cf. Figure 5.2. Hence the new arrival time of τ_{it}, $a_{it}(t) = S_{\sigma(t)}$, is later than $a_{it}(s)$. Since the synchronization is made with τ_{it}, $a_{ip}(s) < a_{ip}(t)$ for

all instances p. This specifically holds for τ_{is} such that $a_{is}(s) < a_{is}(t)$. But since (19) holds it follows that $a_{is}(s) = S_{\sigma(s)} < a_{is}(t)$, and hence (20) does not hold when synchronized with τ_{it}, i.e. $a_{is}(t) \le S_{\sigma(s)}$ is false. This contradicts Lemma 6 and shows that the phasing with τ_{it} cannot be the best-case phasing and therefore the phasing with τ_{is} must be the best-case phasing.

■

Finally, it remains to find $\sigma(s)$, i.e. how many instances of τ_i that arrive in the interval $[a_{is}, f_{iq})$. The execution within this interval is well-defined for every task in the set because the phasing, period and best-case execution time are known for every task. Theorem 5 below summarizes the calculation of the best-case response time. First, a synchronization instance needs to be searched for. When a synchronization instance is given, R_i^b is straight forward to calculate.

Theorem 5. *The best-case response time*. A task set is employed according to the best-case phasing in Theorem 1. Let $p, q \in \{1, 2, \ldots, LCM(\tau_i)/T_i\}$. The best-case response time of the task τ_i is given by

$$\begin{aligned} R_i^b &= \min_q w_i(q) - (q-1)T_i \\ \text{s.t.} \quad & w_i(p) \le w_i(q) - (q-p)T_i \qquad \forall p \end{aligned} \tag{21}$$

For each p or q the recurrence equation

$$w_i(s) = sC_i^b + \sum_{j \in hp(i)} \left\lceil \frac{w_i(s) - J_j - T_j}{T_j} \right\rceil_0 \cdot C_j^b \tag{22}$$

is applied starting from an initial value equal to the worst-case response time, $w_i^0(s) = R_{i\sigma(s)}^w$. $R_{i\sigma(s)}^w$ is given by (1) using a smallest execution time of $C_i^w = sC_i^b$.

Proof. It is assumed that the hyperperiod is well defined. To prove the objective function in (21), we note that the best-case response time of the virtual task $\tau_{v\sigma(s)}$ corresponds to the interval $[a_{is}, a_{iq}]$ followed by $[a_{iq}, f_{iq}]$. The former has the length $(\sigma(s)-1)T_i$ and the latter has the length R_i^b. In Theorem 1 it is stated that the instance with the shortest response time is the one finishing in f_{iq}. As a result, given the best-case response time $R_{v\sigma(s)}^b$ of the virtual task $\tau_{v\sigma(s)}$ this yields $R_i^b = R_{v\sigma(s)}^b - (\sigma(s)-1)T_i$. The condition on the objective function in (21) is given by Theorem 4. Equation (22) is derived in Theorem 2 and the convergence is proved by Lemma 3 with the condition that the initial condition is correct, $w_i^0 \ge w_i(s)$. It is also clear from (1) that $R_{i\sigma(s)}^w \ge w_i(s)$ is always true for a worst-case execution time of $C_i^w \ge sC_i^b$.

■

Remark. Another possible initial value is $w_i^0(s) = sT_i$ because $sT_i \ge w_i(s)$.
Remark. If $R_{i\sigma(s)}^w \le T_i$ then $q = 1$ and Theorem 3 can be applied directly.

If any two different q render the same phasing, thus gives the same R_i^b, any of these q can be chosen. It is not the shortest virtual task that is of interest. Compare the expressions for the best-case response time (21) and the worst-case response time (1). In both cases a search has to be conducted. The condition in Theorem 5 is more complicated, since the start of a virtual task has to be searched for, and the possible self-interference by previous tasks must be resolved. In the worst-case phasing scenario, all tasks are released at the critical instant which marks the start of a busy period, thus no instance of the studied task previous to the critical instant inter-

feres with itself. It is well-known that the first instance in the busy period starting at the critical instant does not have to experience the worst-case response. On the contrary, in the best-case phasing scenario, the last instance will have the best-case response.

6 Pseudo code and an example

To clarify the theory presented in the previous sections and to help increase the understanding from an engineering perspective, we will here give examples of pseudo code, a concrete numerical example and finally also a numerical investigation based on randomly generated task sets. This will give an idea of how necessary it is to calculate the exact best-case response time, and it will also give a hint of the resulting error if the exact calculation is ignored.

The detailed pseudo code of the function `bcrt()` in Table 6.1 calculates R^b for a task i, given a task set τ and an execution time C_i. It is a direct implementation of the recurrence equation (3) in 4, which is applicable when there is no self-interference.

Table 6.1 A piece of pseudo code of a function `bcrt()` for computing R^b if there is no self-interference.

```
/* calc best-case response time */
function bcrt(tau,Ci)
initialize R_best
repeat
     R_best_old:= R_best
     w:= 0
     for j:= 1 to i-1
          temp:= ceil((R_best_old-tau[j].J-tau[j].T)/tau[j].T)
          w:= w + max(0,temp*tau[j].C)
     end
     R_best:= Ci + w
until R_best equals R_best_old
```

For an off-line use, the calculation cost is a minor problem: the for-loop scales with the number of tasks and the condition for terminating the repeat-loop is reached monotonically. The condition of the repeat-loop is guaranteed to be satisfied according to the theory presented, and it is usually reached within a few repetitions. The initial condition of the variable `R_best` holding the result, needs to be set appropriately according to Theorem 3.

The pseudo code in Table 6.2 can be used to calculate R^b for the case of an arbitrarily long worst-case response time, and $R^w \leq T$ in Table 6.1 then becomes a special case. The pseudo code is based on the theory presented in 5. For simplicity, the tasks in the set are assumed to have integer valued periods, which means that the variable `horizon` also is an integer. In the first part of the pseudo code, the synchronization points which correspond to virtual tasks, cf. Lemma 5, are calculated by calling the function `bcrt()` in Table 6.1. A synchronization point is calculated as a response time of a virtual task, i.e. a time interval relative to the favourable instant which is a natural origin.

Table 6.2 A piece of pseudo code for computing R^b when self-interference may occur.

```
/* 1. calc synchronization points */
horizon:= least_common_multiple(tau)/Ti
for s:= 1 to horizon
     synch_points(s):= bcrt(tau,s*Ci)
end

/* 2. check synchronization point */
for s:= horizon downto 1
     for p:= 1 to horizon
          arrival_times(p):= even_spaces(synch_points(s),p,Ti)
     end
     if (for every p:= 1 to horizon
          arrival_times(p) greater or equal to synch_points(p)) then
          feasible_synch_point:= synch_points(s)
     end
end

/* 3. calc R_best */
R_best:= remainder(feasible_synch_point/Ti)
if R_best equals 0 then
     R_best:= Ti
end
```

In the middle part, the correct synchronization point is sorted out. We want to find the smallest s for which Theorem 4 holds, and it is absolutely necessary to go one hyperperiod backwards from the favourable instant to ensure that interference of previous instances are taken into account. The function `even_spaces()` calculates the equidistant arrival times currently synchronized at `synch_points()`. The condition of the first if-clause becomes opposite to Theorem 4, which gives the corresponding condition in absolute time. In the third part, the best-case response time is given directly by Theorem 5.

The computations can be performed in polynomial time, and the variable `horizon` determines to a large extent the computational cost. If the periods within the task set are related to each other as prime numbers or if the quotient between two periods is high, the computational cost increases substantially using this straight forward algorithm.

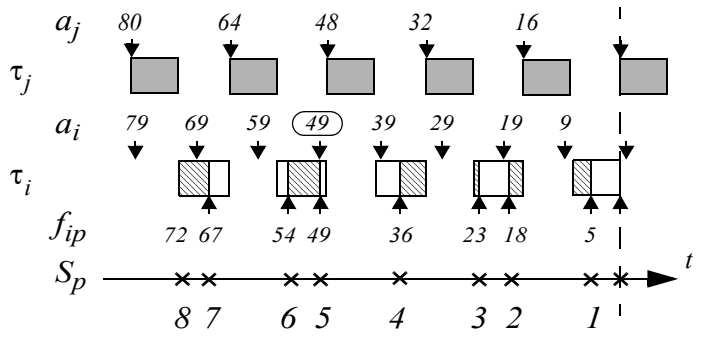


Figure 6.1 Example task set τ_i and τ_j gives $R_i^b = 9$. The numbers in the figure tell how many time units or instances there are left to the favourable instant. For the ease of reading, every second instance of τ_i has been marked.

Here follows a small numerical example with two tasks, for which a solution can be found by visual inspection of Figure 6.1. The parameters are $T_j = 16$, $C_j = 8$ and $T_i = 10$, $C_i = 5$ time units, which give a utilization of $U = 1$, a worst-case response time of $R_i^w = 13 > T_i$ and a hyperperiod of 80, further $J_i = J_j = 0$.

The goal is to shift the 8 equidistantly distributed arrival times a_i to the right as much as possible, since the best-case response time is calculated as $min(a_{ip} - f_{ip}) = min(a_{i1})$. We can see that the 8 instances need all the spare time left by the high priority task, thus the first instance must start no later than 72 time units before the favourable instant. This is the response time of a virtual task with a period of 80 (the hyperperiod) and an execution time of $8C_i$. Further, 7 instances need at least 67 time units. The function `bcrt()` in Table 6.1 is used to calculate all these synchronization points S_p, for $p = 8, 7, \ldots, 1$. In this example, every single instance finishes at a synchronization point because there can be no idle time in the schedule. Now, assume that the evenly distributed vector a_i is shifted. It turns out that at least one arrival time which coincides with a synchronization point $a_{ip} = S_p$ can be found, where the arrivals of a_{ip} cannot come later than S_p for every p. At the point $S_2 = 18$ the condition is not satisfied, since for e.g. $p = 5$ the vector a_i is clearly shifted too much to the right. At the point $S_5 = 49$, every arrival comes before or at its corresponding synchronization point. This means that the last instance must arrive at $a_{i1} = 9$, and the best-case response time thus becomes $R_i^b = 9$.

Finally, we would like to share the results of a straight forward statistical analysis of the best-case response time calculation. The input to the analysis consists of randomly generated task sets. This kind of data gives a broad and varying input space which helps elucidate different perspectives of the best-case calculations. However, the input data might not represent real cases. For a real task set the task parameters are probably related to each other and not totally in random. The theory presented in the previous sections, gives a calculation of the best-case with the interference and self-interference taken into account. It is interesting to compare this approach with the most trivial alternative idea, which is to equate the best-case response time with the best-case execution time. Another approximation which might be tempting, is to neglect the self-interference which adds to the total interference. The validity and applicability of these two approximations will be investigated below. The effect of a rate-monotonic priority assignment, compared to an arbitrary priority assignment, will also be shown.

A task set has been generated in the following way. A random number of tasks, between 5 and 25, are assigned random execution times regardless of the priority. The execution time is bounded within $[1, 10]$ time units, i.e. two tasks differ in their execution demand up to a factor of 10. The periods are drawn in random from the set $T = \{1, 2, 3, 5, 12, 20\}$ of positive integers, which means that the hyperperiod becomes relatively short. The relation between priority and period is random. All random variables used have uniform distributions. The release jitter is zero for every task. The execution times in the task set are scaled by a factor such that a random target utilization is achieved. The relation between the shortest and longest execution time remains after the scaling and the period times are also unaffected. For each task set the response time of the low priority task has been calculated and the result is summarized in Table 6.3, Figure 6.2 and Figure 6.3. In addition, Table 6.4 and Figure 6.4 show the result if the priorities instead are set in a rate monotonic fashion.

Table 6.3 The case with an arbitrary priority assignment. The number of tasks that are subject to interference and self-interference. The maximum and average of the quotient R_i^b / C_i^b.

Utilization range	Inter-ference	Self-inter-ference	Maximum quotient	Average quotient	Task sets
0.01 — 0.30	0	0	1.0	1.00	5000
0.30 — 0.50	0	0	1.0	1.00	5000
0.50 — 0.70	2	0	1.9	1.00	5000
0.70 — 0.80	18	0	2.6	1.00	5000
0.80 — 0.90	62	0	4.4	1.02	5000
0.90 — 0.95	114	2	8.7	1.04	5000
0.95 — 0.99	270	29	21.9	1.19	5000
0.01 — 0.99	**466**	**31**	**21.9**	**1.04**	**35000**

Table 6.3 makes it clear that with a higher utilization, the need for a better estimate increases. The calculation has been grouped according to the utilization; notice the uneven spread in the left column. For each utilization range, 5000 task sets have been generated and the calculated best-case response time has then been divided by its execution time, i.e. R_i^b / C_i^b. The second column shows the number of times the condition $R_i^b \neq C_i^b$ occurred, that is, how often τ_{iq} was interfered by τ_j and self-interfered by τ_{ip}, $p < q$. The third column shows the number of times self-interference occurred. This occurred in about 0.1% of the cases, but when the utilization was less than about 0.9 it didn't show up at all. It is clear that the need to handle self-interference is in practice necessary only for a very high utilization. The maximum recorded quotient among the samples shows that the assumption $R_i^b = C_i^b$ can give a very conservative value of the bound on the best-case response time. For a utilization close to one, the best-case response time becomes underestimated by about one order of magnitude. The occurrence of zero in columns two and three does not necessarily imply that the theory is superfluous for that particular range.

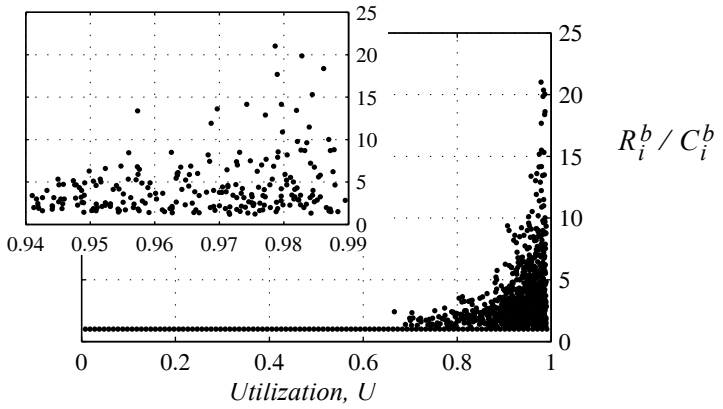


Figure 6.2 The quotient between best-case response time and best-case execution time plotted as a function of the utilization.

In Figure 6.2 the quotient between best-case response time and best-case execution time R_i^b / C_i^b is plotted as a function of the utilization. The plot is based on the data in Table 6.3. It shows a diversity of response times, especially when the utilization is high. The case $R_i^b / C_i^b = 1$ has been left out from the plot for clarity, because it can be represented by a horizontal line.

The actual response time depends in an interwoven way on the task periods and execution times for all tasks in the set. A large difference between the shortest and longest period of two tasks in a set, seems to lead to an increased interference, which implies a longer best-case response time. Conversely, if two tasks have approximately the same period, they are less likely to interfere with each other under the best-case phasing scenario. If the studied low priority task has a short execution time, the chances are higher that it will escape interference compared to the opposite case. By combining these two observations, and defining the partial utilization as $U_k = C_k^b / T_k$, the relative best-case response time R_i^b / C_i^b is plotted in Figure 6.3 as a function of the partial utilization U_i.

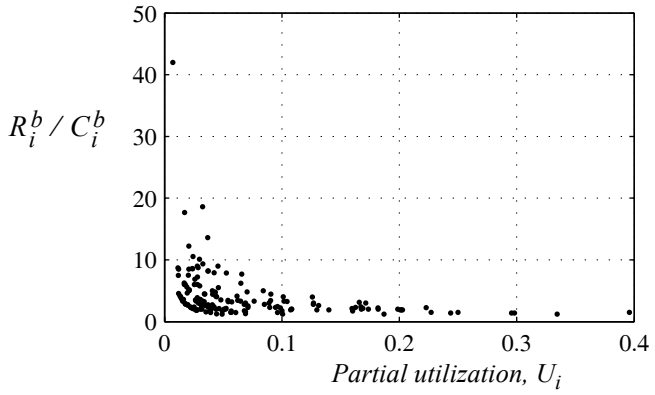


Figure 6.3 The quotient R_i^b / C_i^b as a function of the partial utilization.

For a low partial utilization and also for a low total utilization, the simple and conservative rule of thumb $R_i^b = C_i^b$ might be a reasonable approximation, i.e. it is a good lower bound. However, this can not be taken as a general rule as seen in Figure 6.3. The case $R_i^b / C_i^b = 1$ has been left out from the plot for clarity, because it can be represented by a horizontal line; the centre of mass of the scattered points is therefore deceptive in the figure. No matter how short execution time the low priority task has, a high priority task with a sufficiently short period can still interfere with it, which means that $R_i^b \neq C_i^b$.

Another dimension worth studying is the choice of priority. The rate monotonic priority assignment is a well-known and recognized method to find relative priorities for a task set. A reason might be the fact that a simple schedulability test can be based on an upper bound on the allowable utilization. If a feasible priority assignment exists for some task set, the rate monotonic assignment is feasible for that task set, cf. Liu and Layland (1973). We will here use task sets which violate the utilization levels, but it is nevertheless interesting to study the best-case results.

Table 6.4 The case with rate monotonic priority assignment. The same task sets as in the previous study have been reused, but now the task with the longest period (lowest priority) is studied.

Utilization range	Inter-ference	Self-inter-ference	Maximum quotient	Average quotient	Task sets
0.01 — 0.30	0	0	1.0	1.00	5000
0.30 — 0.50	0	0	1.0	1.00	5000
0.50 — 0.70	9	0	2.0	1.00	5000
0.70 — 0.80	30	0	3.0	1.00	5000
0.80 — 0.90	97	0	6.2	1.03	5000
0.90 — 0.95	218	1	10.3	1.10	5000
0.95 — 0.99	388	30	34.9	1.50	5000
0.01 — 0.99	**742**	**31**	**34.9**	**1.09**	**35000**

The data points in the 5th column of Table 6.3 and Table 6.4 have been drawn in Figure 6.4 to show a comparison of a rate monotonic priority assignment versus an arbitrary priority assignment. The average value of the quotient R_i^b / C_i^b of all task sets is plotted versus the utilization.

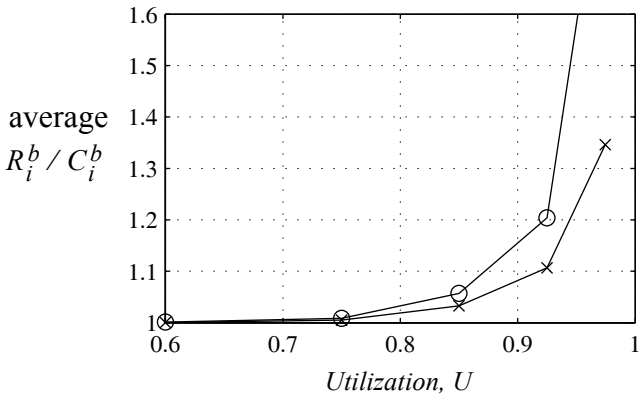


Figure 6.4 The average relative best-case response time R_i^b / C_i^b for rate monotonic (circles) and arbitrary (crosses) priority assignment.

The average value of the quotient R_i^b / C_i^b is clearly higher for rate monotonic compared to the case with arbitrary priority assignment. The difference gets more accentuated when the utilization is high. R_i^b is more likely to be longer if tasks with short periods are given high priorities. This is best understood by considering the fundamental model, the best-case phasing scenario.

7 Discussion

A few aspects, extensions and applications of the theory are worth commenting, especially when applying the theory to circumstances which do not adhere to the premises of the task model in Section 2. Luckily, in many cases where conjectures are made, the exact analysis just turns to a correct lower bound.

The best-case response time can occur for other phasings relative to the studied task than the best-case phasing defined and proved in Theorem 1. There may be other choices of phasings that will give an equally short response time, but none will be shorter. Thus, the best-case phasing is in general not *unique*, but uniqueness is not really of interest.

The most important model assumption is the *independence* of tasks. This assumption simplifies the analysis to a large extent, because it enables the arbitrary phasing of tasks to eventually lead to the best-case phasing scenario. This is equally true for the worst-case calculations, but the violation leads to different problems. Relations between tasks is typically expressed in the form of a task graph — and at the same time the end-to-end response time gets more interesting than the response time for a single task. A phase between two tasks specified at design time, can be used to translate a dependence. The introduction of offsets is of course a violation to the prerequisite of arbitrary phasing.

Consider the following example of *precedence relation*. A high priority task τ_j is constrained to follow after a low priority task τ_k, and typically these tasks have the same period and can be viewed as a task chain. A task τ_i with an intermediate priority, and with an execution time longer than the period of the task chain, starts before or during the execution of τ_k. This combination will block τ_j from interfering with τ_i. The actual best-case response time of τ_i will be shorter than the result of an analysis assuming independent tasks.

Another common dependence is the *mutual exclusion* of tasks, where a low priority task currently executing inside a protected critical section, can block a high priority task from entering this section, cf. the example above with the three tasks τ_j, τ_i and τ_k. Again, it is impossible to disregard from this blocking effect in the analysis. A *priority ceiling protocol* is often applied in this case to counteract a prolonged blocking of the high priority task. This will to a large extent fix the problem with dependencies and pre-emption renders interference as expected. For *non-preemptive* scheduling the best-case response time is simply $R^b = C^b$, because once a low priority task has started it can not be interrupted.

The analysis requires the task set to be periodic, but in reality the use of interrupts is common. The time interval between two interrupts should determine if it is necessary to incorporate the interrupt task as a *sporadic task* in the analysis.

A bound on the best-case execution time, BCET, is of course a prerequisite for calculating the best-case response time. The execution time of a given piece of source code depends on the hardware architecture (e.g. caches and pipelines), the software (e.g. the branching) and the translation from source code to object code (e.g. the compiler). Over the past years, much research effort has been devoted to the problem of estimating the worst-case execution time, WCET. It is plausible that BCET and WCET are equally complex to estimate. A conservative but safe bound on BCET leads of course to a conservative calculation of R^b.

In *distributed scheduling* and in the scheduling of *multi-processors*, a holistic view is often taken, i.e. processing elements and communication channels are scheduled simultaneously. The problem here is primarily to find the end-to-end response time and response jitter for a task chain. This is a natural and tempting direction for extending the response time analysis. Care must of course be exercised if an iterative procedure is used where the phasing of tasks within and between task chains violates the assumptions leading to the best-case phasing scenario.

Context switch time is the time it takes for the dispatcher to stop one task, save its registers, reload another task's registers and start it, i.e. the book-keeping work typical for a dispatcher. Context switching has not been added to the analysis, but it should be possible if required to include it in a similar way as Katcher et al. (1993) have done for the worst-case calculations.

8 Conclusions

Given a periodic task set feasibly scheduled using preemptive fixed priority scheduling on a uniprocessor, and with each independent task having a lower bound on the execution time, the exact best-case response time of a task can be calculated by the methods derived in this paper. The theory allows for release jitter and arbitrarily long response times. The dualism between best-case and worst-case analysis forms an appealing symmetry and it fills out a missing piece in the response time theory. The dualism has made the ideas worth to explore, but the real motive for looking at the best-case response time comes from a need to analyse jitter in fixed priority systems. A feedback controller belongs to a class of real-time systems for which the delay jitter in the response of tasks also is interesting, besides meeting hard deadlines. The theory developed here is a step in the right direction to model and estimate the influences, from a control theoretical perspective, resulting from time-varying response times. The best-case response time is also useful in other situations, for example in admission control.

The best-case response time is calculated using a recurrence equation starting from an initially high value, decreasing iteratively to finally converge at the exact value. The best-case phasing of tasks, which is the starting-point of the analysis, is very similar to the familiar worst-case phasing called critical instant. Because of this, it does not come as a surprise that the resulting recurrence equation resembles the well-known equation found in worst-case analysis. When the worst-case response time is longer than the task period, some extra work is needed. Under this condition, preceding instances of the studied task might still be running when an instance of the studied task arrives and is released. It is necessary to go backwards in time to search for a busy period leading to a correct synchronization point, as close to the favourable instant as possible. Once the synchronization point is found, the best-case response time is easily calculated. In order to clarify the theory presented, both pseudo code and an example has been provided.

A numerical investigation of the relevance of a theory for an exact best-case response time, reveals that using the execution time as a crude estimate of the best-case response time, can give a large error especially for a high utilization.

9 Acknowledgements

This work has partially been supported by VINNOVA, the Swedish agency for innovation systems, through the DICOSMOS project and partially supported by SSF, the Swedish strategic research initiative, through the PICADOR project. We would also like to thank Prof. Jan Wikander and Prof. Martin Törngren.

10 References

Audsley N., Burns A., Tindell K., Richardson M. and Wellings A., “Applying new scheduling theory to static priority preemptive scheduling”, Software Engineering Journal, 8(5):284-292, 1993.

Henderson W., Kendall D. and Robson A., “Improving the accuracy of scheduling analysis applied to distributed systems”, Intl. Journal of Real-Time Systems, Kluwer, 20(1):5-25, 2001.

Joseph M. and Pandya P., “Finding response times in a real-time system“, The Computer Journal, Vol. 29, No. 5, 1986.

Katcher D.I., Arakawa H., Strosnider J.K., “Engineering and analysis of fixed priority schedulers”, IEEE Transactions on Software Engineering, Vol. 19-9, p. 920-934, September 1993.

Klein M., Ralya T., Pollack B., Obenza R. and González Harbour M., “A practitioners handbook for real-time systems analysis”, Kluwer Academic Publishers, 1993.

Lehoczky J.P., “Fixed priority scheduling of periodic task sets with arbitrary deadlines”, Proc. of the 11th IEEE Real-Time Systems Symposium, pp. 201-209, 1990.

Liu C.L. and Layland J.W., “Scheduling algorithms for multiprogramming in a hard real-time environment”, Journal of the ACM, 20(1):46-61, 1973.

Palencia Gutiérrez J.C., Gutiérrez García J.J. and González Harbour M., “Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems”, Proc. of the tenth Euromicro Workshop on Real-Time Systems, Berlin, pp. 35-44, 1998.

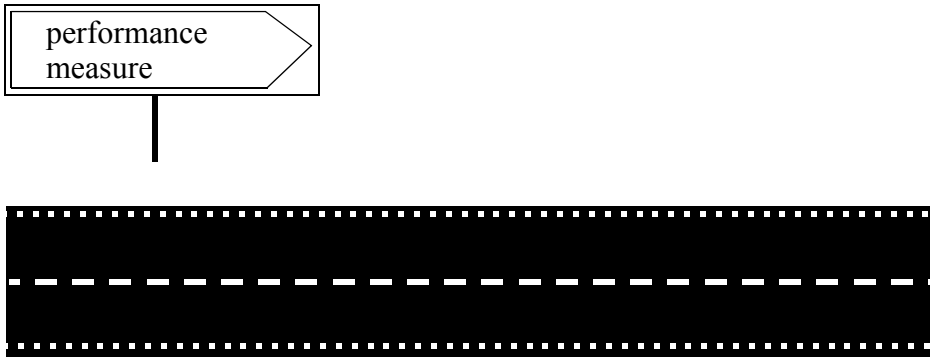
Redell O. and Sanfridson M., “Exact best-case response time analysis of fixed priority scheduled tasks“, Proc. of the 14th Euromicro Workshop on Real-Time Systems, Vienna, 2002.

Tindell K. and Clark J., “Holistic schedulability analysis for distributed hard real-time systems”, Microprocessing & Microprogramming, 50(2-3):117-134, 1994.

Törngren M., “Fundamentals of implementing real-time control applications in distributed computer systems”, Journal of Real-time Systems, Kluwer, 14:219-250, 1998.

Paper B

Martin Sanfridson, “Discretization of loss function for a control loop having time-varying period and control delay”, Technical report, ISRN KTH/MMK/R--03/2--SE, Mechatronics Lab, KTH, May 2004.



Mechatronics Lab
Department of Machine Design
Royal Institute of Technology
S-100 44 Stockholm, Sweden

TRITA-MMK 2003:2
ISSN 1400-1179
ISRN KTH/MMK/R--03/2--SE

Document type
Technical Report

Date
2004-05-07

Author(s)
Martin Sanfridson
mis@md.kth.se

Supervisor(s)
Prof. Jan Wikander
Prof. Martin Törngren

Title
Discretization of loss function for a control loop having time-varying period and control delay

Sponsor(s)
Vinnova and SSF

Abstract

This report describes the derivation of a quadratic control performance measure for the so called timing properties. The timing properties can be categorized into: the sampling period including time-variations in the period, the control delay including time-variations in the delay and finally the transient error. They define an interface between control design and controller implementation, with respect to for example the partitioning, allocation and real-time scheduling of control tasks.

At design time, it is essential for the control engineer and the computer engineer to know how the timing properties affect the stability, robustness and performance of the control loop, in order to make trade-off. A control measure can also be used at run-time by a quality-of-service broker to negotiate an optimal resource distribution. By measuring the timing behaviour of the underlying computer system, the performance is estimated based on a model of the process. This supervisor can be placed on top of a scalable real-time system, which for example admits reallocation and rescheduling of tasks, with the idea that the system should adapt to internal and external changes.

The process and the controller are assumed to be linear and time invariant. The process and the loss function are given in the continuous time domain. The loss function, which is frequently used in optimal control, describes the performance at an infinite time horizon. The controller is allowed to have internal states and it does not have to be optimal. The process and the loss function are discretized and the matrices of the closed loop are formed. The zero-order-hold discretization allows multiple arrivals of a control signal between two sampling instants, and is an extension of the text book procedure.

A Markov chain is used to model sampling period jitter and delay jitter. This is necessary when the delay is allowed to be longer than the period. The Markov chain is set up to preserve the causal ordering of delays. Two techniques can be used to assess performance and stability. The first is a direct calculation of the performance and the second is based on linear matrix inequalities.

The report gives examples of situations where the estimation is applied to a range of typical processes and controllers. This will hopefully give the reader an intuitive understanding of the impact of the timing properties.

Keywords
Feedback control, real-time scheduling, control performance, robustness, time-varying delay, sampled-data control, sampling period, jitter, jump linear system, Markov chain, LMI.

Language
English

Notation

Table 0.1 Notation used throughout this report.

A_p, B_p, C_p, D_p	State space representation of a continuous time dynamic system. The respective sizes of the matrices are: $[np \times np]$, $[np \times nu]$, $[ny \times np]$ and $[ny \times nu]$.
$\Phi_p, \Gamma_{pa}, \Gamma_{pb}$ C_p, M_p	State space representation of a discrete time process. The sizes are: $[np \times np]$, $[np \times nu]$, $[np \times (nd \cdot nu)]$, $[ny \times np]$ and $[nz \times np]$ respectively.
$\Phi_c, \Gamma_c, \Lambda_c$ C_c, D_c, E_c	Discrete time controller. Sizes: $[nc \times nc]$, $[nc \times nz]$, $[nc \times (nd \cdot nu)]$, $[nu \times nc]$, $[nu \times nz]$ and $[nu \times (nd \cdot nu)]$ resp.
Φ_{cl}, Γ_{cl}	Closed loop system matrix and input matrix in discrete time, sizes: $[na \times na]$ and $[na \times ne]$ respectively.
E	Expected value operator.
i, j, r	State of the Markov chain, $\{1, 2, \ldots, nr\}$.
$\bar{J}, J, J_v, J_w$	Loss, sampling dependent loss due to noise v and w respectively, $J \in \Re$.
h	Denotes period, $h \in \Re$.
P	Transition probability matrix of Markov chain. Also denotes process.
$Prob$	Probability.
π, π^{∞}	State probability vector, especially stationary, of a Markov chain.
$\bar{Q}, Q, Q_{cl}$	Weight matrices (non-negative definite) for setting up a continuous time (or discrete time) quadratic cost function. Sizes $[nq \times nq]$ and $[na \times na]$ respectively.
R	Noise variance.
S	A notion of covariance.
T	Time horizon in time units, $T \in \Re$. As a superscript, and only as a superscript, T denotes the matrix transpose.
τ	Any delay, $\tau \in \Re$ usually takes a subindex.
θ	Periodicity $\theta \in \{1, 2, 3, \ldots\}$.
tr	The matrix trace operator, the sum of the main diagonal.
$U(t_k)$	Vector of old outputs. Size $[(md \cdot nu) \times 1]$.
$u(t), u(t_k)$	Control output, time continuous or time discrete. Sizes $[nu \times 1]$ and $[nu \times 1]$ respectively.
$v(t), v(t_{k+1}, t_k)$ $w(t_k), e(t_k)$	State noise and measurement noise. Sizes $[np \times 1]$, $[np \times 1]$, $[ny \times 1]$ and $[(ny + np) \times 1]$.
$x_p(t)\, x_c(t_k), x(t_k)$	The state vector of a process, a controller and closed loop. Sizes $[np \times 1]$, $[nc \times 1]$ and $[na \times 1]$ respectively.
$y(t), z(t_k)$	Process output, measured value. Size $[ny \times 1]$ and $[nz \times 1]$.

A brief annotation of notational conveniences in the report:

- A bar over-lining a constant indicates continuous time. The discrete time equivalent is typically without the bar. A variable usually has an argument indicating what domain it belongs to.
- A subscript is sometimes used as an alternative to a variable within parenthesis, in order to facilitate the reading.
- The subscript c means controller.
- The subscript p means process.
- The subscript cl means closed loop. This subscript is sometimes omitted.
- The subscript 0 means nominal value, or initial value.
- The superscript ∞ denotes stationarity.
- $A_{row \times col}$ is a matrix A of the size $[row \times col]$.
- A continuous time variable typically takes the argument t.
- A discrete time variable typically takes the argument k or t_k.
- k denotes discrete time instances, with index $k = \ldots, -1, 0, 1, \ldots$ or $k = 0, 1, 2, \ldots$.
- t denotes continuous time, and a specific time instant is written t_k for some instance k.
- The notion of time t_k is used as a substitute for the notion of event k in various arguments.
- The sampling period is defined by $h_k = t_{k+1} - t_k$.
- All constants, variables, vectors and matrices are real-valued except indices, which domains should be properly defined in the text.

Table 0.2 The sizes of some sets, matrices and vectors.

np	Number of states of the continuous time process.
nu	Number of inputs to the process.
ny	Number of outputs from the process.
nz	Number of measured outputs from the process used by the controller.
nc	Number of discrete states of the controller.
nd	Number of additional states necessary to describe the delay.
md	Maximal number of additional states necessary to describe the maximum delay.
ng	Number of changes in the control signal during the given sampling period.
cd	Number of additional states due to the control law.
na	Number of elements of the closed loop state vector, $na = np + nu \cdot md + nc$.
ne	Number of elements of the closed loop noise vector, $ne = np + nz$.
nr	Number of states of a Markov chain.
nq	Size of the square discrete weight matrix, $nq = np + nu \cdot nd + nu$.

1 Introduction

In this report the impact of the *timing properties* of a control loop are analyzed. The timing properties of interest are: *the sampling period, constant control delay, sampling period jitter, control delay jitter,* and *transient error*, see Sanfridson (2000a). The timing properties are derived from the underlying digital computer system, on which the controller is implemented. The timing properties arise because of for example scheduling, contention in communication channels, and time-varying execution time. In a fault free analog computer the timing properties do not exist. The assumption of a fault free analog computer is similar to the traditional assumption of continuous time control analysis and synthesis.

The analysis is based on a continuous time loss function, commonly used in optimal control and related to the H_2-norm. The analysis is complicated by the fact that the system is time-varying. The sampled-data system is driven by state noise or measurement noise. The performance measure characterizes the steady state behaviour.

The derivation in this report has a step wise structure, which presents the well-known special cases first, such as analysis in case of a constant period, before moving on to the general problem. An ambition has also been to make the report rather self-contained. It is detailed, however not in the sense that formal theorems are stated. These arrangements aim to lower the threshold, and will make the material easier to penetrate. The theoretical sections of the report are followed by two sections with examples of typical or difficult systems in different scenarios. A brief explanation of the performance measure meant as an introduction follows in Section 1.3. Afterwards, some related work is commented.

1.1 Purpose and aims

The control performance measure is a tool for analysing timing properties of a single sampled-data system. It can be used at design time by a system designer or control engineer as a complement to traditional analysis. It can also be used at run-time e.g. by a quality-of-service negotiator, to automatically accommodate changes in the environment, by rendering a feasible or optimal distribution of the available hardware resources. Below is a list of requirements of a performance metric.

- It must be possible to analyse a time delay longer than the period. This is necessary for transient errors.
- All kinds of linear controllers must be possible to investigate, including non-optimal and those with internal states.
- The same measure should be applicable for all timing properties. This gives a possibility to investigate a trade-off between two different timing properties.

1.2 Methods and tools

First, the process and a time continuous quadratic loss function are discretized and this is described in detail. The theory is based on a weighted quadratic loss functions which is commonplace in optimal control theory. Next, the actual calculation of the measure is described. Besides giving a detailed theoretical description the aim has also been to give a good notion of the impact of the timing properties. A number of examples show how the measure can be used.

Validation has been performed by simulation using the well-known tool Simulink. The performance measure is easily calculated using a computer tool, and even for a simple system it is necessary to resort to numerical computations — hand calculation quickly becomes inconvenient. For the ambition of showing the impact of the timing properties in a more general setup, simulation is a good aid. The report is however directed towards an analytical approach.

The theory presented here is for linear systems only. Both the plant and the controller must be linear time invariant (LTI). It is the time-varying period and delay that make the system time-varying. The theoretical model presented here has not been developed for multirate systems.

1.3 A short summary of the theory

The inputs into the analysis are:

- a continuous LTI process on state space form,
- a discrete time LTI controller on state space form,
- a description of the delay and the sampling period,
- a set of weights, and optionally
- the variance of the continuous time state noise and discrete time measurement noise.

The control performance is in this report measured by the well-known integrated quadratic loss function,

$$\bar{J} = E \lim_{T \to \infty} \frac{1}{T} \int_0^T [x_p^T \bar{Q}_{11} x_p + x_p^T \bar{Q}_{12} u + u^T \bar{Q}_{12}^T x_p + u^T \bar{Q}_{22} u] dt \quad (1)$$

i.e. the expected weighted (with $\bar{Q}_{11}$, $\bar{Q}_{12}$, $\bar{Q}_{21} = \bar{Q}_{12}^T$, $\bar{Q}_{22}$) variance of the state vector x_p of the process, and the control signal vector u over a time horizon T. The result is a scalar value $\bar{J}$ which characterizes the average deterioration or cost due to the variance of the closed loop. A higher value denotes a higher weighted variance and thus worse performance. Loosely speaking, an infinitely high value means that the closed loop system is unstable.

The difference in performance between two values or the direction of a change in the measure says more than the absolute value. This enables a comparison between two sets of timing properties or controllers etc. The performance measure as such does not give the control engineer much of a feeling how well the system performs in reality. It gives a concrete answer of what is better or worse. As it is of interest to relate the performance measure with specifications of the system's performance given in absolute values, other techniques must be applied simultaneously, e.g. time plots.

Starting from the continuous time quadratic loss function above, an equivalent discrete time expression is formed. The continuous process is zero-order-hold sampled for some constant period h and constant delay τ. The time continuous state noise is also sampled; this will add intersample behaviour of the state noise to the loss. The discretized process and controller are connected to form a closed loop. In order to describe delays, it is necessary to augment the state vector by old control signals. The infinite dimensional delayed LTI process is converted and augmented to a finite dimensional LTI discrete time system. This is repeated for every combination of delay and period, since these can be time-varying.

Delay jitter, period jitter and transient errors are expressed by using a Markov chain, which is supposed to be a model of the underlying computer system in terms of e.g. variations in execution and communication times. The state of the Markov chain changes from one sampling instance to the next according to a transition probability matrix. The Markov chain is thought of as describing a random phenomena, but it also admits periodic transitions to be described. A set of LTI systems governed by the transition of the Markov chain is called a jump linear system.

1.4 Related work

The report is rather comprehensive and covers well-known theory. The majority of the theoretical material has been around at least since the 1960's and included here is a reference from the early 1970's which is a study of jitter in computer control systems. In the early days of computer control systems, the timing properties were of course also interesting. A few related works are presented below. For an overview of the timing properties and a longer state-of-the-art report, see Sanfridson (2000a) or Sanfridson (2000b).

The main source of information and inspiration has been Nilsson (1998) who studied random delays, both stochastically independent and governed by a Markov chain. The two delays in the closed loop, from the sensor to the controller $\tau_{sc}(t)$ and from the controller to the actuator $\tau_{ca}(t)$ are stochastically independent and have known probability distributions. The total delay is bounded by the constant period $\tau_{sc}(t) + \tau_{ca}(t) < h$. This constraint is a very practical simplification. The probability distributions depend on the current Markov state r_k, e.g. the network load can be high or low. The current state is known to the controller and all old time delays are assumed to be known for the state estimation. The current value of τ_{ca} is of course unknown. The time delay in the messages are calculated by time stamping using a global clock. The main result of Nilsson's work is an optimal controller including an optimal state estimation for coping with delay jitter. The LQG control law is predicted based on measurements of the computer system's timing behaviour,

$$u_k = -L(\tau_{sc}, r_k)\begin{bmatrix} \hat{x}_k \\ u_{k-1} \end{bmatrix}$$

where L is a gain vector and u_{k-1} is a previous control signal and $\hat{x}_k$ is an estimated state vector.

Optimal control over a network with a queue on the measurement side was studied by Chan and Özgüner (1995). A FIFO queue is modelled at the sampler. The network delay for the measurement is τ_{sc} and the actuation delay is $\tau_{ca} = 0$. The multiplexing of the network is described with a switching function and the switching is governed by a Markov chain. The modelled process is time discrete and linear. An optimal controller for this communication link and jump system is derived from a standard discrete time loss function. A non-switching linear control law is derived.

The choice of sampling period is treated in Zeng-Qi (1981) from the set-out of investigating the effect of long sampling periods on disturbance rejection. The idea is that a long sampling period could do as well as a short one, but with the benefit that the implementation would be facilitated if the computational resource is scarce. The process is described by a continuous time state space equation. To the discrete time optimal controller with continuous time weight matrices, an optional Kalman estimator is added. The choice of sampling period for a number

of discrete time controllers are compared in Lennartson (1985). In order to get a fair comparison every controller is tuned such that the variance of the control output is held fixed and independent of the sampling period. This is done by changing an appropriate tuning parameter; it is either the weight matrix of the control output, the dominating time constant or the natural frequency depending on control algorithm in question. As a measure of performance, a continuous time average output variance is discretized. The SISO systems studied are extended by allowing time delay in the process and integral action in the controller. A time delay in the process is handled by a prediction, which preserves the order of the state matrix. Time-variations are not studied.

Lincoln and Cervin (2002) developed a tool called Jitterbug to study the effect of period, delay and jitter in computer control systems. The timing behaviour is modelled by assigning a so called timing node, describing a vector of discrete delays, to an elementary function, such as the sampler. The elementary functions are either continuous time or discrete time LTI systems. The tool can be used to investigate for example multi-rate systems, aperiodic systems and controllers compensating for jitter. It is possible to create branches, conditioned on a random variable or the total delay, to model the execution of the elementary functions. The tool computes the continuous time cost in (1). If the system is periodic, it is possible to plot the spectral density to investigate e.g. the sensitivity for different amounts of jitter. Each elementary function can be assigned a cost function; the total cost is the sum of all defined cost functions. The input period and delay specification together with the computations are based on a clock tick, which defines the granularity. The continuous elementary functions, and the weight matrices, are sampled by a period equal to the granularity. The discretized parts are then assembled together with the discrete time parts. Additional nodes are inserted into the resulting Markov chain to model the delay. The tool is partly based on the theory for multi-rate sampling. The granularity forces the ratio of two sampling periods to be a rational number. If a period — which is supposed to be the sampling period — is supplied, the system is periodic and the computation is efficient; otherwise a slow iterative method is used to find the steady state cost value. The border between periodic and aperiodic systems is resolved by an assumption regarding the execution model of the computer system: the execution of a periodic system is preempted (skipped) if the total delay exceeds the period. The delay is counted from a specifically marked (periodic) timing node.

Davidson (1973) investigates the impact of timing properties on control systems using quadratic cost functions — both time-varying delays and time-varying periods. The time-variation is modelled by a stochastic process having a known probability distribution. The loss function is set up differently, depending on the problem studied. They are similar to the one used in this report, and show more explicitly how the time variations enter the loss function. The intersample behaviour is captured in a noise term. Two different types of control signals are studied: impulse control and zero-order-hold. The relative change in the variance of the Riccati steady-state solution and in the noise are given for each case. Davidson distinguishes between clock synchronized sampling (centred around a uniform period) and free running sampling (which means the sampling instants drift). Sample rejection is also studied. A lot of the basic theory used by Davidson and the aim of investing the timing properties, are very much the same as in this report.

Integrated communication and control systems (ICCS) were studied by Halevi and Ray (1988), in a way close to the approach presented here. The “method of steps” as they call it, leads to a finite-dimensional time-varying discrete time model. A similarity is the discretization of the process when multiple arrivals of a piecewise constant control signals during one period is possible. However, one notable difference is that no stochastic framework is laid out: the time-varying delay is periodic and deterministic. The lumping of sensor-to-controller delay and

controller-to-actuator delay is discussed, with the aim to allow for a simplified analysis and design. No time-varying period is modelled and the proposed loss function is completely different: it is based directly on period and average delay and the probability of vacant sampling.

2 Treating time

This section describes in detail how the timing properties are modelled and used in the performance measure. The rate of actions and the response times deriving from the computer system are defined in a way that admits a control theoretical analysis.

2.1 The timing properties

The timing behaviour of the computer system is described by the so called timing properties. These also relate to the control system. In order to analyse a given system correctly, it is important to clarify how to model the computer system using the timing properties, see Sanfridson (2000a). The timing properties are:

- *The choice of period*. This determines the rate of actions in the computer system and it has a clear correspondence to the theory of discrete control systems. The choice the control engineer makes is often based on some rule-of-thumb regarding the dynamics of the synthesized closed loop system. The terms sampling period and control period are used as synonyms for single rate systems. The period is often seen as the result of a deliberate choice whereas the origin of the other timing properties are less controllable.
- *Constant delay*. The general rule is that delays should be as small as possible, because of the decrease in phase margin a delay causes in an LTI system. A general delay can be modelled by a constant part plus a time-varying part. A constant delay in an LTI system can conveniently be analyzed by familiar control engineering tools. Keeping the delay in the closed loop as small as possible is always a keen issue in design.
- *Jitter in delay and period*. The sampling action is also susceptible to jitter, i.e. sampling jitter. This is not the same as delay jitter, because a continuous process (signal) is sampled. Compare for example 1) an analog signal sampled with sampling jitter and 2) an analog signal sampled without any sampling jitter, but followed by an element that adds delay jitter to the sampled signal. When sampling jitter is present, the acquired value depends on the sampling instant. Delay jitter, on the other hand, causes a delay of the output value to the continuous process. Delay jitter and sampling jitter can of course both be present in a system simultaneously. Jitter makes the system time-varying and the standard tools for LTI (e.g. Nyquist and Bode plots) are no longer applicable. It is however possible to analyse both delay and period jitter with the method described here. Jitter is practically always present in a computer system, but it is often small enough to be neglected. Severe jitter might arise when the resource utilization of a processing unit or communication channel is high, depending on the scheduling strategy.
- *Transient error*. A transient error is here defined as an infrequent change in the normal sampling scheme, and it results in a prolonged delay of the control signal. It can be deliberate; what seems to be an natural strategy in computer control systems is to skip tasks or messages to clear the processing unit or communication channel from a temporary overload. The rise of a transient error can also be the result of a previous fault in the computer system, an example is vacant sampling (or vacant actuation) because of jitter. Transient errors are typically more devastating from an end-to-end argument perspective, to a system without forgiving dynamics. In a safety critical application it is of interest not to violate defined limits on some variable, e.g. the control output or the measured value.

2.2 The sampled and delayed process, constant period and delay

Here, the modelling problem will be outlined, but only for uniform sampling and constant delay. First, the idealized setup will be described in this section. The closed loop in Figure 2.1 shows the setup to be analyzed. The time continuous signal $z(t)$ from an analog sensor at the process P, is sampled at the constant period h. The control signal u calculated by the controller C is held constant by the hold circuit until the controller calculates a new control signal. The hold circuit holds the signal constant until a new one arrives, so called zero-order-hold. The control signal is subsequently delayed by τ time units, until it reaches the actuator affecting the process P. The constant delay can be viewed as belonging to P. The clock in the figure forces the sample and the hold to be activated simultaneously; the time to calculate the control law is negligible. This is called ideal sampling.

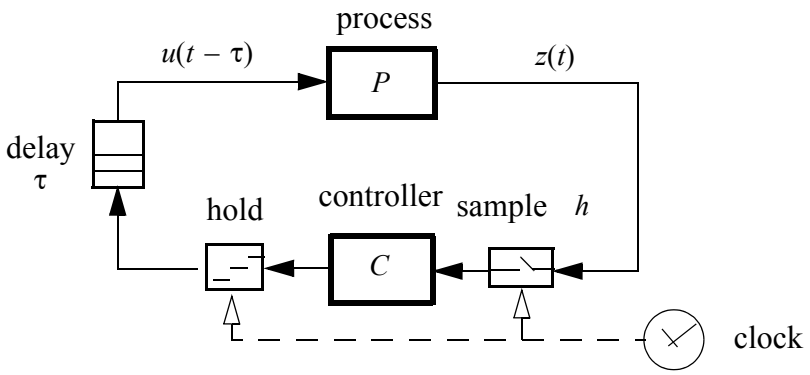


Figure 2.1 A closed loop system with sample and hold circuits, continuous time process, discrete controller, measurement signal and delayed control signal.

The scenario is depicted in Figure 2.2 by a timeline. The upper timeline represents the sampling and the lower represents actuation.The diagonal arrows indicate data processing, causing delays from sampling (at the upper timeline) to actuation (at the lower timeline). The control signal $u(t_k)$ is calculated at time t_k — preferably using the measurement $z(t_k)$, but does not affect the process until $t_k + \tau$.

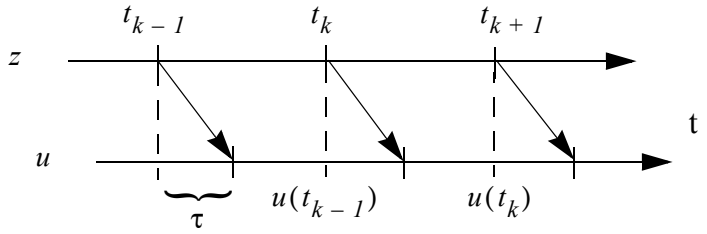


Figure 2.2 A periodic pattern of equidistant sampling and constantly delayed actuation.

The process P must be continuous time LTI and given in state space form. The translation to discrete time is well-known and can be found e.g. in Åström and Wittenmark (1997). The model becomes

$$\begin{bmatrix} x(t_{k+1}) \\ u(t_k) \end{bmatrix} = \begin{bmatrix} \Phi_p & \Gamma_{pb} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x(t_k) \\ u(t_{k-1}) \end{bmatrix} + \begin{bmatrix} \Gamma_{pa} \\ I \end{bmatrix} u(t_k) \tag{2}$$

The sampling period is constant $h = h_k = t_{k+1} - t_k, \forall k$. It is seen that the delay is described by $u(t_{k-1})$ and appended to the state. It should be clear that the longer a delay the more old control signals need to be augmented; this is treated further in Section 2.4.

An idea of the problems faced ahead will now be given. If the delay is time-varying such that it changes after every sampling, several models of type (2) are needed. The delay of the buffered signal cannot increase faster than the time itself, i.e. $\dot{\tau} \leq 1$ in continuous time notation. If $\tau > h$ is possible, this causality (or FIFO-queue) constraint becomes an issue for a sampled system. The change of model (2) must therefore be controlled in some way. To further complicate the setup, consider a time-varying h. The description of a constant delay based on augmenting the state by a set of old control signals is complicated by the fact that the number of control signals needed, depends on the length of each associated preceding period. It also depends on the order. In the general case, the delay should also be allowed to be time-varying at the same time the period is time-varying. This is the mess that has to be sorted out. It is not realistic to use an infinite number of models of the type (2), and to limit the number might be a necessary approximation.

An elaboration of this section is found in Section 4.1 introducing the timing model and discussing its connection to computer timing related issues.

2.3 Simplifying assumptions

The example in the previous section was greatly simplified by assuming a constant period and a constant delay less than the period. Five simplifying assumptions regarding period and delay, commonly made in related work are:

(A1) The sampling period is constant, $h > 0$. (Or else it can be time-varying.)

(A2) The delay is zero, $\tau = 0$. (Or else it can be positive.)

(A3) The delay is constant, $\tau > 0$. (Or else it can be time-varying.)

(A4) The delay is always less than or equal to the sampling period, $\tau(t) \leq h(t), \forall t$. (Or else the delay can be longer than a period.)

(A5) There is only one change of the control signal during one sampling period. (Or else there are none or several changes during one period.)

The idea is of course to relax these restrictive assumptions in order to analyse the general case. For example, in Figure 2.1 A1 and A3 were combined (and A2 relaxed). Assumption A4 covers the case when the total delay always is less than or equal to the possibly time-varying period. This is not equal to A5. The total delay can be longer than a period, but the delay variation can be shorter than a period. One could claim that it makes little sense to have a delay jitter larger than a constant h in a control system. On the other hand, this is more general and admits an analysis of a bursty communication channel. Åström and Wittenmark (1997) assume A1, A3 and relax A4. In Davidson (1973) assumptions A1 to A4 are relaxed at some point, but the general case with jitter simultaneously in both period and delay is not studied. Lennartson

(1985) and Zeng-Qi (1981) looked the choice of periods for the case of A1 and A2. Nilsson (1998) assumes A1 and A4 (relaxing A2 and A3) in the majority of the work but also brings up the case of relaxing A1 but holding A4. The Jitterbug tool by Lincoln and Cervin (2002) relaxes A1 to A4.

2.4 Describing time in the discrete model

The delay in the closed loop is captured in the control signal of the state-space representation. A set of sequential control signals, denoted by the vector u of size $[nu \times 1]$ can be written using the notation:

$$U(t_k) = \begin{bmatrix} u(t_{k-md}) \\ \cdots \\ u(t_{k-1}) \end{bmatrix} \tag{3}$$

where md is the maximum delay. Control signals older than md samples need not be considered. The column vector $U(t_k)$ has the size $[(md \cdot nu) \times 1]$. The time interval between any two rows is in the general case not equidistant. The discrete delayed control output vector is updated by

$$U(t_{k+1}) = \begin{bmatrix} 0 & I_{nu(md-1)} \\ 0 & 0 \end{bmatrix} U(t_k) + \begin{bmatrix} 0 \\ I_{nu} \end{bmatrix} u(t_k) \tag{4}$$

when time progresses from t_k to t_{k+1}. The rows in (3) are, so to speak, shifted upwards, such that the oldest value is discarded and the new value of $u(t_k)$, i.e. the control signal that is calculated at time t_k, is augmented.

The fact that the delay is described using a number of old signals is a bit more awkward when the sampling period is time-varying. Also when the period is allowed to vary with time, the delay must be described by a sequence of contiguous output instants: $t_{k-1}, t_{k-2}, \ldots, t_{k-md}$. A constant delay does not correspond to a fixed number of samples. The delay can of course also be time-varying at the same time, and it will then be described by time-varying periods. It facilitates the actual computations tremendously if $U(t_k)$ has a fixed length. In order to keep the system matrix constant additional "empty" rows and columns are inserted. md covers the worst-case combination of a sequence of τ_k and h_k, thus md might have to be larger than what is necessary for many other cases. Let nd denote the number of additional control signals needed to describe the delays of a particular sampling period. Especially, if τ and h are constant then

$$nd = \left\lceil \frac{\tau}{h} \right\rceil \tag{5}$$

additional signals need to be appended. If τ and h are constant, i.e. assumption A1 and A3 hold, then nd can be used to expressed the total (continuous time) delay:

$$\tau = (nd-1)h + \tau' \qquad 0 < \tau' \le h \tag{6}$$

as in Åström and Wittenmark (1997). As another example, if $0 < \tau_k \le h$ for a constant h, then only $u(t_{k-1})$ is needed, cf. A4. $nd = nd(k)$ is in general time-varying. It must hold that $nd(k) \le md$, with equality $\forall k$ in case τ and h are constant, or put another way

$$md = \max_{\forall k} nd(k) \tag{7}$$

If the controller uses control signals older than nd samples back, (7) should be written $md = max[nd(k), cd(k)]$, $\forall k$, were cd refers to the oldest control signal the controller uses. An alternative is to add internal states to the controller for old control signals too.

3 The discrete Markov chain

The timing of the computer system, i.e. the execution and communication, is modelled by a discrete time Markov chain. This is a common approach in computer engineering and it is related to queuing theory. It follows that any causality conditions must also be implemented by the Markov chain. The material presented in this section can be found in many text books, but it is important to relate it to the timing model. In the analysis, an existing steady state solution, i.e. the equilibrium which is eventually reached when the time goes to infinity, is of interest.

The type of chain of interest in this report has a finite number of states which are all reachable. Further, the chain is homogeneous, irreducible and either aperiodic or periodic. These terms will be defined below. An aperiodic chain can be used to model the general case of random time-variations, and a periodic chain can be used to model e.g. a static schedule. A chain can be periodic even if some state transitions are random.

A simple Markov chain with two states is shown in Figure 3.1 to exemplify the use of the chain. Each state corresponds to a timing situation characterized by a tuple $\{h, \tau\}$. The LTI dynamics of the process and the controller are unchanged. The average period and delay, as well as the performance measure depend on the steady state of the transitions. The probability of a jump from state i to state j is p_{ij}, where $i, j, r \in \{1, 2\}$. In the example, the delay is less than the period, i.e. $\tau_r < h_r$.

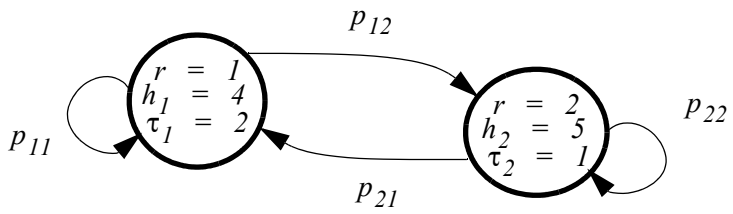


Figure 3.1 Example of a Markov chain with two states.

3.1 Fundamentals of the Markov chain

In a discrete Markov chain, state transitions, i.e. events of the discrete event system, occurs at t_k, $k = 0, 1, \ldots$. The events do not have to be equidistantly spaced in time. The sequence of events generated by a Markov chain is characterized by the so called Markov property: the next state (or mode) depends only on the current state and not on the past history of visited states. This property of uncorrelated states makes the Markov chain useful when analysing probability and expected values. For a stochastic system, the probability of the future states is uniquely determined by the current state.

The number of states in the Markov chain is denoted nr. Let $r, i, j \in \{1, 2, \ldots, nr\}$ be the state of a Markov chain. There is exactly one state active at any time instant. The state can (but does not have to) change every instance k, i.e. $r = r(k) = r(t_k)$ (the time t_k is the preferred argument). The transition probability from state i to state j is defined by:

$$p_{ij}(t_k) = Prob\{r(t_{k+1}) = j \,|\, r(t_k) = i\}$$

where $r(t_k)$ denotes the state of the chain at time t_k. The probability is of course limited by $0 \leq p_{ij}(t_k) \leq 1$. The probability of a jump from i (including a jump to itself) must be one at any time t_k. This can be expressed by:

$$\sum_{j=1}^{nr} p_{ij}(t_k) = 1$$

A Markov chain is said to be *homogenous* if p_{ij} is independent of t_k. The homongeneity is a form of stationarity of the transition probabilities, not of the Markov chain itself, see e.g. Cassandras and Lafortune (1999). As a result of this, one cannot know in advance exactly what state is active at a specific point in time, since $r(t_k)$ does not depend on t_k in a deterministic way. It is however possible to estimate the probability of being in a specific state, when the system is in equilibrium (steady state). The probability of being in state i at some time t_k is

$$\pi_i(t_k) = Prob(r(t_k) = i)$$

A chain is called *irreducible* if every state j can be reached ($p_{ij} > 0$), from any other state i, i.e. the state is not absorbed by some subset (clique) of the chain which it cannot leave. For an *aperiodic* and irreducible Markov chain there exists a limit probability distribution π_i^{∞} of state i

$$\left| Prob(r_k = i) - \pi_i^{\infty} \right| < \delta$$

for an arbitrary small $\delta > 0$. The *state probability vector*

$$\pi^{\infty} = \left[\pi_1^{\infty} \ \ldots \ \pi_{nr}^{\infty} \right]$$

contains the probability of every state in the chain, and has the size $[1 \times nr]$.

To find the steady state value of $\pi_i(t_k)$ a recursion starting from a suitable initial state $\pi_j(t_0)$ $\forall j$, can be written

$$\pi_i(t_{k+1}) = \sum_{j=1}^{nr} p_{ji}\pi_j(t_k) \tag{8}$$

Let the *transition probability matrix* of size $[nr \times nr]$ be defined by

$$P = \begin{bmatrix} p_{11} & \cdots & p_{1(nr)} \\ \cdots & \cdots & \cdots \\ p_{nr(1)} & \cdots & p_{nr(nr)} \end{bmatrix}$$

This will be used many time in the rest of the report. One can say that the state jumps “from a row to a column” in the transition matrix. A convenient way to rewrite recursion (8) is

$$\pi(t_{k+1}) = \pi(t_k)P \tag{9}$$

starting from a valid initial state $\pi(t_0)$. In (8) it must hold that $\sum \pi_j(t_k) = 1 \ \forall k$ or equivalently, that the row sum of P is one. The recursion in (9) will reach an equilibrium after an infinite time, $\pi^{\infty}(t_k) = \pi^{\infty}(t_k)P$ when $k \to \infty$, provided it exists. The recurrence equation in (9) converges for any choice of valid initial vector if the chain is *positive recurrent* and *aperiodic*.

3.2 A causal ordering condition for delays

In Figure 3.2, an example Markov chain modelling a transient error is sketched. State 1 corresponds to normal execution without delay. The period is constant. The transition $1 \rightarrow 2$ has the probability $p_{12} \ll 1$, which means that p_{11} is close to one, and the probability that the system is found in state 1 is high. It is clear from the figure that $p_{23} = p_{34} = p_{41} = 1$. State 2 represents a delay of h, state 3 a delay of $2h$ and state 4 a delay of $3h$. During these states, no new control signal arrives at the process. In this case, a temporary delay which spans over three sampling intervals is modelled.

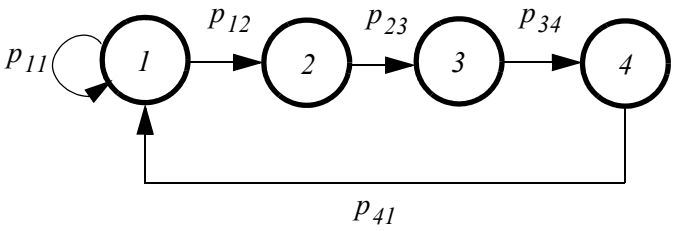


Figure 3.2 Example of a transient error with $p_{12} \ll 1$.

The structure of the transition probability matrix for this example becomes (with $p_{11} = 1 - p$):

$$P = \begin{bmatrix} 1-p & p & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{10}$$

Note that the chain is aperiodic if $p_{11} > 0$.

An example borrowed from Xiao et al (2000) with $nr = 4$, is shown in Figure 3.3, to clarify the ordering condition involved when modelling a delay which extends over several sampling periods. This is an elaboration of Figure 3.2. Assume that h is constant and that $\tau = kh$, $k = 0, 1, \ldots$. The time-varying delay can be described as a one dimensional buffer, a communication channel between a producer and a consumer. It is important to realize that the age of a data in a FIFO buffer can only increase as fast as time itself. In continuous time this can be expressed by $\dot{\tau} \leq 1$. However, there is no bound on the recovery — the delay can vanish instantaneously. To repeat, if condition A4 or A5 in Section 2.3 are satisfied, the ordering is no problem.

The Markov chain must admit and control this causal ordering of events. State 1 has a delay of zero, state 2 has a delay of h, state 3 a delay of $2h$ and 4 a delay of $3h$. Note that there are no transitions $r: 1 \rightarrow 3$, $1 \rightarrow 4$ and $2 \rightarrow 4$. A path in the direction from state 1 to state 4 must pass states 2 and 3. This is a result of the causal ordering (FIFO-ordering) condition.

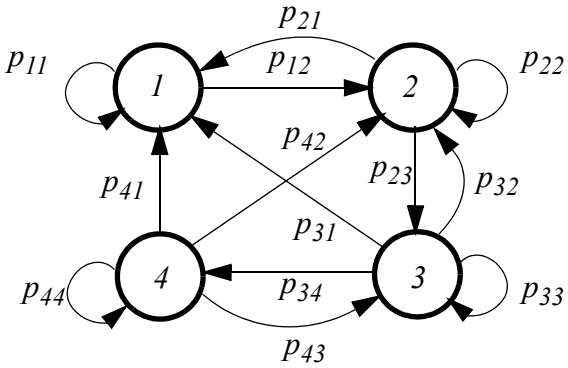


Figure 3.3 Example of impossible transitions, $p_{13} = p_{14} = p_{24} = 0$.

The transition matrix corresponding to Figure 3.3 is

$$P = \begin{bmatrix} p_{11} & p_{12} & 0 & 0 \\ p_{21} & p_{22} & p_{23} & 0 \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix}$$

Note that the upper right triangle is zero $p_{13} = p_{14} = p_{24} = 0$ to reflect the ordering condition.

3.3 The periodic chain

A stochastic process is called a Markov process if it is governed by a Markov chain. The whole idea by Markov chains is to model the stochastic movements between states. The Markov chain can however also describe a deterministic sequence or a sequence with an inherited periodic structure. Consider for example the transition matrix:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{11}$$

This corresponds to the fixed sequence, a circular and repeating pattern $r : 1234123\ldots$, cf. Figure 3.2 with the special case $p_{11} = 0$. The periodicity $\theta \in \{2, 3, \ldots\}$ is clearly 4. The recursion in (9) to calculate the state probability will not converge. The periodic result will also depend on the initial state vector $\pi(t_0)$. But the probability of being at a certain state at a certain time is defined anyway, since every state is visited according to a repetitive deterministic pattern.

Figure 3.4 shows an example of a periodic chain, which has a “mixed” deterministic and stochastic pattern:

$$P = \begin{bmatrix} 0 & p_{12} & p_{13} \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{12}$$

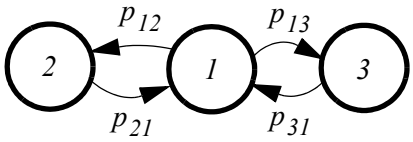


Figure 3.4 Example of a periodic Markov chain. It is necessary that $p_{21} = p_{31} = 1$.

The transition matrix (12) can be deceptive in this case. State $r = 1$ is visited every second (2nd) transition and every other second transition either state 2 or 3 is visited.

If a state j is only reached in a number steps via a set of all possible paths, then state j is periodic (with the period θ) if the greatest common divisor of the paths is equal to or greater than two. If it is equal to one, then the state is defined to be aperiodic (one could define $\theta = 1$). For example, the period in (12) is two, the greatest common divisor of the number of transitions. If a chain is irreducible (in other words, if state j can be reached from i by a path leading via other states), then all of its states must have the same period.

Another example of a 2-periodic chain is

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1-p & 0 & p & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{13}$$

for some probability p. In (12) state number one was always visited every second transition, but for (13) state number one is visited either every second or every fourth transition. Notice that the state cannot be exactly specified at a specific time t_k given an initial condition $\pi(t_0)$. The transition of the chain is not completely deterministic.

If the chain is periodic, the equilibrium must obviously be defined in another way. The probability of finding the chain in a specific state j is π_j^∞. In the examples (11) to (13) it is intuitively clear that a limit π_j^∞ exists. In a steady state it also makes sense that the chain returns to every state i within a bounded time θ_{ii} (counted as a number of transitions), i.e. the chain is recurrent. One can define the *mean recurrence time* $M_i = E(\theta_{ii})$ (or the *expected recurrence time*) and state i is positive recurrent if $M_i < \infty$ but weakly recurrent if $M_i = \infty$. For this type of Markov chain the following definition is useful

$$\pi_i^\infty = \lim_{k \to \infty} \pi_i(t_k) = \frac{1}{M_i} \tag{14}$$

The probability of finding the chain in a specific state is related to the expected recurrence time of that state.

4 Closing the loop

In this section the process will be discretized and together with the discrete controller the loop will be closed. First, the coupling between computer related timing and the proposed model will be clarified, a continuation of Section 2.2 and Section 3.2 with time-variations added to the discussion.

4.1 Sampling and actuation timelines

Recall the sampled-data closed loop of Figure 2.1. Figure 4.1 is an elaboration where the clock does not enforce ideal sampling and where the delay block comes before the hold block, the sampling and hold blocks are time-varying. The generality of the timing model to incorporate the timing behaviour of the computer system will be shown.

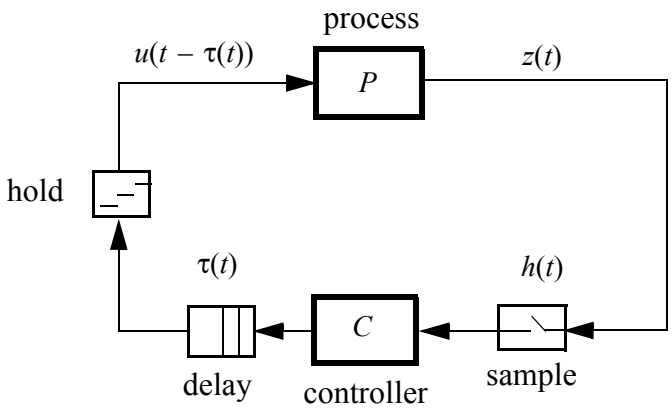


Figure 4.1 The closed loop of a sampled-data system.

The input and output actions are triggered by some means. The triggering can either be periodical (TT — time triggered) or event-driven (ET — event triggered). Assume that we have an ideal clock which emits an exactly periodic trigg pulse. At the trigg pulse some task or message associated with it becomes eligible to run, for example an A/D-conversion interrupt routine. Additional delay comes e.g. from the fact that the task is released having experienced release jitter, interference and that its actual execution time typically is time-varying. Tasks and messages can also form chains, and in distributed systems the chains span multiple hardware units. In a distributed computer system the ideal clock can represent a global clock; in contrast to local clocks which might not be synchronized. The mis-synchronization of two local clocks having a skew can lead to an additional time delay, even if both sender and receiver are TT. Both the TT and ET synchronization are affected by delays, constant or random. In case of a TT task, the actual period can for example be modelled by a constant plus a time-varying part with zero mean. In case a task chain is ET, the actual period can drift.

A task chain has several tasks and messages, but in an end-to-end study the input and output are primarily of interest. The input and output can correspond to the sampling and actuation respectively. The period of the input (sampling period) and the output (control period) is the same; multirate systems are not considered. The intermediate tasks and messages are rationalized in this model, i.e. modelled by a time-varying delay.

Table 4.1 shows four combinations of chains and the progress of the resulting input $in(k)$ and output $out(k)$ sequences in absolute time with $k = 0, 1, \ldots$, starting from some suitable initial values. This is an example of how input and output jitter can be described. It is assumed that the delay jitter $\delta\tau(k) \geq 0$ and period jitter $\delta h(k) \geq 0$ are time-varying, possibly modelled as random processes. Further, τ_0 is an offset (a nominal delay) from the time kh_0 where h_0 is the nominal period.

Table 4.1 Triggering table.

Input	Output	Corresponding model of sequential time instants.
TT	TT	$in(k) = kh_0 \pm \delta h(k)$ $out(k) = kh_0 + \tau_0 \pm \delta\tau(k)$
TT	ET	$in(k) = kh_0 \pm \delta h(k)$ $out(k) = in(k) + \delta\tau(k)$
ET	TT	$in(k) = in(k-1) + h_0 \pm \delta h(k)$ $out(k) = kh_0 + \tau_0 \pm \delta\tau(k)$
ET	ET	$in(k) = in(k-1) + h_0 \pm \delta h(k)$ $out(k) = in(k) + \delta\tau(k)$

Note that every $in(k)$ and $out(k)$ are subject to jitter. For example, in case both input and output are ET, they can both drift according to the recurrence equations, $in(k) = in(k-1) + \ldots$.

In Figure 4.2, both the sampling period and the actuation period are time-varying. This illustrates the most general case with sampling jitter and delay jitter. The diagonal arrows indicate data processing and transmission from sampling to actuation, i.e. the control delay. During the interval $(t_k, t_{k+1}]$ four consecutive control signals affect the process. On the other hand, during the interval $(t_{k-2}, t_{k-1}]$ an old control signal prevails. This depicted burst is possible e.g. for a task chain with event-triggered tasks. The control signal $u(t_k)$ is calculated at the time t_k, but typically affects the process a little later.

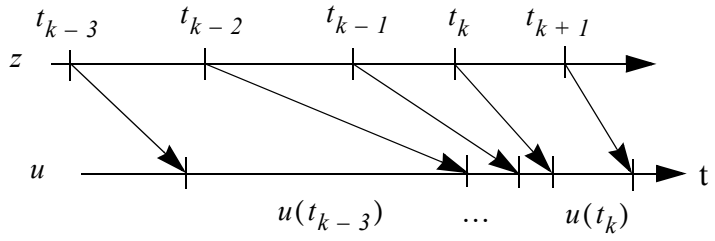


Figure 4.2 A sequence of relatively even sampling but with bursty actuation illustrated on a double timeline.

In contrast to the random sampling and actuation appearance of Figure 4.2, the pattern of Figure 4.3 looks deterministic; it actually has a periodicity θ of two samples. This pattern could for example be the result of a task chain with both TT input and output. The instants $t_{k-2}, t_k, t_{k+2}, \ldots$ are equidistantly spaced. Type *1* corresponds to the interval $(t_k, t_{k+1}]$ and type *2* to $(t_{k+1}, t_{k+2}]$. Both the sampling and the control delay change at every instant k. Note that the sequence of outputs in the figure does not exhibit jitter.

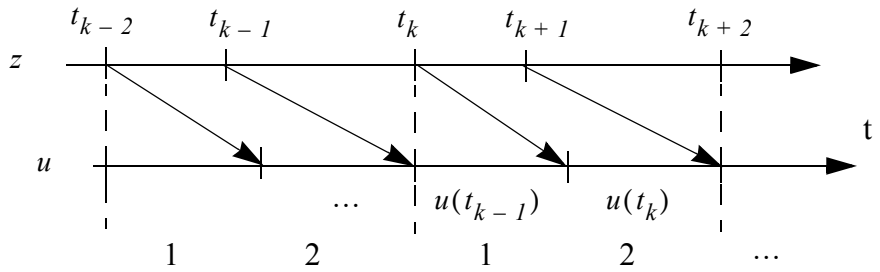


Figure 4.3 Example with a periodic sequence of sampling and actuation.

A periodic sequence with a partly random pattern is also a possible scenario, even though it might at first seem a little far-fetched. Consider Figure 4.4, which has a slight modification compared to Figure 4.3: the actuation instant corresponding to the sampling at t_{k-1} arrives earlier. The difference has been marked by a. Further, assume that the difference is modelled by a random variable, such that either a period of type *2* or type *3* will follow a period of type *1*. An example sequence of states is a pattern: *1213131312*… . This is also periodic: type *1* has a period of two and the others have a (time-varying) period which is a multiple of two. Compare this timeline with the transition matrix in (12).

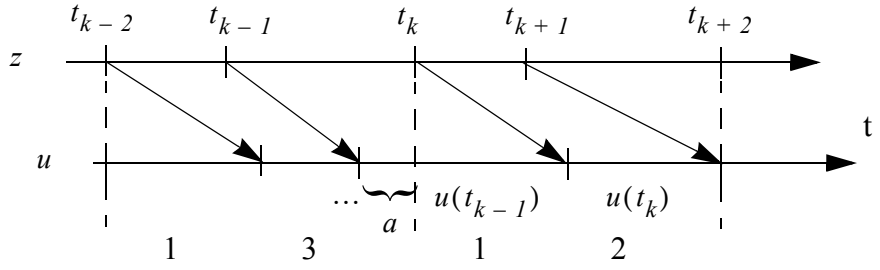


Figure 4.4 Another example of a periodic pattern of sampling and actuation. After a period of type *1* either a type *2* or *3* follows, which means that the sequence is not completely deterministic.

4.2 A model of the closed loop

The loss function will be addressed by continuing with the model of the closed loop first set up in Section 2.2, together with the sampling and actuation time lines in Section 4.1 in mind. Actually, two versions will be discussed but they are quite similar. The first version (called

"cd") will be the primary and the other (called "pd") will always be the second choice and only referred to when it is specifically mentioned.

In Figure 4.5, the process P has the state vector x_p and the controller C has the state vector x_c. z is a measurement used for feedback purpose, y is the process output signal, v is process state noise and w is measurement noise. These will be defined more formally later on.

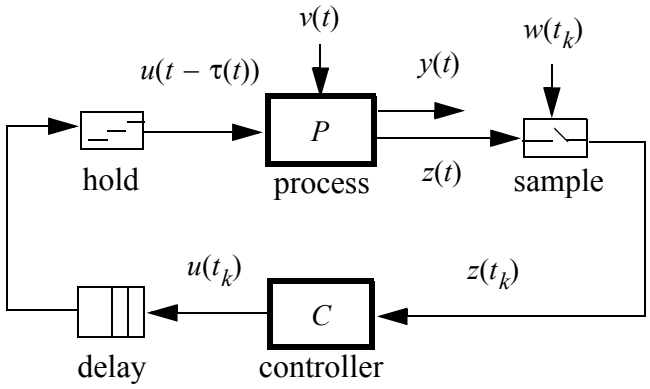


Figure 4.5 Process and controller, with delay before the hold block. The delay is on the controller's side, "cd".

Note that the delay is placed on the controller's side, before the hold circuit. The hold circuit is event-triggered by the delay. The shape of $u(t_k)$ does not change from the hold circuit past the delay; there is only a pure delay. In the model of the closed loop system, there is no delay between the sampling and the calculation of the control output, i.e. $z(t_k)$ and $u(t_k)$ are simultaneous, but $u(t_k)$ is delayed before it reaches the plant.

If the order of the hold and delay blocks in Figure 4.5 are switched, the delayed control signal will not change, and thus not the dynamics of the closed loop either. The hold circuit is event-triggered by the controller. The resulting closed loop is depicted in Figure 4.6.

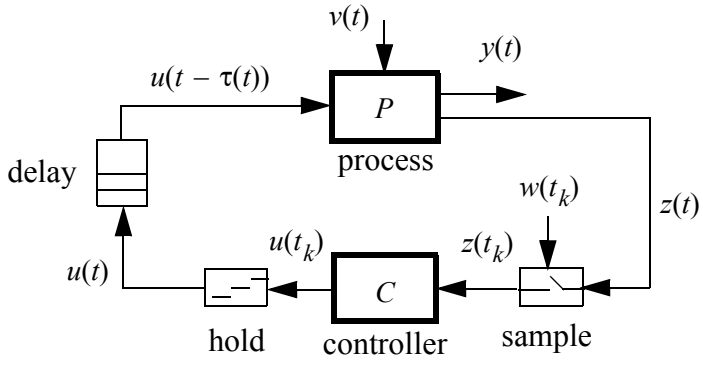


Figure 4.6 Process and controller, etc. The delay is on the process' side, "pd".

Delays in a computer control system do of course arise at the input side of the controller also. The control delay can here be viewed as lumped at the output side of the controller. This moving and fusing of delays will be discussed at the end of the report.

4.3 The closed loop state vector

The states of the process $x_p(t_k)$ are now being augmented by the old control signals $U(t_k)$ and by the states of the controller $x_c(t_k)$. This is a convenient way of expressing both a process and a controller, possibly including auxiliary elements, such as a filter or a state estimator. The resulting augmented state vector x, a column vector of size $[na \times 1]$, with $na = np + nu \cdot md + nc$, becomes

$$x(t_k) = \begin{bmatrix} x_p(t_k) \\ U(t_k) \\ x_c(t_k) \end{bmatrix} = \begin{bmatrix} x_p(t_k) \\ u(t_{k-md}) \\ \dots \\ u(t_{k-nd}) \\ \dots \\ u(t_{k-1}) \\ x_c(t_k) \end{bmatrix} \tag{15}$$

where md is the maximal number of additional states needed to describe the worst-case combinations of periods and delays in U. nd is the number of additional states needed for a particular combination of periods an delays, see Section 2.4.

We need to find an expression for how the state evolves from one instant $x(t_k)$ to the next:

$$x(t_{k+1}) = \begin{bmatrix} x_p(t_{k+1}) \\ U(t_{k+1}) \\ x_c(t_{k+1}) \end{bmatrix} = \begin{bmatrix} x_p(t_{k+1}) \\ u(t_{k-(md-1)}) \\ \dots \\ u(t_{k-(nd-1)}) \\ \dots \\ u(t_k) \\ x_c(t_{k+1}) \end{bmatrix} \tag{16}$$

The system matrix describing the transition from $x(t_k)$ to $x(t_{k+1})$ will in the general case be time-varying and managed by a Markov chain.

4.4 The controller

The linear discrete time-invariant controller is written on state-space form,

$$\begin{aligned} x_c(t_{k+1}) &= \Phi_c x_c(t_k) + \Gamma_c z(t_k) + \Lambda_c U(t_k) \\ u(t_k) &= C_c x_c(t_k) + D_c z(t_k) + E_c U(t_k) \end{aligned} \tag{17}$$

where $x_c(t_k)$ is the state of size $[nc \times 1]$, $U(t_k)$ is a vector of size $[(md \cdot nu) \times 1]$ with old control signals as was defined in (3). $\{\Phi_c, \Gamma_c, \Lambda_c, C_c, D_c, E_c\}$ are matrices of sizes: $[nc \times nc]$, $[nc \times nz]$, $[nc \times (md \cdot nu)]$, $[nu \times nc]$, $[nu \times nz]$ and $[nu \times (md \cdot nu)]$ respectively.

A separation of the output into $y(t_k)$ and $z(t_k)$ has been made. $y(t_k)$ is the output signal and $z(t_k)$ is the measurement signal, cf. the configuration in Figure 4.5 and Figure 4.6. For example, a state feedback controller can use several or every measured state but the interesting output is maybe only one of the states or another combination of states.

The actual discretization requires a sampling period as an input parameter. If the sampling period is constant, then $h = t_{k+1} - t_k$ and $t_{k+1} = kh + h$, of course. The update in (17) is performed in the same way regardless of any modelled jitter in h and τ. Future sampling instants and delays are usually unknown, whereas past sample times and delays are assumed to be fully known. It is always possible to retrieve timing information, e.g. via time stamping, from the underlying computer system. As an extension, but not further discussed here, $\{\Phi_c, \Gamma_c, \Lambda_c, C_c, D_c, E_c\}$ can be time-varying and describe a control strategy which tries to cope with the changes in the tuple $\{h_k, \tau_k\}$. It is possible to use the knowledge of previous control signals $U(t_k)$ in the control law to calculate $u(t_k)$. The reason for doing this is of course to improve the controller's ability to cope with delay and delay jitter. It is also possible for a controller to use older control signals than nd steps back in time, i.e. an over compensation is feasible.

4.5 The process and its discretization

The process can be written as a differential equation

$$dx_p(t) = A_p x_p(t)dt + B_p u(t)dt + dv(t)$$

where $x_p(t)$ is the state vector of size $[np \times 1]$, $u(t)$ is the input of size $[nu \times 1]$. The state noise $v(t)$ is a vector of white noise stochastic processes, has a mean value of zero and uncorrelated increments. The incremental covariance of $v(t)$ is $\bar{R}_v dt$, see Åström and Wittenmark (1997). $\bar{R}_v$ has the size $[np \times np]$.

Since the input signal is delayed, $u(t - \tau(t))$, the LTI process on a traditional state-space form becomes

$$\begin{aligned} \dot{x}_p(t) &= A_p x_p(t) + B_p u(t - \tau(t)) + v(t) \\ y(t) &= C_p x_p(t) \\ z(t_k) &= M_p x_p(t_k) + w(t_k) \end{aligned} \tag{18}$$

where $y(t)$ is the process output of the size $[ny \times 1]$, and $z(t_k)$ is the measured output of the size $[nz \times 1]$. Note that $v(t)$ is used instead of $\dot{v}(t)$. The constant matrices $\{A_p, B_p, C_p, M_p\}$ have the well-defined sizes: $[np \times np]$, $[np \times nu]$, $[ny \times np]$ and $[nz \times np]$ respectively. The process is strictly proper, i.e. there is no direct term D_p from u to y, and not from u to z either. This requirement on the dynamics is typical. On the other hand, a controller usually has a direct term from z to u, i.e. D_c in (17).

The first row in (18) is the state update. The second row is the continuous time output equation, and the third equation is the sampled measurement equation. The sampling action is associated with a measurement noise, $w(t_k)$. The measurement noise $w(t_k)$ is a vector of discrete Gaussian white noise stochastic processes with mean value of zero and a constant covariance of R_w. R_w has the size $[nz \times nz]$. $w(t_k)$ emerges by definition simultaneously with $z(t_k)$ at the sampling instant t_k, i.e. there is no time difference between them. The state and measurement noise, $v(t)$

and $w(t_k)$, are uncorrelated with each other and loosely speaking of other signals too. The implications of this model assumption will show up later.

The block diagram in Figure 4.7 gives an overview of the signals. x_c is the internal state vector of the controller.

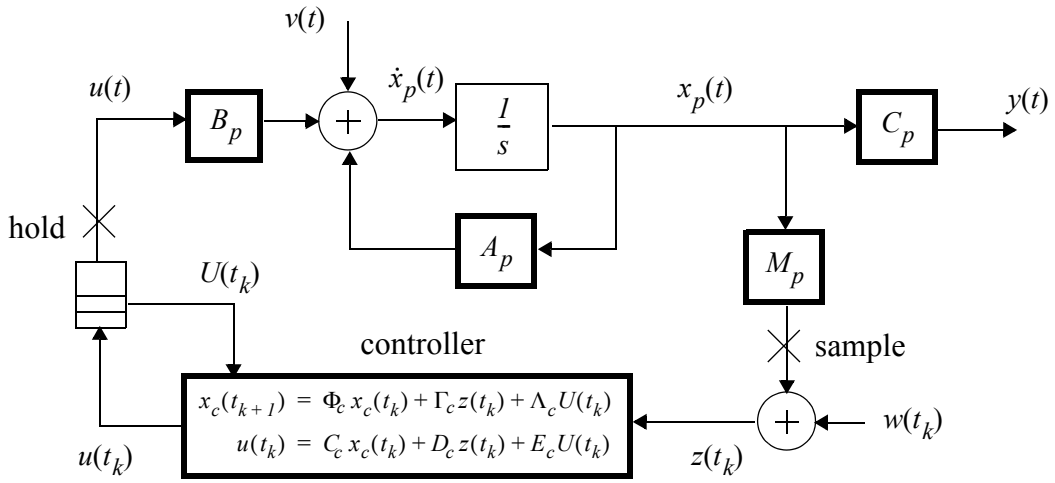


Figure 4.7 Block diagram of the sampled-data system.

In case of state feedback where all states are measurable, this can be captured by $M_p = I$. If some states are not measured, an observer can be used for state feedback, and e.g. $M_p = C_p$.

For the analysis, the process must be converted to discrete time, which means that a sampling period must be provided. The common zero-order hold sampling is used here. To facilitate the understanding and as an introduction and comparison to the presented extended formulations, three special cases will be treated below.

4.5.1 Constant sampling period, no delay

First, we assume that there is no delay $\tau = 0$, that the sampling period h is constant and the state and measurement noises are omitted. The discrete time system is the well-known, see e.g. Åström and Wittenmark (1997):

$$\begin{aligned} x_p(kh + h) &= \Phi_p(h)x_p(kh) + \Gamma_p(h)u(kh) \\ z(kh) &= M_p x_p(kh) \end{aligned} \tag{19}$$

where $\Phi(h) = e^{A_p h}$ and $\Gamma(h) = \int_h e^{A_p h} ds B_p$. The constant matrix gain M_p is unaffected when z is sampled.

4.5.2 Constant sampling period and constant delay

Now, the sampling period is still constant but we relax the assumption that $\tau = 0$. The total delay is given in (6): $(nd - 1)h$ is the integer part of the delay and τ' is the fractional part, i.e. within the bounds $0 < \tau' \leq h$.

The discrete time zero-order-hold sampled equivalent formulation of (18) with delay becomes,

$$x_p(kh + h) = \Phi_p x_p(kh) + \Gamma_{pa} u(kh - (nd - 1)h) + \Gamma_{pb} u(kh - (nd \cdot h)) + v(kh + h, kh) \tag{20}$$

for some $nd = 1, 2, \ldots$. The control signal is broken in two pieces, the moment $kh - (nd \cdot h)$ immediately precedes $kh - (nd - 1)h$, and they are both approximately a distance nd away from the instant kh. The system matrix Φ_p is the same as in the previous section and expressions for Γ_{pa} and Γ_{pb} can be found in Åström and Wittenmark (1997), where they are called Γ_0 and Γ_1 respectively.

The expression for the state update in (20) can be generalized into

$$x_p(t_{k+1}) = \Phi_p(t_{k+1}, t_k) x_p(t_k) + \Gamma_{pa}(t_{k+1} - t_k, \tau_k) u(t_{k-(nd-1)}) + \Gamma_{pb}(t_{k+1} - t_k, \tau_k) u(t_{k-nd}) + v(t_{k+1}, t_k) \tag{21}$$

The elements of the vector $v(t_{k+1}, t_k)$ are discrete Gaussian white-noise processes with zero mean and the covariance of $v(t_{k+1}, t_k)$ is R_v, see Section 4.5.7. However, this formulation with more detailed arguments permits simultaneous delay jitter and sampling jitter to be described, but with the restriction that only at the most one change of $u(t)$ is made between two sampling instants, i.e. assumption A5 in Section 2.3 must still hold. A Markov process could be used to model the time-variations.

The state update in (21) will be extended in the following section where the time interval between t_k and t_{k+1} does not need to be equidistant, and τ_k does not have to be constant. Further, multiple changes of u between t_k and t_{k+1} will be allowed. If $md = 0$ no extra state is needed (19), and if $md = 1$ (21) is applicable. On the other hand, if $md > 1$ there can be more than one change of u during some interval $h_k = t_{k+1} - t_k$. Recall that $md = max(nd(t_k))$.

4.5.3 The general case

In the general case both the period and the delay are time-varying. When the sampling period or the delay (or both) is allowed to vary, there can be more than one change of control signal during one sampling interval. The sampling period is no longer constant and each sample must be treated separately. Since the delay is described as a number of whole periods plus a fraction of a period, the non-constant period complicates the discrete time state space form. A Markov chain is used as a mechanism to “keep track of” the previous sampling instants for a particular delay. This comprises the timing interaction and causal ordering conditions. A complementary view is to consider the sampling instants and the output instants as two sequences, where instances in the output sequence occur more or less independently of the events in the input sequence. During normal operation, an output will follow an input instant suite, such that in a normal execution, every second event or so there is an input, and all the other events are the corresponding outputs.

The LTI continuous time process in (18) can be translated into discrete time by

$$\begin{aligned} x_p(t_{k+1}) &= \Phi_p(t_{k+1}, t_k)x_p(t_k) + \Gamma_{pa}(T_k)u(t_k) + \Gamma_{pb}(T_k)U(t_k) + v(t_{k+1}, t_k) \\ z(t_k) &= M_p x_p(t_k) + w(t_k) \end{aligned} \tag{22}$$

with matrices $\{\Phi_p, \Gamma_{pa}, \Gamma_{pb}, M_p\}$ of appropriate sizes: $[np \times np]$, $[np \times nu]$, $[np \times (nd \cdot nu)]$, and $[nz \times np]$ respectively. Note that Γ_{pb} is of another size than in the previous section. The control signal $u(t_k)$ corresponds to the sampling at t_k. Recall that in the model of the closed loop system, there is no delay between the sampling and the calculation of the control output, i.e. $z(t_k)$ and $u(t_k)$ are simultaneous, but $u(t_k)$ is delayed before it reaches the plant. $U(t_k)$ is a vector of old control signals defined by (3) and τ_k requires a number of elements in $U(t_k)$, in order to be correctly described. If (18) is strictly proper, then (22) will be strictly proper too.

Let $ng(k) \geq 0$ denote the number of changes of the control signal during the studied period $(t_k, t_{k+1}]$. Recall that $nd(k)$ refers to the oldest of the control signals arriving, and that $md = max(nd(k))$, $\forall k$. It must hold that $ng(k) \leq nd(k)$, $\forall k$, since $nd(k)$ denotes the number of control signals currently put on hold (queued in the FIFO buffer). The argument k is sometimes dropped in the sequel when there is no risk of confusion which instance is referred to.

Figure 4.8 shows a geometrical relation of the triple $\{ng, nd, md\}$ used to describe the delays and the switching of u. Let the instances $a \leq b \leq c$ be related to the sampling instance denoted $k \geq c$. Since it must hold that $nd(k) \leq md$ and $ng(k) \leq nd(k)$, this uniquely defines the instances b and c. The studied period starts at t_k. $u(t_b)$ is the oldest control signal to arrive during the studied period and $u(t_c)$ is the most recent to arrive. The most recently calculated control signal is $u(t_k)$. The arrows of the relative distances $\{ng, nd, md\}$ point in the respective positive directions. For example, an increase of ng makes the last control signal to arrive during the studied period to be $u(t_{c'})$ with $c' > c$; the distance between k and b is not changed.

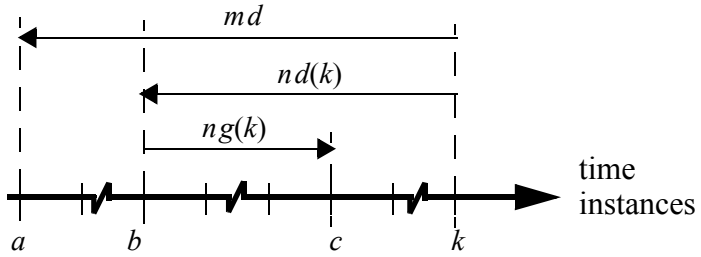


Figure 4.8 The total number of additional states md, the current number of additional states nd and the number of control signals due to ng. These define the instances b and c.

T_k in (22) is a set of delays, which describes the switches of the control signal during the studied sampling period $(t_k, t_{k+1}]$. Next, T_k will be described, and the input matrices $\Gamma_{pa}(T_k)$ and $\Gamma_{pb}(T_k)$ derived.

4.5.4 Multiple changes of the control signal

An alternative to the state update in (22) is the formulation:

$$x_p(t_{k+1}) = \Phi_p(t_{k+1}, t_k)x_p(t_k) \tag{23}$$

$$+ \sum_{g=0}^{ng} \Gamma_{pg}(h_k, \tau_{k,g+1}, \tau_{k,g})u(t_{k-(nd-g)}) + v(t_{k+1}, t_k)$$

where the vector $U(t_k)$ has been replaced by a summation. Each Γ_{pg} has the size $[np \times nu]$, and $g = 0, 1, \ldots, ng$. With $ng(k)$ switches, the sampling period is thus divided into $ng(k) + 1$ sub-intervals with $\Gamma_{p0}, \Gamma_{p1}, \ldots, \Gamma_{p(ng)}$. The choice of index g makes the corresponding $u(t_{k-(nd-g)})$ come in chronological order: $u(t_{k-nd}), u(t_{k-(nd-1)}), \ldots$. The sequence does not have to end by $u(t_k)$, but since the order of any two control signals cannot be reversed $u(t_{k-nd})$ must be the first.

Note that there is always a control signal acting on the process (possibly an old signal), i.e. the process is not described as an autonomous system, should there be no new control signal arriving during the studied period.

The set T_k has so far been vaguely defined. Now, let T_k be a sorted set of (partial) delays $\tau_{k,\gamma}$ or offsets from the corresponding t_k, with $\gamma = 0, 1, \ldots, ng + 1$. The offset $\tau_{k,g}$ is the time interval from t_k until $u(t_{k-(nd-g)})$ switches in, see Figure 4.9. The elements of T_k form a monotonically increasing sequence,

$$0 = \tau_{k,0} \le \tau_{k,1} \le \ldots \le \tau_{k,ng} \le \tau_{k,ng+1} = h_k \tag{24}$$

Normally, there are equally many periods (sampled events) as there are delays (output events), but they can be spread unevenly over time. If a period h_k is divided into several subsections, then a corresponding number of preceeding periods (not necessarily in a continguous sequence) had no outputs at all. Let some control signal $u(t_{k-c})$ end at $t_k + \tau_{k,g}$, with a known relation between c and g. To repeat, $\tau_{k,g}$ can be regarded as a delay or an offset relative to the sampling instant t_k. Each delay $\tau_{k,g}$ relates to one $u(t_{k-c})$ for some c which arrives at the input to the process (the hold circuit) during the studied period, cf. Figure 4.9.

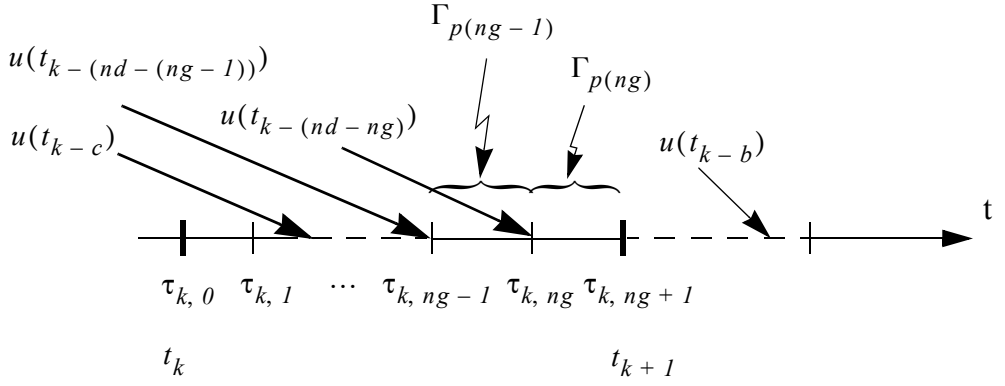


Figure 4.9 A sampling period is divided into $ng + 1$ sub intervals, one for each fractional delay $\tau_{k,g}$ corresponding to $u(t_{k-c})$ for some c. The relation between g and c is specified here. $u(t_{k-b})$ has been added to the timeline to represent control signals that do not arrive before t_{k+1} because $nd > ng$.

The forced definition of $\tau_{k,\,0} = 0$ and $\tau_{k,\,(ng+1)} = h_k$, is a useful construction in order to "complete" the interval $(t_k,\, t_{k+1}]$ properly. If no new control signal arrives then $ng = 0$, $\tau_{k,\,0} = 0$ and $\tau_{k,\,1} = h_k$. If $ng = 0$, then the sampling period is uninterrupted but the size of nd still determines the delay, e.g. if $nd = 1$ the delay is equal to the current period and if $nd > 1$, the delay spans multiple periods (but it is the studied period which is of interest).

A relation between (22) and (23) is established by (25)(26)(27) defining the input matrices:

$$\Gamma_{pa}(T_k) = \Gamma_{p(ng)}(h_k,\, \tau_{k,\,ng+1},\, \tau_{k,\,ng}) \tag{25}$$

which holds only if the last signal to arrive before t_{k+1} is $u(t_k)$, i.e. $ng = nd$, otherwise it is a zero matrix, $\Gamma_{pa}(T_k) = 0$. Further,

$$\Gamma_{pb}(T_k) = \left[0 \;\; \Gamma_{p0}(h_k,\, \tau_{k,\,1},\, \tau_{k,\,0}) \;\; \cdots \;\; \Gamma_{p(ng-1)}(h_k,\, \tau_{k,\,ng},\, \tau_{k,\,ng-1})\right] \tag{26}$$

when $ng = nd$. The number of leading zeros is due to the difference $md - nd$. Again, the other possibility, $ng < nd$, gives

$$\Gamma_{pb}(T_k) = \left[0 \;\; \Gamma_{p0}(h_k,\, \tau_{k,\,1},\, \tau_{k,\,0}) \;\; \cdots \;\; \Gamma_{p(ng)}(h_k,\, \tau_{k,\,ng+1},\, \tau_{k,\,ng}) \;\; 0\right] \tag{27}$$

The number of trailing zeros of the same size as Γ_{pg} corresponds to the difference $nd - ng$. To comfort the discrepancy between the oldest signal modelled md and the oldest signal of the current model nd, additional zeros have to be inserted, to keep the size of the closed loop state vector constant. The leading zero matrix of (26) and (27) has the size $[np \times (md - nd) \cdot nu]$, which actually is independent of ng.

4.5.5 Three special cases

First a straight forward example. In Figure 4.10, there is jitter in the sampling but the control output is uniform. The time-varying delay is a little longer than one sampling period. During the interval $(t_k,\, t_{k+1}]$, the control signal $u(t_{k-2})$ is switched to $u(t_{k-1})$ at the time instant $t_k + \tau_{k,\,1}$, as the control output affecting the continuous time process.

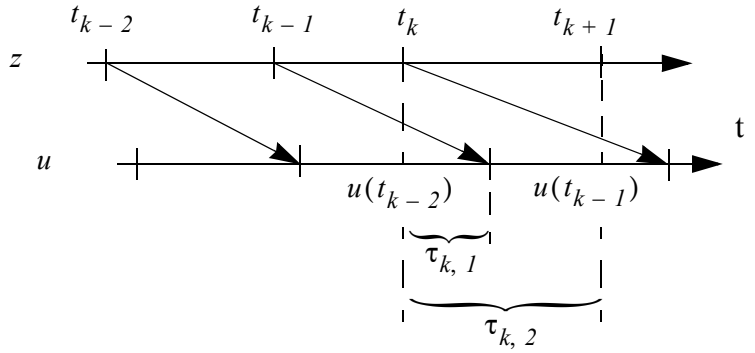


Figure 4.10 Example with $nd = 2$ and $ng = 1$ for the period $h_k = t_{k+1} - t_k$. Note that $u(t_k)$ is delayed such that it arrives at the process after the sampling at t_{k+1}.

The logical order of $u(t_k)\ \forall k$ is left untouched such that it is also the chronological order, but rejection can of course occur. Rejection means that an instance of $u(t_k)$ never reaches the process, since a later signal, $u(t_{k+a})$ for some a, is delivered first. This behaviour ultimately depends on the computer architecture, and it is easily coped with by the controller program by the use of a counter. Figure 4.11 shows a situation where the calculated control signal $u(t_{k-1})$ is discarded because $u(t_k)$ is a more recent value. From a control point of view it does not make sense to output $u(t_{k-1})$ after $u(t_k)$. This causal ordering is a basic assumption in the model. The situation depicted in the figure can be called *actuation rejection*, since a new control signal arrives prematurely and overrides the "normal" sequence. This term is derived from Halevi and Ray (1988) who used the term "message rejection" (or sample rejection) to denote the fact that only the most recent sensor data is accepted as input to the controller. Here, a sampled $z(t_{k-1})$ can still be used to produce $u(t_k)$ etc., even if its corresponding output $u(t_{k-1})$ was overwritten.

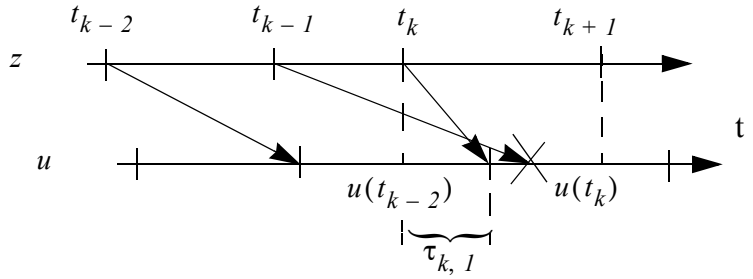


Figure 4.11 Actuation rejection. $u(t_{k-1})$ is not allowed to be output after $u(t_k)$.

Figure 4.12 shows another situation, which can be called *vacant actuation*. Due to a delivery omission of the control signal for some reason, $u(t_{k-1})$ does not reach the actuator. The output at time $t_k + \tau_{k,1}$ does not occur. This forces the previous output $u(t_{k-2})$ to be held a little longer, until a new u can be successfully sampled, calculated and output.

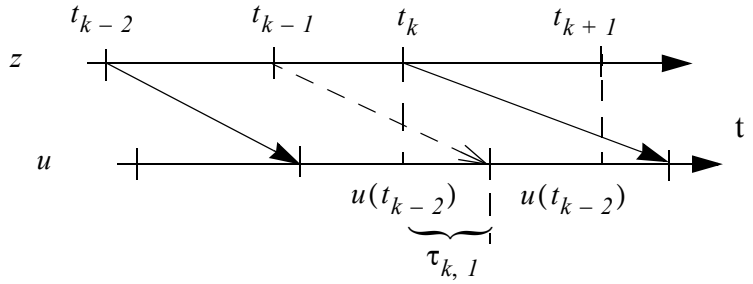


Figure 4.12 Example of vacant actuation. The value of $u(t_{k-2})$ is held and used instead of the vacant $u(t_{k-1})$.

4.5.6 Calculating the integrals

In this section it will be shown how the previously identified matrices Φ_p and Γ_{pg} are calculated. For the actual numerical calculations, see Appendix F. The solution to the differential

equation in (18) without delay and noise, i.e. $\dot{\bar{x}}_p(t) = A_p\bar{x}_p(t) + B_p u(t)$ becomes after a multiplication by $e^{-A_p t}$:

$$\frac{d}{dt}(e^{-A_p t}x_p(t)) = e^{-A_p t}B_p u(t)$$

and with an initial condition of $x_p(t = 0) = x_p(t_k)$ this yields

$$x_p(t) = e^{A_p(t - t_k)}x_p(t_k) + \int_{t_k}^{t} e^{A_p(t - s)}u(s)ds B_p \tag{28}$$

It follows from (28) that the discretization of the system matrix is computed by

$$\Phi_p(t_{k+1}, t_k) = e^{A_p(t_{k+1} - t_k)} \text{ or } \Phi_p(h_k) = e^{A_p h_k} \tag{29}$$

for a specific value of the sampling period. Φ_p does not depend on the delay. Γ_p on the other hand, depends on the delay of u. When there is no delay the input matrix is given by:

$$\Gamma_p(t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} e^{A_p(t_{k+1} - s)}B_p ds \text{ or } \Gamma_p(h_k) = \int_{0}^{h_k} e^{A_p s}ds B_p \tag{30}$$

for a specific value of the sampling period. The discretized Φ_p and Γ_p fit into (19).

For a delay $\tau \le h$, which will split the interval h_k into two parts by a fractional delay. Åström and Wittenmark (1997) give the integrals for $\Gamma_{pb} = \Gamma_{p0} = \Gamma_1$ and $\Gamma_{pa} = \Gamma_{p1} = \Gamma_0$, which correspond to $u(t_k)$ and $u(t_{k-1})$ respectively. Please note the difference between the order of the second index compared to Åström and Wittenmark (1997). u is piecewise constant over the interval h_k and right-continuous. The integrals are computed by

$$\Gamma_{pa}(h_k, \tau_k) = \int_{t_k + \tau_k}^{t_{k+1}} e^{A_p(t_{k+1} - s')}ds' B_p = \int_{0}^{h_k - \tau_k} e^{A_p s}ds B_p$$

$$\Gamma_{pb}(h_k, \tau_k) = \int_{t_k}^{t_k + \tau_k} e^{A_p(t_{k+1} - s')}ds' B_p = e^{A_p(h_k - \tau_k)}\int_{0}^{\tau_k} e^{A_p s}ds B_p$$

The discretized Γ_{pa} and Γ_{pb} fit into (21).

For multiple delays as described previously in (24), see Figure 4.9, a generalization of (30), is given by

$$\Gamma_{pg}(t) = \int_{t_k + \tau_{k,g}}^{t_k + \tau_{k,g+1}} e^{A_p(t - s'')}B_p ds'' = \int_{\tau_{k,g}}^{\tau_{k,g+1}} e^{A_p(t - t_k - s')}B_p ds' \tag{31}$$

$$= e^{A_p(t - t_k - \tau_{k,g+1})}\int_{0}^{\tau_{k,g+1} - \tau_{k,g}} e^{A_p s}ds B_p$$

for some $g = 0, 1, \ldots, ng$ over the time $(t_k + \tau_{k,\,g},\ t_k + \tau_{k,\,g+1}]$. It can be noted that $\Gamma_p(t_{k+1},\ t_k)$ in (30) with constant h is effectively split into a number of consecutive integrals such that:

$$\Gamma_p(t_{k+1},\ t_k) = \sum_{g=0}^{ng} \Gamma_{pg} \tag{32}$$

$$= \Gamma_{p0}(t_{k+1},\ \tau_{k,\,1},\ \tau_{k,\,0}) + \ldots + \Gamma_{p(ng)}(t_{k+1},\ \tau_{k,\,ng+1},\ \tau_{k,\,ng})$$

A special case arises when the actual value of $u(t_{k-c})$ for some $c > 0$, does not change over the corresponding time horizon, with the sum of $\Gamma_{pg}u(t_{k-c})\ \forall g$ in (23) equal to $\Gamma_p u$ in (19).

4.5.7 The state and measurement noise

Both v and w were characterized in the beginning of Section 4.5. To repeat, it is assumed that v and w do not correlate with each other, i.e. there is no cross-term in the covariance. The discrete noise inputs $v(t_k)$ and $w(t_k)$ arise simultaneously at the sampling instance k. Up to this point, the deterministic translation into discrete time has been treated, but the continuous time state noise must be discretized too, see e.g. Åström (1970) for the sampling of stochastic differential equations. The solution to the differential equation in (18) will once again be the starting point, but this time for simplicity without the control signal, $\dot{x}_p(t) = A_p x_p(t) + \dot{v}(t)$ rewritten into the form of a stochastic differential equation

$$dx_p(t) = A_p x_p dt + dv(t) \tag{33}$$

where the noise was characterized in the beginning of Section 4.5. Solving (33) by multiplying with $e^{-A_p t}$ and integrating over a period $(t_k,\ t]$ with appropriate initial conditions yields:

$$x_p(t) = e^{A_p(t-t_k)}x_p(t_k) + \int_{t_k}^{t} e^{A_p(t-s)}dv(s) = \Phi_p(t,\ t_k)x_p(t_k) + v(t,\ t_k) \tag{34}$$

The integral term of (34) corresponds to $v(t,\ t_k)$, the discrete equivalent of $v(t)$. It has a mean value of zero, it does not depend on the delay, and it does neither correlate with x_p nor with itself for two non-overlapping time-intervals, i.e. $(t_k,\ t_{k+1}] \cap (t_q,\ t_{q+1}] = \varnothing$ for $k \neq q$. The result $v(t_{k+1},\ t_k)$ will not be used directly, but rather to calculate the expected value of its covariance:

$$R_v(t_{k+1},\ t_k) = E[v(t_{k+1},\ t_k)v^T(t_{k+1},\ t_k)]$$

$$= \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-s')}\overline{R}_v e^{A^T(t_{k+1}-s')}ds' = \int_{0}^{h_k} e^{As}\overline{R}_v e^{A^T s}ds$$

or $R_v(t_k)$ for short, over a period h_k, see for example Åström (1970). In the following, the explicit notation of $(t_k,\ t_{k+1}]$ is dropped and represented by t_k only. The size of $R_v(t_k)$ is

$[np \times np]$, related of course to the size of x_p. The covariance matrix of the continuous time noise $\bar{R}_v = E[dv(s)dv^T(s)]$, is an input to the analysis. It can be remarked that $R_v(t_k)$ does neither depend on the delay nor the arrival pattern of the control signal.

The discrete Gaussian white noise process $w(t_k)$, $k = 0, 1, \ldots$ has zero mean value and a covariance matrix

$$R_w = E(w(t_k)w^T(t_k))$$

of the size $[nz \times nz]$, and R_w is an input to the analysis.

The augmented discrete time noise vector for the closed loop becomes

$$e(t_k) = \begin{bmatrix} v(t_k) \\ w(t_k) \end{bmatrix}$$

and has the size $[ne \times 1]$, with $ne = np + nz$. For notational convenience, let R_v and R_w be assembled into the discrete noise covariance matrix R_e, of size $[ne \times ne]$, defined by

$$R_e(t_k) = E(e(t_k)e^T(t_k)) = \begin{bmatrix} R_v(t_k) & 0 \\ 0 & R_w(t_k) \end{bmatrix}$$

having no cross-term.

4.6 The closed loop

The loop is closed by inserting the controller (17) into the process (22). The closed loop yields the state evolution (16). The measurement equation inserted into the control law gives:

$$u(t_k) = D_c M_p x_p(t_k) + E_c U(t_k) + C_c x_c(t_k) + D_c w(t_k) \tag{35}$$

The state update of the process (22) after insertion of (35) then yields

$$x_p(t_{k+1}) = (\Phi_p + \Gamma_{pa} D_c M_p) x_p(t_k) + (\Gamma_{pa} E_c + \Gamma_{pb}) U(t_k) + \Gamma_{pa} C_c x_c(t_k) \tag{36}$$

The state update of the delay vector in (3) becomes, also using (35):

$$U(t_{k+1}) = \begin{bmatrix} 0 \\ D_c M_p \end{bmatrix} x_p(t_k) + \begin{bmatrix} 0 & I \\ 0 & E_c \end{bmatrix} U(t_k) + \begin{bmatrix} 0 \\ C_c \end{bmatrix} x_c(t_k) \tag{37}$$

By combining the expressions (36) and (37), plus the controller state update (22), and by augmenting the states the fusion can more conveniently be written

$$\begin{bmatrix} x_p(t_{k+1}) \\ U(t_{k+1}) \\ x_c(t_{k+1}) \end{bmatrix} = \begin{bmatrix} \Phi_p + \Gamma_{pa} D_c M_p & \Gamma_{pa} E_c + \Gamma_{pb} & \Gamma_{pa} C_c \\ \begin{bmatrix} 0 \\ D_c M_p \end{bmatrix} & \begin{bmatrix} 0 & I \\ 0 & E_c \end{bmatrix} & \begin{bmatrix} 0 \\ C_c \end{bmatrix} \\ \Gamma_c M_p & \Lambda_c & \Phi_c \end{bmatrix} \begin{bmatrix} x_p(t_k) \\ U(t_k) \\ x_c(t_k) \end{bmatrix} + \begin{bmatrix} I & \Gamma_{pa} D_c \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ D_c \end{bmatrix} \\ 0 & \Gamma_c \end{bmatrix} \begin{bmatrix} v(t_k) \\ w(t_k) \end{bmatrix} \tag{38}$$

or for short

$$x(t_{k+1}) = \Phi_{cl} x(t_k) + \Gamma_{cl} e(t_k) \tag{39}$$

Φ_{cl} is the closed loop system matrix of the size $[na \times na]$ with $na = np + nu \cdot md + nc$, and with the closed loop state vector $x_{cl}(t_k)$ (or $x(t_k)$ for short) of the size $[na \times 1]$. Γ_{cl} is the input matrix and it has the size $[na \times ne]$. The sizes of the submatrices of Φ_{cl} and Γ_{cl} are given in detail in Appendix A.

5 Discretization of the loss function

The discretization of the process was treated in the previous section, and together with the controller the closed loop has been constructed. This gives the possibility to calculate the steady state covariance of the sampled-data system when driven by noise. The covariance will be weighted such that some states of the process contribute more to the variance than other.

5.1 The continuous time loss function

The loss function is in continuous time given by the expected value of a quadratic function over some time period T (not to confuse with the token used to denote a matrix transpose). It is defined either by

$$\bar{J}_{cd} = E \lim_{T \to \infty} \frac{1}{T} \int_0^T \begin{bmatrix} x_p(t) \\ u(t - \tau(t)) \end{bmatrix}^T \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix} \begin{bmatrix} x_p(t) \\ u(t - \tau(t)) \end{bmatrix} dt \tag{40}$$

with subscript “cd” for controller delay, or

$$\bar{J}_{pd} = E \lim_{T \to \infty} \frac{1}{T} \int_0^T \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix}^T \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix} \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix} dt \tag{41}$$

with subscript “pd” for process delay. These two versions will be treated in the following, but with the focus on “cd”, which will be the standard choice from here on — if “pd” is used it will be specially remarked. See Appendix D and Appendix E for details.

The given symmetric and positive semi-definite weight matrices $\bar{Q}_{11} = \bar{Q}_{11}^T \geq 0$ and $\bar{Q}_{22} = \bar{Q}_{22}^T \geq 0$ punish the state and control signal respectively. In addition $\bar{Q}_{12}$ should be chosen such that the assembled matrix $\bar{Q}$ is positive definite. $\bar{Q}_{11}$ is of size $[np \times np]$, $\bar{Q}_{22}$ $[nu \times nu]$ and $\bar{Q}_{12}$ $[np \times nu]$. The lower the value of the scalar $\bar{J} > 0$ the better, or with an alternative interpretation, the lower the value — the less worse. The weighting factor

$$\bar{Q} = \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix}$$

is a part of the specification, and the question how to select a suitable weighting factor will be discussed briefly.

Figure 5.1 shows the idea that x_p and u are tapped from the closed loop to calculate $\bar{J}$ by “inserting probes” into the loop — without affecting the closed loop. A time continuous loss function will now be discretized and this will render the time discrete weighting factor. A discrete loss function is needed because the controller is expressed in the discrete time domain. The purpose is to assess control performance and not to develop a method to find an optimal controller. Consequently, it is not enough to start directly with a time-discrete loss function.

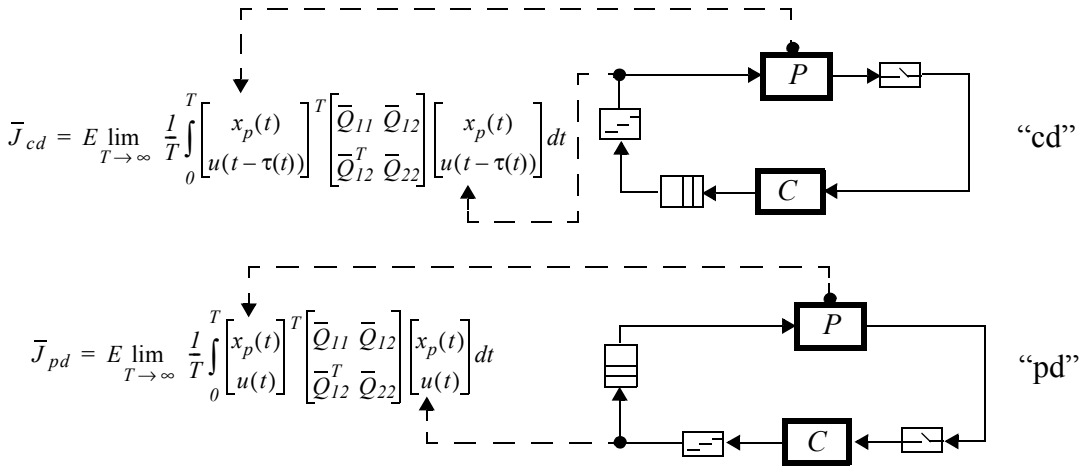


Figure 5.1 The closed loop is tapped and the signals are weighted. The delay is either at the controller's side (top) or at the process' side (bottom).

Starting with continuous time weight matrices $\bar{Q}_{11}$, $\bar{Q}_{22}$, $\bar{Q}_{12}$, the discrete time zero order hold sampled equivalents are to be calculated. In the general case, an additional term J_v has to be added to the loss function compared to the continuous version, in order to account for the inter-sample behaviour of the state noise. As pointed out by Åström and Wittenmark (1997), this is not needed when the only purpose of translation is to find the optimal controller. In a closed loop, the measurement noise will also contribute by a term called J_w via the controller.

The basic idea is to take the expected value of (40) or (41) during a time period T that is long enough for $\bar{J}$ to reach a steady state value, with an expected value of the stochastic function. If the limes of (40) or (41) does not exist ($\bar{J}$ is unbounded), the system is considered unstable in the mean square sense, see Section 7.1. The scaling by $1/T$ admits an interpretation "loss per time unit" in steady state, as if it was an average dissipation of energy. In the well-known case of a sampled system without time-varying delays or periods, T need not be longer than the sampling period, h. An equilibrium can also be achieved if the system is periodic with another T, i.e. a least common multiple of a repeating sequence of h and τ. The closed loop system can also be aperiodic. In that case the closed loop can be a modelled as a stochastic function even if the state and output noise terms are both nil. This motivates the use of the expected value operator.

The weighted covariance in (41) is the common way to define a loss function, but it is quite possible to construct other forms which are equally valid for the aim of estimating a measure of goodness, as (40) of course. In this context of time-varying sampling periods and delays, Davidson (1973) gives examples of other constructions, which models the phenomena similarly. It is important to understand that (40) or (41) together with a matrix $\bar{Q}$ defines what is considered good, so the scalar measure $\bar{J}$ is not a universal truth. However, this construction admits comparisons of e.g. different controllers for a fix $\bar{Q}$.

5.2 Discretization, no delay

This section contains the discretization case without delay, found in text books, such as Åström and Wittenmark (1997) and Franklin et al. (1990). The repetition of the material serves as an

introduction to the extensions presented later. It follows the structure in the previous section, where the simplified case is treated before the more general and elaborated case.

Let J be the discrete time counterpart of $\bar{J}$. With a constant sampling period h and without delay, the loss function is often written

$$J = E \lim_{N \to \infty} \frac{1}{N} \sum_{k=0}^{N-1} J_k + J_N \tag{42}$$

which is an average value over N samples of J_k starting from zero, followed by an end-value J_N. $J_N = x^T(Nh)\bar{Q}_0 x(Nh)$ is the final covariance, and it is a useful term when solving for an LQ-controller by dynamic programming backwards from N to 0; here $\bar{Q}_0 = 0$ is the simplest choice. When sampled, the continuous system inherits the uniform sampling period and becomes periodic with the period h. It is enough to take the expected value over one sampling period since the expected state evolution is equivalent for every such time interval. It is clear from (41) and (42), that

$$\bar{J}_k = \frac{1}{h} E \int_{kh}^{(k+1)h} \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix}^T \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix} \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix} dt \tag{43}$$

is the loss during a time period of h. The sampled equivalence of $\bar{J}_k$, $\forall k$ can be written:

$$J_k = E \begin{bmatrix} x_p(kh) \\ u(kh) \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix} \begin{bmatrix} x_p(kh) \\ u(kh) \end{bmatrix} + J_v \tag{44}$$

For convenience, the factor $1/h$ has been moved into Q_{ij} in (44). The product Nh represents the time horizon. The extra term J_v, is the intersample variance due to the noise $v(t)$, see Section 6.1. An identification of the new composite weight matrix

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix} \tag{45}$$

is made possible simply by comparing (43) with (44). Q is symmetric, $Q_{11} > 0$ is of the size $[np \times np]$, $Q_{22} \geq 0$ of $[nu \times nu]$ and Q_{12} of the size $[np \times nu]$. The discrete time weight matrices are translated by the integrals below.

$$\begin{aligned} Q_{11} &= \frac{1}{h} \int_{kh}^{(k+1)h} \Phi_p^T(t)\bar{Q}_{11}\Phi_p(t)dt \\ Q_{12} &= \frac{1}{h} \int_{kh}^{(k+1)h} \Phi_p^T(t)[\bar{Q}_{11}\Gamma_p(t) + \bar{Q}_{12}]dt \\ Q_{22} &= \frac{1}{h} \int_{kh}^{(k+1)h} \Gamma_p^T(t)\bar{Q}_{11}\Gamma_p(t) + 2\Gamma_p^T(t)\bar{Q}_{12} + \bar{Q}_{22}dt \end{aligned} \tag{46}$$

where

$$\Phi_p(t) = e^{A_p(t-kh)}$$

$$\Gamma_p(t) = \int_{kh}^{t} e^{A_p(t-s)} B_p ds$$

come from (29) and (30) respectively. Since all periods have the same length h, letting $k = 0$ is a simple choice. With $\Gamma_p(t)$, there will be a double integral in the equations above. It can be noted that Q_{12} arises even though $\bar{Q}_{12}$ is identical to a zero matrix. Q_{12} can be eliminated by a change of variables, see Åström and Wittenmark (1997), this is however not done here.

When the loop is closed, the discrete measurement noise will add J_w to the variance like J_v does. The dimensions of the state and control signal do not change when the loop is closed — except if the state of the controller x_c is appended, but these states are not punished anyway.

5.3 Discretization, constant delay

Assume that both the sampling period and the delay are constant. For simplicity, the delay is less than the sampling period. At the beginning of a period, the previous control signal will act on the process. A new control signal, based on the current sampling instant, will be switched in after some $\tau < h$. Every period, there will be one switch from an old to a new (or one period less old) control signal. This will actually be the case even if there is a constant delay longer than the period. Thus, all periods are identical and the discretized system inherits the period h, exactly as in the previous section. The problem in this section is, as it was in the previous section, how to discretize $\bar{Q}$ into Q. Q must fit with the discrete state vector x_p, now augmented by the previous control signal $u(kh - h)$. The treatment in this section is a special case of the general setup, and some obvious calculations will not be repeated when it comes to the general case.

With constant h and a constant τ, a solution to the (retarded) differential equation $\dot{x}_p(t) = A_p x_p(t) + B_p u(t-\tau)$ can be written:

$$x_p(t) = e^{A_p(t-kh)} x_p(kh) + \int_{kh}^{t} e^{A_p(t-s')} B_p u(s'-\tau) ds' \qquad (47)$$

which gives the value of $x_p(t)$, $t \geq kh$, cf. (28). The time interval of interest is $t \in (kh,\ kh + h\ ]$. The state noise has zero mean and is uncorrelated with both x_p and u. Again, the term J_v arises due to the state noise and it will be treated separately, see Section 6.1. Luckily, J_v does not depend on the delay, but on h. Recall that $x_p(kh)$ is constant and $u(s'-\tau)$ is piecewise constant over the studied time interval, see Figure 5.2.

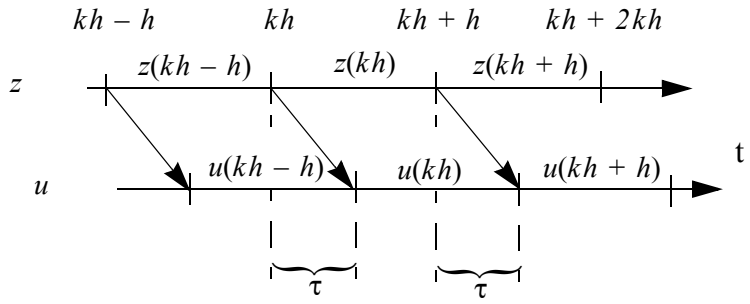


Figure 5.2 A special case with a constant delay less than the constant period.

The loss function (40) taken over one period,

$$\bar{J}_k = \frac{1}{h} E \int_{kh}^{(k+1)h} \begin{bmatrix} x_p(t) \\ u(t-\tau) \end{bmatrix}^T \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix} \begin{bmatrix} x_p(t) \\ u(t-\tau) \end{bmatrix} dt \tag{48}$$

can be rewritten as $J_k = E(I_{11} + 2I_{12} + I_{22})$, and three integrals need to be studied:

$$\begin{aligned} I_{11} &= \frac{1}{h} \int_{kh}^{(k+1)h} x_p^T(t) \bar{Q}_{11} x_p(t) dt \\ I_{12} &= \frac{1}{h} \int_{kh}^{(k+1)h} x_p^T(t) \bar{Q}_{12} u(t-\tau) dt \\ I_{22} &= \frac{1}{h} \int_{kh}^{(k+1)h} u^T(t-\tau) \bar{Q}_{22} u(t-\tau) dt \end{aligned} \tag{49}$$

The discrete time correspondence to (48) is conveniently written

$$J_k = \begin{bmatrix} x_p(kh) \\ u(kh-h) \\ u(kh) \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{12}^T & Q_{22} & Q_{23} \\ Q_{13}^T & Q_{23}^T & Q_{33} \end{bmatrix} \begin{bmatrix} x_p(kh) \\ u(kh-h) \\ u(kh) \end{bmatrix} + J_v \tag{50}$$

with the new composite square positive definite and symmetric weight matrix Q of size $[np + nu \cdot nd + nu]$ (cf. nq), where $nd = 1$ is the number of extra states necessary to describe the delay. Q can be identified by comparing (48) via (49) with (50) and by inserting $x_p(t)$ given by (47). This is exactly the same way it was done in the previous section, but this time more details of the calculations are given in Appendix D. The expected value operator has been dropped, since the stochastic noise is disregarded for the moment being and the discrete system is aperiodic.

The identification yields,

$$
\begin{aligned}
Q_{11} &= \frac{1}{h}\int_{kh}^{(k+1)h} \Phi_p^T(t)\bar{Q}_{11}\Phi_p(t)dt \\
Q_{12} &= \frac{1}{h}\int_{kh}^{kh+\tau} [\Phi_p^T(t)\bar{Q}_{11}\Gamma_{pb}(t) + \Phi_p^T(t)\bar{Q}_{12}]dt \\
Q_{13} &= \frac{1}{h}\int_{kh+\tau}^{kh+h} [\Phi_p^T(t)\bar{Q}_{11}\Gamma_{pa}(t) + \Phi_p^T(t)\bar{Q}_{12}]dt \\
Q_{22} &= \frac{1}{h}\int_{kh}^{kh+\tau} [\Gamma_{pb}^T(t)\bar{Q}_{11}\Gamma_{pb}(t) + 2\Gamma_{pb}^T(t)\bar{Q}_{12} + \bar{Q}_{22}]dt \\
Q_{33} &= \frac{1}{h}\int_{kh+\tau}^{kh+h} [\Gamma_{pa}^T(t)\bar{Q}_{11}\Gamma_{pa}(t) + 2\Gamma_{pa}^T(t)\bar{Q}_{12} + \bar{Q}_{22}]dt \\
Q_{23} &= 0
\end{aligned}
\tag{51}
$$

where

$$
\begin{aligned}
\Phi_p(t) &= e^{A_p(t-kh)} \\
\Gamma_{pb}(t) &= \int_{kh}^{t} e^{A_p(t-s)}B_p ds \qquad kh < t \le kh+\tau \\
\Gamma_{pa}(t) &= \int_{kh+\tau}^{t} e^{A_p(t-s)}B_p ds \qquad kh+\tau < t \le kh+h
\end{aligned}
\tag{52}
$$

There is no cross term ($Q_{23} = 0$) between the two piecewise constant parts $u(kh-h)$ and $u(kh)$ of $u(t-\tau)$. This result has been derived without assumptions about the controller, and it generalizes the case without delay (46).

5.4 Discretization, the general case

In the general case, both the delay and the period are allowed to be time-varying and $\tau(t) > h(t)$ is also possible, i.e. there might be more than one change of control signal between two sampling instants.

The starting point is the continuous time loss function in (40), and again the discrete time correspondence will be written as the expected value of the sum of individual contributions,

$$
J = E \lim_{N\to\infty} \frac{1}{N}\sum_{k=0}^{N-1} J_x(t_k) + J_x(t_N) \tag{53}
$$

It is the steady state behaviour that is interesting. The challenge here, compared to the previous sections, is that $J_x(t_k)$ will not be identical $\forall k$, i.e. the common case here is that $J_x(t_k) \neq J_x(t_l)$ for $k \neq l$. As before, the final value $J_x(t_N)$ is disregarded. It is not enough only to take the expected value of a single J_k (as previously defined), since the behaviour of the system is not uniform over some period corresponding to $J_x(t_k)$. As mentioned earlier, the closed loop will be modelled as a jump linear system, which is typically thought of as being aperiodic. Provided the closed loop it is stable, (53) will converge to a value of J in the mean square sense of a stochastic process. In case the jumps of the linear system exhibit a periodic pattern, caution must be taken to get the notion of convergence correct. In any case, it is of interest to reformulate the state vector as

$$J_x(t_k) = \begin{bmatrix} x_p(t_k) \\ u(t_{k-md}) \\ \cdots \\ u(t_{k-nd}) \\ \cdots \\ u(t_{k-1}) \\ u(t_k) \end{bmatrix}^T Q(t_k) \begin{bmatrix} x_p(t_k) \\ u(t_{k-md}) \\ \cdots \\ u(t_{k-nd}) \\ \cdots \\ u(t_{k-1}) \\ u(t_k) \end{bmatrix} + J_v(t_k) \tag{54}$$

and the problem here is of course to find the discrete time $Q(t_k)$. Note that the discretized weight matrix now depends on the time. There are md extra states allocated to old control signals, compared to the continuous time correspondence. Again, in order to undertake an analysis using constant sizes, the matrices and vectors must be dimensioned for the worst-case. The constant md corresponds to the maximum value of nd for any time instant t_k. The time-varying $J_v(t_k)$ will be treated later in Section 6.1.1.

Using calculations analog to previous subsections, the following integrals are readily found

$$\begin{aligned} q_{x,x} &= \frac{1}{t_{k+1}-t_k} \int_{t_k}^{t_{k+1}} \Phi_p^T(t)\bar{Q}_{11}\Phi_p(t)dt \\ q_{x,g} &= \frac{1}{t_{k+1}-t_k} \int_{t_k+\tau_{k,g}}^{t_k+\tau_{k,g+1}} [\Phi_p^T(t)\bar{Q}_{11}\Gamma_{pg}(t) + \Phi_p^T(t)\bar{Q}_{12}]dt \\ q_{g,g} &= \frac{1}{t_{k+1}-t_k} \int_{t_k+\tau_{k,g}}^{t_k+\tau_{k,g+1}} [\Gamma_{pg}^T(t)\bar{Q}_{11}\Gamma_{pg}(t) + 2\Gamma_{pg}^T(t)\bar{Q}_{12} + \bar{Q}_2]dt \end{aligned} \tag{55}$$

They all depend on the current instance k and control signal g with

$$\begin{aligned} \Phi_p(t) &= e^{A_p(t-t_k)} && t_k < t \leq t_{k+1} \\ \Gamma_{pg}(t) &= \int_{t_k+\tau_{k,g}}^{t_k+\tau_{k,g+1}} e^{A_p(t-s)}B_p ds && t_k+\tau_{k,g} < t \leq t_k+\tau_{k,g+1} \end{aligned}$$

The assembled weight matrix becomes

$$Q(t_k) = \begin{bmatrix} q_{x,\,x} & 0 & q_{x,\,0} & \cdots & q_{x,\,ng} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ q_{x,\,0}^T & 0 & q_{0,\,0} & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ q_{x,\,ng}^T & 0 & 0 & \cdots & q_{ng,\,ng} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{56}$$

Additional rows and columns are inserted to cater for the difference within the triple $\{md,\, nd,\, ng\}$. $Q(t_k)$ is symmetric and positive semi-definite and has the size $[nq \times nq]$. The column vector marked with a dashed box has the size $[nq \times (md - nd)]$ and the column vector marked with a dotted box has the size $[nq \times (nd - ng)]$. The cross terms q_{ij} for indices $j \neq i > 1$ are zero, that is, there is no covariance between $u(t_{k-i})$ and $u(t_{k-j})$.

The general case is easily specialized to the previous cases treated in Section 5.2 and Section 5.3. With for example for $md = nd = 0$ and $ng = 0$, (45)-(46) will result, by

$$Q = \begin{bmatrix} q_{x,\,x} & q_{x,\,0} \\ q_{x,\,0}^T & q_{0,\,0} \end{bmatrix}$$

Similarly, when $md = nd = 1$ and $ng = 1$, (56) is equivalent to the case with constant delay less than a period,

$$Q = \begin{bmatrix} q_{x,\,x} & q_{x,\,0} & q_{x,\,1} \\ q_{x,\,0}^T & q_{0,\,0} & 0 \\ q_{x,\,1}^T & 0 & q_{1,\,1} \end{bmatrix}$$

where the corresponding weight matrix is found in (50)-(51).

See Appendix E for the similar case of the delay at the process' side, found in Figure 5.1 and (41). For that case it holds that $Q_{23} \neq 0$ in general. The derivation of (55) is made in a similar way as that found in Appendix E.

5.5 Inserting the control signal

Finally, the loss with respect to the current control signal can be calculated by inserting the control law (35) into (54) in order to eliminate $u(t_k)$ from the equations. In Section 4.6 an expression for the closed loop was derived based on this method. The state update in (38) describes the evolution of the closed loop state vector. Here, on the other hand, the weight matrix is of interest. The scalar loss can be written:

$$J_x(t_k) = \tilde{x}^T(t_k)Q(t_k)\tilde{x}(t_k) + J_v(t_k)$$

$$= \begin{bmatrix} x_p(t_k) \\ U(t_k) \\ u(t_k) \end{bmatrix}^T Q(t_k) \begin{bmatrix} x_p(t_k) \\ U(t_k) \\ u(t_k) \end{bmatrix} + J_v(t_k)$$

For $J_v(t_k)$ see Section 6.1.1. The weight matrix should fit the state vector in (16), i.e.:

$$J_x(t_k) = x^T(t_k)Q_{cl}(t_k)x(t_k) + J_v(t_k) + J_w(t_k) \tag{57}$$

$$= \begin{bmatrix} x_p(t_k) \\ U(t_k) \\ x_c(t_k) \end{bmatrix}^T Q_{cl}(t_k) \begin{bmatrix} x_p(t_k) \\ U(t_k) \\ x_c(t_k) \end{bmatrix} + J_v(t_k) + J_w(t_k)$$

where the scalar $J_w(t_k)$ is an additional loss term due to the weighted noise following from $E[u^T(t_k)Q_{uu}u(t_k)]$ for some Q_{uu}, see Section 6.1.2. Q_{cl} has the size $[na \times na]$. The states of the controller are not punished. The matrix manipulation

$$Q_{cl}(t_k) = M^T Q(t_k) M$$

based on the transformation $\tilde{x}(t_k) = Mx(t_k)$ with

$$M = \begin{bmatrix} I_a & 0 & 0 \\ 0 & I_b & 0 \\ D_c M_p & E_c & C_c \end{bmatrix}$$

where the constant matrix M of the size $[nq \times na]$, does the conversion trick. The unit matrix I_a has the size $[np \times np]$ and I_b has a size of $[md \cdot nu \times md \cdot nu]$.

5.6 On the choice of weight matrix

Besides the process model, the choice of controller, the description of the timing behaviour, the noise variance, and the weight matrix $\bar{Q}$ affect the result of the calculations. The question is how to make a fair choice of the weights (and the noise covariance). The problem here is similar to that in LQ design. The choice of weights is based on knowledge of the system and rules-of-thumb. There is a relation between the $\bar{Q}_{11}$ and $\bar{Q}_{22}$ (and also $\bar{Q}_{12}$), which relates the punishment of the state with the punishment of the control signal.

A convenient way of assigning weights is the following setup. The first matrix can be assigned $\bar{Q}_{11} = C_p{}^T C_p$, which gives a loss function where the process output y is punished, rather than the state of the process, x_p. The continuous time cross matrix $\bar{Q}_{12}$ can be set to zero. Recall that $\bar{Q}$ must be positive definite. The weight on the control signal $\bar{Q}_{22}$ is however a lit-

tle bit more difficult to give good advice on. One option is to set it to the zero matrix. The performance is now a measure of the expected variance of the measured value.

The choice of $\bar{Q}$ above means that the control signal is viewed as totally unimportant. Another choice is to use unit matrices. For a real application this might not be a good idea, because a control signal usually has a more narrow working range than a control engineer would like it to have. If an optimal controller is designed by this choice of $\bar{Q}_{22}$, trouble with the limits is likely to occur. Of course, the performance can always be evaluated in this way, but if two design choices are to be compared, e.g. different controllers or different sampling periods, a relevant measure must of course be used. Another option would be to simulate the system or to calculate the state evolution from worst-case initial values, and try different relations between $\bar{Q}_{11}$ and $\bar{Q}_{22}$.

5.7 Discussion

In this section the loss function has been discretized. Two versions were proposed. In the first case the probe was inserted such that the delay came to belong to the controller, “cd”, and in the second case the delay was on the process’ side, “pd”. A third version called “nd”, for no delay, will be set up.

One thing seldom pointed out in research papers, is that one is free to select a loss function that is suitable for the circumstances. The loss function is the objective function during optimization and defines what is good (or optimal).

In Section 5.1, following the model in Figure 5.1 (top), the loss function “cd”,

$$\bar{J}_{cd} = E \lim_{T \to \infty} \frac{1}{T} \int_0^T \begin{bmatrix} x_p(t) \\ u(t - \tau(t)) \end{bmatrix}^T \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix} \begin{bmatrix} x_p(t) \\ u(t - \tau(t)) \end{bmatrix} dt \tag{58}$$

was suggested, see Appendix D for the derivation. It means that the control signal is tapped after the delay, at the entrance of the process. Another continuous time loss function was used in the derivation of “pd”. It is the one commonly used in optimal control theory,

$$\bar{J}_{pd} = E \lim_{T \to \infty} \frac{1}{T} \int_0^T \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix}^T \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{12}^T & \bar{Q}_{22} \end{bmatrix} \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix} dt \tag{59}$$

see Appendix E for the derivation in the general case.

Equation (59) and (58) will give similar discretized weight matrices, but not quite, since the cross terms between control signals of different age will be slightly altered. By combining the discretized submatrices for the case “pd”,

$$\{q_{x,\,x},\ q_{x,\,g},\ q_{x,\,u},\ q_{g,\,g},\ q_{g,\,u},\ q_{u,\,u}\}$$

then applying the discretization of Section 5.5, i.e. inserting the control signal; one will recognize (56), which is Q derived from (58). As a matter of fact, when the cross-term $\bar{Q}_{12}$ is zero, there is no difference. The derivation of the closed loop covariance will stay the same. The closed loop remains unaltered, it is only eavesdropped in another way.

A third alternative, “nd”, is to translate $\bar{Q}$ into Q assuming $\tau = 0$ even though $\tau > 0$ and expand Q to cater for the triple $\{md, nd, ng\}$ placing $\{Q_{11}, Q_{12}, Q_{21} = Q_{12}^T, Q_{22}\}$ in the “corners” of Q:

$$Q = \begin{bmatrix} Q_{11} & 0 & Q_{12} \\ 0 & 0 & 0 \\ Q_{12}^T & 0 & Q_{22} \end{bmatrix}$$

It will punish the state and only the most recent control signal. This can be viewed as a discrete time approach, with a preceeding translation. Compare this discretization approach with Nilsson (1998), where a discrete time controller to cope with delay jitter $\tau(t) < h$ is derived.

6 A performance measure

In this section the derivation of a quadratic performance measure will be finalized using the discretized process model and weight matrix derived in the previous sections. First, the state and measurement noise have to be discretized. However, the derived model requires a more elaborated approach, called jump linear system, since the closed loop is time-varying.

The relation between an input and an output signal can be characterized by a system norm. In control engineering a common system norm is the H_2-norm. This temporal norm can be defined both in the time and frequency domains for an LTI system and it is applicable also to MIMO-systems. The H_2-norm is defined by

$$\|G(s)\|_2 = \sqrt{\frac{1}{2\pi}\int_{-\infty}^{\infty} tr(G^H(j\omega)G(j\omega))d\omega} \Leftrightarrow \sqrt{\int_0^{\infty} tr(g^T(\tau)g(\tau))d\tau} = \|g(\tau)\|_2 \tag{60}$$

where $G(s) = C(Is - A)^{-1}B + D$ should be strictly proper (i.e. $D = 0$). $G^H(j\omega)$ is a constant complex matrix where H denotes the Hermitian transpose. The trace operator tr, is the sum of the main diagonal. It can be seen that the norm is defined by integrating a "squared" matrix over the whole frequency domain or over an infinite time horizon. This is clearly a measure of the energy contents in the dynamic system $G(s)$. Skogestad and Postlethwaite (1996) show how the optimal control problem can be rewritten and interpreted as a special case of optimizing an H_2-norm. A stochastic interpretation of the H_2-norm is that (60) is the integrated squared value of the output when the system is excited by white noise.

6.1 Covariance and loss of the state and measurement noises

The noise model is important for the performance measure, since the primary interpretation of the H_2-norm is that the dynamical system is "driven by noise". The noise can be seen as a way to specify and weight uncertainty entering the system. It can also be used to model variance in the process and sensors. It is difficult to give advice on how to select the noise covariance $\bar{R}_v$ and R_w. The ultimate idea is that there should be a good relation between the noise of the studied process and the model. The noise makes the model of the closed loop a stochastic process, even though there is no Markov chain modelled. During design a good idea is to try different values and simulate time plots of the system to get a feeling for the size and impact of the noise.

As before, let the loss function be written on the form

$$\bar{J} = \frac{1}{h}\int_h (x_p^T\bar{Q}_{11}x_p + x_p^T\bar{Q}_{12}u + u^T\bar{Q}_{12}^T x_p + u^T\bar{Q}_{22}u)dt \tag{61}$$

x_p and u will introduce the continuous time state noise v and discrete measurement noise w, with the variance $\bar{R}_v$ and R_w, respectively. Loosely speaking, just to provide an overview, the first term of (61) will give an additional term $1/h\int v^T\bar{Q}_{11}vdt$, when sampled. The fourth term will render an additional term $1/h \cdot w^TQ_{22}w$, when the control law, corrupted by noise from the measurement, is inserted. These terms are called J_v and J_w, respectively. By defining the noise to be uncorrelated for disjoint time intervals and uncorrelated with other signals (e.g. $E(x_p\bar{Q}_{12}w) = 0$), all other cross-terms vanish. It can be argued that this kind of noise is unre-

alistic, but the model approximation leads to a great simplification. Thus, it is favourable if this property holds for time-varying periods and delays, too.

Due to the correlation structure, the covariance for the noise terms Evv^T and Eww^T can be isolated, but the state covariance $Ex_p(t_k)x_p^T(t_k)$ must be found by solving a recurrence equation.

6.1.1 The state noise

$\bar{Q}$ was discretized into Q in Section 5, but the noise was neglected. The state noise must be discretized, and it is of interest to include the variance of the signal between the sampling instants, not only the variance at the sampling instants. The state noise will depend on the sampling period, which can be time-varying. The discretization of $v(t)$ was discussed in Section 4.5.7.

The state noise is present as the additional term J_v in e.g. (44), (50) and (54). The variance of $v(t)$ will not show up in the Q-matrix, but it will show up as this additional term:

$$J_v(t_{k+1}, t_k) = \frac{1}{t_{k+1} - t_k} E\left(\int_{t_k}^{t_{k+1}} v^T(t_{k+1}, t_k) \bar{Q}_{11} v(t_{k+1}, t_k) dt \right) \tag{62}$$

$$= \frac{1}{t_{k+1} - t_k} tr\bar{Q}_{11} \int_{t_k}^{t_{k+1}} \int_{t_k}^{t} e^{As} \bar{R}_v e^{A^T s} ds\, dt$$

$$= \frac{1}{t_{k+1} - t_k} tr\bar{Q}_{11} \int_{t_k}^{t_{k+1}} (t_{k+1} - s) e^{As} \bar{R}_v e^{A^T s} ds$$

$J_v(t_k)$ does not depend on the delay or arrival pattern of the control signal at the process. As mentioned before, the covariance of the continuous time noise, $\bar{R}_v$ is a design specification. The trace operator is a handy trick in this situation, and it will be used more in the sequel.

6.1.2 The measurement noise

It is assumed that $w(t_k)$ is a white noise stochastic process, which does not depend on absolute time or time intervals, again see Section 4.5.7. It can for example be used to model a noisy sensor, as an uncertainty entering the closed loop. Equation (57) contains the scalar $J_w(t_k)$, which is an additional loss term due to the weighted measurement noise:

$$J_w(t_k) = E[w^T(t_k) D_c^T Q_{uu} D_c w(t_k)] = tr(D_c^T Q_{uu} D_c R_w(t_k)) \tag{63}$$

This is found by inserting the control law. In Section 5.5, the loop was closed by using the transformation matrix M. In so doing, (63) will fall out. The submatrix Q_{uu} is the one corresponding to $u(t_k)$, i.e. $u^T(t_k) Q_{uu} u(t_k)$. Q_{uu} depends on nd and ng if the loss function J_{cd}, J_{pd} or J_{nd} is used.

6.2 Covariance in case of a constant sampling period

The familiar case with a constant period and zero delay is treated first, as an introduction to the more general approach. In this case a Markov chain is not needed, or it can be regarded as being degenerated to one state only.

Recall Figure 5.1, where the state of the process and the control signal of the closed loop are tapped and weighted, to calculate a scalar index by integration over time. In the case of constant h, the average over one sampling period is taken, since this is the inherited period of the sampled system, every such period will be equal. The system and input matrices Φ_{cl} and Γ_{cl} are constant $\forall k$. The composite noise vector $e(kh)$ is uncorrelated with the state $x(kh)$. The controller (17) with $u(kh)$ and measurement noise is substituted into the discrete time correspondence of J_k in (44), and the discrete time weight matrix Q is given by (45). Assume for simplicity that $\tau = 0$ and that the controller has no states, e.g. a gain only, such that Q does not have to be augmented ($x = x_p$ is the closed loop state vector and $Q = Q_{cl}$). This yields from (44) and (57),

$$\begin{aligned} J &= tr(E[x^T(kh)Qx(kh)] + J_v + J_w) \\ &= tr(QE[x(kh)x^T(kh)]) + tr(J_v) + tr(J_w) \\ &= tr(Q_{cl}S^\infty) + J_v + J_w \end{aligned} \tag{64}$$

The trace operator is applied to the left and right hand sides in the second equality; it is a useful trick in this situation, which admits a separation of Q_{cl} and S^∞, the steady state covariance of $x(kh)$, two pieces which can been found separately. The scalars J_v and J_w are defined by (62) and (63) both with and without the trace operator. The state covariance evolves governed by the closed loop state update (39), and can thus be found by the recurrence equation:

$$\begin{aligned} S(kh+h) &= Ex(kh+h)x^T(kh+h) \\ &= E\Phi_{cl}x(kh)x^T(kh)\Phi_{cl}^T + E\Gamma_{cl}e(kh)e^T(kh)\Gamma_{cl}^T \\ &= \Phi_{cl}S(kh)\Phi_{cl}^T + \Gamma_{cl}R_e(kh)\Gamma_{cl}^T \end{aligned} \tag{65}$$

The equilibrium value of the covariance matrix $S(kh) = E[x(kh)x^T(kh)]$, i.e. the value at an infinite time horizon $k \to \infty$, is denoted S^∞. Assume a convergence of $S^\infty < \infty$ when $k \to \infty$, then

$$S^\infty = \Phi_{cl}S^\infty\Phi_{cl}^T + \Gamma_{cl}R_e^\infty\Gamma_{cl}^T \tag{66}$$

gives the stationary value by straight forward matrix algebra. The expected noise variance $R_e(kh)$ is written R_e^∞ under stationary conditions. An alternative to a direct calculation is to update the recurrence equation in (65) until a sufficient convergence is detected. $S(kh)$ will converge to $S^\infty < \infty$, if and only if the closed loop is (quadratically) stable, cf. the discrete Lyapunov equation. The stability condition can also be expressed by stating that the eigenvalues of the closed loop system matrix should be less than one, $eig(\Phi_{cl}) < 1$. Both input v and w affect S^∞, but have nothing to do with the eigenvalues of Φ_{cl}.

The straight forward derivation above is easily extended to encompass a constant $\tau > 0$ (to relax assumption A2) with a constant h. In that case Q will have to be augmented to fit with $U(kh)$ and $x_c(kh)$, since $x \neq x_p$ will result. When $S^{\infty} < \infty$ is found, (64) gives the loss.

6.3 Stochastic jump linear system

Finally, the transition from a timeline into a Markov chain will be completed. The Markov chain was introduced in Section 3. The infinite sequence of sampling instances is modelled by a finite set of Markov states. The transition probability matrix captures the stochastic behaviour of the timeline. In this section we assume that the Markov chain is positive recurrent and aperiodic. This kind of system can be called a stochastic jump linear system. The loss at a particular time interval is determined by the actual period and the changes in the control signal during that interval, i.e. the corresponding discrete LTI model. Despite this stochastic jumping, the loss is calculated in analogy with the previous section; the outline of this section will be more top-down than the traditional approach. The treatment is not exhaustive and the reader might want to consult the references for further details and formal proofs. In order to set up a state covariance evolution similar to that of (65) but for a jump linear system, the so called *vector notation* combined with the *Kronecker product* is a useful mathematical tool.

6.3.1 The Markov chain

To repeat, in order to do a numerical analysis using constant sizes of matrices and vectors, these must be dimensioned for the worst-case combination of periods and delays. The constant md corresponds to the maximum value of nd for any time instant t_k of the timeline or state r of the Markov chain. Note that the number of states in the Markov chain is not equal to the number of distinct combinations of h and τ. It is the jitter (both in h and τ) that increases the number of states. An excessive delay does not have to increase the number or Markov states. The worst-case md that needs to be modelled, depends on the size of the periods during the delays. Figure 6.1 shows an example of a timeline and the corresponding piece of a Markov chain. The continuous time LTI model is discretized for each state of the chain.

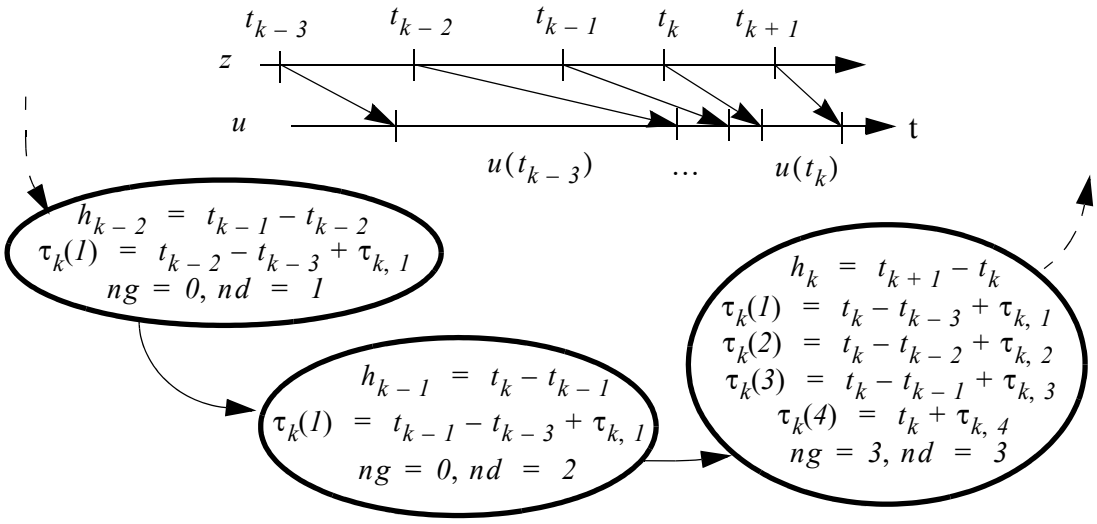


Figure 6.1 A sequence of relatively even sampling and bursty actuation instances illustrated by a timeline and the corresponding states.

6.3.2 Calculating the loss

The loss function depends on the closed loop model for every state and the state transition probability matrix P. Every Markov state r is associated with a sampling period h_r and a set of partial delays, $\{\tau_{r,0}, \ldots, \tau_{r,(ng)}\}$. Every state is also associated with a weight matrix $Q_{cl}(r)$ and additional loss due to the noise $J_v(r)$ and $J_w(r)$. These have previously been derived for a period and partial delays originally associated with a specific state, i.e. without the proper context of random or deterministic jumps. The jumping between states makes the system stochastic, and the expected behaviour can be characterized at equilibrium.

A (Markov chain) state-dependent (conditional) covariance S_i at time t_k is defined by

$$S_i(t_k) = E(x(t_k)x^T(t_k) \,|\, r(t_k) = i)$$

The loss can be written

$$\begin{aligned} J &= E \lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} [x^T(t_k)Q_{cl}(t_k)x(t_k) + J_v(t_k) + J_w(t_k)] \\ &= E \lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} [tr(Q_{cl}(t_k)x(t_k)x^T(t_k)) + J_v(t_k) + J_w(t_k)] \\ &= \sum_{i=1}^{nr} [tr(Q_{cl}(i)S_i^\infty) + \pi_i^\infty J_v(i) + \pi_i^\infty J_w(i)] \end{aligned} \tag{67}$$

At the third equality, the domain changes from the timeline with arguments of t_k to the Markov chain with states i, when stationarity is considered. Recall that the subindex i refers to a Markov state, not to a time instant, i.e. $X_i = X(r(t_k) = i)$ at the time t_k for some X. The state-independent covariance (the total covariance without weights), is the sum over all nr states:

$$S^\infty = \sum_{i=1}^{nr} S_i^\infty \tag{68}$$

In analogy with the previous section, the evolution of the covariance can be written:

$$\hat{S}(t_{k+1}) = \hat{A}\hat{S}(t_k) + \hat{B} \tag{69}$$

where the matrices $\hat{A}$, $\hat{B}$ and the column vector $\hat{S}(t_k)$ are given below. $\hat{A}$ has the size $[nr \cdot (na)^2 \times nr \cdot (na)^2]$, $\hat{B}$ has the size $[nr \cdot (na)^2 \times 1]$ and $\hat{S}(t_k)$ has the size $[nr \cdot (na)^2 \times 1]$. Equation (69) will converge to an equilibrium under certain circumstances, i.e. the limes of $\hat{S}(t_k)$ equals $\hat{S}^\infty$ when $k \to \infty$. The steady state conditional covariance S_i^∞ is then retrieved by unfolding the column vector $\hat{S}^\infty$. Equation (67) will then render the loss.

The state probability vector π_i is found in (67) and in the matrices $\hat{A}$ and $\hat{B}$. A property of a positive recurrent and aperiodic Markov chain is that its state probability vector converges according to the update function $\pi(t_{k+1}) = \pi(t_k)P$. The convergence of (69) is of course important. Similar to the previous section, it can be shown that the system is stable if $eig(\hat{A}) < 1$, see e.g. Costa and Fragoso (1993) — see further in Section 7.1. It remains to show how $\hat{S}^\infty$ is found, before the loss can be computed.

6.3.3 The Kronecker and vector notation

Let X be of the size $[b \times c]$. The vector operation $vec(X)$ is equal to stacking all c columns of the matrix X on top of each other, which result in a column vector of the size $[bc \times 1]$. The vector notation is a simple rearrangement of the columns of a matrix. The reverse operation requires a specification of b (or c). Let the matrix V be of the size $[b \times c]$ and the matrix H of the size $[p \times q]$. The Kronecker product is defined by

$$V \otimes H = \begin{bmatrix} v_{11}H & v_{12}H & \dots & v_{1n}H \\ v_{21}H & v_{22}H & \dots & v_{2n}H \\ \dots & \dots & \dots & \dots \\ v_{m1}H & v_{m2}H & \dots & v_{mn}H \end{bmatrix}$$

which is of the size $[bp \times cq]$. This is quite a simple combination of the elements. It is used to express a permutation of products using matrix notation, instead of repetition over a sum. A drawback is that the size of the resulting matrix becomes quite large. Several calculation rules for the Kronecker product can be setup, but the important calculation rule that will be used here, is

$$vec(VXH) = (H^T \otimes V)vec(X)$$

with matrices V, H, X of appropriate sizes (see Lancaster (1969) for a proof). Let $H = V^T$, and it is clear that the Lyapunov equation $VXV^T - X + Q = 0$ is conveniently written using vector notation and the Kronecker product, thus subsequently solved for X by ordinary matrix manipulation.

6.3.4 Calculating the state-dependent covariance

The presentation in this section is quite brief. For a more thorough treatment see especially Nilsson (1998), from whom a lot of the theory in this section is taken, but see also Costa and Fragoso (1993), and e.g. Ji and Chizeck (1990).

The update of the conditional state covariance follows the state evolution and one way to write this is

$$\tilde{S}_i(t_{k+1}) = \sum_{i=1}^{nr} p_{ij}\pi_i(t_k)E[\Phi_{cl}(i)x(t_k)x^T(t_k)\Phi_{cl}^T(i) + \Gamma_{cl}(i)e(t_k)e^T(t_k)\Gamma_{cl}^T(i) \,\big|\, r(t_k) = i] \tag{70}$$

$$= \sum_{i=1}^{nr} p_{ij}E[\Phi_{cl}(i)\tilde{S}_i(t_k)\Phi_{cl}^T(i) + \pi_i(t_k)\Gamma_{cl}(i)R_e^{\infty}(i)\Gamma_{cl}^T(i) \,\big|\, r(t_k) = i]$$

where $\tilde{S}_i(t_k) = \pi_i(t_k)S_i(t_k)$ and $R_e^{\infty}(i) = E(e(t_k)e^T(t_k) \,|\, r(t_k) = i)$ are the covariance of the state and noise vectors. $\Gamma_{cl}(i)$ and $\Phi_{cl}(i)$ denote the dynamic descriptions specific for state i.

The column vector in (68) with all states augmented is defined by

$$\hat{S}(t_k) = \begin{bmatrix} vec(\tilde{S}_1(t_k)) \\ \dots \\ vec(\tilde{S}_{nr}(t_k)) \end{bmatrix}$$

The system matrix for the first order difference equation (68) becomes

$$\hat{A} = (P^T \otimes I_1) diag(A_i) \tag{71}$$

with $A_i = E(\Phi_{cl}(i) \otimes \Phi_{cl}(t))$, and the probability transition matrix P. A_i has the size $[(na)^2 \times (na)^2]$. Further

$$\hat{B} = (P^T \otimes I_2)(diag(\pi_i^{\infty}(t_k)) \otimes I_2)\hat{G}$$

with

$$\hat{G} = \begin{bmatrix} vec(G_1) \\ \dots \\ vec(G_{nr}) \end{bmatrix}$$

and in turn, $G_i = E(\Gamma_{cl}(i) R_e^{\infty}(i) \Gamma_{cl}^T(i))$ defines the update. $\hat{G}$ has the size $[nr \cdot (na)^2 \times 1]$. The unit matrices $\{I_1, I_2\}$ have the sizes $[(na)^2 \times (na)^2]$ and $[na \times na]$ respectively. The (block) diagonal of a square matrix or scalar ω_i is defined by

$$diag(\omega_i) = \begin{bmatrix} \omega_1 & \dots & 0 \\ \dots & \dots & \dots \\ 0 & \dots & \omega_{nr} \end{bmatrix}$$

independent of i. The sum of every state variance renders the total variance

$$S(t_k) = \sum_{i=1}^{nr} \pi_i(t_k) S_i(t_k) = \sum_{i=1}^{nr} \tilde{S}_i(t_k)$$

6.4 Periodic systems

The periodic Markov chain was introduced in Section 3.3. If the Markov chain is periodic, the state probability vector π in (9) does not converge in an asymptotic manner, but exhibits a non-converging cyclic pattern. Let $\theta = 1, 2, \dots, nr$ denote the periodicity of the chain with the special case $\theta = 1$ for an aperiodic Markov chain. It is clear that $\theta \leq nr$ since the period is defined as the greatest common divisor of possible paths going from and returning to the same arbitrary state. The periodicity of the state probability vector can be described by $\pi^{\infty}(t_{k+\theta}) = \pi^{\infty}(t_k)$ for some k in steady state. If $\theta > 1$, the period is longer than a single sampling period h. θ could be called the meta-period in comparison to h. Note that equidistant sampling is not assumed.

It is possible to describe a steady state behaviour of a periodic system, provided the loss (especially the state covariance) is bounded. *Lifting* and *switch decomposition* are primarily associated with multi-rate systems, systems whose sampling and control periods are different. If all

transitions are deterministic, a lifting procedure can be used to expand the periodically time-varying system into a time-invariant one, by augmented states describing intermediate dynamics, see Bittanti et al. (1991). This can be applied when the ratio of e.g. the sampling period and control period is a rational number, in order to obtain a finite dimensional discrete time representation. The time-invariant formulation can be used for control synthesis, to find a fixed feedback gain controller. For a fully deterministic periodic system with $\theta = 2, 3, \ldots$, the Markov chain becomes an unnecessary theoretical overhead. However, in this report a periodic system will be treated as a special case of the presented theory. All transitions in a periodic chain do not necessarily have to be deterministic, cf. Figure 3.4. In fact, the framework developed for random jump linear systems is applicable also when the Markov chain is periodic. The transition probability matrix is degenerated, but the Markov property still holds. The state independent covariance will converge (seen over a meta-period) on the same premises as that of a completely aperiodic Markov chain.

If there is no jump linear system $nr = 1$, and the periodicity (from a continuous time viewpoint) is $\theta = 1$, the discretized loss function can be written:

$$J = E \lim_{N \to \infty} \frac{1}{N} \sum_{k=0}^{N-1} [x^T(t_k) Q_{cl} x(t_k) + J_v + J_w]$$

where x contains all states of closed loop, and Q_{cl}, J_v, J_w do not depend on time. Since all periods are uniform, it is sufficient to take the expected value over one period. For a θ-periodic chain, with deterministic jumps only, the loss can be written more generally:

$$J = E \lim_{N \to \infty} \frac{1}{N} \sum_{k=0}^{N-1} \frac{1}{\theta} \sum_{\kappa=k}^{k+\theta-1} [x^T(t_\kappa) Q_{cl}(t_\kappa) x(t_\kappa) + J_v(t_\kappa) + J_w(t_\kappa)]$$

All θ-periods are uniform and it is sufficient to take the expected value over one such period, instead of the average over N number of θ-periods. It must be assured that the covariance $E[x(t_\kappa)x^T(t_\kappa)]$ with $\kappa = k, \ldots, k + \theta - 1$ converges, since it is found by a recurrence equation. By applying a lifting procedure, the θ-periodic system can be written as a 1-periodic (or aperiodic from a sampled time viewpoint) equivalence, (Bittanti et al., 1991). Consider the closed loop system (39),

$$x(t_{\kappa+1}) = \Phi_{cl}(t_\kappa) x(t_\kappa) + \Gamma_{cl}(t_\kappa) e(t_\kappa) \tag{72}$$

with deterministically periodic time-varying matrices $\Phi_{cl}(t_{\kappa+\theta}) = \Phi_{cl}(t_\kappa)$ and $\Gamma_{cl}(t_{\kappa+\theta}) = \Gamma_{cl}(t_\kappa)$ for some reference instance κ. Let $K = 0, 1, \ldots$, and an equivalent time-invariant formulation of (72) can be written

$$\hat{x}(t_{K+1}) = \hat{\Phi}_{cl}\, \hat{x}(t_K) + \hat{\Gamma}_{cl}\, \hat{e}(t_K) \tag{73}$$

with the lifted state vector $\hat{x}(t_K) = x(t_{\kappa+K\theta})$ and lifted input vector

$$\widehat{e}(t_K) = \begin{bmatrix} e(t_{\kappa + K\theta}) \\ e(t_{\kappa + K\theta + 1}) \\ \dots \\ e(t_{\kappa + (K+1)\theta - 2}) \\ e(t_{\kappa + (K+1)\theta - 1}) \end{bmatrix}$$

and with the constant system and input matrices

$$\widehat{\Phi}_{cl} = \Phi_{cl}(t_{\kappa + \theta - 1}) \cdot \Phi_{cl}(t_{\kappa + \theta - 2}) \cdot \dots \cdot \Phi_{cl}(t_\kappa) \tag{74}$$

$$\begin{aligned} \widehat{\Gamma}_{cl} = [& \Phi_{cl}(t_{\kappa + \theta}) \cdot \dots \cdot \Phi_{cl}(t_{\kappa + 1})\Gamma_{cl}(t_\kappa) \\ & \Phi_{cl}(t_{\kappa + \theta}) \cdot \dots \cdot \Phi_{cl}(t_{\kappa + 2})\Gamma_{cl}(t_{\kappa + 1}) \\ & \dots \\ & \Phi_{cl}(t_{\kappa + \theta})\Gamma_{cl}(t_{\kappa + \theta - 2}) \\ & \Gamma_{cl}(t_{\kappa + \theta - 1})] \end{aligned}$$

of which the latter is a row vector. System (73) is stable if and only if (74) is asymptotically stable according to Bittanti et al. (1991). The expected covariance of the closed loop is

$$\widehat{S}(t_K) = E(\widehat{x}(t_K)\widehat{x}^T(t_K)), K = 0, 1, \dots$$

and (73) satisfies a linear recursion

$$\widehat{S}(t_{K+1}) = \widehat{\Phi}_{cl}\,\widehat{S}(t_K)\,\widehat{\Phi}_{cl}^T + E\left[\widehat{\Gamma}_{cl}\widehat{e}(t_K)\widehat{e}^T(t_K)\widehat{\Gamma}_{cl}^T\right]$$

Here it is assumed that the input noise vector $\widehat{e}(t_K)$ is uncorrelated with the state $\widehat{x}(t_K)$. The recursion above converges provided the eigenvalues of $\widehat{\Phi}_{cl}$ are inside the unit disc. More generally, the θ-periodic chain can also have stochastic jumps, cf. (12), and the lifting procedure is not enough. Even though a periodic system might have a reference instance κ, when modelled as a Markov chain, the time instant of a particular state of the chain, is not in general deterministic, cf. (13).

7 Stability and robustness

In the previous section the performance was studied based on white noise disturbance acting as input signals. Before it becomes relevant to speak about performance, stability must be guaranteed. The performance measure is related to a concept called *mean square stability*, which is used for stochastic systems with parametric jumps. In many engineering disciplines, energy is typically modelled using quadratic terms to express the variance of the system. The term *second moment stability* is also used (Ji et al., 1991). Besides performance and stability, robustness is an interesting property for the control engineer. In this context the robustness against the timing properties are of interest. For example, it is valuable to investigate if the system will become unstable or not if the actual delay (jitter) differs from the nominal delay by a specified amount. One can speak about stability margin (stability robustness) and performance margin (performance robustness).

7.1 Stability concepts, second moment stability

Stability in the sense of Lyapunov is based on the well recognized energy principle, a principle that is applicable also to stochastic systems. The principle of energy conservation can be used to model systems that can store energy; the momentary input energy does not have to equal the momentary output energy at every moment in time. For such a *dissipative system*, the amount of energy that is dissipated from the system over some time period, can not exceed the amount of energy that has been supplied and stored in it, i.e. it cannot create energy. Many real-world systems are dissipative, and this is understood by considering the physical interaction of the system with its environment. Energy is typically dissipated in the form of heat or other kinds of radiation, friction and similar phenomena. It is also a matter of choosing where the boundaries of the system should be drawn.

The studied sampled-data system in Figure 4.5 is converted into a discrete time system. For a discrete time LTI system, a useful stability criterion is that the spectral radius must be inside the unit disc. Consider the following discrete time system

$$x(t_{k+1}) = \Phi_{cl}(r(t_k))x(t_k) + \Gamma_{cl}(r(t_k))e(t_k) \tag{75}$$

where $r(t_k)$ is the state of the Markov chain at time t_k for the sampling instance k. The Markov chain has the initial state r_0 and the dynamic system has the initial state x_0 for time t_0 at $k = 0$. In order for a system to be stable, the state evolution of a system should be such that the energy, on the average, decreases with time and approaches an equilibrium where it ultimately stops decreasing. The decrease of energy does not have to be strictly monotone. Stability in the sense of Lyapunov is usually classified in four categories depending on the state trajectories of (75). First, the system can be *unstable* and the trajectory goes to infinity. Second, the system can be *stable* and the trajectory remains within a bounded region. Third, the system can be *asymptotically stable* and the trajectory approaches the equilibrium, conveniently placed in the origin. Finally, the system can be *exponentially stable*: the trajectory approaches the stable origin in a time which can be described by an exponential function.

Ji et al. (1991) define stochastic stability and mean square stability in the following way. The system in (75) is *stochastically stable* (at the origin) if for every initial state there exists a finite number $M(x_0, t_0) > 0$ such that (with $\|x\|^2 = x^T x$):

$$E\sum_{k=0}^{\infty}\left\|x(t_k\,|\,x_0,\,r_0)\right\|^2 < M(x_0,\,r_0) \tag{76}$$

Stochastic stability implies *mean square stability*:

$$E\left\{\lim_{k\to\infty}\left\|x(t_k\,|\,x_0,\,r_0)\right\|^2\right\} = 0$$

Further, Ji et al. (1991) show that mean square stability is equivalent to stochastic stability for systems of type (75) (with $e(t_k) = 0,\ \forall k$), something which is not true in general. Similar to the traditional stability classes in the sense of Lyapunov, exponential mean square stability can also be defined. A collective term is *second moment stability*. Lyapunov stability of a non-autonomous system is similar to stability of an autonomous one, but depends also on the initial conditions, (Khalil, 1992). Costa and Fragoso (1993) have a similar definition for mean square stability, where the norm of the state covariance should be bounded. A noise input vector $e(t_k)$, uncorrelated with the state vector $x(t_k)$ is allowed, (75). In Costa and Fragoso (1993) it is also shown that the condition $eig(\hat{A}) < 1$, i.e. that the maximum spectral radius of the matrix $\hat{A}$ in (71) should be inside the unit disc, is equivalent to mean square stability of a jump linear system. The test condition $eig(\hat{A}) < 1$ is attractive since it is a simple-to-test condition similar to discrete systems without jumps and it is valid both for autonomous and non-autonomous systems.

7.2 Coupled matrix equations

An alternative to a direct calculation of the covariance, is to formulate the closed loop system as a set of coupled quadratic Lyapunov functions. The coupling is described by the transition probability matrix P, which makes the total system stochastic in nature, hence one can talk about stochastic Lyapunov functions.

A linear matrix inequality, LMI, is an equation system formulated by matrices. A main advantage by formulating a problem as an LMI, is that it can be solved by efficient numerical methods derived from convex optimization theory. See Sanfridson (2003) for an overview on LMIs in control. Besides the feasibility problem of checking stability of the feedback loop, the formulation of an LMI is handy for optimization. In the context of optimal control, finding optimal gains of a feedback controller is a direct application, as in e.g. Xiao et al (2000). Efficient methods for solving LMIs using convex optimization were developed in the late 1980's. The formulation of an LMI can here be regarded as being based on the principle of energy. Consider the following autonomous discrete time jump linear system:

$$x(t_{k+1}) = \Phi_{cl}(r(t_k))x(t_k) \tag{77}$$

where the LTI description for the instance k depends on the Markov chain.

Let the Markov chain state dependent covariance be defined by

$$S_i(t_k) = E[x(t_k)x^T(t_k)I_i(t_k)] \tag{78}$$

where the Markov state indicator function (or 1-function) is defined by

$$\begin{cases} I_i(t_k) = I, & r(t_k) = i \\ I_i(t_k) = 0, & r(t_k) \neq i \end{cases}$$

The expected value of the indicator function depends on the state jumps, and can with some abuse of notation be written

$$E[I_j(t_{k+1}) \,|\, I_i(t_k)] = \sum_{i=1}^{nr} Prob\{r(t_{k+1}) = j \,|\, r(t_k) = i\} E[I_i(t_k)]$$

$$= \sum_{i=1}^{nr} p_{ij} E[I_i(t_k)]$$

When time progresses, $S_i(t_k)$ is updated according to the following recursion

$$S_j(t_{k+1}) = \sum_{i=1}^{nr} p_{ij} \Phi_{cl}(i) S_i(t_k) \Phi_{cl}^T(i) \tag{79}$$

This is repeated $\forall j \in 1, \ldots, nr$. A small example with two states in Figure 7.1 shows how it works. The expected evolution of the covariance of Markov state number one ($j = 1$), becomes

$$S_1(t_{k+1}) = E[x(t_{k+1}) x^T(t_{k+1}) I_1(t_{k+1})]$$

$$= \Phi_{cl}(1) E[x(t_k) x^T(t_k) I_1(t_{k+1})] \Phi_{cl}^T(1) + \Phi_{cl}(2) E[x(t_k) x^T(t_k) I_1(t_{k+1})] \Phi_{cl}^T(2)$$

$$= p_{11} \Phi_{cl}(1) S_1(t_k) \Phi_{cl}^T(1) + p_{21} \Phi_{cl}(1) S_2(t_k) \Phi_{cl}^T(1)$$

and similarly for $S_2(t_{k+1})$. The total expected covariance is $S(t_k) = S_1(t_k) + S_2(t_k)$. This also holds for the special case when the example chain is periodic, with $p_{11} = p_{22} = 0$ and $p_{12} = p_{21} = 1$.

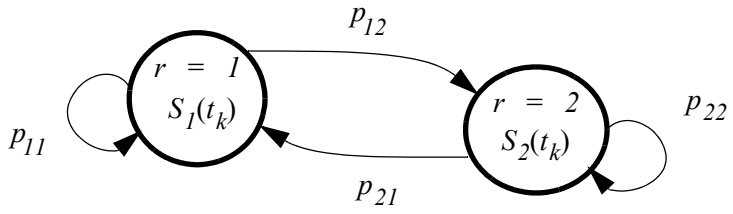


Figure 7.1 Example with two states.

There are three alternative formulations to (79) which do not necessarily give the same result, but are equivalent when investigating the mean square stability (Costa and Fragoso, 1993). These model the evolution of the state dependent covariance in different ways. One alternative formulation is given by Ji and Chizeck (1990), who start out with a stochastic Lyapunov function instead. Let the unknown variables S_r be symmetric and positive definite matrices (S_r now has another interpretation than above) and assume a quadratic and stochastic Lyapunov candidate function

$$V(x(t_k),\, r(t_k)) \;=\; x^T(t_k) S_r(t_k) x(t_k) \tag{80}$$

which is continuous in x, and has a minimum point at the origin $V(0,\, r(t_k)) \;=\; 0$. The idea of Lyapunov's (direct) stability criteria is that the derivative of a generalized energy function such as (80), should have the opposite sign to the function itself for the origin to be a stable point. A quadratic Lyapunov candidate is sufficient for LTI systems, both time continuous and discrete. However, for continuous time systems having delay (a retarded differential equation), additional terms usually are appended or another structure is used, see for example Kharitonov (1999) for an overview.

The expected difference $E[\Delta V]$ should be negative if $E[V]$ is positive. The condition on the expected difference becomes with (77) combined with (80), and $\forall i$, $i \in \{1,\, \ldots,\, nr\}$:

$$E[\Delta V] \;=\; E[V(x(t_{k+1}),\, r(t_{k+1}) \,|\, x(t_k),\, r(t_k)) - V(x(t_k),\, r(t_k))] \tag{81}$$

$$= \; x^T(t_k) \left[\Phi_{cl}^T(i) \left(\sum_{j=1}^{nr} p_{ij} S_j(t_k) \right) \Phi_{cl}(i) - S_i(t_k) \right] x(t_k) \;=\; -x^T(t_k) W_i x(t_k) < 0$$

and it can be concluded that V is indeed a Lyapunov function for some given $W_i > 0$, see Ji and Chizeck (1990) for the proof. Recall that the sum of p_{ij} over all j is equal to one. Thus, the difference is negative if the lumped matrix inequality composed by

$$\Phi_{cl}^T(i) \left(\sum_{j=1}^{nr} p_{ij} S_j^{\infty} \right) \Phi_{cl}(i) - S_i^{\infty} < 0 \tag{82}$$

$\forall i$, is satisfied. Provided it is bounded, S_i^{∞} denotes the steady state covariance, when $k \to \infty$. The coupled nr number of equations of (82) are lumped to one LMI and solved numerically by tools developed for convex optimization. This is a feasibility problem.

7.3 Robustness to the timing properties

LMIs can also bee used to define and validate conditions for stability robustness or performance robustness, when there is uncertainty in the model. A relatively simple method, which is a continuation of the previous section, will be used here. The delay and period jitter are of course the modelled uncertainty compared to the nominal model used for nominal design.

Assume a sampled-data system as described previously. Let the loss be described by

$$\bar{J} \;=\; E \int_0^{\infty} \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix}^T \bar{Q} \begin{bmatrix} x_p(t) \\ u(t) \end{bmatrix} dt \tag{83}$$

It is assumed that $x_p(t) \to 0$ and $u(t) \to 0$ when $t \to \infty$ from an arbitrary initial state vector $x_p(t = 0)$ and $u(t = 0)$ respectively, such that $\bar{J}$ is bounded. There is no noise input modelled

here. Discretizing (83) and the process followed by a closing of the loop with a discrete time controller gives an autonomous system $x(t_{k+1}) = \Phi_{cl}x(t_k)$ and the loss function

$$J = E\sum_{k=0}^{N-1} x^T(t_k)\tilde{Q}_{cl}(t_k)x(t_k) + J_N = E\sum_{k=0}^{N-1} J_k + J_N \tag{84}$$

where the final end-value J_N is dropped once again. Note the difference compared to (42): this is the total energy and not the average. The discretized weight matrix is denoted $\tilde{Q}_{cl} = \tilde{Q}_{cl}^T \geq 0$. It is clear from (83) that $\tilde{Q}_{cl}(i) = h_i Q_{cl}(i)$, with a change from time to Markov state.

In (81), the objective is to prove that $E\{\Delta V\} < 0$, i.e. that the virtual energy decreases. The condition can be strengthened by a further restriction on the rate of change, by the condition, $\forall i$:

$$E\{\Delta V\} < -J_k$$

$$x^T(t_k)\left[\Phi_{cl}^T(i)\left(\sum_{j=1}^{nr} p_{ij}S_j(t_k)\right)\Phi_{cl}(i) - S_i(t_k)\right]x(t_k) < -x^T(t_k)\tilde{Q}_{cl}(i)x(t_k)$$

This requirement is typically more difficult to satisfy, than the stability requirement. The positive semi-definite weight matrix can be interpreted as a performance margin or robustness requirement. Define

$$S = \sum_{i=1}^{nr} \pi_i^\infty S_i^\infty$$

and consider the following optimization problem

$$\begin{aligned} &\text{minimize} && f(S) = x_0^T S x_0 \\ &\text{such that} && S_i^\infty > 0,\ \forall i \\ & && \Phi_{cl}^T(i)\left(\sum_{j=1}^{nr} p_{ij}S_j^\infty\right)\Phi_{cl}(i) - S_i^\infty < -\tilde{Q}_{cl}(i),\ \forall i \end{aligned} \tag{85}$$

The smallest upper bound on J is found by minimizing the function $f(S)$, i.e. $J \leq x_0^T S x_0$. The initial vector of the process $x_p(t=0)$ is incorporated in x_0. The system can be said to be robust against the modelled timing properties, if the conditions are satisfied. This can be used as a performance measure to evaluate different timing properties.

7.4 Discussion

Three calculation techniques have been described. The two first are based on the fact that the weight and the covariance in $Ex^TQ_{cl}x$ can be separated into $(Exx^T)Q_{cl}$ using the trace operator. The state covariance Exx^T evolves with $k \to k+1$ as governed by the closed loop state update. The covariance can thus be found by a recurrence equation, letting the iteration con-

tinue until a sufficient convergence is detected. This is a slow method and is only mentioned as an alternative; an advantage might be the memory usage. Another faster calculation method is based on ordinary matrix manipulation assuming an equilibrium. This also works for jump linear systems by writing sums and products using vector notation and Kronecker products. J_v and J_w are appended to cater for the intersample behaviour and the sampling noise. These two techniques use noise as a way to specify and weight uncertainty entering the system. However, it is difficult to give advice on how to select the noise covariance $\bar{R}_v$ and R_w. The third calculation technique is based on LMIs. This formulation does not rely on noise as input, but instead on an arbitrary initial state vector. The same setup is cast as a standard convex optimization problem, which offers another view of the system. The discretization of the loss function is decoupled from the problem of finding the steady state covariance. A common stability criterion for a jump linear system is called mean square stability. In essence it says that as long as the state covariance is bounded, the loop is stable. The stability does not depend on the noise level.

A large number of Markov states is necessary if a variety of periods and long delays are to be modelled. In such a case, it might be necessary to impose approximations to limit the number of states. The size of the problem and the computational cost depend of course on the number of states. One way to approximate is to remove a Markov state that can be considered similar to another one. Another approach is to impose a time quanta to limit the input space. A third way is to use a constant period or a constant delay. The decision should depend on the problem at hand.

8 The effect of a single timing property

The effect of the different timing properties will be investigated by the aid of seven example systems. These are described in Appendix A. An overview of the example systems is found in Table 8.1.

Table 8.1 A List of example systems, pairs of processes and controllers.

Nb	Process	Controller
1	2nd order damped	PI-controller
2	DC-motor	LQ controller with delay compensation
3	3rd order process	LQG controller
4	Non-minimum phase	LQ controller
5	1st order with delay	Smith predictor
6	Double integral	Deadbeat controller
7	Inverted pendulum	Minimum variance controller

The example systems represent either common or intrinsically difficult processes and controllers. In this section, the timing properties will be studied one by one in isolation. The timing properties not currently studied will be absent or held fixed. The rest of this section is divided into the following five subsections:

(1) A constant sampling period,

(2) constant delay,

(3) delay jitter,

(4) sampling jitter, and

(5) transient error, skip.

For each sample system, the controller is designed based on its corresponding nominal period, which is denoted h_0, if not otherwise stated. For system 2 and 5, there is also a compensation of a nominal delay $\tau_0 > 0$. The time unit is custom defined.

8.1 The constant sampling period

Assume an equidistantly sampled system without any control delay. This corresponds to a combination of assumptions A1 and A2 in Section 2.3. The control design is based on the actual period, h. The question here is what a good choice of h would be. This is discussed in e.g. Åström and Wittenmark (1997), who give rules-of-thumb for selecting h for different types of systems and controllers. In Lennartson (1985) the problem is studied based on a quadratic loss function, and this line is pursued here.

The choice of period is strongly connected to the capability or bandwidth of the processing and communication resources. A change of periods usually alters the steady state utilization of a processing unit or communication channel in a straight forward way. But there are also other

implications by choosing an extreme value of h, such as the implementation issue of achieving well-conditioned numerical computations.

In the comparison below, the weight matrix is taken to be

$$\bar{Q} = \begin{bmatrix} \frac{1}{c} C_p^T C_p & 0 \\ 0 & c\|C_p\|_2 \end{bmatrix} \tag{86}$$

for a constant $c \in \{1/10, 10\}$ and where C_p comes from each process (18).

The relative loss as a function of the sampling period is shown in Figure 8.1 for the choice $c = 1/10$. The calculated loss of each system has been scaled by its average value over the range of periods $J_{rel} = J/mean(J)$, and thereafter shifted such that the loss equals one at the nominal period, $J_{rel}(h_0) = 1$. The actual period is chosen as $h = \alpha h_0$ with $\alpha \in [1/2, 8]$, for each system respectively, see also Table A.1 in Appendix A. This might make the longest periods become a bit longer than what is advisable for some example systems.

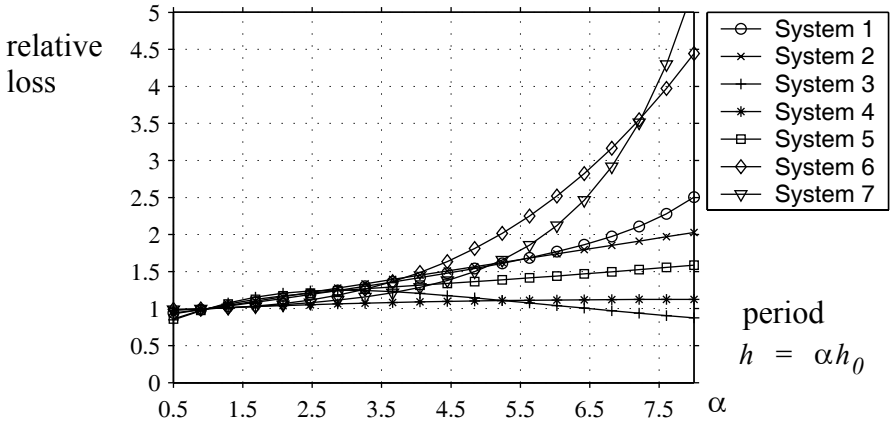


Figure 8.1 Relative loss as a function of an increasing sampling period, $c = 1/10$.

It can be seen that systems 1, 6 and 7 explode with increasing h, i.e. when the sampled controller closes the loop more seldom, leaving the process in an open loop. System 6 (double integral) and system 7 (inverted pendulum) are open loop unstable. System 1 (2nd order process) is open loop stable, but has a PI-controller, where the internal state depends on h.

In Figure 8.2 the setup is the same as it was in the previous figure. The difference is that the actual period is a factor of the nominal period, $h = kh_0$ with $k \in [1/8, 2]$ instead, for each system respectively. The lines are scaled and shifted as described above.

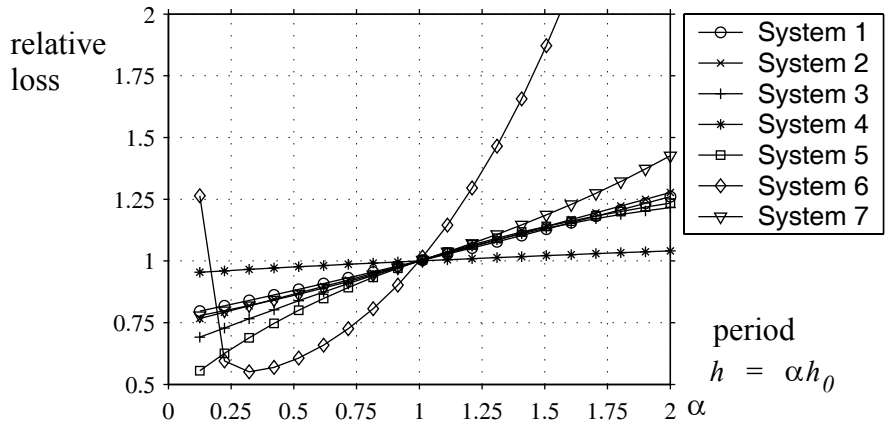


Figure 8.2 Relative loss as a function of a decreasing sampling period, $c = 1/10$.

Not surprisingly, the deadbeat controller (system 6) is very sensitive to the choice of sampling period. The other systems, which can be said to have corresponding continuous time controllers, decrease further with h at various rates.

The graph of Figure 8.3 is calculated in exactly the same way as Figure 8.1, except that $c = 10$. Note that the loss is never negative, the relative loss has been scaled and shifted. The shape of each line is approximately the same as Figure 8.1 and systems 1, 6 and 7 still go to infinity for an increasing h. There is at least one notable difference compared to Figure 8.1: the loss of the DC-motor and discrete optimal feedback controller (system 2) goes down instead of up. This also seems to be the case for the non-minimum phase process and discretized optimal controller, system 4. When it comes to interpreting the result, the choice of weight matrix is not unimportant.

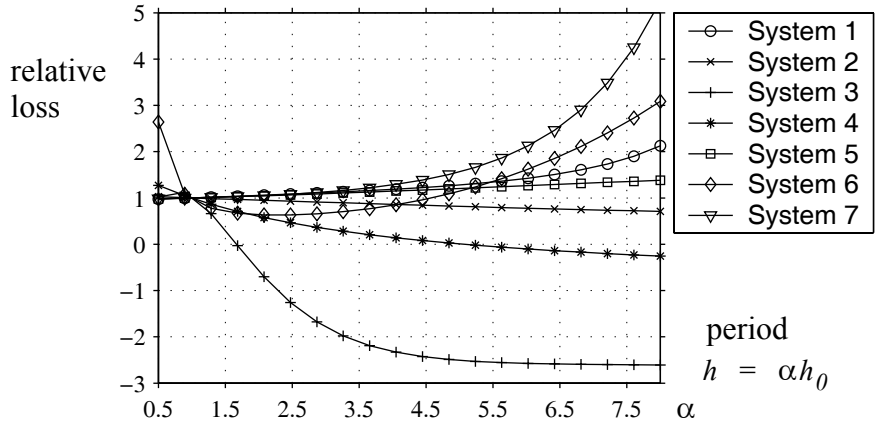


Figure 8.3 The same setup as previous, relative loss as a function of the period, but here with $c = 10$.

Finally, Figure 8.4 is calculated in exactly the same way as Figure 8.2, except that $c = 10$. The most striking difference is that of system 3, the 3rd order process with an LQG-controller, having a clearly visible peak at about $\alpha = 0.75$. Again, both system 2 (and 4) differ from

Figure 8.2 by going up for decreasing h instead of down (and vice versa). It can be noted that these three systems all have optimal controllers, i.e. the design method is based on a trade-off between the variance of the state and the variance of the control signal.

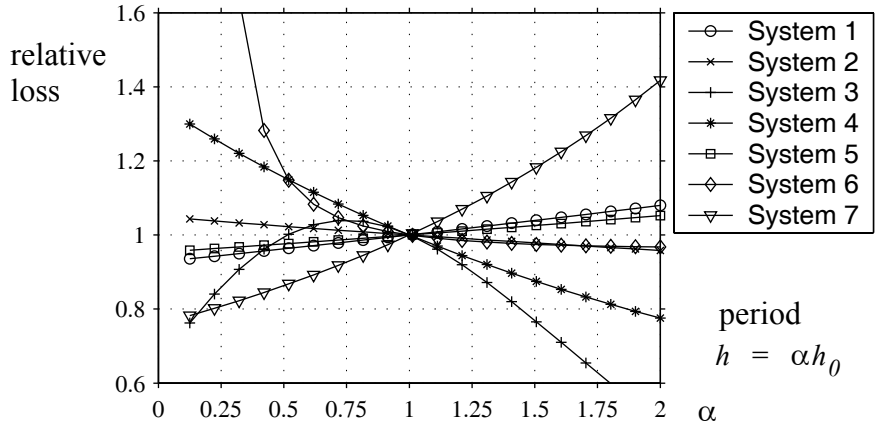


Figure 8.4 The same setup as previous, relative loss as a function of the period, with $c = 10$.

8.2 A constant delay in the closed loop

In this section the effect of a constant delay will be studied. Assume constant nominal h_0 and a constant $\tau > 0$, i.e. assumptions A1 and A3 combined. Each system has a different h_0, see Table A.1 in Appendix A. For systems 2 and 5, there is already a compensation of a nominal delay $\tau_0 > 0$. The loss function is weighted by $\bar{Q} = I$.

Figure 8.5 shows the effect of static delays for the sample systems. The actual delay τ is a factor of the nominal period, $\tau = \alpha h_0$ with $\alpha \in [0.05, 0.2]$. For each system, the loss has been scaled and shifted such that the lines coincide for the shortest delay.

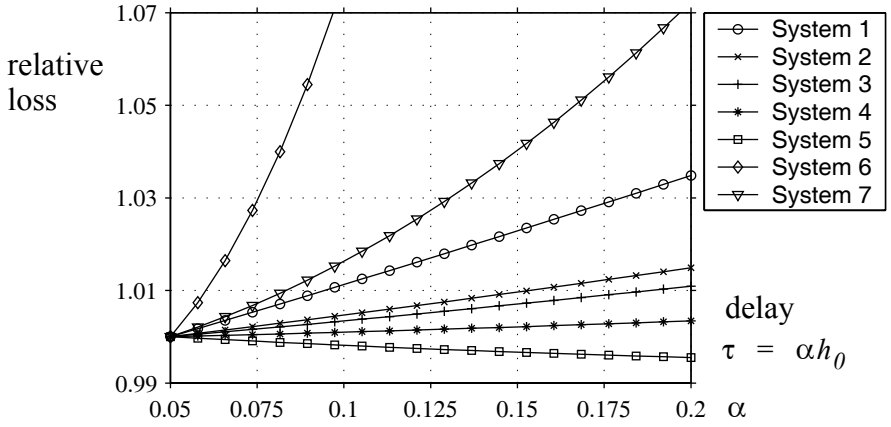


Figure 8.5 Relative loss as a function of constant delay.

Figure 8.6 is similar to the previous figure, but with longer delays. The most sensitive systems, number 6 and 7, are now unstable. The actual delay τ is a factor of the nominal period, $\tau = \alpha h_0$ with $\alpha \in [1, 3]$. In this case the controller in system 5 does not compensate for a nominal delay of $\tau = 5h_0$ neither in Figure 8.5 nor Figure 8.6.

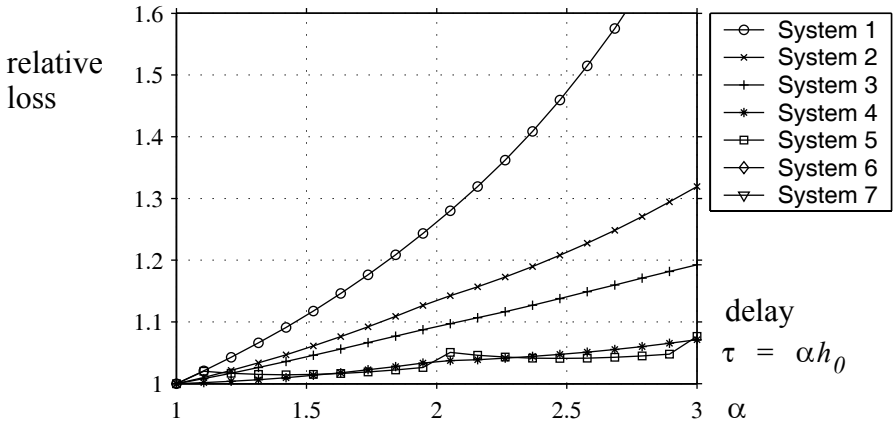


Figure 8.6 Relative loss as a function of constant delay.

8.3 Delay jitter

In this section the delay jitter is studied, while the period is held constant at the respective nominal period h_0. The delay jitter is modelled by a Markov process. The delay takes on discrete values between and including the extreme pair $\{(1/4)h_0, (3/4)h_0\}$, that is $max(\tau) < h_0$ at all times, compare with assumption A4 in Section 2.3. Let the split $s = 1, \ldots, 10$ be equal to the number of points in the distribution. This is also equal to the number of Markov states.

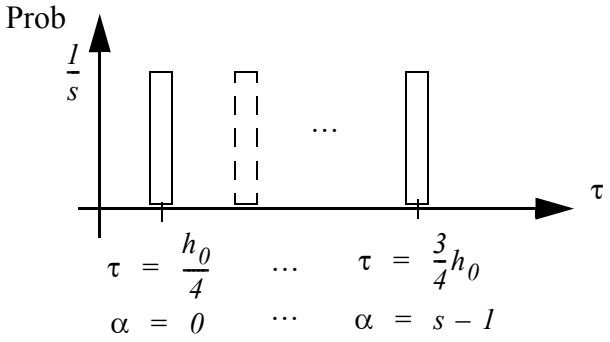


Figure 8.7 Discrete probability density function.

The delays can be written as $\tau(\alpha) = (1/4)h_0 + \alpha/(s-1)(1/2)h_0$ with $\alpha = 0, \ldots, s-1$, see Figure 8.7. The exception is $s = 1$, when the delay is defined to be $\tau = h_0/2$, i.e. there is no jitter. The transition probability matrix of the size $[s \times s]$ is written:

$$P = \frac{1}{s}\begin{bmatrix} 1 & \dots & 1 \\ \dots & \dots & \dots \\ 1 & \dots & 1 \end{bmatrix}$$

The state probability vector becomes $\pi^{\infty} = (1/s)\begin{bmatrix} 1 & \dots & 1 \end{bmatrix}$. Thus, the average delay is kept at $h_0/2$ independent of s by the jitter construction.

Figure 8.8 shows an interesting result. The relative loss have been scaled and shifted such that all lines start at $s = 1$ where the relative loss is equal to one, which corresponds to a constant delay of $h_0/2$. The value is highest for the split $s = 2$, i.e. when the delay changes between just the two extreme values, then gradually decreases asymptotically, with an increasing split. This suggests that the worst type of uniform delay jitter, assuming that the average delay is constant, is the jumping between two extreme values. System 7 is affected the most. System 6 with the deadbeat controller, is unstable at all times and is not shown. The lines for systems 2, 3, 4 and 5 almost coincide in the plot — these systems are hardly affected by the delay jitter.

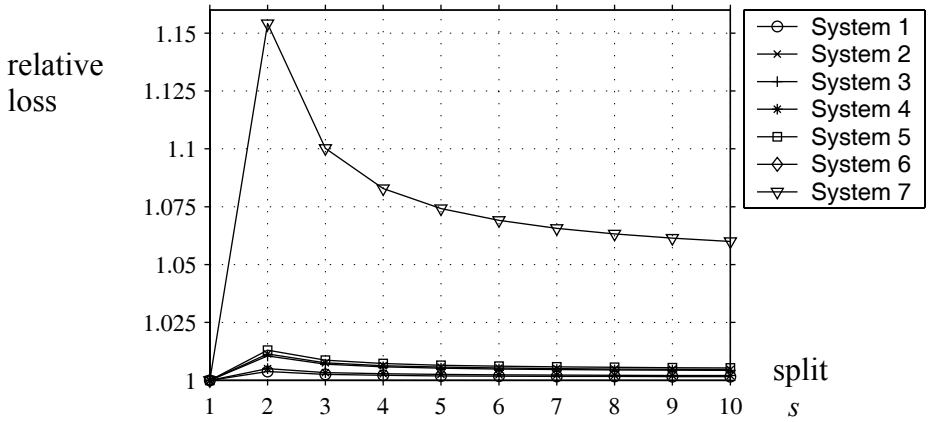


Figure 8.8 Delay jitter: relative loss as a function of delay jitter for each sample system.

8.4 Sampling period jitter

In this section, a time-varying sampling period will be studied and the delay will be held constant. The jitter is defined similarly to the previous section. The change in period is modelled by a Markov chain with $s = 1, \dots, 10$ states. The average period will be the nominal period h_0 for each system respectively, and the extreme values are $\{(3/4)h_0, (5/4)h_0\}$ for any choice of the split s. The corresponding periods are $(3/4)h_0 + \alpha/(s-1)(1/2)$, $\alpha = 0, \dots, s-1$, except for $s = 1$ where there is no jitter. For simplicity it is assumed that the probability of changing to a new state is uniform, and since $max(\tau) < min(h_0)$, there is no causality condition needed to make allowance for. For simplicity system 5 has in this case a nominal delay of five periods, rather than a nominal delay of $5h_0$.

The result plotted in Figure 8.9, is quite similar to the previous result on delay jitter, in Section 8.3. The scaling of the relative performance in Figure 8.9 is made in exactly the same way as in Figure 8.8, i.e. the result of each system has been divided by its mean value over the $s = 10$ experiments. The maximum values occur at $s = 2$, when the jitter changes between the two extreme values only. Thereafter, the loss decreased towards what seems to be an

asymptote. The lines for systems 2, 3, 4 and 5 are almost impossible to distinguish in the figure. System 7 (inverted pendulum) is now stable ($\tau = 0$). System 1, the 2nd order process with a PI-controller, seems to be more sensitive for variations in the period than variations in the delay. The one most sensitive to the jitter is system 6 (the double integrator with deadbeat controller).

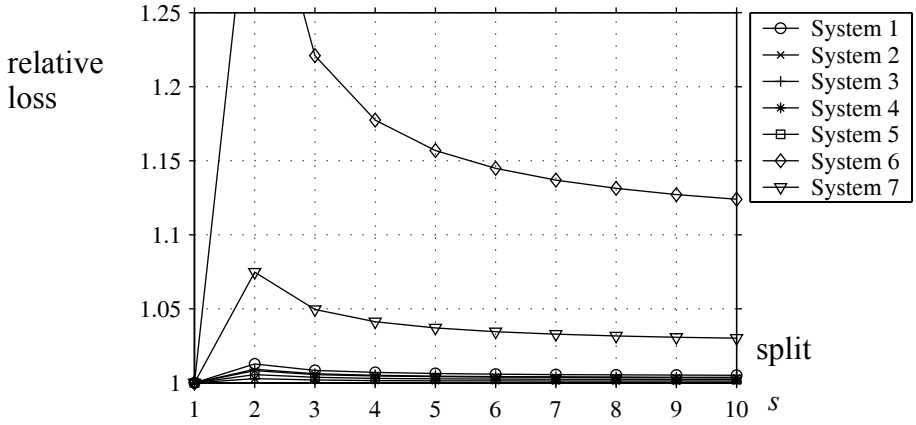


Figure 8.9 Sampling jitter: relative loss as a function of uniform period jitter for each sample system.

8.5 Transient error

In this section a vacant actuation will be studied. The period of each example system is held constant at h_0. The transient error occurs at the output side of the controller, i.e. the control signal is not delivered to the actuator for some reason. The delay grows larger at the highest possible rate, then holds and immediately ceases within a moment. A difference compared to delay jitter, is the sporadic nature of transients. State 2 corresponds to a delay of h_0 and state 3 to a delay of $2h_0$, i.e. once a transient error has occurred, it takes $2h_0$ before the error is recovered in a deterministic sequence, and the execution returns to a normal state with $\tau = 0$. The transition probability matrix is

$$P = \begin{bmatrix} 1-p & p & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

where $1 - p$ is the probability of not having an error, i.e. the normal state. The Markov chain is aperiodic if $p \neq 1$.

Figure 8.10 shows the performance as a function of p. System 6 is unstable for approximately $p > 0.01$ and has been left out from the figure. Next, system 7 is as expected quite sensitive too. System 1 is also rather sensitive compared to the other systems.

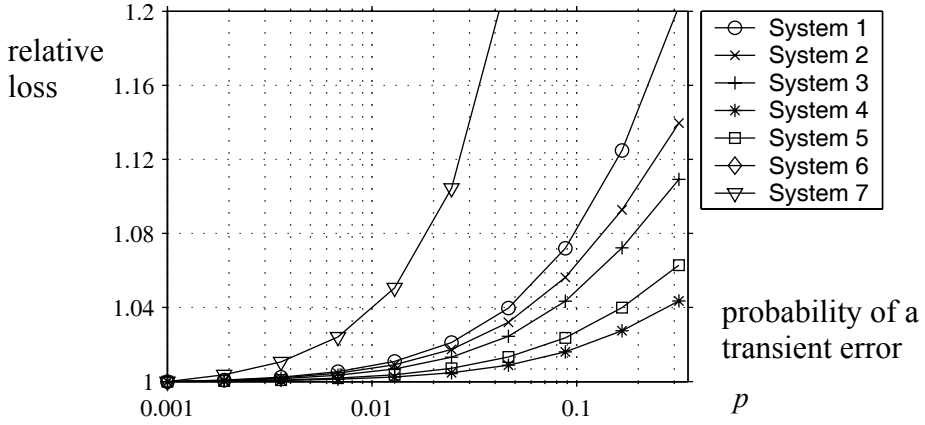


Figure 8.10 Loss as a function of the probability of a transient error, $J(p)$, for all sample systems.

8.6 Discussion

In Table 8.2 the result for each timing property has been collected. For each type of timing property, the systems have been ordered according to the perceived severity (how much a system changes compared to the other systems), with credits 7 down to 1. Credit 7 means that the system is highly affected. The table ranks the systems against each other, using data from Figure 8.1, Figure 8.9, Figure 8.5, Figure 8.8, and Figure 8.10 respectively. Not surprisingly, system 6 and 7 are the worst in all respects, and system 4 is the least sensitive.

Table 8.2 Impact of the timing properties.

System	Period, constant	Period, jitter	Delay, constant	Delay, jitter	Transient error	Total
1	*5*	*5*	*5*	*1*	*5*	**21**
2	*4*	*3*	*4*	*4*	*4*	**19**
3	*1*	*4*	*3*	*3*	*3*	**14**
4	*2*	*1*	*2*	*2*	*1*	**8**
5	*3*	*2*	*1*	*5*	*2*	**13**
6	*6*	*7*	*7*	*7*	*7*	**34**
7	*7*	*6*	*6*	*6*	*6*	**31**
Total	**28**	**28**	**28**	**28**	**28**	**140**

A comparison of this kind has to be interpreted cautiously. The result in Section 8.1 showed that the choice of weight matrix cannot be ignored. The two different weights gave totally different impressions on the sensitivity to the period for some of the systems. This might be less of a problem if two very similar systems are compared. One might be tempted to draw the conclusion that if a system is sensitive to one timing property, it is in general sensitive to the other properties too. This will be investigated more carefully in the following section.

9 Applications of the performance measure

In this section, more complex situations will be studied. The effect of the different timing properties will be investigated by studying the example systems described in Appendix A. Combinations of timing properties will be explored. Some of following experiments are typical for the codesign of computer control systems, and others represent interesting special cases. In the following subsections, these topics will be studied:

(1) Constant period vs. constant delay.

(2) Constant delay vs. delay jitter.

(3) A comparison between sampling and delay jitter.

(4) A comparison between input and output jitter.

(5) The worst and the best types of delay jitter.

(6) Deliberate delay with compensation.

(7) The value of making skips.

(8) Random and periodic jitter.

(9) Periodic schemes.

The numbers above correspond to the numbering of the subsections below. Again, the time unit is customer defined.

9.1 Constant period vs. constant delay

The loss as a function of both constant period and constant delay can for example help a system designer to make a trade-off in the real-time scheduling. An example is static cyclic scheduling, where the jitter typically is small or another situation where the influence of jitter is judged to be of minor importance. In the following, $\bar{Q} = I$ and $\bar{R}_v = I$ are some of the input parameters.

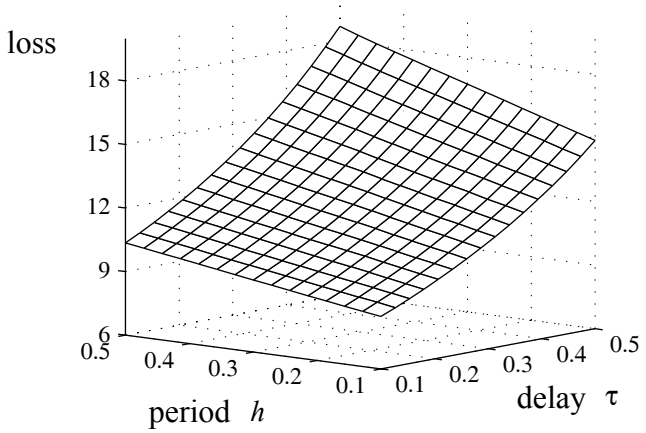


Figure 9.1 Combinations of constant periods and constant delays, system 1.

The result is best visualized in three dimensions. The gross features are shown in Figure 9.1 for system 1, the 2nd order process with a PI-controller. The change of loss due to a change of h is less sensitive than a change of τ. The shape of the three dimensional plot could be approximated by a tilted plane, i.e. a linear approximation would do fine.

The result using example system 6 instead, i.e. the inverted pendulum with a deadbeat controller, is shown in Figure 9.2. The shape of the three dimensional plot does no longer resemble a plane. The loss increases for increasing τ. However, for a constant delay (e.g. $\tau = 0.3$), the loss actually decreases to a minimum for an increasing h.

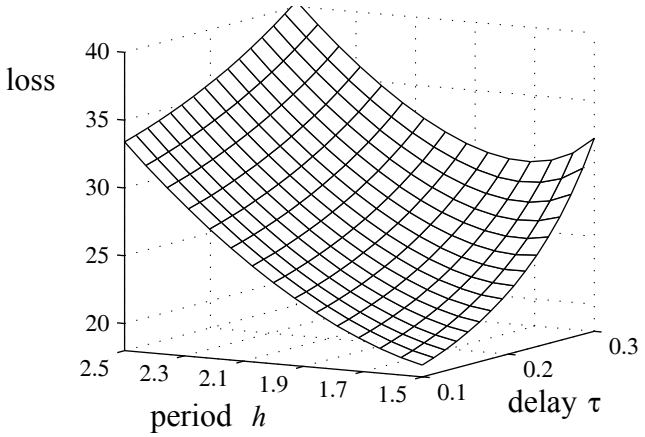


Figure 9.2 Combinations of constant periods and constant delays, system 6.

9.2 Constant delay vs. delay jitter

Constant delay will be compared to delay jitter in this section. Like the previous experiment, this is also a typical trade-off problem. One way to remove delay jitter is to literally soak it up by a buffer. However, the actual average delay will increase and the question is if this is better or worse than having a delay jitter.

Jitter can be characterized by a probability density distribution, which here for convenience is fully specified by a single parameter suitable for plotting. Consider the following transition probability matrix,

$$P = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{87}$$

The delay varies around a nominal delay of $\tau_0 = \alpha h_0$, plus or minus a delay $\Delta\tau = w\tau_0$, where $\alpha \in [0, 0.5]$, $w \in [0.1, 0.9]$ and h_0 is the nominal period. Example system 2, the LQ-controller that compensates for a delay of $h_0/2$, is used in this experiment.

In Figure 9.3 the loss due to delay jitter $\tau = \tau_0 \pm \Delta\tau$ is visualized. The loss corresponding to a constant delay $\tilde{\tau} = \tau_0 + \Delta\tau < h_0$, has been subtracted first. That delay represents the shortest delay a buffer would need to remove the delay jitter. The surface is on the negative side, that is, with an increasing $\Delta\tau$ (and τ_0) it pays off to accept jitter, since the average delay is reduced.

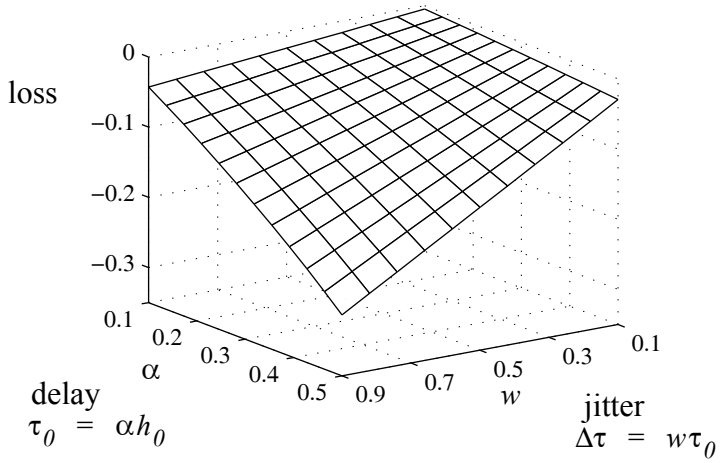


Figure 9.3 Difference in loss because of delay jitter $\tau = \tau_0 \pm \Delta\tau$ compared (subtracted) to a constant delay $\tilde{\tau} = \tau_0 + \Delta\tau < h_0$.

9.3 A comparison between sampling and delay jitter

An interesting experiment is to compare two kinds of jitter, for example sampling jitter and delay jitter. A problem is to set up a fair comparison. In order to simplify the setup, the constraint A4 in Section 2.3 is used, i.e. $max(\tau) < min(h)$ is enforced. All periods and delays will be related to the system's nominal period, h_0. System 4, the non-minimum phase system with an LQ-controller, is used in this example with $\bar{Q} = I$ and $\bar{R}_v = I$.

Consider P in (87) again. The same variation is used for both cases. With a constant h_0, the delay varies around a nominal delay of $\tau_0 = h_0/2$, plus or minus a delay $\Delta\tau = wh_0$, with $w \in [0.1, 0.4]$, such that the average delay remains at τ_0. Likewise, with a constant $\tau_0 = h_0/2$, the period varies around the nominal h_0, plus or minus $\Delta h = wh_0$, such that the average period remains h_0.

The result in Figure 9.4 shows that the sampling jitter is more detrimental than delay jitter in this case. The difference is very small for this example system, cf. Table 8.2 in Section 8.6.

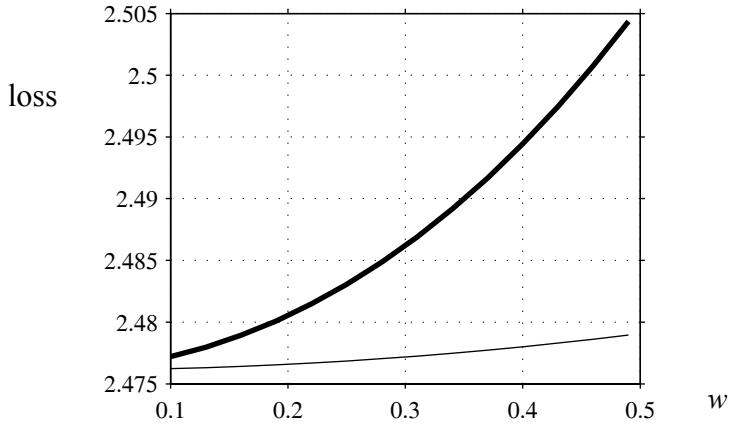


Figure 9.4 Loss vs. sampling jitter $h = h_0 \pm w h_0$ (bold) and delay jitter $\tau = \tau_0 \pm w h_0$ (thin).

9.4 A comparison between input and output jitter

An experiment similar to the previous one, is to compare input jitter (sampling jitter) and output jitter (actuation jitter). In case of input jitter, the delay is changed such that it cancels any output jitter perfectly. Vice versa, in case of output jitter, the input has a uniform sampling period. System 4 is used in this experiment.

Consider the following probability transition matrix

$$P = \begin{bmatrix} 1-2p & p & 0 & p & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1-2p & p & 0 & p & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1-2p & p & 0 & p & 0 \end{bmatrix} \tag{88}$$

with the parameter $p \in [0,\ 1/2]$. The corresponding Markov chain is visualized in Figure 9.5 and its states are explained in Table 9.1. The jitter is again a jumping between two extreme values (a long delay or sampling period and one short respectively), as described in the figure and table.

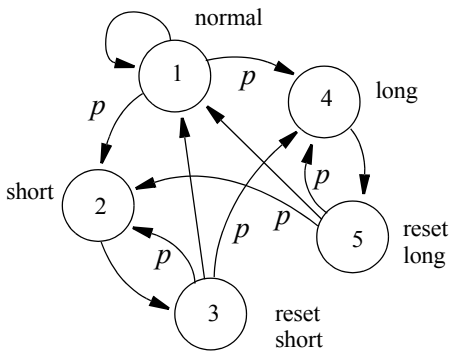


Figure 9.5 Markov chain for the experiment with input and output jitter.

The "nominal" delay is $\tau_0 = h_0/2$ and jitter is described by $\Delta\tau = w\tau_0$ with a dimensionless magnitude of $w \in [0.1, 0.3]$, which gives a delay switching described by $\tau = \tau_0 \pm \Delta\tau$. Similarly $\Delta h = w(h_0/2)$ which in turn gives a switching $h = h_0 \pm \Delta h$. Thus, the magnitude of the jitter is the same in both cases.

Table 9.1 Summary of states, input and output jitter.

		Output jitter		Input jitter	
State nb	**Function**	**h**	**τ**	**h**	**τ**
1	normal	h_0	τ_0	h_0	τ_0
2	short	h_0	$\tau_0 - \Delta\tau$	$h_0 - \Delta h$	τ_0
3	short reset	h_0	$\tau_0 + \Delta\tau$	$h_0 + \Delta h$	$\tau_0 + \Delta h$
4	long	h_0	$\tau_0 + \Delta\tau$	$h_0 + \Delta h$	τ_0
5	long reset	h_0	$\tau_0 - \Delta\tau$	$h_0 - \Delta h$	$\tau_0 - \Delta h$

Let $J(\Delta\tau)$ denote the loss as a function of output jitter and $J(\Delta h)$ the loss of the input jitter. The relative loss is here defined by

$$J_{rel} = \frac{J(\Delta h) - J(\Delta\tau)}{J(\Delta h \,|\, p = 0,\, w = 0.1)}$$

The result shown in Figure 9.6 reveals that the output jitter is worse than the input jitter, when it comes to performance degradation. However, the difference is rather small for this example system, and could be considered negligible in a practical situation.

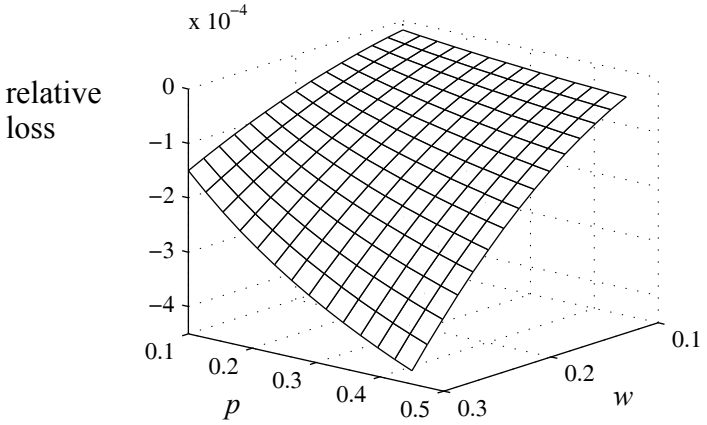


Figure 9.6 Relative difference between input jitter and output jitter as a function of the relative magnitude w and the probability p.

9.5 The worst and the best types of delay jitter

What kind of jitter distribution will give the worst performance? This is a very interesting question, both from a theoretical point of view but also from a practical one. An idea is that, knowing the worst kind may simplify analysis and synthesis. If the performance is adequate even for the worst type of jitter, the performance can only be better for other types. It thus becomes an upper bound. The average delay is indeed important, thus a fair way to formulate the problem is to hold the average delay constant and characterize the shape of the distribution.

The searching for a worst or best distribution of delay jitter will here be made by brute force. One technique suitable for combinatorial problems where the structure is unknown, is simulated annealing. A pseudo code listing is found in Figure 9.7.

```
old_value = low_or_high;
Tcur = Tstart;
while Tcur > Tfinal
     for k = 1 to maxtry
          old_mch = mch;
          ind = pick_random();
          mch = change_mch(mch,ind);
          value = calc_energy(mch);
          valid = is_change(value-old_value,Tcur);
          if valid
               old_value = value;
          else
               mch = old_mch;
          endif
     endfor
     Tcur = Tcur*Trate;
endwhile
```

Figure 9.7 Pseudo code for a simulated annealing algorithm. The abbreviation `mch` means Markov chain.

The tuning parameters are: the start temperature (`Tstart`), the final temperature (`Tfinal`) and the cooling rate (`Trate`). At each temperature level, at most `max_try` number of tries are made in this way: a new modified transition matrix P is picked and changed randomly using functions `pick_random` and `change_mch` (each time, a row is changed in random, but there are alternative ways to do this). The loss is calculated by `calc_energy`. If it results in a decrease and in case a minimum value is searched for, it will be accepted. If instead it results in an increase, a random variable that depends on the current temperature (`Tcur`) will determine if the change is valid or not. The lower the temperature, the smaller the chance of accepting a change in the "wrong" direction. A clear drawback of this method is that it does not help asserting that an optimal solution has been found. Thus, the result has to be interpreted cautiously. The simulating annealing needs a tuning of temperatures and the method of changing parameters to yield an outcome which is satisfactory.

System 7 used in this example has a nominal period h_0. The delay jitter is modelled as a discrete random variable controlled by a Markov chain. The delay takes on values between and including the extreme pair $\{(1/4)h_0, (3/4)h_0\}$ with the split $s > 1$. The split is equal to the number of points in the distribution and also equal to the number of Markov states. The average delay is kept at $h_0/2$ by the jitter construction.

From an arbitrary initial transition probability matrix for a fixed s, two final transition probability matrices are seen in Figure 9.8 and Figure 9.9, for a (local) minimum and maximum

respectively. In Figure 9.8 there are $s = 2$ states and the figure reveals that if the state switches often, the loss is less than it would be otherwise. In Figure 9.9 there are $s = 4$ states. The bars are highest at the edges. Recall that the row sum must equal one. The simulated annealing has been interrupted prematurely to yield results that are intermittent, but show the general direction.

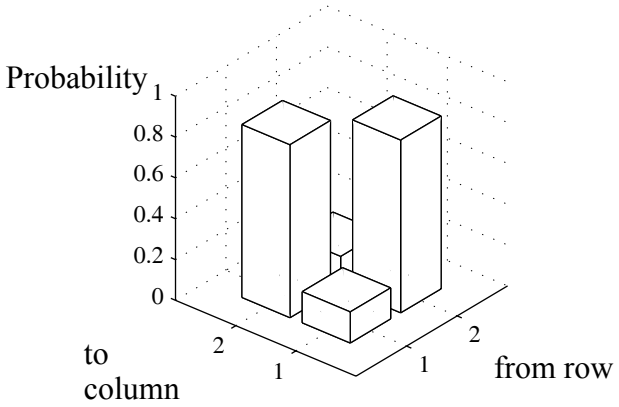


Figure 9.8 Final transition probability matrix, P. Searching for the minimum loss with the split $s = 2$.

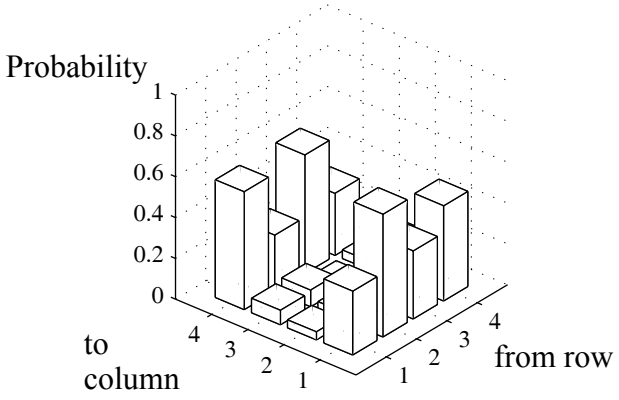


Figure 9.9 Final transition probability matrix, P. Searching for the maximum loss with $s = 4$.

9.6 Deliberate delay with compensation

If it is relatively easy to compensate for a constant delay, then why bother about an excessive constant delay if it can be handled? A constant delay could for example be created by a buffer which also can make control delay jitter disappear. The cost of adding and compensating for a constant delay will be studied in this section. System 2 is used in this experiment, with $\bar{Q} = I$ and $R_w = I$. The controller compensates for a delay of $\tau' = h_0/2$. As before the control delay is switching between low and high values, with $\Delta\tau = h_0/4$. Table 9.2 shows the five

different configurations. In case 5, the delay is over-compensated, i.e. its highest possible delay $(\tau'/2 + \Delta\tau)$ corresponds to τ'.

Table 9.2 Five cases of delay configurations.

Case nb	Actual delay or jitter	Compensated	Average delay
1	none, $\tau = 0$	$\tau_0 = 0$	0
2	constant, $\tau = \tau'$	$\tau_0 = 0$	$h_0/2$
3	constant, $\tau = \tau'$	$\tau_0 = \tau'$	$h_0/2$
4	jitter, $\tau = \tau' \pm \Delta\tau$	$\tau_0 = \tau'$	$h_0/2$
5	jitter, $\tau = (\tau'/2) \pm \Delta\tau$	$\tau_0 = \tau'$	$h_0/4$

The result is shown in Figure 9.10. The best is of course to have no delay at all as in case 1, and the worst is to have a large delay, case 4. Case 3 is lower than case 4, i.e. a constant delay is less worse than jitter having an average value equivalent to the constant delay. In case 5, the average delay is lower than in case 4, which can help explain a slightly lower value, despite the over-compensation.

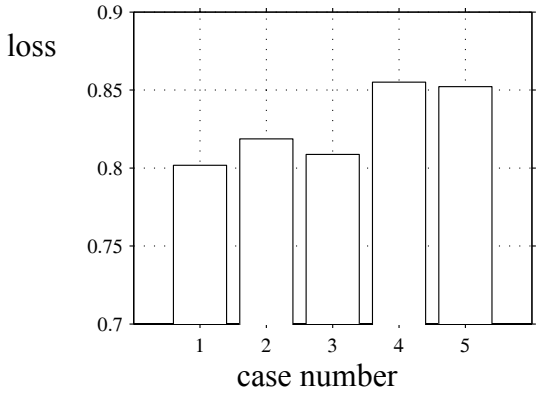


Figure 9.10 Delay compensation for five different configurations.

9.7 The value of making skips

In this section the value of making a skip will be compared to the effect of an increased period. A skip is a very simple idea and has probably been used in many computer control systems to counteract temporary overload. The skip is in essence a delay. With skip it is here meant that the control signal is temporally discarded, but any internal states of the controller are updated as before — in this report the delay is modelled to be situated between the controller and the

process. The control output can be held constant in case of a lost control signal; this does not require any computation by the elementary output function and can be seen as a prediction.

A skip can be frequently occurring as a deliberate way to compute the control law, or it can be very infrequent, e.g. as a last resort in case of temporary overload or delivery omission.

The inverted pendulum, system 7, is used here to study the effects of a lost control signal. Consider the following transition probability matrix

$$P = \begin{bmatrix} 1-p & p & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

where p is the probability that a skip will occur. If a skip occurs, it takes two sampling periods before the normal conditions are restored.

In Figure 9.11 two loss functions are plotted. One is the loss as a function of the probability p of a skip. The loss increases for increasing p; in this case $h = h_0 = 0.1$. The other loss function shows the loss as a function of the period but without skips. These are independent of p and drawn as horizontal lines. The possibility of skips and the choice of sampling period are interconnected, since they both are related to the utilization and eventual overload of a processing or communication unit.

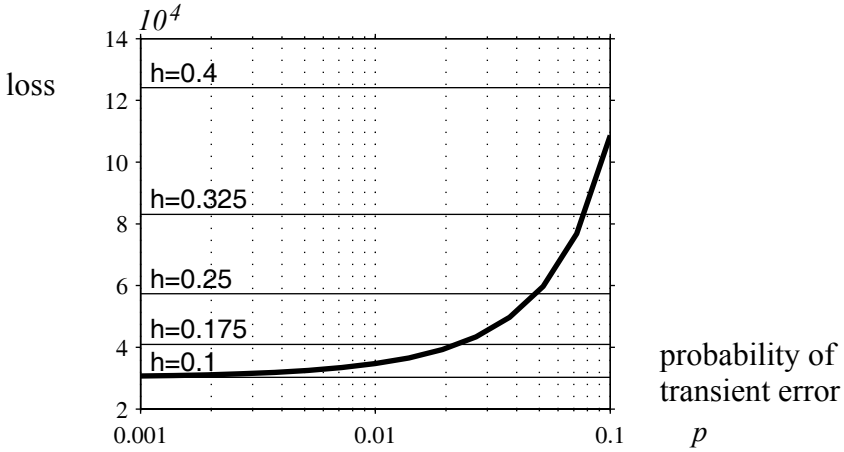


Figure 9.11 The loss as a function of the probability of a skip (bold line) and for longer periods without skips (horizontal lines).

9.8 Random and periodic delay jitter

In this section random delay jitter will be compared to periodic (deterministic) delay jitter. Consider the following transition probability matrix

$$P = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

with the probability $p \in [0.01, 0.99]$ of a switch. The delay τ switches within the set $\{0.2h_0, 0.8h_0\}$. System 7 is used in this experiment, with $h_0 = 0.1$. In case of deterministic jitter, the delay switches between high and low at every sampling instance ($p = 1$).

Let $J(\tau = 0.2h_0)$ denote the loss in case of a constant and short delay, and let $J(\tau = 0.8h_0)$ denote the loss in case of a constant and long delay. Further, let $J(p)$ denote the loss when the delay varies randomly according to the Markov process and finally let $J(det)$ denote the loss when the delay varies periodically in a deterministic manner.

The result is shown in Figure 9.12. $J(\tau = 0.2h_0)$ and $J(\tau = 0.8h_0)$ do not depend on p and are plotted as limits, with $J(\tau = 0.8h_0) > J(\tau = 0.2h_0)$, naturally. The loss of the system with the randomly varying delay, $J(p)$, decreases asymptotically, apparently from an average value$J(p \to 0) \Rightarrow [J(\tau = 0.2h_0) + J(\tau = 0.8h_0)] / 2$ to a loss corresponding to the deterministic uniform switching, $J(p \to 1) = J(det)$. The relative loss has been calculated by the division of $J(\tau = 0.2h_0)$.

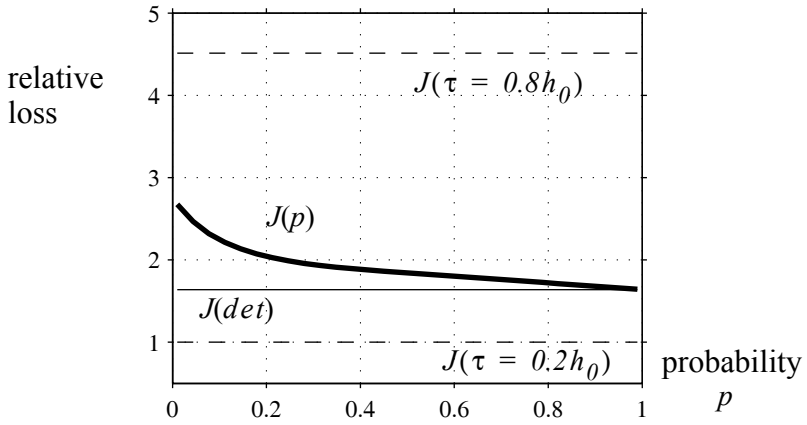


Figure 9.12 Relative loss versus probability, p, for random jumps (bold) and periodic scheme (thin). The lower and upper dashed lines are the loss for constant short and constant long delay respectively.

9.9 Periodic schemes

Deterministic sequences of delays are investigated in this section. A periodic sequence with jitter could be the result of a static cyclic schedule. Assume that the delay τ switches between a short (a) and a long (b) delay, corresponding to $0.2h_0$ and $0.8h_0$ respectively. Consider an nr-periodic sequence, with exactly two instances of a and all other instances of the type b. This setup corresponds to different static schedules. An interesting question is whether the relative distance between the two instances of type a matters or not. A transition probability matrix of size $[nr \times nr]$ can be written

$$P = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{89}$$

where the order of states corresponds to the nr-periodic sequence. Let the first instance of a have position 1, and let $\alpha = 2, \ldots, nr$ denote the position of the second instance of type a. Consider the sequence $s = s_1 \ldots s_w s_\alpha s_{(w+2)} \ldots$ (for some index w), i.e. with the first instance $s_1 = a$ and another instance $s_\alpha = a$. Since the sequence is periodic, it does not matter what instance comes first, for a steady state analysis.

In Figure 9.13, system 7 has been used to exemplify the static sequences, with $nr = 16$. The loss for different choices of the position α is plotted. The result is symmetric since the relative distance between s_1 and s_α reaches an apex half-ways. The mid position is $\alpha = 9$, since $s_1 = s_{nr+1}$. The result is contra-intuitive to the idea that the s_1 and s_α should be evenly spread; there seems to be three better options.

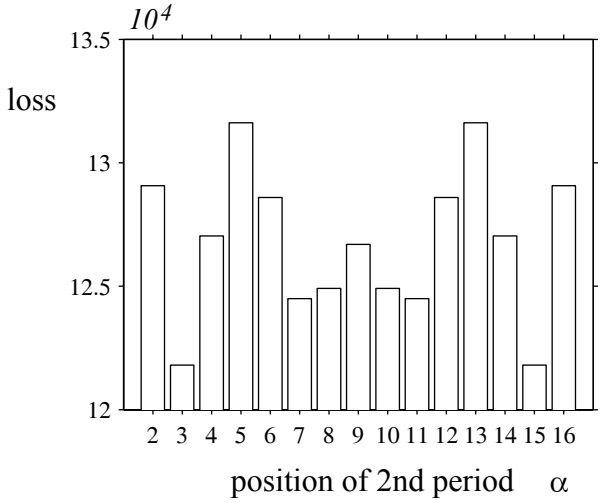


Figure 9.13 The loss of different sequences as determined by the position α.

9.10 Discussion

In the preceding Section 8 and Section 9 several experiments have been shown. Some of the seven example systems are sensitive to delay and jitter, whereas other are less sensitive. In a design context, a lucky control engineer would conclude that the small amount of jitter present in the computer system is irrelevant as judged by the loss. If not, there are several ways to cope with jitter, e.g. by adding a buffer or by changing control gains, as proposed by Nilsson (1998).

It is difficult to set up a fair comparison of different types of jitter, especially when the delay is allowed to be larger than the period. The jitter has been modelled starting from timelines. Another approach, made by e.g. Davidson (1973), Nilsson (1998) and Lincoln and Cervin (2002), is to associate a probability density function with the jitter. That is of course a natural way to model the stochastic time-variations. However, there is a potential problem to ensure causality ordering (i.e. that the delay of a buffer does not increase faster than time itself, and that sending/delivery order is preserved).

10 Summary and discussion of the report

The discretization of a continuous process and a loss function, in case of time-varying sampling period and time-varying control (or process) delay, has been derived in this report. The derivation is quite detailed with a number of examples. The intended use of the discretized loss function is as a performance measure to study the impact of the timing properties, above all delay jitter and period jitter. Jitter is a kind of uncertainty, which is valuable to model in order to assess the robustness of a nominal LTI analysis and synthesis.

The model of the sampled-data system is a single closed loop with a time continuous process and a discrete controller. The control signal is delayed. Since the sampling is non-uniform, the discretization of the process and the continuous time loss function were derived from first principles. The ordinary case with uniform sampling and zero delay falls out naturally as a special case. A noteworthy additional problem is that a time-varying period or a time-varying delay, can lead to multiple arrivals of a control signal at the process during one sampling period. The solution has been to integrate signals over two adjacent sampling instants and use the fact that the control signal is right-continuous and piecewise constant between arrivals.

The approach to the present model has been based on the consideration of quite simple time lines. Since the delay can be longer than the period, a Markov chain is used to keep track of combinations of periods and delays. A practical use is of course to compose the Markov chain from the timing sequences generated by e.g. a schedule simulation. It is necessary to keep the size of the problem small to reduce the computational complexity which increases with the number of states in the chain. In contrast to the proposed model using a Markov process, another approach is to model the jitter as a stochastic process. This is also natural, but when the delay can become longer than the sampling period, there will be a problem to satisfy the causality condition that the delay cannot increase faster than time itself.

As a special case, the Markov chain can be periodic. This means that at least some state is visited regularly and that the least common multiple of the number of events between two visits, is greater than or equal to two. A chain can be strictly periodic if all transitions are deterministic. When the chain is periodic, the closed loop system becomes periodic too. Strictly periodic systems can be analysed by a lifting procedure. However, a periodic chain can have transitions which are governed by some probability. It turns out that periodic systems can be treated using the same framework as for aperiodic systems.

An assumption is that the discrete time controller has parameters that are constant and do not change from one Markov state to another. An alternative control synthesis method is to construct a controller that varies its parameters according to the current or historical delays and sampling instants. These can be measured on-line in an actual implementation. If the timing properties are modelled by a Markov chain, a problem for the controller is to “know” what Markov state it is in, or going to. In Nilsson (1998), an LQG controller with a time-varying feedback gain was proposed. The gain was a function of both the measured and expected delay. The delay from the sensor to the controller can be measured by time-stamping, but the delay from the controller to the process is not known at the time when the control law is calculated. It is quite possible to relax the assumption of a constant control parameter setting. The proposed analysis in this report does not need any extension to deal with time-varying parameters which depend on the actual Markov state. However, some care must be taken regarding the prerequisites: the controller still has to be LTI and the Markov property must of course be satisfied. The aim of this report has been to study a performance measure for the timing properties, and control synthesis is beyond the scope.

In the model, the delay is placed at the output side of the controller but before the process. An idea is to move delays situated elsewhere and lump them together at this location. A delay can occur at virtually any place in a computer control system and it can also be present in the process. Care must be exercised when moving a delay or lumping delays together, and is relevant to discuss the conditions for when this preprocessing of the model is exact.

The delay of the measurement signal is especially relevant in this context. A control delay on the measurement side is situated after the sampling circuit, but before the controller. Assume that the controller is event-triggered, and that the measurement delay is also a delay of the trigger. Assume as before that the output circuit also is event-triggered in the same way. Take a position at the actuator, look upstream towards the controller, and judge if it is possible to determine the origin of the delay. If not, the move is probably correct. The cross-coupling of signals within the loss function must also be considered, not only the state evolution of the closed loop. If the controller does not have internal states (memory), it is clear that the move is exact even if the delay is time-varying and longer than one sampling period, since a gain-only controller only scales the signal. If the controller has internal states, a move is an exact model transformation if the delay is less than the sampling period at all times, even if they both are time-varying, cf. the assumptions in Nilsson (1998).

A transient error longer than one sampling period, can occur at the input side (vacant sampling), within the controller, or at the output side of the controller (deliberate skip, vacant actuation). These are not fully equivalent if the controller has internal states: if the timing error occurs at the output, the event-driven update of its internal states is unaffected by the interrupt. At the end of the transient delay, the state of the controller will not be the same for the two cases. This reasoning could possibly be elaborated including a time-varying (Markov state depending) controller.

Swapping the order of the sampler and a measurement delay is not covered by the presented theory, not even for a gain-only controller. Delaying a continuous time signal to be sampled, is not the same as delaying the sampled undelayed signal. It is of course always possible to view a lumping as an approximation, even if it isn't exact, but the consequences of the approximation should be investigated. Compare this discussion with an internal time delay element, placed between two time-continuous blocks, see e.g. Åström and Wittenmark (1997).

The closed loop with a single period and a single control delay could be extended. The measurement delay could also be described by a Markov chain. Multiple inputs and outputs with different delays for every signal, even for the measurement signal, internal delays and with multiple periods will add complexity to the analysis with the number of variables. The moving of delays becomes more complicated for a general MIMO system, since in a general MIMO system individual signals have different delays.

In multi-rate sampling a delayed sampled-data system of the type discussed here has a finite dimensional discrete representation if the quotient h / τ is a rational number. Also the quotient between any two sampling periods in a non-delayed multi-rate system is generally assumed to be a rational. A tick defining the granularity can be used for this purpose. Åström and Wittenmark (1997) propose that delayed sampling also can be handled by an oversampling technique. This would be an alternative to the model proposed in this report, cf. Lincoln and Cervin (2002). In this report, a Markov chain with a finite number of states is used to switch between individual finite dimensional discrete-time systems.

Even though the increase in loss might be small for a transient error, there is a significant difference between the steady state behaviour and the transient behaviour. The range of measured values, control signals etc. must be checked in a practical application. If the control signal sat-

urates, the system becomes nonlinear and the prerequisite of the discretization developed here does not hold. The user or operator might be able to detect a transient errors and judge the quality of the system out of that. This calls for a worst-case analysis rather than the average-case analysis proposed here.

A few related topics not studied will be commented below. Sliding mode is a motion that exists in a manifold of state trajectories, traversed in a finite time. Sliding mode can exist also in sampled-data systems. Here, it is assumed that sliding mode does not occur. Observability and detectability has not been treated at all. These are important properties, at least when it comes to control synthesis. Since jump linear systems theory is being used, results regarding observability and detectability could be taken from that research area, see e.g. the works by Costa and Fragoso (1993), Ji and Chizeck (1990) and Ji et al. (1991). It is of course assumed here that the sampled-data system positively have these properties.

11 References

Bittanti S., Laub A. J. and Willems J. C., “The Riccati equation“, Springer Verlag, ISBN 0-387-53099-1, 1991.

Cassandras C. G. and Lafortune S., “Introduction to discrete event systems”, Kluwer, ISBN 0-7923-8609-4, 1999.

Chan H. and Özgüner Ü., “Optimal control of systems over a communication network with queues via a jump system approach”, IEEE 1995, p. 1148-1153, 1995.

Costa O. and Fragoso M., “Stability results for discrete-time linear systems with Markovian jumping parameters”, Journal of mathematical analysis and applications 179, p. 154-178, 1993.

Davidson C., “Random sampling and random delays in optimal control systems”, PhD thesis, KTH dissertation 21429, TRITA-MAT-1973-8, May 1973.

Franklin, Powell and Workman, “Digital control of dynamic systems“, 2nd edition, ISBN 0-201-11938-2, 1990.

Halevi Y. and Ray A., “Integrated communication and control systems: Part 1 and Part 2“, Journal of dynamic systems, measurement and control, Vol. 110, pp. 367-381, December 1988.

Ji Y., Chizeck H. J., “Jump linear quadratic gaussian control: steady state solutions and testable conditions“, Control-Theory and Advanced Technology, Vol. 6, No. 3, pp. 289-319, September 1990.

Ji Y., Chizeck H. J., Feng X., Loparo K.A., “Stability and control of discrete-time jump linear systems“, Control-Theory and Advanced Technology, Vol. 7, No. 2, pp. 247-270, June 1991.

Khalil H., “Nonlinear Systems”, ISBN 0-02-946336-X, Macmillan, 1992.

Kharitonov V. L., “Robust stability analysis of time delay systems: a survey“, Annual reviews in control, 23 (1999) 185-196, Pergamon, 1999.

Lancaster P., “Theory of matrices”, Academic press, New York, 1969.

Lennartson B., “On the choice of controller and sampling period for typical process models”, Teknisk rapport 85:11, Inst. f. Reglerteknik, Chalmers, 1985.

Lincoln B. and Cervin A., “Jitterbug: A tool for analysis of real-time control performance”, In Proceedings of the 41st IEEE Conference on Decision and Control. Sydney, Australia, 2002.

Nilsson J., “Real-time control systems with delays”, PhD thesis, ISSN 0280-5316, ISRN LUTFD2/TFRT--1049--SE, January 1998.

Sanfridson M., “Timing problems in distributed real-time computer control systems”, Technical Report, Dept. of Machine Design, TRITA-MMK 2000:11, 2000.

Sanfridson M., “Timing problems in distributed control“, Licentiate thesis, ISRN KTH/MMK--00/14--SE, Mechatronics Lab, May 2000.

Sanfridson M., “An overview of the use of LMI in control to assess robustness and performance”, Technical report, ISRN KTH/MMK--03/8--SE, Mechatronics Lab, KTH, 2003.

Seto D., Lehoczky J. P., Sha L., Shin K. G., “On task schedulability in real-time control systems”, Proceedings RTSS 96, p.13-21, 1996.

Skogestad and Postlethwaite, “Multivariable feedback control - Analysis and design”, ISBN 0-471-94277-4, Wiley, 1996.

Van Loan, C., “Computing integrals involving the matrix exponential“, IEEE Trans. on automatic control, Vol. AC-23, No. 3, June 1978.

Xiao L., Hassibi A., How J. P., “Control with random communication delays via a discrete time jump system approach“, Proceedings of the American Control Conference, June 2000.

Zeng-Qi S., “Sampling interval selection and feedback reduction for the control of time continuous processes”, PhD Thesis no. 379, Chalmers, 1981.

Åström K. J., “Introduction to stochastic control theory“, ISBN 0-12-065650-7, Academic Press, 1970.

Åström K. J. and Wittenmark B., “Computer controlled systems”, Prentice-Hall, ISBN 0-13-314899-8, 1997.

Appendix A: Description of the example systems

A set of seven sample systems are used in this document to exemplify and compare different aspects of the presented theory. The purpose of this section is to describe these in such a detail that the computations easily can be repeated. An ambition has been to collect a spectrum of typical and problematic processes and controllers. About half of the systems are well-known to represent some difficulty in control engineering, for example the non-minimum phase process. The other processes and controllers are typically found in the industry, for example the DC-motor. All systems have approximately the same time scale (e.g. time constants) with the parameters given in this section. The time unit is customer defined, and this indefiniteness is denoted by c.u., customer units. Millisecond is otherwise a time unit which puts the dynamics of the process in relation with that of today's ordinary computer control devices. An overview of the example systems is found in Table 8.1.

The general controller model is given in Section 4.4 on state-space form by the set of constant matrices $\{\Phi_c, \Gamma_c, \Lambda_c, C_c, D_c, E_c\}$ of appropriate sizes. The control parameters will be constant, i.e. they are derived from the nominal period and delay, if nothing else is stated.

If not otherwise specified, the following weight matrices have been used throughout this report: $\bar{Q}_{11} = I$, $\bar{Q}_{12} = 0$ and $\bar{Q}_{22} = I$. In the LQ control design, other weight matrices have been used, see details below. The default equation noise variance is $\overline{R_v} = I$ and the measurement noise variance is $R_w = I$.

Table A.1 contains some key figures. For each example system, a range of suitable sampling periods is indicated. The range is based on visual inspection of simulation and the rules-of-thumb found in Åström and Wittenmark (1997). The nominal period and nominal delay are used for the analysis and synthesis. A maximum allowable delay is also indicated. This is the delay which at the nominal sampling period gives a 50% increase of the loss as defined by the weight and noise in the previous paragraph. This is to give the reader an idea of the sensitivity to a constant control delay, compare this with the "phase margin" of a system. It is also valuable to acquire a coupling between the value of a loss function and e.g. the step response.

Table A.1 Nominal periods and delays. Customer time units (c.u.).

System	Process	Nominal period, h_0	Period range	Nominal delay	τ at 50% increase of J	Open-loop stable
1	2nd order damped	0.25	[0.1, 0.5]	0	$1.5h_0$	yes
2	DC-motor	0.4	[0.2, 0.7]	$h_0/2$	$3.2h_0$	no
3	3rd order process	0.1	[0.05, 0.2]	0	$3.2h_0$	yes
4	Non-minimum phase	0.2	[0.1, 0.3]	0	$11h_0$	yes
5	1st order with delay	0.2	[0.1, 0.3]	1	$7h_0$	yes
6	Double integral	2	[1, 3]	0	$0.18h_0$	no
7	Inverted pendulum	0.1	[0.05, 0.2]	0	$0.5h_0$	no

With open-loop stability in the right most column, it is here meant that the process value remains bounded when the control signal is held constant. This is the case for example when

the control signal cannot be updated, because of a broken feedback path. The numbers in the column on the left hand side correspond to the subsections below.

A.1 Damped second order process with a PI controller

A damped second order process is a classical example. Like the first order process, it can be used to model many physical phenomena, but with more interesting characteristics. The structure is,

$$G(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 + \omega_0^2}$$

with ω_0 for the natural frequency and ζ for the damping. A PID is an industrial de facto standard and a traditional first choice of controller. The derivative part is often omitted. A simple discrete controller with the essential parameters, can be translated from a continuous time form into

$$\begin{aligned} P(kh) &= K(-y(kh)) \\ I(kh + h) &= I(kh) + \frac{Kh}{T_i}(-y(kh)) \end{aligned}$$

with the output equation $u(kh) = P(kh) + I(kh)$, see e.g. Åström and Wittenmark (1997). The controller can be written on state space form having the state $I(kh)$.

The parameters of the process are chosen to be: $\omega_0 = 1$ and $\zeta = 0.7$. The PID is tuned manually and the parameters are: $K = 0.9$, $T_i = 1.3$. A suitable nominal sampling period is $h_0 = 0.25$ and a reasonable choice should lie in the range $[0.1, 0.5]$. With the nominal sampling period, an additional control delay of $\tau = 1.5h_0$ renders a 50% increase of J. This is shown in the simulated step responses in Figure A.1.

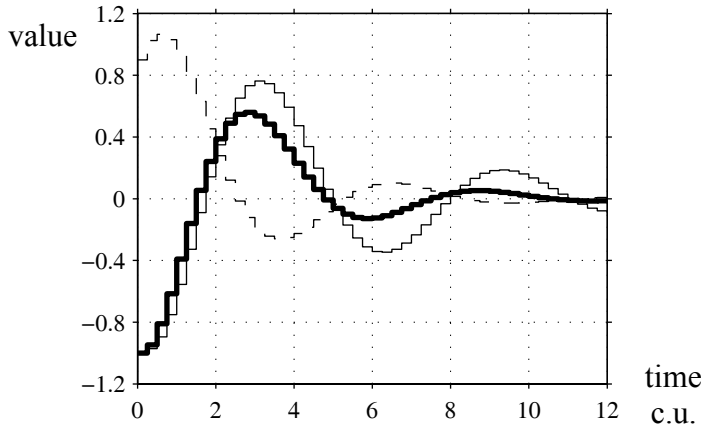


Figure A.1 Step responses of the closed loop system. Process output (bold) and control signal (dashed) vs. time, for nominal sampling period and no delay. The thin line represents the process output having an additional delay.

The matrices for the controller become

$$\Phi_c = 1 \qquad \Gamma_c = -\frac{Kh}{T_i} \qquad \Lambda_c = 0_{1 \times md}$$
$$C_c = 1 \qquad D_c = -K \qquad E_c = 0_{1 \times md}$$

The sizes of Λ_c and E_c depend on the length of the composite control signal U, see Section 4.4.

A.2 DC-motor with a state feedback controller

The DC-motor is a ubiquitous machine element in a mechatronic system and it is a true reference example system just like the previous one. The DC-motor is a 2nd order system with the transfer function

$$G(s) = \frac{1}{s(s+1)}$$

In this discrete design, the process is first discretized, assuming a constant nominal control delay of $\tau = h_0/2$. A state feedback controller to compensate for some delay can be written:

$$u(kh) = -L_x x(kh) - L_u u(kh-h)$$

and the static feedback gains can be found by optimization of the standard discrete loss function using the weights

$$Q_{xx} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad Q_{uu} = 3$$

The discrete time weight matrices are found by trial and error for a nominal sampling period such that the overshoot due to a step disturbance is less than 5%. Examining the rules-of-thumb and the simulation, a suitable sampling period is found to be $h_0 = 0.4$ and a suitable range is $[0.2,\ 0.7]$. With the nominal sampling period and nominal delay, a control delay of $\tau = 3.1 h_0$ renders a 50% increase of J (compared to an actual delay corresponding to the nominal one). This is shown in the simulated step responses in Figure A.2.

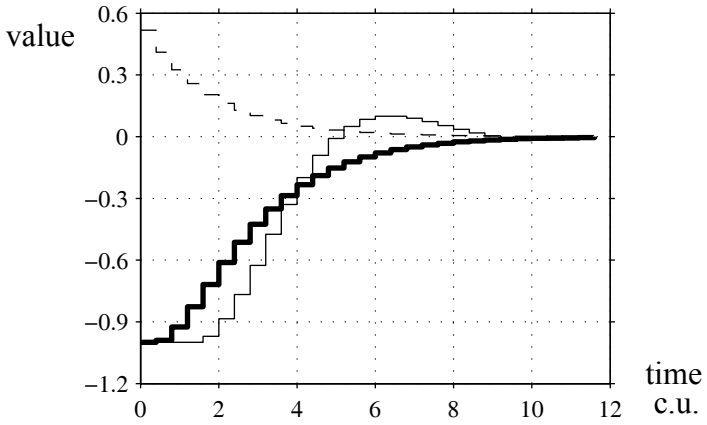


Figure A.2 Step load responses of the closed loop. Process output (bold) and control signal (dashed) vs. time for the nominal sampling period and nominal delay; process output with additional delay (thin).

The controller becomes

$$\Phi_c = 0_{0\times 0} \quad \Gamma_c = 0_{0\times 2} \quad \Lambda_c = 0_{0\times md}$$
$$C_c = 0_{1\times 0} \quad D_c = -L_x \quad E_c = -L_u$$

and the length of Λ_c and E_c depends on the length of the composite control signal U, see Section 4.4.

A.3 Third order process with an LQG controller

The third order process with an LQG controller is given by

$$G(s) = 75\frac{s+1}{(s+2)(s+3)(s+4)}$$

This will be controlled by an optimal controller (gain L) with a Kalman state estimator (gain K) estimating the noisy states through the noisy measurement. For the synthesis, the nominal period is assumed, and the weight matrices are

$$Q_{xx} = 4I \qquad Q_{uu} = 1$$

together with the noise covariance:

$$\overline{R_v} = \left(\begin{bmatrix}2\\2\\2\end{bmatrix}\begin{bmatrix}2 & 2 & 2\end{bmatrix}\right)^{\frac{1}{2}} \qquad R_w = 1$$

A suitable sampling period is $h_0 = 0.1$ or it should at least be in the range $[0.05,\ 0.2]$. With the nominal sampling period, an additional control delay of $\tau = 3.2h_0$ renders a 50% increase of J. This is shown in the simulated step responses in Figure A.3.

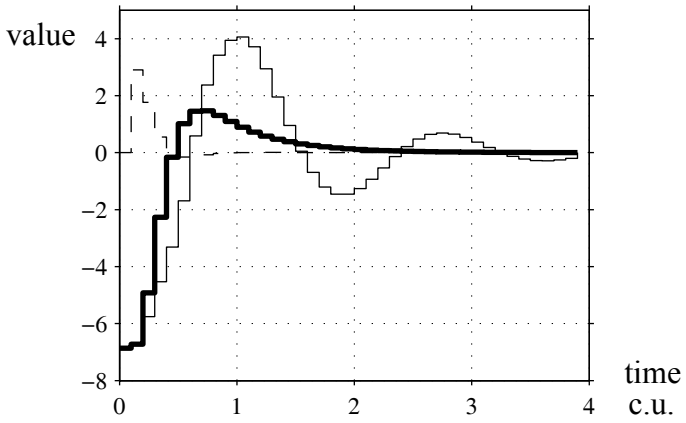


Figure A.3 Step responses of the closed loop system. Process output (bold) and control signal (dashed) vs. time for the nominal sampling period and no delay. Process output with additional delay (thin).

The process, controller and a straight forward estimator can be written

$$\begin{aligned} x_p(kh+h) &= \Phi_p x_p(kh) + \Gamma_p u(kh) + v(kh) \\ y(kh) &= C_p x_p(kh) + w(kh) \\ u(kh) &= -L\hat{x}_p(kh\,|\,kh-h) \\ \hat{x}_p(kh+h\,|\,kh) &= \Phi_p \hat{x}_p(kh\,|\,kh-h) + \Gamma_p u(kh) + K[y(kh) - C_p \hat{x}_p(kh\,|\,kh-h)] \end{aligned}$$

and the matrices of the controller become

$$\Phi_c = \Phi_p - \Gamma_p L - KC_p \quad \Gamma_c = K \quad \Lambda_c = 0_{2\times md}$$
$$C_c = -L \quad D_c = 0_{1\times 1} \quad E_c = 0_{1\times md}$$

A.4 Non-minimum phase process with an LQ controller

The non-minimum phase process can be found in mechanical systems. A vehicle typically has a non-minimum phase behaviour in the steering. This type of process represents a difficulty because of the zero in the right half plane. The RHP-zero limits the achievable bandwidth of the closed loop system. The process,

$$G(s) = \frac{1 - 0.5s}{s^2 + 3s + 1}$$

will be controlled by an LQ controller,

$$u(kh) = -Lx(kh)$$

of continuous time weight matrices

$$\bar{Q}_{xx} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \qquad \bar{Q}_{uu} = 1$$

A suitable sampling period is $h_0 = 0.2$ and a reasonable range is $[0.1, 0.3]$. With the nominal sampling period, a control delay of as much as $\tau = 11h_0$ renders a 50% increase of J. This is shown in the simulated step responses in Figure A.4.

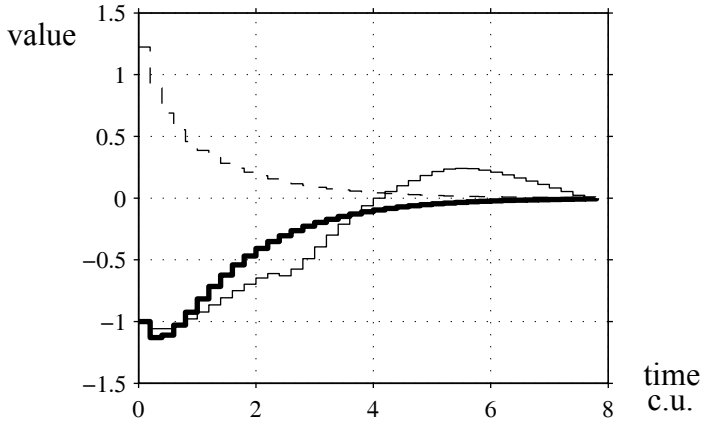


Figure A.4 Step responses for the closed loop system, with its typical non-minimum phase behaviour. Process output (bold) and control signal (dashed) vs. time, for the nominal sampling period and no delay. Process output with additional delay (thin).

The matrices of the controller become:

$$\Phi_c = 0_{0\times 0} \quad \Gamma_c = 0_{0\times 2} \quad \Lambda_c = 0_{0\times md}$$
$$C_c = 0_{1\times 0} \quad D_c = -L \quad E_c = 0_{1\times md}$$

A.5 First order with delay and Smith predictor

The first order process with delay is a model frequently seen in the process industry. It can represent a pipe and tank system, i.e. a transport delay of a flow combined with an accumulation or dilution of the fluid. A long delay makes the process difficult to control and limits the achievable bandwidth of the closed loop system. Let the process be described by

$$G(s) = \frac{1}{sT_p + 1} e^{-s\tau_0}$$

where the constant process delay $\tau_0 = 1$ is long compared to its time constant, $T_p = 1$. A modified Otto-Smith predictor is typically used for this process. The rational transfer function part of the model is zero-order-hold-translated into the form $y(kh + h) = ay(kh) + bu(kh)$ and used in the feedback together with a PI-controller in the direct path, see e.g. Åström and Wittenmark (1997). The parameters of the controller are a, b, $K = 0.9$ and $T_i = 1.3$.

A suitable sampling period is $h_0 = 0.2$ time units and a reasonable range is $[0.1, 0.3]$. With the nominal sampling period, an additional control delay of $\tau = (12 - 5)h_0$ renders a 50% increase of J. This is shown in the simulated step responses in Figure A.5. The model mismatch of the delay makes the process value look rough.

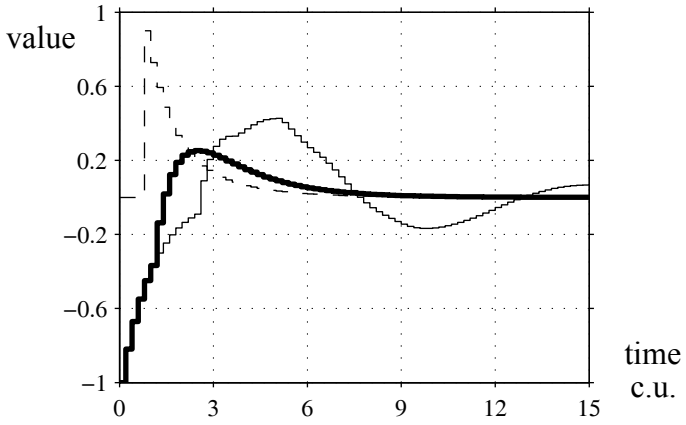


Figure A.5 Step responses of the closed loop system. Process output (bold) and control signal (dashed) for the nominal sampling period and with process delay only. Process output with control delay (thin).

The controller becomes:

$$\Phi_c = \begin{bmatrix} 1 & -\frac{Kh}{T_i} & \frac{Kh}{T_i} \\ b & a - bK & bK \\ 0 & 0 & a \end{bmatrix} \qquad \Gamma_c = \begin{bmatrix} -\frac{Kh}{T_i} \\ -bK \\ 0 \end{bmatrix} \qquad \Lambda_c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ b & 0 \end{bmatrix}$$

$$C_c = \begin{bmatrix} 1 & -K & K \end{bmatrix} \qquad D_c = -K \qquad E_c = 0_{1 \times d}$$

with the state vector $\begin{bmatrix} I(kh) & y(kh) & y(kh - dh) \end{bmatrix}^T$. For simplicity it is assumed that $\tau_0 = dh_0$, $d = 1, 2, \ldots$ and the multiple $d = \tau_0 / h_0$ is a positive integer. Note that $d > nd$ is possible, however $md \geq d$ determine the size of Λ_c and E_c.

A.6 Double integrator and deadbeat controller

The double integrator is a traditional text book example

$$G(s) = \frac{1}{s^2}$$

and this second order system is rather difficult to control. It will be controlled by a deadbeat controller, which is very special because it does not have a continuous time counterpart. The behaviour of the controller is sensitive to the choice of period. The name deadbeat comes from the bang-bang nature when trying to force the system to the origin in a minimum number of sampling periods. Understandably, this control strategy is often avoided. The process is zero-order-hold discretized. The deadbeat controller is obtained by placing the discrete poles at the origin. In this case, the control law becomes

$$u(kh) = -Lx_p(kh) = -\begin{bmatrix} \frac{1}{h^2} & \frac{3}{2h} \end{bmatrix} x_p(kh)$$

A suitable sampling period is $h_0 = 2$, and a reasonable range is $[1, 3]$. With the nominal sampling period, an additional control delay of $\tau = 0.18 h_0$ renders a 50% increase of J. It is clear from the simulated step responses in Figure A.6, that this makes the closed loop more volatile — the process is oscillating. One can suspect that this system is very sensitive to jitter and model mismatch.

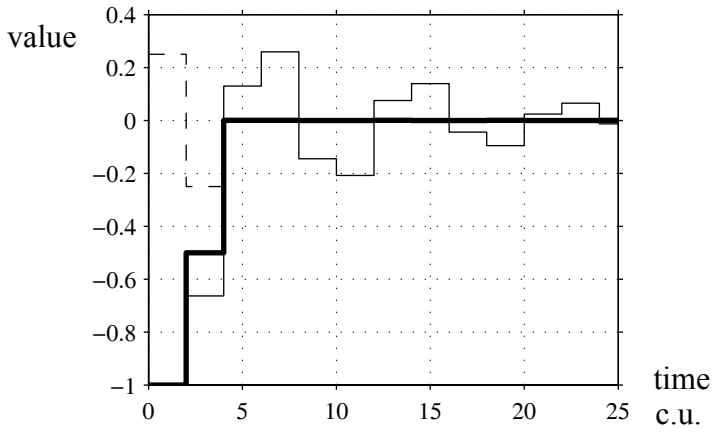


Figure A.6 Step response of the closed loop system. Process output (bold) and control signal (dashed) vs. time for the nominal sampling period and no delay. Process output with additional delay (thin).

The controller becomes:

$$\Phi_c = 0_{0\times 0} \quad \Gamma_c = 0_{0\times 2} \quad \Lambda_c = 0_{0\times md}$$
$$C_c = 0_{1\times 0} \quad D_c = -L \quad E_c = 0_{1\times md}$$

A.7 Inverted pendulum with a minimum variance controller

This process is a favourite in the control community. It is derived from Newtons' equations of motion, where suitable states are the position x and the velocity $\dot{x}$ of the cart, the angle of the pendulum φ (an angle of zero means vertical position) and its angular velocity, $\dot{\varphi}$. Input u is a force acting in positive x-direction. The differential equations describing the dynamics of the mechanical system become

$$\ddot{x} = \frac{u + ml\dot{\varphi}^2 \sin\varphi - mg\cos\varphi\sin\varphi}{M + m - m(\cos\varphi)^2}$$

$$\ddot{\varphi} = \frac{u\cos\varphi - (M+m)g\sin\varphi + ml\dot{\varphi}\sin\varphi\cos\varphi}{ml(\cos\varphi)^2 - (M+m)l}$$

The parameters are $m = 0.1$ for the mass of the pendulum, $M = 10$ the mass of the sliding cart the pendulum is connected to, $l = 1$ is the distance from the joint to the centre of mass of the bar and $g = 10$ is the acceleration of gravity. For small angles, a linearized pendulum model in continuous time can then be written in state space form,

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(M+m)g}{M} & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{M \cdot l} \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

including $D = 0$, and with the state vector $x_p = \begin{bmatrix} x & \dot{x} & \varphi & \dot{\varphi} \end{bmatrix}^T$. An optimal single output controller is assigned the process using weights $Q_{xx} = C^T C$ (since $y = Cx$, $y^T y = x^T C^T C x$) and $Q_{uu} = 0.001$. The variance of the process output is minimized with this formulation. A suitable sampling period is $h_0 = 0.1$ and a possible but large range is $[0.05, 0.2]$. With the nominal sampling period, an additional control delay of $\tau = 0.5 h_0$ renders a 50% increase of J. However, this step response is hard to distinguish from the non-delay case, see Figure A.7.

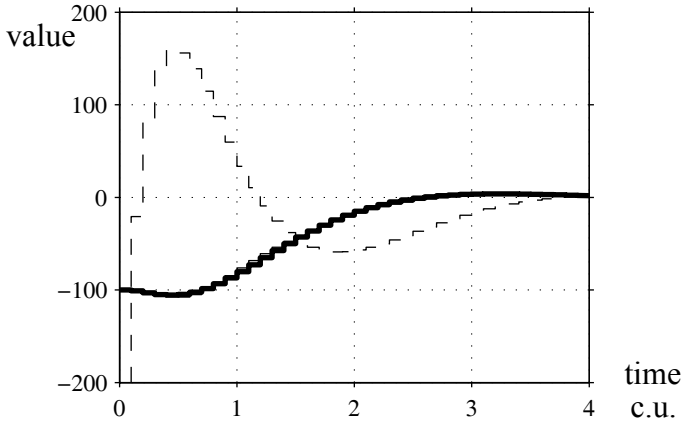


Figure A.7 Step response of an inverted pendulum. Process output (bold) and control signal (dashed) vs. time for the nominal sampling period and without delay; process output with additional delay (thin).

The controller becomes:

$$\Phi_c = 0_{0 \times 0} \qquad \Gamma_c = 0_{0 \times 4} \qquad \Lambda_c = 0_{0 \times md}$$
$$C_c = 0_{1 \times 0} \qquad D_c = -L \qquad E_c = 0_{1 \times md}$$

Appendix B: Validation by simulation

In this section, the performance measure will be validated by means of simulation. The tool suite Matlab and Simulink is excellent for combined continuous time and discrete event simulation. The simulation setup is the single closed loop shown in Figure 4.7. The closed loop is driven by state or measurement noise. The control and measurement signals are tapped, squared, weighted and continuous time integrated. The steady state (final) value of the integrated value is the result from one simulation run. The simulation is straight forward since no parameter of the model is altered (except the time-varying periods and delay). The time horizon for each simulation run has to be long enough for the integrated measure to settle. With more repetitions and longer simulation times, the accuracy can be improved. The standard deviation of the result decreases slowly with the length of each run and for repetitive runs, typical for Monte Carlo simulation. The more time-variations, the harder it is to achieve satisfactory accuracy. To validate the calculations by simulation can be a challenge and take a lot of time.

A zero-order-hold S-function for time-varying sampling periods has been implemented. The measurement noise is added at the sampling instant. The delay block supplied with Simulink has been avoided. The delay buffer has instead been implemented as an S-function with variable sample time. The “next hit” can either be the next sampling instant (when a new sample is added to the buffer) or the next dispatch of an element in the buffer. Continuous time white noise is created using the built in block, with a sampling period shorter than that of the controller.

An overview of the validation is provided in Table B.1. The number of a validation in the first column corresponds to a subsection below. A new system has been selected for every experiment.

Table B.1 List of validation presented below.

Number	System	h	τ	Altered parameter(s)
1	1	constant	constant	h, and re-discretization of controller
2	4	constant	constant	τ
3	6	random	random	τ and h, with $\tau < h$
4	2	random	random	τ and h, arbitrary τ
5	7	constant	periodic	τ

The systems are found in Appendix A. For all experiments, the weight matrix is composed of $\bar{Q}_{11} = I$, $\bar{Q}_{12} = 0$ and $\bar{Q}_{22} = I$, if not otherwise stated. Also, if $R_w = 0$ then $\bar{R}_v = I$ and vice versa. Validation have been performed with the delay on the controller’s side, but also with the delay on the process’ side and the difference $\bar{J}_{cd} - \bar{J}_{pd}$ matches well with the simulations.

B.1 Constant period

In Figure B.1, the calculated and the simulated loss functions can be compared. System 1 with its nominal period of $h = 0.25$ and zero delay. The driving noise is $\overline{R_v} = I$ and $R_w = 0$. The controller is discretized for each constant value of h. The shape of the plot is typical. The plot has been generated using short simulation times, hence the low accuracy which neatly visualizes the difference between the calculated and the simulated values.

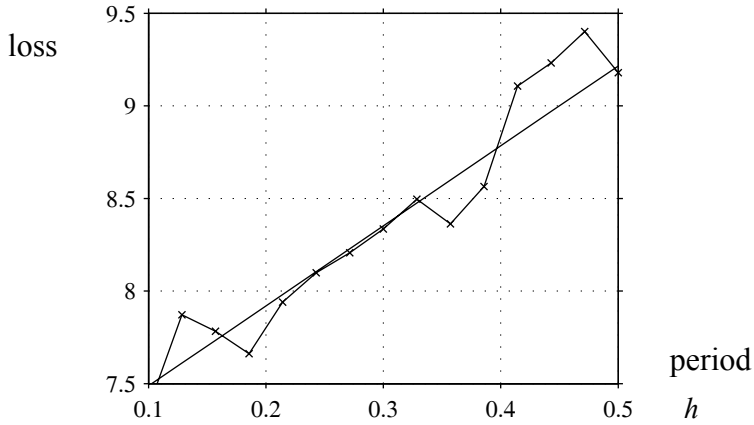


Figure B.1 Validation, calculated and simulated (crosses) loss as a function of constant period.

The sampled system for small periods, has also been compared with the calculated loss of a continuous time equivalence. This is especially easy for optimal controllers, since the covariance matrix is given by the Riccati equation (e.g. Matlab's *lqr.m*), but is not shown here.

B.2 Constant delay

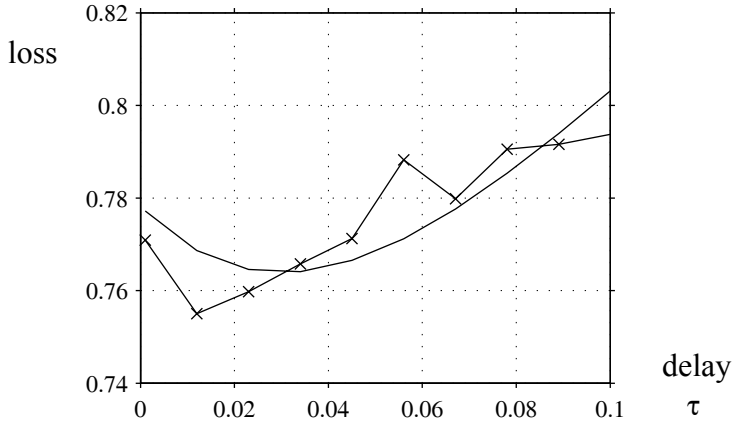


Figure B.2 Validation of constant delay. Calculated and simulated (crosses) loss as a function of constant delay.

The loss as a function of the delay has also been validated. The following example has been chosen for its somewhat unexpected result, and not because it exhibits a typical behaviour. System 4 with its nominal period of $h = 0.2$, $R_w = I$ and $\overline{R}_v = 0$ has been used to generate Figure B.2. The weight matrix is composed of $\overline{Q}_{11} = 10I$ and $\overline{Q}_{22} = 0$. The shape of the line depends very much on the weight matrix.

B.3 Period and delay jitter, restricted

The next exercise is to examine the time-varying delay and the period to see if the calculation matches the simulation. System 6 with

$$\overline{R}_v = \begin{bmatrix} 3 & \sqrt{3} \\ \sqrt{3} & 1 \end{bmatrix} \text{ and } R_w = 0$$

has been used to generate Figure B.3. For simplicity $\tau(t) < h(t)$ for all times, such that assumption A4 in Section 2.3 holds. Let the transition probability matrix be

$$P = \begin{bmatrix} 1-3p & p & p & p \\ p & 1-3p & p & p \\ p & p & 1-3p & p \\ p & p & p & 1-3p \end{bmatrix}$$

such that when $p \leq 1/3$ is small, the probability of changing state is low. The state probability becomes $\pi^{\infty}(r) = 0.25$, $r = 1, 2, 3, 4$. The four states correspond to the combinations $\{h_r, \tau_r\}$, $\forall r$: low/low $\{1.7, 0.05\}$, low/high $\{1.7, 0.1\}$, high/low $\{2.3, 0.05\}$ and high/high $\{2.3, 0.1\}$ respectively.

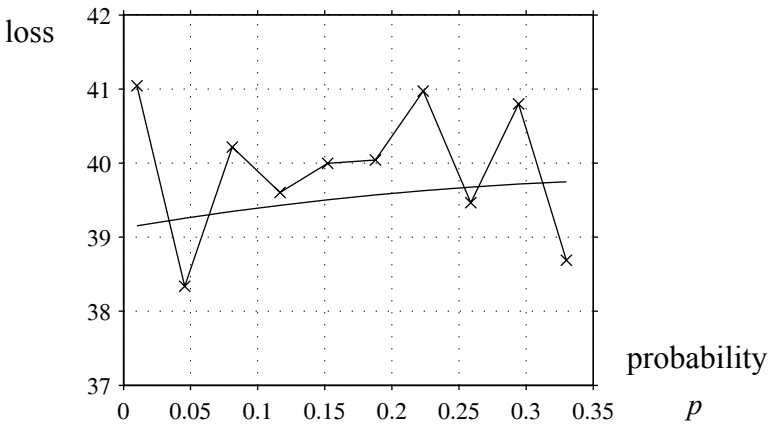


Figure B.3 Validation of time-varying period and delay, computed and simulated (crosses).

B.4 Period and delay jitter, relaxed

Assumptions A4 of Section 2.3 will now be relaxed compared to the previous section, i.e. the constraint $\tau(t) < h(t)$ is removed. The Markov chain, modelling the temporal relationship of a timeline of sampling and actuation, must be constructed.

Consider the Markov chain in Figure B.4. The delay changes between the values a short τ_s and a long $\tau_l > \tau_s$ value. Likewise, the period changes between a short h_s and a long $h_l > h_s$ value. In order hold the number of states in the Markov chain low, the following constraints are imposed: a long delay must be followed by a short delay and the inequalities $\tau_l \leq 2h_s$, $\tau_l \geq h_l$ and $\tau_s \leq h_s$ must be satisfied. States 3 and 4 are the only states with a long delay; no control signal will arrive during their corresponding period. States 5 to 8 can be called "reset states", since two control signals will arrive during the period modelled by these states. For states 1 and 2, exactly one control signals arrive during the respective period.

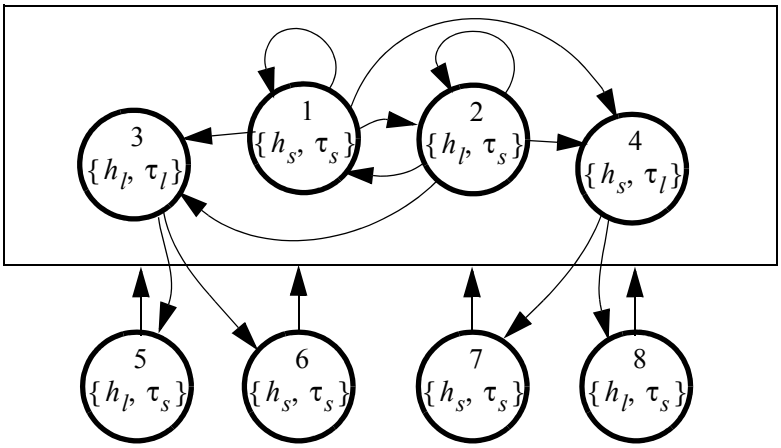


Figure B.4 A Markov chain for period and delay jitter. The box is used to substitute a set of arrows and it means that the states 5 to 8 can go to any of the states 1 to 4.

Assume that the probability of a transition of a specific state is $1/n$, where n is the number of states possible to jump to. The following transition probability matrix

$$P = \begin{bmatrix} p & p & p & p & 0 & 0 & 0 & 0 \\ p & p & p & p & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2p & 2p & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2p & 2p \\ p & p & p & p & 0 & 0 & 0 & 0 \\ p & p & p & p & 0 & 0 & 0 & 0 \\ p & p & p & p & 0 & 0 & 0 & 0 \\ p & p & p & p & 0 & 0 & 0 & 0 \end{bmatrix}$$

with $p = 1/4$, corresponds to Figure B.4. Now, in order to find suitable values of the delays and periods, let $\tau_s = h_s$ and define the relation $\tau_l = h_l + \alpha(2h_s - h_l)$, i.e. a linear increase with $\alpha \in [0, 1]$. The delays $\{\tau_s, \tau_l\}$ are uniquely specified by the choice of $\{h_s, h_l\}$, and the conditions above implies that $h_l < 2h_s$.

In Figure B.5, system 2 with a nominal period of $(h_l + h_s)/2$ and delay compensation, driven by a noise covariance of $\overline{R_v} = I$ and $R_w = 0$ has been used to illustrate the jitter.

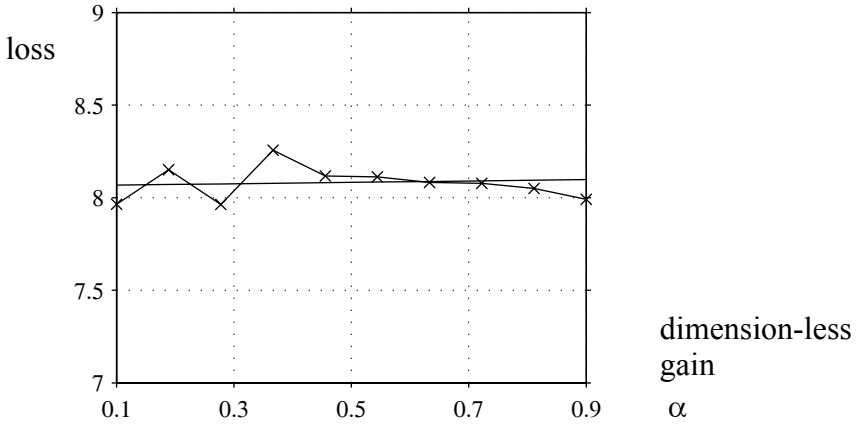


Figure B.5 Period and delay jitter. Loss as a function of the dimension-less factor α.

B.5 Periodic schedule

In Figure B.6, periodic schemes are validated using system 7 with its nominal sampling period. The setup is the same as the experiment in Section 9.9. Here the length of the schedule is $nr = 8$. As can be seen from the figure, the calculation and simulation fit together.

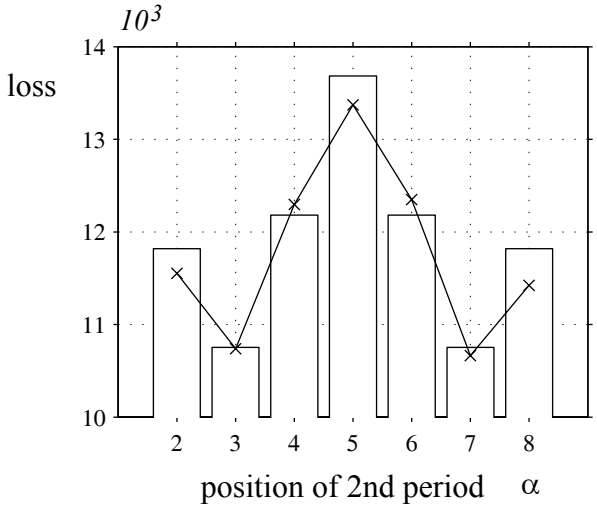


Figure B.6 Validation of periodic schemes, calculated (bars) and simulated (crosses) values.

Appendix C: Some matrix sizes

The size of the closed loop system matrix Φ_{cl} is $[na \times na]$ and it grows linearly with md, see Section 4.6. It has the following submatrices:

$$\Phi_{cl} = \begin{bmatrix} (\Phi_{cl})_{11} & (\Phi_{cl})_{12} & (\Phi_{cl})_{13} \\ (\Phi_{cl})_{21} & (\Phi_{cl})_{22} & (\Phi_{cl})_{23} \\ (\Phi_{cl})_{31} & (\Phi_{cl})_{32} & (\Phi_{cl})_{33} \end{bmatrix}$$

of the respective sizes

$$[(\Phi_{cl})_{11}] = [np \times np] + [np \times nu][nu \times nz][nz \times np]$$

$$[(\Phi_{cl})_{12}] = [np \times (nu \cdot md)] + [np \times nu][nu \times (nu \cdot nd)]$$

$$[(\Phi_{cl})_{13}] = [np \times nu][nu \times nc]$$

$$[(\Phi_{cl})_{21}] = \begin{bmatrix} [(nu \cdot (md-1)) \times np] \\ [nu \times nz][nz \times np] \end{bmatrix}$$

$$[(\Phi_{cl})_{22}] = \begin{bmatrix} [nu \cdot (md-1) \times nu] & [nu \cdot (md-1) \times nu \cdot (md-1)] \\ [0 \times 0] & [nu \times (nu \cdot md)] \end{bmatrix}$$

$$[(\Phi_{cl})_{23}] = \begin{bmatrix} [nu \cdot (md-1) \times nc] \\ [nu \times nc] \end{bmatrix}$$

$$[(\Phi_{cl})_{31}] = [nc \times nz][nz \times np]$$

$$[(\Phi_{cl})_{32}] = [nc \times (md \cdot nu)]$$

$$[(\Phi_{cl})_{33}] = [nc \times nc]$$

The noise input to the closed loop is taken by the matrix Γ_{cl} of size $[na \times ne]$

$$\Gamma_{cl} = \begin{bmatrix} (\Gamma_{cl})_{11} & (\Gamma_{cl})_{12} \\ (\Gamma_{cl})_{21} & (\Gamma_{cl})_{22} \\ (\Gamma_{cl})_{31} & (\Gamma_{cl})_{32} \end{bmatrix}$$

having the following sizes of its six submatrices respectively:

$$[(\Gamma_{cl})_{11}] = [np \times np]$$

$$[(\Gamma_{cl})_{12}] = [np \times nz]$$

$$[(\Gamma_{cl})_{21}] = \begin{bmatrix} [nu \cdot (md-1) \times np] \\ [nu \times np] \end{bmatrix}$$

$$[(\Gamma_{cl})_{22}] = \begin{bmatrix} [nu \cdot (md-1) \times nz] \\ [nu \times nz] \end{bmatrix}$$

$$[(\Gamma_{cl})_{31}] = [nc \times np]$$

$$[(\Gamma_{cl})_{32}] = [nc \times nz]$$

Appendix D: Integrals, delay at the controller's side

This appendix contains the intermediate calculations of integrals I_{11}, I_{12}, I_{22} stated in Section 5.3. They give the discretized weight matrix for the case with the delay situated at the controller's side. The first integral of (49) becomes, by inserting (47),

$$
\begin{aligned}
I_{11}h = & \int_{kh}^{(k+1)h} [e^{A(t-kh)}x_p(kh)]^T \overline{Q}_{11} [e^{A(t-kh)}x_p(kh)]dt \\
& + \int_{kh}^{kh+h} [e^{A(t-kh)}x_p(kh)]^T \overline{Q}_{11} \left[\int_{kh}^{t} e^{A(t-s')}B_p u(s'-\tau)ds' \right] dt \\
& + \int_{kh}^{kh+h} \left[\int_{kh}^{t} e^{A(t-s')}B_p u(s'-\tau)ds' \right]^T \overline{Q}_{11} [e^{A(t-kh)}x_p(kh)]dt \\
& + \int_{kh}^{kh+h} \left[\int_{kh}^{t} e^{A(t-s')}B_p ds' u(s'-\tau) \right]^T \overline{Q}_{11} \left[\int_{kh}^{t} e^{A(t-s')}B_p ds' u(s'-\tau) \right] dt
\end{aligned}
\tag{90}
$$

The integration is split up over two intervals to let the piecewise constant $u(kh)$ and $u(kh-h)$, from $u(s'-\tau)$ be moved out from the integral. However, the state $x_p(t)$ is constant over a sampling period, sampled at the time instant kh. Equation (90) can be written,

$$
\begin{aligned}
I_{11}h = & \, x_p^T(kh) \int_{kh}^{(k+1)h} \Phi_p^T(t,kh)\overline{Q}_{11}\Phi_p(t,kh)dt \cdot x_p(kh) + x_p^T(kh) \int_{kh}^{kh+h} \Phi_p^T(t,kh)\overline{Q}_{11}\Gamma(t,kh)dt \cdot u(kh-h) \\
& + x_p^T(kh) \int_{kh}^{kh+h} \Phi_p^T(t,kh)\overline{Q}_{11}\Gamma(t,kh+\tau)dt \cdot u(kh) + u^T(kh-h) \int_{kh}^{kh+h} \Gamma^T(t,kh)\overline{Q}_{11}\Phi_p(t,kh)dt \cdot x_p(kh) \\
& + u^T(kh) \int_{kh}^{kh+h} \Gamma^T(t,kh+\tau)\overline{Q}_{11}\Phi_p(t,kh)dt \cdot x_p(kh) + u^T(kh-h) \int_{kh}^{kh+h} \Gamma^T(t,kh)\overline{Q}_{11}\Gamma(t,kh)dt \cdot u(kh-h) \\
& + u^T(kh) \int_{kh}^{kh+h} \Gamma^T(t,kh+\tau)\overline{Q}_{11}\Gamma(t,kh+\tau)dt \cdot u(kh)
\end{aligned}
$$

Using the notation in (52), $\Gamma(t,\, kh) = \Gamma_{pb}(t)$ and $\Gamma(t,\, kh+\tau) = \Gamma_{pa}(t)$ the integration intervals are valid to the domains $(kh,\, kh+\tau\,]$ and $(kh+\tau,\, kh+h\,]$ respectively.

The second integral of (49) becomes

$$
I_{12}h = \int_{kh}^{kh+h} [e^{A(t-kh)}x_p(kh)]^T \overline{Q}_{12}u(t-\tau)dt + \int_{kh}^{kh+h} \left[\int_{kh}^{t} e^{A(t-s')}B_p u(s'-\tau)ds' \right]^T \overline{Q}_{12}u(t-\tau)dt \tag{91}
$$

and there is also an integral I_{21} due to the symmetry. Again the integration is split up over two intervals with piecewise constant $u(s'-\tau)$ to be moved out from the integral.

Equation (91) can be simplified using the notation in (52) into

$$I_{12}h = x_p^T(kh)\int_{kh}^{kh+\tau}\Phi_p^T(t)\overline{Q}_{12}dt\cdot u(kh-h) + x_p^T(kh)\int_{kh+\tau}^{kh+h}\Phi_p^T(t)\overline{Q}_{12}dt\cdot u(kh) +$$

$$u^T(kh-h)\int_{kh}^{kh+\tau}\Gamma_{pb}^T(t)\overline{Q}_{12}dt\cdot u(kh-h) + u^T(kh)\int_{kh+\tau}^{kh+h}\Gamma_{pa}^T(t)\overline{Q}_{12}(dt)\cdot u(kh)$$

Finally, the third integral of (49) becomes

$$I_{22}h = u^T(kh-h)\int_{kh}^{kh+\tau}\overline{Q}_{22}dt\cdot u(kh-h) + u^T(kh)\int_{kh+\tau}^{kh+h}\overline{Q}_{22}dt\cdot u(kh)$$

Appendix E: Integrals, delay on the process' side

Similar to Appendix D, the case with delays at the process' side is treated here. Multiple arrivals of the control signal is possible as well as $\tau > h$, of course. As before, the integrals I_{11}, I_{12}, I_{22} come from Section 5.3. However, in this general case, the sampling is not uniform and there can be multiple arrivals of a control signal during one sampling interval. From

$$
\begin{aligned}
I_{11}(t_{k+1}-t_k) &= x_p^T(t_k)\int_{t_k}^{t_{k+1}} \Phi_p^T(t)\bar{Q}_{11}\Phi_p(t)dt\cdot x_p(t_k) \\
&+ x_p^T(t_k)\sum_{g=0}^{ng}\left[\int_{t_k+\tau_{k,g}}^{t_k+\tau_{k,g+1}} \Phi_p^T(t)\bar{Q}_{11}\Gamma_{pg}(t)dt\cdot u(t_{k-(nd+g)})\right] \\
&+ \sum_{g=0}^{ng}\left[u^T(t_{k-(nd+g)})\cdot\int_{t_k+\tau_{kg}}^{t_k+\tau_{k,g+1}} \Gamma_{pg}^T(t)\bar{Q}_{11}\Phi_p(t)dt\right]x_p(t_k) \\
&+ \sum_{g=0}^{ng}\left[u^T(t_{k-(nd+g)})\cdot\int_{t_k+\tau_{k,g}}^{t_k+\tau_{k,g+1}} \Gamma_{pg}^T(t)\bar{Q}_{11}\Gamma_{pg}(t)dt\cdot u(t_{k-(nd+g)})\right]
\end{aligned}
\tag{92}
$$

and

$$
I_{12}(t_{k+1}-t_k) = x_p^T(t_k)\int_{t_k}^{t_{k+1}} \Phi_p^T(t)\bar{Q}_{12}dt\cdot u(t_k) + \sum_{g=0}^{ng} u^T(t_{k-(nd+g)})\int_{t_k+\tau_{k,g}}^{t_k+\tau_{k,g+1}} \Gamma_{pg}^T(t)\bar{Q}_{12}dt\cdot u(t_k)
$$

and

$$
I_{22}(t_{k+1}-t_k) = u^T(t_k)\int_{t_k}^{t_{k+1}} \bar{Q}_{22}dt\cdot u(t_k)
$$

and by identification of x_p and u, the submatrices of the discretized weight matrix become

$$
\begin{aligned}
q_{x,x} &= \frac{1}{t_{k+1}-t_k}\int_{t_k}^{t_{k+1}} \Phi_p^T(t)\bar{Q}_{11}\Phi_p(t)dt \\
q_{x,g} &= \frac{1}{t_{k+1}-t_k}\int_{t_k+\tau_{k,g}}^{t_k+\tau_{k,g+1}} \Phi_p^T(t)\bar{Q}_{11}\Gamma_{pg}(t)dt \\
q_{x,u} &= \frac{1}{t_{k+1}-t_k}\int_{t_k}^{t_{k+1}} \Phi_p^T(t)\bar{Q}_{12}dt
\end{aligned}
\tag{93}
$$

$$q_{g,g} = \frac{1}{t_{k+1} - t_k} \int_{t_k + \tau_{k,g}}^{t_k + \tau_{k,g+1}} \Gamma_{pg}^T(t) \bar{Q}_{11} \Gamma_{pg}(t) dt$$

$$q_{g,u} = \frac{1}{t_{k+1} - t_k} \int_{t_k + \tau_{k,g}}^{t_k + \tau_{k,g+1}} \Gamma_{pg}^T(t) \bar{Q}_{12} dt$$

$$q_{u,u} = \frac{1}{t_{k+1} - t_k} \int_{t_k}^{t_{k+1}} \bar{Q}_2 dt$$

with

$$\Phi_p(t) = e^{A(t - t_k)} \qquad t_k < t \le t_{k+1}$$

$$\Gamma_{pg}(t) = \int_{\tau_{k,g}}^{\tau_{k,g+1}} e^{A_p(t - t_k - s')} B_p \, ds' \qquad t_k + \tau_{k,g} < t \le t_k + \tau_{k,g+1}$$

the weight matrix can be assembled. The size and the assembly of the weight matrix depend on the triple $\{md, nd, ng\}$ and it must hold that $md \ge nd \ge ng \ge 0$. If $nd > 1$, then

$$Q(t_k) = \begin{bmatrix} q_{x,x} & 0 & q_{x,0} & \cdots & q_{x,ng} & 0 & q_{x,u} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ q_{x,0}^T & 0 & q_{0,0} & \cdots & 0 & 0 & q_{0,u} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ q_{x,ng}^T & 0 & 0 & \cdots & q_{ng,ng} & 0 & q_{ng,u} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ q_{x,u}^T & 0 & q_{0,u}^T & \cdots & q_{ng,u}^T & 0 & q_{u,u} \end{bmatrix}$$

where the inserted zeros depends on ng and md. If $nd = 1$ and $ng = 1$ then

$$Q(t_k) = \begin{bmatrix} q_{x,x} & q_{x,0} & q_{x,1} + q_{x,u} \\ q_{x,0}^T & q_{0,0} & q_{0,u} + q_{1,u} \\ (q_{x,1} + q_{x,u})^T & (q_{0,u} + q_{1,u})^T & q_{1,1} + q_{u,u} \end{bmatrix}$$

The submatrices in (93) can be combined to form the corresponding submatrices in the case of delay at the controller's side (that derivation is not shown here).

Appendix F: Calculating the integrals

With the numerical tool Matlab, the integrals can easily be calculated using matrix exponentials, see Van Loan (1978) and also Franklin et al. (1990) and compare with Matlab's control toolbox function *lqrd.m*. There are also other ways to calculate the integrals. The matrix exponential is denoted *exp*. The symbol $\varnothing$ acts as a place holder for parts of the result not useful when obtaining the intended integral. $\varnothing$ does not represent any particular expression and an eventual subindex denoting position in a matrix is suppressed. Numerical issues might have to be considered when calculating the matrix exponentials.

F.1 System and input matrices

The matrices describing the discrete time state update equation is found by:

$$\Phi_p(h) = \exp(A_p h) = s_0 s_1 \cdots s_{ng}$$

and

$$\Gamma_{pg}(t, \tau_{k,(g+1)}, \tau_{k,g}) = s_{ng} s_{ng-1} \cdots s_{g+1} \sigma_g$$

and

$$\exp\left(\begin{bmatrix} A_p & B_p \\ 0 & 0 \end{bmatrix} (\tau_{k,g+1} - \tau_{k,g}) \right) = \begin{bmatrix} s_g & \sigma_g \\ \varnothing & \varnothing \end{bmatrix} \tag{94}$$

Recall that $\exp(0) = I$.

F.2 Sub-weights for delay at the controller's side

The matrix exponential is split up and calculated for each partial delay, $g = 0, \ldots, ng$:

$$\left(\begin{bmatrix} q_{11} & q_{12} \\ q_{12}^T & q_{22} \end{bmatrix} \right)_g = (s_{22}^T)_{g+1} (s_{12})_{g+1} - (s_{22}^T)_g (s_{12})_g$$

and

$$q_{x,x} = \frac{1}{t_{k+1} - t_k} \sum_{g=0}^{ng} (q_{11})_g$$

$$q_{x,g} = \frac{1}{t_{k+1} - t_k} (q_{12})_g$$

$$q_{g,g} = \frac{1}{t_{k+1} - t_k} (q_{22})_g$$

using

$$\exp\left(\begin{bmatrix} -A_p^T & 0 & \bar{Q}_{11} & \bar{Q}_{12} \\ -B_p^T & 0 & \bar{Q}_{21} & \bar{Q}_{22} \\ 0 & 0 & A_p & B_p \\ 0 & 0 & 0 & 0 \end{bmatrix} \tau_{k,\,g}\right) = \begin{bmatrix} \varnothing & (s_{12})_g \\ \varnothing & (s_{22})_g \end{bmatrix}$$

Φ_p and Γ_{pg} could also be extracted from this equation, as an alternative to (94). Recall the convention $\tau_{k,\,0} = 0$ and $\tau_{k,\,(ng+1)} = h$.

F.3 Sub-weights for delay at the process' side

The six different parts of the loss matrix are given by the expressions below, with $h = t_{k+1} - t_k$.

$$q_{u,\,u} = \frac{1}{h} s$$

calculated with the matrix exponential *exp*

$$\exp\left(\begin{bmatrix} 0 & \bar{Q}_{22} \\ 0 & 0 \end{bmatrix} h\right) = \begin{bmatrix} \varnothing & s \\ \varnothing & \varnothing \end{bmatrix}$$

and

$$q_{x,\,x} = \frac{1}{h} \exp(A_p^T h) s$$

with

$$\exp\left(\begin{bmatrix} -A_p^T & \bar{Q}_{11} \\ 0 & A_p \end{bmatrix} h\right) = \begin{bmatrix} \varnothing & s \\ \varnothing & \varnothing \end{bmatrix}$$

and

$$q_{x,\,u} = \frac{1}{h} \exp(A_p^T h) s$$

with

$$\exp\left(\begin{bmatrix} -A_p^T & \bar{Q}_{12} \\ 0 & 0 \end{bmatrix} h\right) = \begin{bmatrix} \varnothing & s \\ \varnothing & \varnothing \end{bmatrix}$$

and

$$q_{x,\,g} = \frac{1}{h}\left[\sigma_1^T s_1 - \sigma_0^T s_0 \;\; \cdots \;\; \sigma_{g+1}^T s_{g+1} - \sigma_g^T s_g \;\; \cdots \;\; \sigma_{ng+1}^T s_{ng+1} - \sigma_{ng}^T s_{ng}\right]$$

with

$$\exp\left(\begin{bmatrix} -A_p^T & \bar{Q}_{11} & 0 \\ 0 & A_p & B_p \\ 0 & 0 & 0 \end{bmatrix}\tau_{k,\,g}\right) = \begin{bmatrix} \varnothing & \varnothing & s_g \\ \varnothing & \sigma_g & \varnothing \\ \varnothing & \varnothing & \varnothing \end{bmatrix}$$

and

$$q_{g,\,u} = \frac{1}{h}\begin{bmatrix} s_1 - s_0 & \cdots & s_{g+1} - s_g & \cdots & s_{ng+1} - s_{ng} \end{bmatrix}^T$$

with

$$\exp\left(\begin{bmatrix} 0 & B_p & 0 \\ 0 & -A_p^T & \bar{Q}_{12} \\ 0 & 0 & 0 \end{bmatrix}\tau_{k,\,g}\right) = \begin{bmatrix} \varnothing & \varnothing & s_g \\ \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing \end{bmatrix}$$

and finally

$$q_{g,\,g} = \frac{1}{h}\begin{bmatrix} 2B_p^T(\sigma_1^T s_1 - \sigma_0^T s_0) & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & 2B_p^T(\sigma_{ng+1}^T s_{ng+1} - \sigma_{ng}^T s_{ng}) \end{bmatrix}$$

with

$$\exp\left(\begin{bmatrix} -A_p^T & I & 0 & 0 \\ 0 & -A_p^T & \bar{Q}_{11} & 0 \\ 0 & 0 & A_p & B_p \\ 0 & 0 & 0 & 0 \end{bmatrix}\tau_{k,\,g}\right) = \begin{bmatrix} \varnothing & \varnothing & \varnothing & s_g \\ \varnothing & \varnothing & \sigma_g & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \end{bmatrix}$$

F.4 State noise covariance and additional loss

The state noise covariance and the loss due to this disturbance can be calculated by

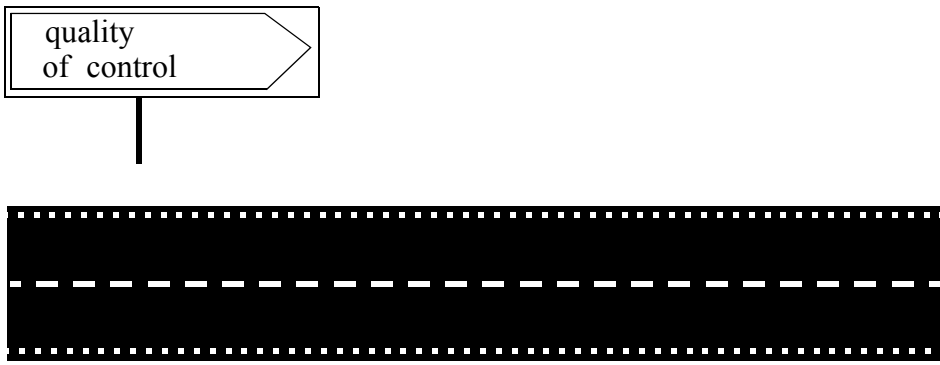
$$R_v = \frac{1}{h}\exp(A_p h)\sigma \quad \text{and} \quad J_v = \frac{1}{h}\operatorname{tr}(\exp(A_p h)s)$$

with

$$\exp\left(\begin{bmatrix} 0 & I & 0 \\ 0 & -A_p & \bar{R}_v \\ 0 & 0 & A_p^T \end{bmatrix}h\right) = \begin{bmatrix} \varnothing & \varnothing & s \\ \varnothing & \varnothing & \sigma \\ \varnothing & \varnothing & \varnothing \end{bmatrix}$$

Paper C

Martin Sanfridson, Martin Törngren and Jan Wikander, “A quality of control architecture and codesign method“, work in progress conference paper at RTAS’04, May 2004.



A quality of control architecture and codesign method

Martin Sanfridson, Martin Törngren and Jan Wikander
{mis, martin, jan}@md.kth.se
Mechatronics Lab, Royal Institute of Technology, KTH, Stockholm

Abstract: An architecture and a method supporting codesign of flexible computer control systems are proposed in this paper. We call this Quality of Control to reflect the emphasis on control performance. Key components of the architecture include negotiation to optimize overall quality, admission control, explicit specifications of control characteristics and timing constraints, and on-line estimation of the control performance as a function of the actual system timing. The purpose is to give flexibility in terms of scalability, maintainability, configurability and graceful degradation. We describe and exemplify the implementation of the conceptual architecture and its codesign approach.

Keywords: Quality of Control, feedback control, control performance, admission, embedded real-time system, scalability, QoS.

1 Introduction

The traditional design of an embedded system is rather static: a fixed design specification is mapped on a fully known platform, where a good a priori schedulability analysis requires the task set to be well-known, Stankovic et al. (1996). A change in the specification or the platform will force a redesign. The implementation effect of the *sampling period, delay and their jitter* respectively, is not — if ever — investigated until the system is integrated and tested. In a scalable system, vital information regarding the real-time aspects becomes available first at a very late design stage, or maybe not until runtime. There are several sources of variations in the configuration and real-time behaviour, e.g. functionality added on-line, variations in the execution time due to application characteristics, or due hardware failure.

No one would endure an office LAN incapable of scalability and most people expect a personal computer to have some plug-and-play capabilities. Such complex systems evolve step by step rather than being completed in the first design specification. Loosely speaking, our intention is to require these properties for dedicated computer control systems, where the real-time, cost and dependability requirements are tougher. This calls for a more dynamic approach based on a higher abstraction in the design specification, i.e. quality. An increased number of functionalities in an *embedded distributed* system threatens to lead to an increased number of nodes. For example, today's mid-range car has several tens of nodes and the number increases for every new model. Such a trend is obviously not sustainable although many functionalities need dedicated hardware.

Decomposition into modules is a way to handle complexity and it also offers scalability and reuse. *Scalability* is the ability of a computer system to be extended, changed and reduced. Scalability is the foundation, but in order to enable scalability for control systems too, the *codesign* of control and real-time issues must be handled in a wider context. The purpose of the

proposed architecture is to cope with the complexity of software development for flexible control systems, primarily with respect to real-time requirements and control performance. To this end the architecture includes an adaptation mechanism with the aim of increasing the overall satisfaction, by rescheduling available resources. *Quality of Control*, QoC, is a runtime strategy where the performance of a set of applications with real-time requirements is managed. The controller is often only one part of a large system, but the fact that a controller is present should not stop future use of scalability.

Many embedded systems are assembled from components *off the shelf*, by proprietary or open standards, possibly as a cooperative product development effort between several divisions, subcontractors and customers. The time to market is an important parameter, but the design process is a complex affair. It should be clear that reusable components can shorten the lead time (Artist road map on component based design, 2003). The scalability and the proposed middleware should reduce the complexity to give flexibility for the customer, even though the initial design effort to make proper modules can be high. However, considering state of practice, improvements are also possible for traditional static designs, especially in terms of codesign.

The traditional text book view of the codesign, is that the implemented controller has a fixed *period* and the resulting *latency* is assumed to be constant. Since for example scheduling and synchronization introduce time-variations in periods and response times, it is clear that this model indeed aims to simplify the codesign. This has been pointed out in many works recent years, e.g. Cervin (2003), Martí et al. (2002), and Sanfridson (2000b). A real-time scheduling viewpoint of tackling the problem is to reduce the jitter. The solution is specific to the chosen scheduling strategy, but typically increases the average delay by what can be modelled as a buffer. Another way is to accept jitter and account for it in the control design, see e.g. Nilsson (1998) and Rehbinder and Sanfridson (2004). Typical for the codesign is that there are many design parameters and complex relations between them and the resulting control performance.

1.1 Contributions

The proposed architecture is a continuation of the work in Sanfridson (2000b), which focused on a specific hardware and scheduling policy. An experimental scalability layer was implemented on top of the commercial RTOS called OSE Epsilon including a link handler for the communication bus (Österman, 2001). It was proven successful, but the quality negotiation was never fully tested. The contribution of the currently proposed architecture is the top-down approach trying to tackle the problem of providing scalability, which in the best-case scenario could help decrease time to market, encompass applications having different kinds of real-time requirements, and facilitate both a product family based on the assembly of modules and product upgrades.

Control applications admit scalability and can thus participate in negotiations to increase the steady state control performance. The control performance requirements and the allowable signal domains are explicitly documented in the specification. A corner stone is the *temporal separation* of the optimization and scheduling, i.e. the computationally costly optimization is either event triggered or time triggered, and invoked quite rarely compared to the applications.

This *separation of concerns* does not only make the architecture easier to understand but also enables a construction of a middleware on top of many existing RTOSes for distributed systems. A return to familiar grounds is the need for worst-case execution time estimations and hard deadlines, for the sake of predicting real-time performance in the short term, avoiding uncontrollable bursty overload that threatens to violate the *allowable state space*.

2 Architecture preliminaries

2.1 Application characteristics

In a computer control system, several kinds of applications can be found. The intention of the architecture is to provide explicit support for flexible controllers and to allow them to coexist with other applications. First, a few assumptions regarding the applications.

For a typical control law implemented as a task, the execution time is fairly constant from one instance to the next. The partitioning and allocation are primarily static in an embedded system, and the scalability features are relatively seldom used. A worst-case design is both desirable and workable in such a playground, open only to known and well specified extensions and replacements. In contrast to many other kinds of applications, controllers are sensitive to the actual period, period jitter, delay and delay jitter. The worst-case design should be accompanied by a consideration of the average behaviour which defines the quality or steady state control performance. A computer control system is typically safety-critical, which means that critical tasks, including tasks closing the loop, cannot be allowed to exhibit excessive latencies.

By carrying out design using different periodic *task models,* an application's requirements can be reflected better and the hardware resources may be used more efficiently. Although the inherent inertia of a dynamic system can explain its tolerance to occasional deadline misses, it cannot be modelled by a soft real-time task in the design, without a constraint on the likelihood of consecutive deadline misses. It may be necessary in many scheduling strategies to impose hard real-time to get predictable latencies, leading to predictable application behaviour.

A task is here characterized in three dimensions: mission-criticality, deadline requirement and timeliness requirement, see Figure 2.1. We informally define timeliness as the tightness of jitter measured relative to a suitable time constant. Strong timeliness implies difficulties in scheduling. The idea to distinguish between urgency and importance was pointed out in Jensen (1992).

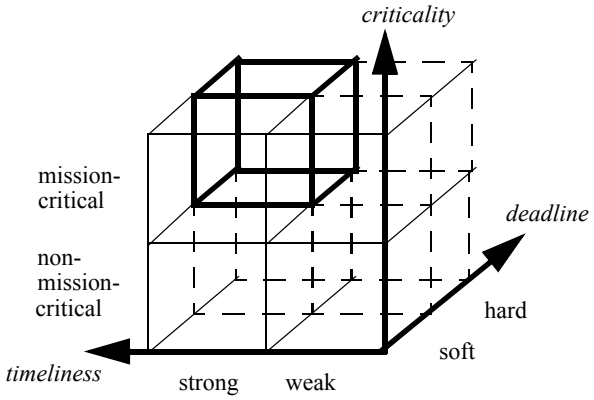


Figure 2.1 Task models: mission-critical vs. deadline vs. timeliness.

The cube drawn with bold lines contains the most important and strongly timeliness sensitive applications, requiring hard real-time guarantees. The idea behind a differentiation is that applications should not be assigned to a more demanding category than necessary, in order to facilitate schedulability and an increase of the resource utilization. Not all types of applications

are susceptible to jitter and for some types of applications it does not make sense to optimize for e.g. period and latency, since it will not improve the perceived or measured quality. The implementation rarely considers importance to be a separate dimension, and in for example priority driven scheduling strategies, the attempt is to lump the three dimensions in Figure 2.1 together. With a large number of soft, weak and non-mission-critical applications, it becomes easier to maximize the utilization and to find a feasible schedule. Control tasks can have strong timeliness and soft deadline requirements (which covers so called firm tasks).

2.2 Definitions

In QoC, the requirements of a controller, in terms of performance with respect to real-time behaviour, are supervised and the available hardware resources are administered, e.g. by negotiation between all applications, to render a feasible or optimal distribution. QoC can be combined with a *best-effort* strategy, for example, if the delay jitter increases, a robust control law designed for that situation could be deployed.

The QoC management, especially the optimization, represents the *long term* perspective, whereas the scheduler represents the s*hort term*. The *temporal isolation* makes it possible to combine well-known scheduling strategies and optimization methods.

QoC differs from the control performance assessment found primarily in the process industry, where the variance of various signals is measured and the estimation is affected by many disturbances other than time-varying periods and computational delays. Also, QoC should not be mixed up with model predictive control, which aims to find an optimal control law at every sampling instance. Finally, QoC should not be confused with Quality of Service, which is associated with soft real-time, quantitative resource reservation, and multimedia over the internet. For QoS in general and terminology, see Sanfridson (2000a) or Artist road map on adaptive real-time systems (2003).

2.3 Requirements

A list of requirements for the architecture can be set up:

(A1) *A scalable computer system*. The ability to add, remove and replace hardware and functionality is a property strongly associated with a modular design concept.

(A2) *Adaptive scheduling strategy.* It must be possible to affect periods and delays, since the performance of a control application can be described as a function of these. It should be possible to optimize the use of e.g. processing units and communication channels.Tasks and messages should be possible to reconfigure, start and stop.

(A3) *Timing guarantees*. The global real-time scheduling must allow for a predictable processing in the short term, possibly combined with application specific overload handling. The timing behaviour must be measurable. This effectively means that a worst-case design is needed, i.e. a schedulability test is a necessity.

(A4) *Separation of concerns*. A generic supervisory QoC manager should be functionally separated from the applications as much as possible.

(A5) *Separation of time scales*. The cascaded QoC manager should be invoked less often than the applications. This will preclude propagation of timing disturbances from the adaption of the scheduling to the closed loop dynamics, and it will also allow computationally intensive optimization algorithms.

3 Architecture for QoC

3.1 Building blocks, architecture overview

The basic building blocks of the architecture are depicted in Figure 3.1. At the bottom we find the hardware and RTOS. The scalability layer, if not part of the RTOS, manages the discovery, join and disconnection of applications and hardware. In the basic layer, the *elementary functions* (EF) are found. They perform the application specific work at a low level, that would be executed even on a bare bone system, e.g. a a sensor reading or a control law computation. The *application function* (AF) constitutes in principle a specification of an application at an abstraction level above the EF. The AF contains a task graph describing the logical and temporal coupling of its EFs. The AF also contains a specification of how the performance for the specific application is computed, and how the latency and jitter is measured. Every EF is part of at least one AF and some EFs might require mutual exclusion.

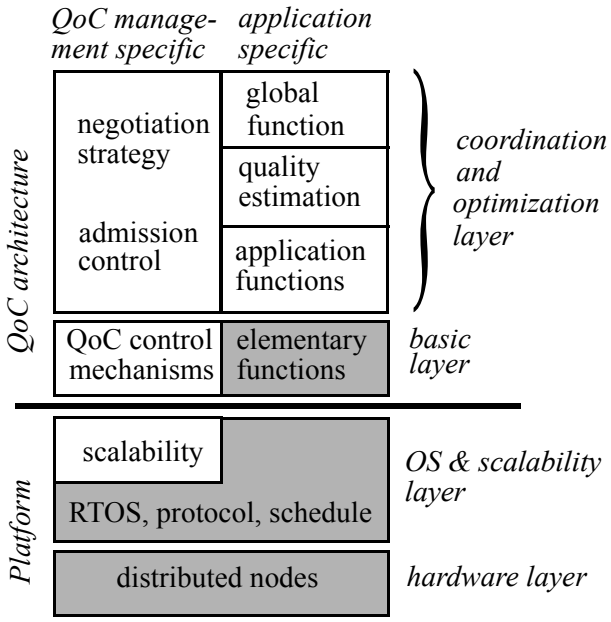


Figure 3.1 Basic building blocks of the QoC architecture.

The *global function* GF, is a specification of the whole system, at an abstraction level above the AF. It contains the function logic for cooperating AFs, functional mutual exclusion of AFs, exception handling, etc. The global optimization strategy and admission control are generic but depend of course to some extent on the applications. In order to negotiate, basic *QoC control mechanisms* are needed to gather information regarding the timing behaviour, for book-keeping to minimize the management overhead, for monitoring deadline misses, for effectuating changes, etc. The QoC control mechanisms are non-application specific, but depend on the choice of RTOS, the protocols, the scheduling strategy, etc.

The union of the shaded boxes in Figure 3.1, as well as parts of the configuration and functional logic both in AF and GF, can be said to comprise the traditional setup, which excludes QoC related features. The left side of the figure, depicts the *QoC management specific* parts and the right side shows the *application specific* parts. The middleware is also split into one basic layer and an optimization layer. The AF and EF have no information about the schedul-

ing or optimization strategy being used. The EF does not know how to compute the performance, but can help measure the latency and jitter. To handle e.g. product variants or graceful degradation, alternative versions of AFs having the same purpose but using an alternative setup of EFs, can be deployed.

The diagram in Figure 3.2 depicts the data flow of the supervisory feedback loop. A control loop (AF_1) using three EFs, is found at the top. The thick arrows represent the execution strategy control (triggering, scheduling and communication), and the supervision to measure the timing behaviour of an AF's task chain. The measured actual timing is used for the model based performance estimation. The negotiator compares the demands of different AFs with the estimated performance and tries to find new values for scheduling parameters such as period, priority, offset.

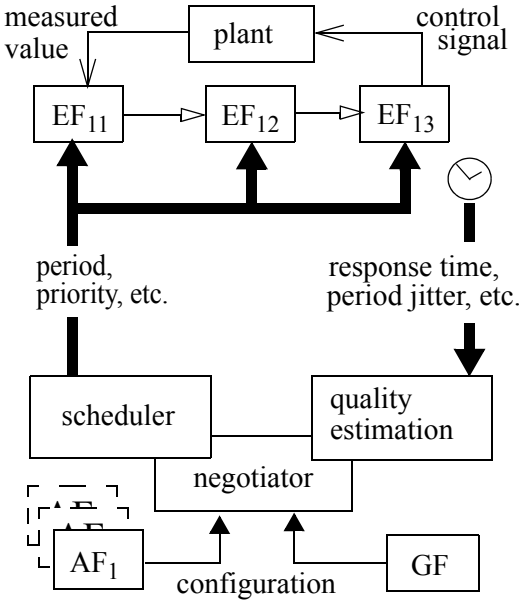


Figure 3.2 A simplified flow diagram of the cascaded negotiation (bold lines).

3.2 Quality estimation and optimization

Figure 3.3 shows three examples of *benefit functions* B, each normalized to a unit peak value, as a function of some timing property — time-varying delay $\tau(t)$ or period $h(t)$. A multi-dimensional benefit function, which is hard or impossible to visualize, extends the mapping to combinations of delays, periods and their jitter respectively.

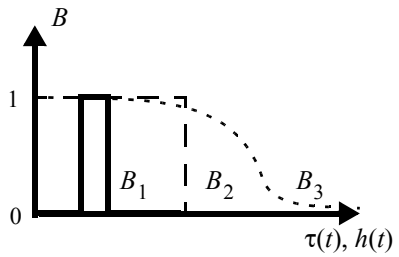


Figure 3.3 Three examples of benefit functions.

Now, we need a strategy to optimize the total benefit or quality, B_{tot}. Applications typically have different objectives but they contribute to the same global goal. Since there are several types of cooperating applications (a controller is just one of these) using the same hardware, a so called *tutti-frutti problem* must be tackled. The way to express this mix of apples and bananas usually boils down to a sum $B_{tot} = \sum_k w_k B_k$ (or similar), with a normalized benefit function B_k scaled by the weight w_k, for an application function AF_k. The drawback is naturally that the choice of w_k is left to the discretion of the system designer or user. On the other hand, this abstraction helps decouple the short term timing properties ($\tau(t), h(t)$) from the long term quality, of which the latter is closer to the heart of the matter.

A vital part of the negotiation is the *optimization routine*. There are many ways to device an optimization routine which uses the benefits as objective functions, constrained by the allowable ranges (or values) of h and τ given by the design specification. Issues to consider are e.g. if B_k is convex, if the optimization routine is periodic or event triggered, the computational cost, and the acceptance of suboptimal solutions. The optimization can be based on convex optimization, simulated annealing, fuzzy logic, micro economic models, etc. There are numerous of references on usable ideas like these in the vast literature of QoS, see Artist road map on adaptive real-time systems (2003), but the choice of optimization routine is hardly decisive for the success of the scalability concept.

A benefit function is needed for every AF participating in the negotiation. As a benefit function for a control application, a *performance function* integrating the continuous time squared and weighted control and measurement signals can be the origin, see e.g. Cervin (2003), Nilsson (1998), and Sanfridson (2000b). This has been described in previous literature and will not be elaborated in this paper, but a few points have to be clarified. The control performance measure is calculated based on a discretized model of the controller and the plant. The performance is a function on the possibly time-varying period $h(t)$ and delay $\tau(t)$, see Figure 3.4. An example of this dependence is given Section 4. A global clock, or a similar concept, is needed to accurately calculate the latencies and jitter on-line. It is the quality associated with an AF that is of interest, and a single EF within the task chain cannot estimate the control performance, since it is normally based on an end-to-end delay.

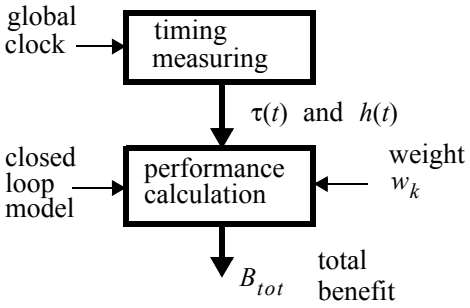


Figure 3.4 Model based estimation of the total benefit B_{tot} based on control performance.

The performance function is based on steady state conditions, i.e. the average performance — the performance during a *long term* perspective. Care must be taken that a transient error, e.g. adjacent skips due to temporary overload, does not lead to a violation of the allowable range of the state vector, measured value or control signal. The performance function, based on a special form of the H_2-norm, is well applicable to time-varying periods and delays including transient timing errors. However, it is not sufficient to measure the impact of transient timing

errors. A resulting individual signal peak — possibly violating limits causing system failure, but not instability — is blurred out when its variance is considered over a sufficiently long time horizon. The system gain, or the H_∞-norm, is actually better suited for this robustness issue, see e.g. Skogestad and Postlethwaite (1996), but there is a lack of supporting theory here. To complicate matters more, the allowable range of the state variable etc. typically depends on the current operating conditions, cf. Seto et al. (1998). In order to guarantee the steady state performance, timing behaviour in the *short term* perspective has to be well-behaved and predictable. This can be enforced by introducing hard relative deadlines, which of course can be longer than the period. The deadline is determined at design time by the sensitivity to transient errors, delay jitter and constant delay. In an advanced strategy, time-varying deadlines are possible. Violation of a hard (end-to-end) deadline implies violation of the domain over which the performance (benefit) function is specified. Thus, flexibility is achieved while staying inside the allowable ranges of period and latency.

3.3 Admission control

Admission control belongs to the long term time scale and is a part of the QoS management. Admission control is a necessary regulation of hardware resources and has more to do with mission critical issues than timeliness and benefit, but there is a connection between mission-criticality and benefit. In order to uphold hard real-time requirements, it will occasionally be necessary to gracefully terminate a non-mission-critical AF, or to deny start-up. An AF with tough real-time requirements can on the other hand be of relatively minor importance, and be shut down to the benefit of mission-critical applications. It is convenient to arrange AFs in order of *mission-critical priority* or importance. Two classes can be distinguished within this range: a mission-critical and a non-mission-critical. The non-mission-critical AF does not cause a system failure if forced to terminate or denied admittance.

Different admission policies are of course possible. An example is if two sets of AFs render the same total benefit, then the set with the greatest number of elements should be selected. An EF is in the simplest case mapped directly to a task or message. The task graph of an AF needs to be taken into account when its EFs are forced to terminate, since more than one AF can use the same EF, e.g. a (read only) sensor. Using multiple mutually exclusive AF having the same purpose but different configurations of EF is a way to implement *graceful degradation* (limp home mode).

Overload handling belongs, on the contrary, to the short time scale. A deadline miss can be interpreted as a situation of a temporary or even an imminent permanent overload. For a hard real-time task (i.e. where the specified deadline must be met) it should not occur, but it could for example be that the execution time becomes longer than the assumed worst-case. A traditional and simple solution to this unusual situation is skipping instances, but the situation can violate the allowable state space and jeopardize the stability of the control loop.

3.4 Negotiation and activation

The QoC management can be described as a cascaded feedback loop supervising the quality over a long time horizon based on steady state models, cf. the process optimization in Skogestad and Postlethwaite (1996). The separation of activation rates between optimization and task chains has several benefits: the optimization routine can run in the background being a computationally expensive soft and weak task (see Figure 2.1), the QoC management stays more logically separated from the scheduling algorithm, and additional modes in the dynamics of the

application are avoided. Of course, there can still be problems with a transition from one set of scheduling parameters (or strategy) to another. First, the transition must be schedulable and second, a switch of control laws must be *bumpless*, to deny transient upsets in the control signal.

The activation can be *periodic* or *event-triggered*. The choice depends on the purpose and the information available, see Table 3.1. If the effect on period and delay after a change in the scheduling or synchronization is predictable, the quality can be estimated for a hypothetical tuple $\{h(t), \tau(t)\}$, which is an advantage. However, since the mapping from a change of parameters to control performance is difficult to obtain, the optimization is better based on a periodical feedback approach: the scheduling parameters are altered, the result is observed and fed back. A combination is suggested: at configuration time a default configuration is used and at runtime the information is updated. The idea of a *one shot* negotiation is that parts of the temporal design is postponed until a very late design time or even run time, when vital information readily becomes available.

Table 3.1 Activation, purpose and information needs.

Activation	**Purpose**	**Information needs**
One shot	Configuration of an assembly of modules	The real-time behaviour is *predictable* and the quality estimated. Default values can be used.
	Reconfiguration after upgrade/repair	
Event triggered	Scalability	
	Mode changes	
Periodic	Continuous supervision	The RT behaviour is *measurable* and quality estimated.

A state chart of the negotiation is depicted in Figure 3.5. The negotiation can be aborted e.g. if the expected benefit of the negotiation only leads to an insignificant improvement. If a periodic invocation is used, latency and jitter measurement and statistical processing can be performed continuously, even when in the idle state. The activation interval of the periodic negotiation should be long enough to record enough data to analyse the control performance, and considerably long compared to the dynamics of the application, not to risk inducing additional dynamics into the control loop. The event triggered negotiation can run in special start-up or configuration modes and be invoced manually or automatically by the designer or the end-user.

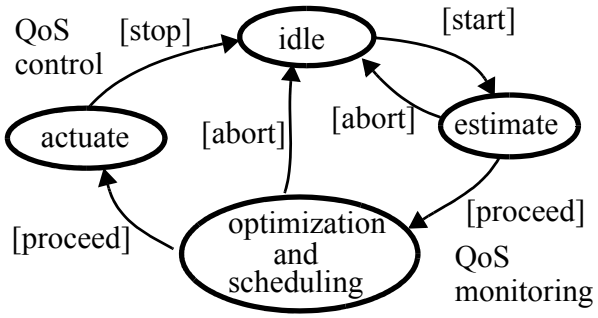


Figure 3.5 State chart of the negotiation, with benefit estimation.

3.5 Codesign method

The proposed architecture is aimed at facilitating the design process, but specifications and functional design including integration, still have be made at design time.

A main principle is the division of work in the codesign, cf. Figure 3.1. The role of the control engineer is to make a control synthesis and specify the allowable domain of the timing properties. For each AF, a benefit function must be supplied together with a domain over which the benefit function is valid. The deadline is determined at design time by the sensitivity to transient errors, delay jitter and constant delay.

The tasks of the computer engineer include among other things the scheduling activity. Every EF has to be partitioned and allocated, presumably to dedicated hardware. Specific combinations of EF must be approved, implemented, verified and tested to encompass any planned use of the scalability feature. Worst-case execution time still needs to be estimated and suitable deadlines assigned. It must still be assured that there is enough hardware resources for the minimum requirements.

The responsibility of the system engineer is to assign priorities for mission-criticality, together with weights to set the relative quality of each application in the global negotiation. In a flexible system, we want to adapt scheduling parameters on-line, which means that a set of control laws must be designed to encompass predictable conditions.

Another idea of the separations outlined in Figure 3.1, is that the programmer implementing EFs should be liberated from reinventing generic functionality that could belong to a configurable middleware.

3.6 Implementation

The proposed architecture is not bound to any specific RTOS, middleware or hardware, and any suitable scheduling policy can be considered. The architecture however requires that the implementation supports the requirements listed in Section 2.3.

Given these requirements there are several implementation alternatives that could be based on a combination of state of the art techniques. Some of the required techniques are already available in commercial RTOSes, for example for supervising and measuring the execution times of threads, for starting and stopping threads, and supervising overruns, see e.g. Artist road map on adaptive real-time systems (2003). The communication can be efficiently implemented using

for example real-time publisher-subscriber communication styles (see e.g. Seto et al. (1998) and the Artist road map on component based design, 2003).

No layer is needed between the OS and the elementary functions, i.e. a message containing application specific data can be sent directly from one EF to another, and the receiver's address is mediated by the AF. An alternative, which probably means more overhead, is that all communication is handled through the scalability layer.

The availability of a global clock in a distributed system largely facilitates measuring the actual timing as well as tailoring the desired timing (through scheduling and synchronization). However, measurements and estimations of end-to-end delays are also possible in asynchronous systems (Törngren et al., 1993) based on run-time measurements of varying sub-delays being part of the end-to-end delay.

The overall modularization of the architecture can also be accomplished in several ways. Using individual tasks for each EF is one straightforward solution, while it is also possible to combine several EFs into one task. The implementation must however ensure that the partitioning supports the timing measurements such that end-to-end delays can be captured properly.

4 Example

The purpose of this section is to give a short example of the proposed architecture and to bring up the issues discussed. We will illustrate that the architecture is flexible and permits an optimization of the resource usage.

4.1 Scenario

The computer system controlling the dynamic system consists of three nodes interconnected by a bus. The task graph in Figure 4.1 also shows the static partitioning and allocation of elementary functions to tasks. EF_{X1}, EF_{A1} and EF_{B1} are periodic tasks that can be phased relative to each other by defining offsets, and the other EFs are mapped to event triggered tasks.

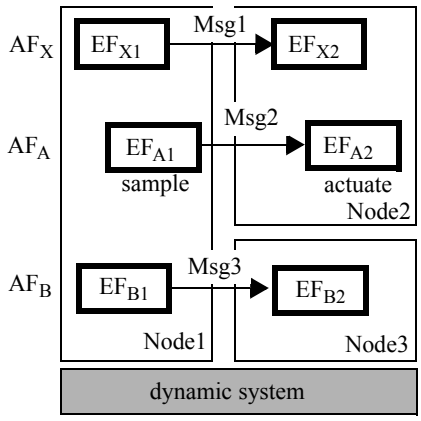


Figure 4.1 Deployment diagram and task graphs of AF_X, AF_A and AF_B.

The overhead of the OS and middleware is omitted for simplicity. The scheduling strategy is chosen to be preemptive fixed priority scheduling (FPS) on the nodes and non-preemptive FPS on the bus. Global schedulability is guaranteed by calculating the worst-case end-to-end

response times. The end-to-end response times are calculated from the arrival of the first EF to the finish of the last EF in a task chain.

4.2 Applications

AF_X is a hard weak mission-critical application, e.g. a supervision. AF_A is a soft strong mission-critical control application with timeliness requirements specified by a scaled quadratic loss function. The loss depends on the delay and period — the higher the period and delay the lower the benefit, see Figure 4.2. The plant is a second order system with a relative damping of 0.5 and a natural frequency of 0.02. It is controlled by a PI-controller having a proportional gain 0.02 and integral gain 100 time units.

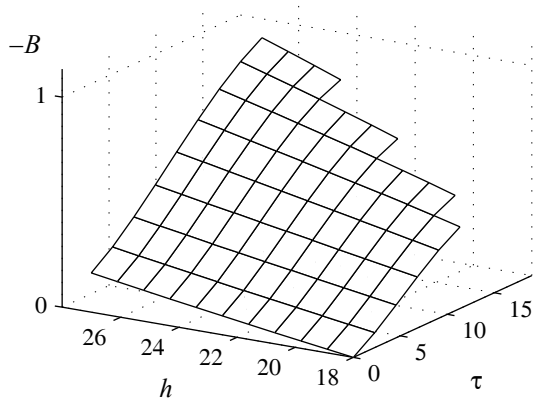


Figure 4.2 Normalized benefit of AF_A as a function of constant h and constant τ.

AF_B is a soft weak non-mission-critical application with a normalized benefit function of $-B = (1/(35-15))(h-15)$, i.e. linear in the period. Table 4.1 provides more information of the AFs.

Table 4.1 End-to-end figures in custom time units.

	AF_X	**AF_A**	**AF_B**
Mission-critical	yes	yes	no
Hard real-time	hard	soft	soft
Timeliness	weak	strong	weak
Execution times	2, 1, 2	3, 1, 2	6, 4, 2
Deadline, e2e	2h	h	n/a
Weight	0	0.9	0.1
Periods, h	9, 16	18, 27	15, 23, 35
FPS priority	high	intermediate	low

4.3 Use cases

The architecture is exemplified by four consecutive use-cases:

(i) *Power up*. A one-shot negotiation based on defaults is performed. EF belonging to AF_X and AF_A start (but not AF_B), given the periods 9 and 27 time units respectively without a proceeding negotiation. The schedulability test is made on-line.

(ii) *Admission control*. The AF_B is discovered and a negotiation based on defaults is triggered by this event, then AF_B joins after an on-line schedulability test.

(iii) *Periodic negotiation*. The periodic recording and analysis of response times helps fine-tune the control performance with respect to period and time-varying delays. AF_A is mixed with AF_B in this tutti-frutti problem by computing their benefit functions, normalized to a maximum value of one. In the succeeding negotiation, altering the offsets of EF_{X1}, EF_{A1} and EF_{B1}, combined with altering the periods AF_{A1} and AF_{B1}, is the method to affect the timely behaviour. An on-line schedulability test is done before the changes are effectuated. The process is repeated until the optimization does not yield any significant improvement on the margin. The optimization is based on a simulated annealing algorithm, suitable for a combinatorial problem lacking intuitive features, but instead of simulation the actual system is used. The period of the negotiation is set two orders of magnitude longer than the dynamics of the plant.

(iv) *Overload handling*. The overload handling is very application specific. For example, for AF_X, a temporal graceful degradation is a resort to make the best in a bad situation. Instead of shutting down the whole system, every AF is immediately forced to take on their lowest resource usage levels as an emergency precaution. This can be combined with a functional graceful degradation. Another example of exception handler, but for AF_A, is just doing nothing if the deadline is violated and a single instance is forced to be skipped. This case is outside the domain of the benefit function and the success depends on the closed loop dynamics, the current and future reference values, the current and future plant disturbances and near future timing errors of the computer system.

The implementation of the optimization starts with a schedule simulation, which gives the timing behaviour (in place of the real execution). This is in turn fed into a function which computes the control performance the same function that was used to generate Figure 4.2. The performance gives the benefit, which is weighted and scaled. A random task chain is chosen and its offset and period are changed. If the change improves the total benefit it is accepted. If not, it can be accepted anyway if the “temperature” in the simulated annealing procedure is sufficiently high. The procedure is repeated and a local minima is typically the result when the system gradually “cools”.

4.4 Results

As a result of the optimization, the total benefit has increased, see Figure 4.3. The encircled roman numbers refer to the start of these two use-cases respectively.

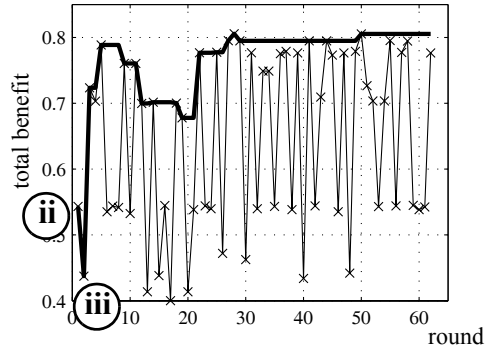


Figure 4.3 Total benefit. Bold line represents the best value that the algorithm keeps, and thin reversed combinations.

The simulated annealing algorithm reverses unfavourable changes to some extent, which explains the dip of the bold line between round 10 and 20, when the “temperature” is low. The volatility of the thin line is due to the fact that the periods of AF are almost multiples of each other and that the imposed changes are totally random. Another type of optimization could render a smother convergence. After 60 rounds there is no point in changing parameters, but the timing behaviour can still be supervised since this is a central error detection mechanism.

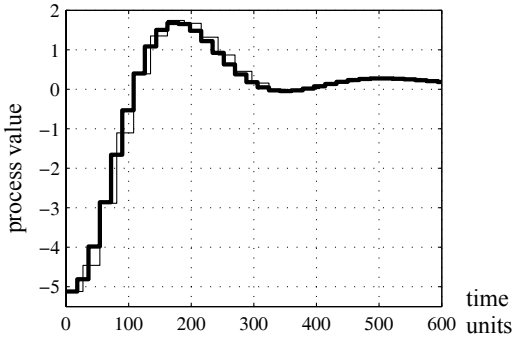


Figure 4.4 Step response of AF_A before (thin) and after (bold) the optimization.

The two step responses in Figure 4.4 reveal that the sampling period of AF_A has been decreased from 27 to 18 time units, but the end-to-end delay stays almost the same. The period of has AF_B also decreased from 35 to its minimum specified value of 15 time units.

5 Related work

In recent years, a research direction in the real-time community has been based on the idea to utilize application specific properties of a dynamic system, in order to device new scheduling or codesign strategies. The aim is often to maximize the utilization of a processing or communication unit by altering task periods, optimizing control performance at the same time. An example of optimization with respect to period only, using monotone convex functions, is found in Seto et al. (1996), but since delays are neglected the strategy rambles. In Cervin (2003) a scheme called feedback scheduling is proposed that allows highly variable and unknown task execution times. It accomodates the utilization via task periods by estimation based on a forgetting factor, combined with feedforward in case of detectable mode changes.

An iterative heuristic off-line design procedure in Ryu et al. (1997) takes both the period and the worst-case end-to-end latency into account, by using the so called period calibration method to find intermediate deadlines. However, period jitter and delay jitter are not considered in the optimization. The so called control server in Cervin (2003) is based on statically scheduled inputs and outputs. The EDF scheduler relies on skips in case of overrun, when optimizing the utilization with respect to control performance. In Sanfridson (2000b) a QoS negotiator measures the time-varying latencies on-line, estimates the control performance and alters task periods and priorities in the distributed computer control system.

In Abdelzaher et al. (1997) a QoS management routine is used together with a service that manages a pool of distributed processing resources. The contention between different applications is handled by rewards depending on the guaranteed level of service. Rejection penalties determine if a new task is to be serviced or not, i.e. a mission critical task should have a rejection penalty such that the probability of rejection becomes low. The centralized broker tries to maximize the total reward. In case of overload the broker can renegotiate QoS levels with the applications. From a control point of view, a good feature is that pre-constructed code modules are invoked depending on the selected service level, e.g. depending on the control period. No explicit mapping between control performance and period or latency is described.

A concept for QoC is developed in Martí et al. (2002) where it is argued that scheduling should be based on flexible timing constraints, i.e. on the period and the latency. No specific scheduling algorithm is proposed, but the scheduling problem is formulated. A central issue is the so called perturbation reaction interval during which the period should be decreased to the shortest allowable value. When a new equilibrium follows, the period returns to the longest allowable value, in order to release resources to other more important tasks.

Another QoS architecture, aiming at the end-to-end guarantees typical for computer control systems, is described in Diot and Seneviratne (1997). Here, each task in a task chain has a reward function transforming input quality to output quality, but no example shows how this works for a control theoretical measure. Other sources of inspiration has been the highly enlightening overview in Stankovic et al. (1996) and the automotive Volcano concept (Rajnak and Ramnefors, 2002) which is not scalable, though.

In many works (Cervin, 2003; Martí et al., 2002; Nilsson, 1998; Rehbinder and Sanfridson, 2004; Sanfridson, 2000b; Sanfridson, 2004; Seto et al., 1996 and Seto et al., 1998), a quadratic performance function is used as a quality measure, but Ryu et al. (1997) is an exception, working with transient over-shoot and steady-state error measures of a step function. Less control related but aiming at control applications, many scheduling papers propose statistical guarantees using the latency or the deadline miss-ratio as a measure of quality, optimizing utilization by finding suitable periods (Abdelzaher et al., 1997; Bernat et al., 2001; Cervin, 2003 and Kang et al., 1997).

6 Summary and future work

The supervisory QoC management is cascaded on top of application functions to negotiate benefit with respect to above all period, latency and jitter. The purpose of the QoC management is to achieve scalability in a computer control system, which typically hosts different kinds of applications. A subordinate aim is to find a workable use of the hardware resources that maximizes the benefit of the system as a whole.

An architecture has been proposed, and a short example demonstrates how the scalability can be realized. The separation of concerns between applications and the QoC middleware, is

accompanied by a separation of time scales. Despite the ability of sustaining a certain amount of timing errors due to the internal memory effects of a dynamic system, a timing error such as a transient or intermittent vacant sampling (or vacant actuation), can lead to failure if the state vector and control signal cannot stay within their allowable domains. The end-to-end response time of a controller is allowed to be excessive, but it has to be predictable, which typically means that the worst-case must be met. However, the control performance depends on the average timing behaviour, not the worst-case. Since the mapping between task parameters and timing behaviour is not straight forward for most scheduling strategies, a periodic activation interval of the QoC management and an on-line model based performance assessment as a function of the actual timing behaviour, is a possible solution. The proposed architecture is not bound to a specific RTOS, middleware or hardware, and any suitable scheduling policy can be considered.

The architecture presented here can be elaborated in many ways. One aspect to look more into is dependability, since it is generally thought of as being adversary to flexibility, and control applications are often safety-critical (Artist road map on adaptive real-time systems, 2003). From a control point of view, a problem to study is that the switch of control laws implemented for different operating conditions (e.g. sampling periods) should be bumpless. The scalability feature also demands predictable timing behaviour, and not only for control applications. A key research topic in the real-time scheduling area is on-line schedulability tests.

7 Acknowledgement

This work has been supported in part by the Flexcon project funded by SSF, the Swedish Strategic research Foundation.

8 References

Abdelzaher T., Atkins E. and Shin K. G., “QoS negotiation in real-time systems and its application to automated flight control”, RTAS’97, pp.228-238, 1997.

Aurrecoechea C., Campbell A. T. and Hauw L., “A survey of QoS architectures“, Report from Columbia University New York NY 10027, 1997.

Bernat G., Burns A. and Llamosí A., “Weakly hard real-time systems“, IEEE Transactions on Computers, Vol. 50, No.4, April 2001.

Cervin A., “Integrated control and real-time scheduling“, PhD Thesis, ISRN LUTFD2/TFRT--1065--SE, Lund, 2003.

Diot C., Seneviratne A., “Quality of service in heterogeneous distributed Systems“, IEEE 1060-3425/97, p. 238-247, 1997.

Jensen D., “Asynchronous decentralized realtime computer systems”, NATO Advanced study Institute on Real-Time Computing, Sint Maarten, 21p, October 1992.

Kang D.-I., Gerber R. and Saksena M., “Performance-based design of distributed real-time systems”, IEEE Proceed. 3rd RT Techn. and Appl. Symp., 1997.

Martí P., Fuertes J. M., Fohler G., and Ramamritham K., “Improving quality-of-control using flexible timing constraints: metric and scheduling issues”, Proceedings RTSS’02, 2002.

Nilsson J., “Real-time control systems with delays”, PhD thesis, ISSN 0280-5316, ISRN LUTFD2/TFRT--1049--SE, January 1998.

Rajnak A. and Ramnefors M., “The Volcano communication concept“, SAE 2002-21-0056, 2002.

Rehbinder H., and Sanfridson. M, “Scheduling of a limited communication channel for optimal control”. Automatica, scheduled for the March, issue (40:3), 2004,.

Ryu M., Hong S. and Saksena M., “Streamlining real-time controller design: from performance specifications to end-to-end timing constraints”, IEEE Proceed. 3rd RT Techn. and Appl. Symp., p.91-99, 1997.

Sanfridson M., “Problem formulation for QoS management in automatic control”, Technical Report TRITA-MMK 2000:3, Mechatronics Lab, KTH, March 2000.

Sanfridson M., “Timing problems in distributed control”, Licentiate Thesis, TRITA-MMK 2000:14, Mechatronics Lab, KTH, May 2000.

Sanfridson M., “Discretization of loss function for a control loop having time-varying period and control delay”, Technical report, ISRN KTH/MMK/R--03/2--SE, Mechatronics Lab, KTH, May 2004.

Seto D., Lehoczky J. P., Sha L. and Shin K. G., “On task schedulability in real-time control systems”, Proceedings RTSS 96, p.13-21, 1996.

Seto D., Krogh B. H., Sha L. and Chutinan A., “Dynamic control system upgrade using the simplex architecture“, IEEE Control Systems, 1998.

Skogestad and Postlethwaite, “Multivariable feedback control - Analysis and design”, ISBN 0-471-94277-4, Wiley, 1996.

Stankovic et al., “Strategic directions in real-time and embedded systems“, ACM computing surveys, Vol. 28, No. 4, December 1996.

Törngren M., Garbergs B. and Berggren H. “A distributed computer testbed for real-time control of machinery”, Proc. of the 5th Euromicro Workshop on Real-Time Systems, Oulu, Finland, June 1993.

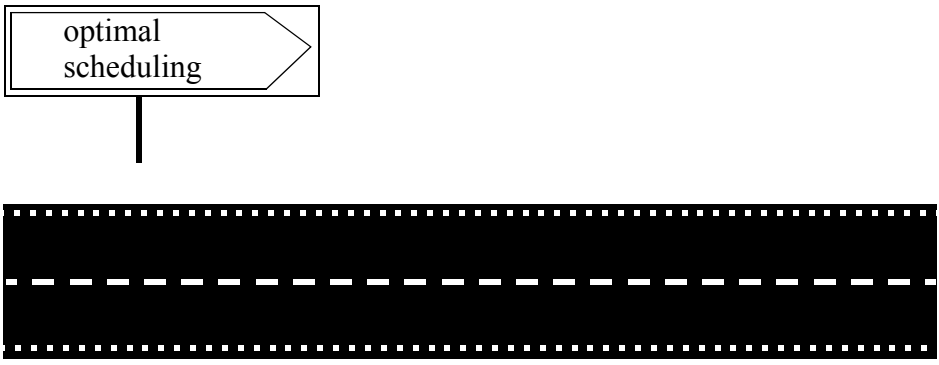
Österman J., “Scalability and QoS for embedded distributed control systems“, MMK 2001:81 MDA181, KTH, Stockholm, 2001.

Artist road map on “Component-based design and development”, www.artist-embedded.org/Roadmaps/A2-roadmap.pdf

Artist road map on “Adaptive real-time systems for quality of service management”, www.artist-embedded.org/Roadmaps/A3-roadmap.pdf

Paper D

Henrik Rehbinder and Martin Sanfridson, “Scheduling of a limited communication channel for optimal control“, Automatica, Vol. 30, No. 3, pp. 491-500, March 2004.



Scheduling of a limited communication channel for optimal control

Henrik Rehbinder

henrik.rehbinder@raysearchlabs.com

Optimization and Systems Theory
Royal Institute of Technology
100 44 Stockholm, Sweden

Martin Sanfridson

mis@md.kth.se

Mechatronics Laboratory
Royal Institute of Technology
100 44 Stockholm, Sweden

Abstract. In this paper a method for optimal off-line scheduling of a limited resource used for control purposes is presented. For various reasons, real-time communication channels are prone to have limited bandwidth. To overcome this obstacle, the rate of actions must be chosen accordingly at design time, both with respect to the limitation of the resource and to control performance. A resulting off-line schedule implements the rate of actions as a repetitive sequence of communication instants. Periodic control theory is used to define a cost functional for LQ-control, that measures the performance of a sampled-data system in relation to a desired continuous time performance. In contrast to uniform sampling, the communication sequence is here allowed to be time-varying. This approach results in a complex combinatorial optimization problem, whose solution gives the optimal off-line schedule, i.e. the sequence in which the actions should take place. The optimization problem is solved by a neighbourhood search method where a heuristic method is used to generate initial guesses close to the optimum. The optimal schedule is typically such that the sampling is non-uniform, but the resulting LQ-control law is time-varying and takes this non-uniform sampling into account.

Key words: Limited communication, periodic control, sampled-data control, LQ-control, combinatorial optimization, timing jitter, scheduling.

1 Introduction

In a traditional view of the development process, the control engineer designs a controller and leaves among other things the scheduling for the software engineer to carry out. This leads to an overall suboptimal design so a more involved co-design of control and scheduling could lead to valuable improvements. Törngren (1998) provides an overview of implementation aspects of controllers in distributed real-time systems. When implementing a controller on some hardware, a number of error sources cause what Wittenmark et al. (1995) call the timing problems: delay, jitter in delay, jitter in sampling period and spurious loss of data, which will decrease the control performance, relative to an ideal continuous time implementation. Because of e.g. physical limitations or economy of scale, a hardware resource often gets a high utilization, i.e. it has a limited capacity. Examples of physical bandwidth limitation are wireless networks (see for example Haartsen (1998)) and acoustic underwater communication. A limited hardware resource can either be a communication channel or a single processing ele-

ment in an embedded dedicated system. On these resources, messages and tasks have to be scheduled in order to guarantee the real-time constraints imposed by the design specifications. We state that it would be possible to increase control performance if approaches could be derived, where the constraints of a limited resource are modelled and compensated for in the control design.

In this paper we consider the following model of a computer control system with limited communication. The scheduling model is periodic and this can be realized in different ways. One well-known and intuitive way is to design a static schedule off-line. A pre-runtime design enables the designer to use existent information more carefully, because there is time to carry out expensive computations. A static schedule is a limited sequence of ticks, which is repeated eternally. The time interval between any two adjacent ticks is constant, and it determines the resolution. In a tick, a specific action is scheduled to take place. Here, a tick corresponds to an instant in time when a communication channel transmits a message. In the following analysis, the timely behaviour of the computer system are assumed to be caused by the static scheduling strategy. Thus, there is no unmodelled delay or delay jitter between the transmission and delivery of a message.

A multi-input sampled-data system with a number of scalar inputs is to be controlled by state feedback. The scheduling model constrains the sampled-data controller in the following way: *Only one of the control signals can be updated at any one tick*. This constraint models the case where all actuators share a common communication channel or when a single processor is used to compute all control actions. We study the following problem: Given a continuous-time LQ-controller, design a sampled-data LQ-controller and a communication sequence such that the performance deviation between the underlying ideal continuous time LQ-controller and the resulting sampled-data controller is minimal. For the sake of simplicity we assume full-state information; the subject of scheduling sensors is discussed in Section 5. The optimal schedule is given by the solution to a combinatorial optimization problem which is solved using sound heuristics. The sampled-data controller corresponding to this optimal schedule is given via standard periodic LQ-theory. With a numerical experiment we demonstrate how the optimal sequence is in agreement with the intuitive idea that a controller with a fast loop characteristic should be assigned a high effective sampling rate, but that simple rules of thumb might be misleading.

A main contribution of this paper lies in the fact the ubiquitous control period is no longer used explicitly in the implementation. It has been extended to a sequence of ticks in a schedule. The period is an important piece of information because it determines directly both the utilization of the resource and the control performance. The control period is now given implicitly by the schedule resulting from an optimization of control performance. When the time interval between two adjacent control actions is not equidistant we simply have jitter. Another contribution is the identification of the structure inherited in the model, leading to a heuristic optimization which greatly reduces the computational cost.

A control theoretic treatment of limited communication control has been undertaken by among others Brockett (1995), Brockett (1997), Wong and Brockett (1997), Wong and Brockett (1999), Tatikonda et al. (1998), Walsh (1999), Hristu and Morgansen (1999), Hristu (1999) and by Rehbinder and Sanfridson (2000a), and in Rehbinder and Sanfridson (2000b). Related research in the real-time systems community regarding task scheduling has been done by Choi et al. (1997), by Seto et al. (1996), by Seto et al. (1998) and by Ryu and Hong (1999). A related study where optimal switching in a hybrid problem is considered has been conducted by Lincoln and Bernhardsson (2000). Our sampled-data controller and optimal schedule can be time varying. The non-uniform sampling is a consequence of the limited communication and the

control law is computed to take this into account. This is different from the work by Hristu and Morgansen where fixed stabilizing controllers are considered and where the question of finding an optimal schedule is not considered. As opposed to Lincoln and Bernhardsson we consider steady state performance and a quite different method for finding the optimal schedule. Further, none of these papers consider sampled-data control.

2 Problem formulation

Consider a given continuous time LQ problem

$$\begin{aligned} \min_u \quad & x'Q_0^c x(t) + \int_0^T x'Q_1^c x(t) + u'Q_2^c u(t)dt \\ \text{s.t.} \quad & \dot{x}(t) = A^c x(t) + B^c u(t) \\ & x(0) = 0 \end{aligned} \tag{1}$$

where $x(t) \in \Re^n$, $u(t) \in \Re^m$ and where we consider the weighting matrices $Q_1^c \geq 0$, $Q_2^c > 0$ and $Q_0^c \geq 0$ to be designed for a desired closed loop performance. To avoid technical difficulties we assume that (A^c, B^c) is a reachable pair and that (A^c, Q_1^c) is an observable pair. Associated with the problem is the optimal value of the objective functional $V(t, T, x) = x'P^c(t)x$ where $P^c(t)$ is the positive semi definite solution to the Riccati equation $\dot{P} = -PA^c - A^{c\prime}P - Q_1^c + PB^{c\prime}(Q_2^c)^{-1}B^cP$, $P(T) = Q_0^c$. Due to reachability and observability this solution is guaranteed to exist for all $t \leq T$. Also,

$$\lim_{t \to -\infty} P^c(t) = P_\infty^c > 0 \tag{2}$$

where P_∞^c is the solution to the Riccati equation $0 = -PA^c - A^{c\prime}P - Q_1^c + PB^{c\prime}(Q_2^c)^{-1}B^cP$. The convergence in (2) is monotone.

2.1 Sampled-data LQ-control

Consider now sampled-data control of (1) with a given sampling interval T_s, that is, let $u(t) = u(kT_s)$ for $kT_s < t < (k+1)T_s$ and assume for simplicity $T = MT_s$. The sampling interval T_s is given by the time between the previously mentioned ticks, that is, the instances when the communication channel is able to transmit messages. The sampled data representation of the control problem (1) is

$$\begin{aligned} \min_u \quad & x'Q_0x(M) + \sum_{k=0}^{M-1} \begin{bmatrix} x \\ u \end{bmatrix}' \begin{bmatrix} Q_1 & Q_{12} \\ Q_{12}' & Q_2 \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \\ \text{s.t.} \quad & x(k+1) = Ax(k) + Bu(k) \\ & x(0) = x_0 \end{aligned} \tag{3}$$

where we use the notation $x(k) = x(kT_s)$, $u(k) = u(kT_s)$. Expressions for the matrices A, B, Q_0, Q_1, Q_2, Q_{12} can be found in the book Åström and Wittenmark (1997) p. 408ff. Note that this discrete time representation is exact and that there is no approximation involved. Especially, the Q-matrices take into account any inter-sample ripple that might occur. Still, the optimal cost for the sampled data system is dependent on the sampling interval. For small sampling

times, the cost is increasing with T_s as demonstrated by Melzer and Kuo (1971), but for larger sampling times this does not hold, as pointed out by Eker (1999).

Our model of the limited communication is, as previously mentioned, that only one control signal can be updated at each tick. We can therefore associate numbers to each tick, defining what control signal will be updated at that sampling instance. Note that a "control signal" does not have to be a scalar, it could just as well be a vector. We will however stick to a scalar exposition. Let the *communication sequence* (previously introduced in Brockett (1995)) $s = s_1 s_2 s_3 s_4 \ldots$ be defined in the natural way

$$s_k = i \qquad \text{if control } i \text{ is updated at } t = kT_s \tag{4}$$

Define also the *update functions*

$$\sigma_i(j) = \begin{cases} 1, & j = i \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

and the matrices

$$\Sigma(k) = diag(\sigma_1(s_k), \ldots, \sigma_m(s_k)) \tag{6}$$

$$\Sigma^*(k) = I - \Sigma(k) \tag{7}$$

Now, those $m - 1$ control signals that can not be updated due to the limited communication, must instead be held constant. To model this we introduce the extra state ξ defined by

$$\begin{aligned} \xi(k+1) &= \Sigma(k)u(k) + \Sigma^*(k)\xi(k) \\ \xi(0) &= 0 \end{aligned} \tag{8}$$

Thus $\xi_i(k)$ is the value of the control command last sent to the i:th actuator. The extended state equations now assume the form

$$\begin{cases} x(k+1) = Ax(k) + B[\Sigma(k)u(k) + \Sigma^*(k)\xi(k)] \\ \xi(k+1) = \Sigma(k)u(k) + \Sigma^*(k)\xi(k) \end{cases} \tag{9}$$

In the same way, the LQ-cost can be written in terms of the update functions σ, σ^* and if we augment the state space according to

$$\hat{x}(k) = \begin{bmatrix} x(k) \\ \xi(k) \end{bmatrix} \text{ and } \hat{u}(k) = u(k) \tag{10}$$

and let

$$\begin{aligned} \hat{A}(k) &= \begin{bmatrix} A & B\Sigma^*(k) \\ 0 & \Sigma^*(k) \end{bmatrix} \\ \hat{B}(k) &= \begin{bmatrix} B\Sigma(k) \\ \Sigma(k) \end{bmatrix} \\ \hat{x}_0 &= \begin{bmatrix} x_0' & 0' \end{bmatrix} \end{aligned} \tag{11}$$

$$\hat{Q}_1(k) = \begin{bmatrix} Q_1 & Q_{12}\Sigma^*(k) \\ (Q_{12}\Sigma^*(k))' & Q_2\Sigma^*(k) \end{bmatrix}$$

$$\hat{Q}_{12}(k) = \begin{bmatrix} Q_{12}\Sigma(k) \\ 0 \end{bmatrix}$$

$$\hat{Q}_2(k) = Q_2$$

$$\hat{Q}_0 = \begin{bmatrix} Q_0 & 0 \\ 0 & 0 \end{bmatrix}$$

then we have the LQ-problem

$$\begin{aligned} \min_{u} \quad & \hat{x}'\hat{Q}_0\hat{x}(M) + \sum_{k=0}^{M-1} \begin{bmatrix} \hat{x} \\ \hat{u} \end{bmatrix}' \begin{bmatrix} \hat{Q}_1(k) & \hat{Q}_{12}(k) \\ \hat{Q}'_{12}(k) & \hat{Q}_2 \end{bmatrix} \begin{bmatrix} \hat{x}(k) \\ \hat{u}(k) \end{bmatrix} \\ \text{s.t.} \quad & \hat{x}(k+1) = \hat{A}(k)\hat{x}(k) + \hat{B}(k)\hat{u}(k) \\ & \hat{x}(0) = \hat{x}_0 \end{aligned} \tag{12}$$

Note that for a given sequence s, this is a time-varying LQ problem and the solution is well known. Therefore, the optimal controller is a function of s and it is derived as follows. The cross-term $\hat{Q}_{12}$ can be eliminated by letting $M(k) = \hat{Q}_{12}(k)\hat{Q}_2^{-1}$ $\vec{u}(k) = \hat{u}(k) + M'(k)\hat{x}(k)$, $\vec{Q}_1(k) = \hat{Q}_1(k) - \hat{Q}_{12}(k)\hat{Q}_2^{-1}\hat{Q}'_{12}(k)$ and $\vec{A}(k) = \hat{A}(k) - \hat{B}(k)M'(k)$. Now, (12) can be written as a standard time-varying LQ problem

$$\begin{aligned} \min_{\vec{u}} \quad & \hat{x}'\hat{Q}_0\hat{x}(M) + \sum_{k=0}^{M-1} \hat{x}'\vec{Q}_1(k)\hat{x}(k) + \hat{u}'\hat{Q}_2\hat{u}(k) \\ \text{s.t.} \quad & \hat{x}(k+1) = \vec{A}(k)\hat{x}(k) + \vec{B}(k)\vec{u}(k) \\ & \hat{x}(0) = \hat{x}_0 \end{aligned} \tag{13}$$

and the solution to (13) is the discrete time LQ-feedback

$$\vec{u}(k) = \vec{K}(k)\hat{x}(k) = -(\hat{Q}_2 + \hat{B}'(k)\hat{P}(k+1)\hat{B}(k))^{-1}\hat{B}'(k)\hat{P}(k+1)\vec{A}(k)\hat{x}(k) \tag{14}$$

where $\hat{P}(k)$ is the solution to the discrete time Riccati equation

$$\begin{aligned} \hat{P}(k) &= \vec{A}'(k)\hat{P}(k+1)\vec{A}(k) + \vec{Q}_1(k) \\ &\quad -\vec{A}'(k)\hat{P}(k+1)\hat{B}(k)(\hat{Q}_2 + \hat{B}'(k)\hat{P}(k+1)\hat{B}(k))^{-1}\hat{B}'(k)\hat{P}(k+1)\vec{A}(k) \\ \hat{P}(n) &= \hat{Q}_0 \end{aligned} \tag{15}$$

The optimal cost associated with (12) is $V(k, M, \hat{x}) = \hat{x}'\hat{P}(k)\hat{x}$ and the feedback law is $\hat{u}(k) = (\vec{K}(k) - M'(k))\hat{x}(k)$.

2.2 Periodic systems

We will eventually let the time interval considered be infinite and from an implementation perspective it is then more or less necessary to consider only periodic sequences s. There are also major theoretical advantages with such a restriction. The LQ-problem (13) will be periodic and it can be shown that the steady state solution is given by a periodic controller. Further, if controllability and observability are fulfilled, the solution to the periodic Riccati equation (15) will converge to a periodic solution. It can also be shown that this periodic solution can be found via lifting and by solving an algebraic Riccati equation for the lifted system. We will here review these results and refer to Bittanti et al. (1990) and Bittanti et al. (1991) for details.

It simplifies matters to let $\hat{C}'(k)\hat{C}(k) = \vec{Q}_1(k)$ and $L'L = \hat{Q}_2$ be full rank factorizations. Now as $\hat{Q}_2 = Q_2 > 0$, L is invertible and we may let $\vec{B}(k) = B(k)L^{-1}$ and $\vec{u} = Lu$. Thus (13) may be rewritten as

$$\begin{aligned} \min_{\vec{u}} \quad & \hat{x}'\hat{Q}_0\hat{x}(n) + \sum_{k=0}^{n-1} \|\hat{y}(k)\|^2 + \|\vec{u}(k)\|^2 \\ \text{s.t.} \quad & \hat{x}(k+1) = \vec{A}(k)\hat{x}(k) + \vec{B}(k)\vec{u}(k) \\ & \hat{y}(k) = \hat{C}(k)\hat{x}(k) \\ & \hat{x}(0) = \hat{x}_0 \end{aligned} \tag{16}$$

Let now the set of p-periodic sequences be

$$S_p = \{s = s_1 s_2 \cdots \,|\, s_i = s_{i+p},\ \forall i \geq 1\} \tag{17}$$

The idea is to use *lifting* (see Chen and Francis (1995) for a general tutorial on lifting) and down-sample the system states with periodicity p — thus getting a time-invariant reformulation of the problem. By letting

$$\bar{x}_k(t) = \hat{x}(k+pt) \tag{18}$$

$$\bar{u}_k(t) = \left[\vec{u}(k+pk)' \ \ldots \ \vec{u}(k+p(k+1)-1)'\right]' \tag{19}$$

$$\bar{y}_k(t) = \left[\hat{y}(k+pk)' \ \ldots \ \hat{y}(k+p(k+1)-1)'\right]' \tag{20}$$

the problem (16) can be rewritten in an equivalent *time-invariant* formulation as

$$\begin{aligned} \min_{\bar{u}} \quad & \bar{x}'\hat{Q}_0\bar{x}(N) + \sum_{k=1}^{N-1} \|\bar{y}_k(t)\|^2 + \|\bar{u}_k(t)\|^2 \\ \text{s.t.} \quad & \bar{x}_k(t+1) = \bar{A}_k(t)\bar{x}_k(t) + \bar{B}_k(t)\bar{u}_k(t) \\ & \bar{y}_k(t) = \bar{C}_k\bar{x}_k(t) + \bar{D}\bar{u}_k(t) \\ & \bar{x}_k(0) = (\bar{x}_k)_0 \end{aligned} \tag{21}$$

where we have assumed that $M = Np + k$. The corresponding time-invariant Riccati equation is

$$\begin{aligned} \bar{P}_k(t) = {} & \tilde{A}_k'\bar{P}_k(t+1)\tilde{A}_k + \tilde{C}_k'\tilde{C}_k \\ & -\tilde{A}_k'\bar{P}_k(t+1)\bar{B}_k(I + \bar{D}_k'\bar{D}_k + \bar{B}_k'\bar{P}_k(t+1)\bar{B}_k)^{-1}\bar{B}_k'\bar{P}_k(t+1)\tilde{A}_k \\ \bar{P}(N) = {} & \hat{Q}_0 \end{aligned} \tag{22}$$

where the non-proper part of the system ($\bar{D}\bar{u}_k(t)$) has been eliminated by introducing

$$\tilde{A}_k = \bar{A}_k - \bar{B}_k(I + \bar{D}_k'\bar{D}_k)^{-1}\bar{D}_k'\bar{C}_k \tag{23}$$

$$\tilde{C}_k'\tilde{C}_k = \bar{C}_k'(I + \bar{D}_k\bar{D}_k')^{-1}\bar{C}_k \tag{24}$$

The expressions for the matrices $\bar{A}_k$, $\bar{B}_k$, $\bar{C}_k$ and $\bar{D}_k$ can be found in Bittanti et al. (1991). The following intuitive relation is true. The solution to the Discrete-time Periodic Riccati Equation corresponding to (16) converges to a p-periodic solution as $t \to \infty$ and this solution can be achieved by solving (22). This is formalized in the following lemma and proposition by Bittanti et al. (1990) and Bittanti et al. (1991).

Lemma 2.1 (Bittanti) If $\bar{P}_k(N) = \hat{P}_k(k + Np) = \hat{Q}_0$ then

$$\bar{P}_k(t) = \hat{P}_k(k + tp), \quad \forall t \le n \tag{25}$$

Proposition 2.1 (Bittanti) If $\hat{P}$ is a Symmetric, Periodic and Positive Semi definite (SPPS) solution to (15) then $\bar{P}_k = \hat{P}(k)$ is a positive semi definite solution of the algebraic version of (22). Conversely, if $\hat{P}$ is a positive semi definite solution of the algebraic (22), then the solution of (15) with $\hat{P}(k) = \bar{P}_k$ is SPPS.

2.3 Optimality function

Consider from now on the infinite time horizon case, that is let $T, N \to \infty$ and let

$$S_p^{co} = \{ s \in S_p : (16) \text{ controllable and observable} \} \tag{26}$$

We will comment on how to obtain S_p^{co} from S_p in Remark 3.1. In continuous time, the optimal cost for a given initial value x_0 is $V_\infty^c(x) = x_0'P_\infty^c x_0$ from (2). For a given $s \in S_p^{co}$ denote by $\hat{P}_k(s, t)$, the periodic solution to (15). As $s \in S_p^{co}$, $P_k(s, t)$ is guaranteed to exist. The corresponding optimal cost is also p-periodic and is $V_\infty(\hat{x}(0), t) = \hat{x}(0)'\hat{P}_k(s, t)\hat{x}(0)$. Note that from (8) $\hat{x}(0)' = \begin{bmatrix} x_0 & 0 \end{bmatrix}$ so we have $V_\infty(x, t) = x'P_k(s, t)x$ where $P_k(s, t)$ is the top left block of $\hat{P}_k(s, t)$ corresponding to x_0. For given x, t, s, the sampled-data cost normalized with the continuous cost is $x'P(s, t)x / x'P^c x$. We consider a worst-case performance measure and define the objective function as follows:

Definition 2.1 The worst-case objective function $f_p : S_p^{co} \to \Re > 0$ is

$$f_p(s) = \max_{t \in \{1, \ldots, p\}} \max_{x \neq 0} \frac{x'P_k(s, t)x}{x'P^c x} \tag{27}$$

$P^c > 0$ so the quotient is well defined. The maximizing x is the worst possible initial value if the system is started at the worst possible position t in the schedule.

Remark 2.1 It might be argued that the quotient $(x'P_k(s, t)x - x'P^c x) / x'P^c x$ is more natural to use. As this quotient can be rewritten as $x'P_k(s, t)x / x'P^c x - 1$, it does not matter which one is used.

The maximization over x has an analytic solution.

Observation 2.1

$$f_p(s) = \max_{t \in \{1, \ldots, p\}} \text{maxeig} \left\{ (P^c)^{-1} P_k(s, t) \right\} \tag{28}$$

Proof. Let U and V be symmetric matrices where $U \geq 0$, $V > 0$ and consider $\phi(x) = x'Ux / x'Vx$. Let W be a full rank factorization of V and let $Wx = z$. We have $\phi(x) = \phi(W^{-1}z) = z'W^{-T}UW^{-1}z / z'z$. Now, $x \neq 0 \Leftrightarrow z \neq 0$ so

$$\max_{x \neq 0} \phi(x) = \max_{z \neq 0} \frac{z'W^{-T}UW^{-1}z}{z'z} = \text{maxeig} \{ W^{-T}UW^{-1} \}$$

The eigenvalues of $W^{-T}UW^{-1}$ and $V^{-1}U$ are the same as $W^{-T}UW^{-1} = WV^{-1}UW^{-1}$.

□

It might now be tempting to consider optimizing over all possible periods. If the best possible schedule is non-periodic then such a formulation will be an ill-posed problem. The larger periodicity, the closer to the optimum we get but the optimum will never be achieved. This undesirable situation is avoided by introducing a maximal allowed periodicity $T_p < \infty$.

Finally we state the following combinatorial optimization problem

Problem 2.1 The optimal control and scheduling integration problem can be formulated as

$$\begin{aligned} &\min_{s,\, p} \; f_p(s) \\ &s \in S_p^{co} \\ &p \in \{1, \ldots, T_p\} \end{aligned} \tag{29}$$

3 Solving the optimization problem

The optimization problem (29) is extremely expensive to solve for large schedule periodicity T_p and input dimensions m. The reason for this is twofold, the size of the search space $s \in S_p^{co}$, $p \in \{1, \ldots, T_p\}$ grows non-polynomially and also the cost of evaluating $f_p(s)$ grows. To experiment with small scale problems on a standard workstation, exhaustive search can be used but for larger problems some more efficient algorithm have to be used. Based on experiences from experimenting with exhaustive search and intuitive ideas we propose a heuristic method for solving (29). Due to the problem complexity, it is very hard to estimate how close the solution found by this heuristic algorithm is to the true optimal solution. As has been pointed out, the optimization problem can be solved by exhaustive search. For the example in Section 4 the solutions have been compared to the optimal ones and for this case, the algorithm did find the optimal solution. Further, the algorithm is based on some quite intuitive reasoning. The heuristics are based on certain structural properties of the problem that we will describe first.

3.1 Structural properties

The search space has an algebraic structure and can be split into disjoint parts. This partitioning of the search space will suggest a heuristic and intuitive method for solving the optimization problem. The structure is in the sets (17). If we, for simplicity of notation, denote a p-periodic sequence s by its first p elements $s = (s_1 s_2 \ldots s_p)$ then we can associate m non-negative inte-

gers $k_1, \ldots, k_m$ with s such that k_i is the number of instances of control action i in s, $k_i = n^\circ(s_j = i, j = 1, \ldots, p)$. Naturally, $0 \le k_i \le p$ and $\sum k_i = p$. The search space can now be parameterized by these k_i and this parameterization induces a natural partitioning. Let $k = \begin{bmatrix} k_1 & \ldots & k_N \end{bmatrix}'$ and let $Z_p(k)$ be the set of schedules corresponding to k

$$Z_p(k) = \{s \in S_p : n^\circ(s_j = i, j = 1, \ldots, p) = k_i, i = 1, \ldots, m\} \tag{30}$$

Now, in vector notation, we can partition S_p according to

$$\begin{aligned} S_p &= \bigcup_{k \in K_p} Z_p(k) \\ K_n &= \{k : 1^T k = p, 0 \le k \le p, k_i \in \aleph\} \end{aligned} \tag{31}$$

As clearly $Z_p(k) \cap Z_p(l) \neq \varnothing$ (the empty set) this really is a partition of S_p. The interpretation of this partitioning is that k represents the amount of resources (number of sampling instances) devoted to the different inputs. The set $Z_p(k)$ is all the possible ways to distribute these given resources. The subsets $Z_p(k)$ have an algebraic structure and can be partitioned into equivalence classes that will prove very useful for the optimization. The same structural properties hold for all S_p. We will review some basic results from group theory and refer to for example the book by Fraleigh (1999) for details.

Consider the group of permutations G of a finite set X of p elements $G = \{g : X \to X, \text{ g is 1-1 and onto}\}$ and the subgroup of cyclic permutations $C = \{e, r, r^2, \ldots, r^{p-1}\}$ where e is the identity element and where

$$r : (x_1 x_2 \ldots x_p) \to (x_p x_1 \ldots x_{p-1}) \tag{32}$$

is the one step shift. Recall that an *action* $*$ G on X is a map $* : G \times X \to X$ such that $e*x = x$ for all $x \in X$, where e is the identity element and such that $(g_1 g_2)*(x) = g_1*(g_2*x)$ for all $x \in X$ and all $g_1, g_2 \in G$.

Group actions on sets define an equivalence relation according to the following

Lemma 3.1 Let X be a G-set. For $x_1, x_2 \in X$, let $x_1 \sim x_2$ if and only if there exists $g \in G$ such that $gx_1 = x_2$. Then $\sim$ is an equivalence relation on X.

In our case, the cyclic group C defines an equivalence relation on S_p. The map $* : C \times S_p \to S_p$ is defined by

$$\begin{aligned} es &= s \\ rs &= (s_p s_1 s_2 \ldots s_{p-1}) \end{aligned} \tag{33}$$

Condition 1 from Theorem 3.1 is trivial and Condition 2 is easily checked.

Actually, the quotient group $S_p/\sim$ (the set of equivalence classes) is well known and is described as the set of "necklaces" composed by p beads of at most m different colours Biggs (1989). A necklace is still the same even if it is rotated around the neck.

The equivalence classes are of importance to our problem as it can be shown that the following quite intuitive statement holds.

Observation 3.1 The optimality function is constant on the equivalence classes,

$$(f_p(s) = f_p(\sigma)), \forall s, \sigma \in S/\sim \tag{34}$$

Proof: Consider the cost formula from Observation 3.1 and fix t. Then $c = x_0' P_c^{-1} P(t, s) x_0$ is the cost for starting the system at position t in the sequence $(s_1 s_2 \ldots s_p)$. Just by changing time notation, c is also the cost for starting the system at position 1 with the schedule $(s_t s_{t+1} \ldots s_p s_2 \ldots s_{t-1})$. From (32), $r^{t-1} s = (s_t s_{t+1} \ldots)$. As x_0 was arbitrary we have that for any k

$$\{P_c^{-1} P(t, s)\}_{t=0}^{t=p-1} \qquad \{P_c^{-1} P(t, r^k s)\}_{t=k}^{t=p-1+k}$$

and therefore the observation is true.

□

The optimization over $Z_p(k)$ can now be replaced by optimization over $Z_p(k)/\sim$. We restate the problem (29) as

$$\begin{aligned} &\min\ J_p(s) \\ &s \in \bigcup_{k \in K_p} Z_p^{co}(k)/\sim \\ &\text{s.t.} \qquad K_p = \{k \in \aleph^m : 1^T k = p,\ 0 \le k \le p\} \\ &\qquad\quad p \in \{1, \ldots, T_p\} \end{aligned} \tag{35}$$

3.2 Optimization heuristics

Based on the above partitioning of the search space (31) we split the problem into three different sub problems where each will be solved in a natural, yet heuristic way. Let

$$h(p) = \min_{k \in K_p} g_p(k) \text{ and } g_p(k) = \min_{s \in Z_p(k)/\sim} f_p(s)$$

Then, the entire problem (29) can be written as

$$\min_{p \in \{1, \ldots, T_p\}} h(p) \tag{36}$$

The advantage of writing the problem as above is that two of the sub problems can be solved with local search methods where very reasonable initial values can be computed and one can be solved by exhaustive search.

(1) First we note that the set $\{1, \ldots, T_p\}$ only contains T_p elements so (36) can be solved by exhaustive search.

(2) Second, the solution $\hat{k}$ to
$\min_{q \in \{1, \ldots, p\}} h(q)$
should provide a close initial guess k_0 to the problem of computing $h(p+1)$.

(3) It is most reasonable that the optimal point $\hat{s}$ for $g_p(k)$ should be the most "evenly distributed"' schedule $\hat{s} \in Z_p(k)/\sim$. For example, the sequence 1010 should be better than 1100. Therefore, the local search should be started with an "evenly distributed" schedule. The control theoretic motivation for this is that using a clustered schedule such as 1100

corresponds to leaving the control signals unattended for longer times than using 1010. That the optimal sequences really are such has been noted when experimenting with exhaustive search.

In order to compute $h(p)$ and $g_p(k)$ we propose a neighbourhood search method. The neighbourhood search is done in the easiest possible way

```
1. start with initial guess x
2. compute the best neighbour x_n
3. if cost(x_n<cost(x), goto 2 else, end
```

Neighbours are defined in the following way

Definition 3.1 For given m and p, let $k \in K_p$. A neighbour $\bar{k}$ to k is an element in K_p such that $\|\bar{k} - k\|^2 = 2$.

Definition 3.2 For given m, p and k, let $s = (s_1 s_2 \ldots s_p) \in Z_p(k)/\sim$ A neighbour $\bar{s}$ to s is an element in $Z_p(k)/\sim$ such that for some i, $\hat{s} = (s_1 s_2 \ldots s_{i-1} s_{i+1} s_i s_{i+2} \ldots s_p)$, that is, the i^{th} and $(i+1)^{\text{th}}$ elements are interchanged.

Based on the heuristic ideas presented here, an algorithm for computing the optimal schedule and the corresponding optimal controller had been implemented in matlab.

Remark 3.1 The set S_p^{co} does not have to be generated to solve the optimization problem. Instead, when the neighbours of a sequence are searched, they are simply tested for observability and controllability. If a neighbour happens not to be in S_p^{co}, it is simply omitted from the search.

4 Examples

To demonstrate the algorithm presented we will consider the simultaneous control of two uncoupled inverted pendulums. We consider the system controllers to be designed as LQ-controllers and we let all control nodes share a common communication channel. The reason for choosing such a simple example is that the complexity of a more realistic example would obscure the sight. The example presented consists of two parallel SISO-systems but the algorithm could just as well handle a general linear MIMO-system (see Rehbinder (2001)). We will study the optimal sequences as a function of the desired closed loop performances and see how the optimal resource allocation is related to these closed loop performances.

Consider the problem of stabilizing two mechanically independent inverted pendulums controlled by torques, T_i, at the base joints (Figure 4.1). The two pendulums have exactly the same dynamics, i.e. the masses and lengths are the same ($l = 1$, $m = 1$. The deviation from the upright position is denoted by the angle ϕ. The linearized dynamics for an inverted pendulum of length l is

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ g/l & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ g/l \end{bmatrix} u(t) \tag{37}$$

where $x' = \begin{bmatrix} \phi & \dot{\phi} \end{bmatrix}'$ is the state and $u = T/(mgl)$ is the scaled control signal. The model (37) is of the same form as (1) and the methods described can readily be applied.

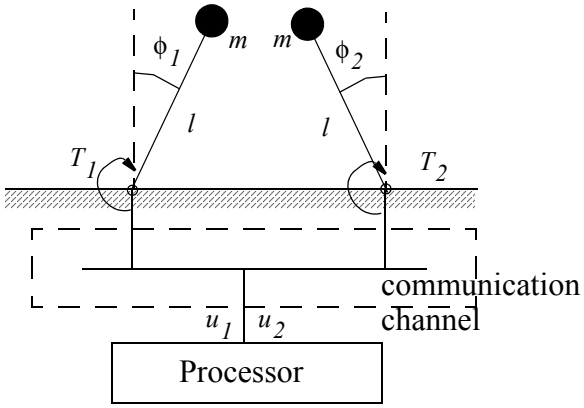


Figure 4.1 Two inverted pendulums sharing a communication channel.

4.1 Control design

We will as before consider initial value problems where the control objective is to achieve $\phi = 0$, upright position. The two pendulums are, except for the communication channel, independent so the control specifications are given for each of the pendulums. We will let the specifications for pendulum 1 be constant and vary the specification of pendulum 2, according to Table 4.1. The specification are given as approximate closed loop system time constants for some initial value ϕ_0, $\dot{\phi} = 0$ and as maximal overshoots $M = 5\%$. The desired time constant for pendulum 1 is $T_c^{(1)} = 0.5\text{s}$ and for pendulum 2 it will vary between 0.5s and 0.005s. To obtain the desired closed loop performances the control weight matrices are taken as $(Q_2^c)_1 = (Q_2^c)_2 = 1$ and the state weight matrices Q_1^c as

$$(Q_1^c)_1 = \begin{bmatrix} 1.72 & 0 \\ 0 & 0 \end{bmatrix} \qquad (Q_1^c)_2 = \begin{bmatrix} \beta^2 & 0 \\ 0 & 0 \end{bmatrix} \tag{38}$$

where we vary β^2 to penalize the angle ϕ and thus obtain the desired time constant $T_c^{(2)}$. The β^2-values generating controllers that fulfil the specifications have been found by numerical experiments.

Table 4.1 Optimal sequences for the two inverted pendulums. $T_s = 0.01$. The time constant for the first pendulum is $T_c^{(1)} = 0.5\text{s}$

$T_c^{(2)}$	β^2	$\hat{s}$	$f_p(\hat{s})$
0.5	1.72	12...	1.0753
0.15	250	122...	1.2027
0.05	17000	12222222...	1.6711
0.015	2000000	122222222222222...	5.2620
0.005	170000000	1222222222222222...	69.9013

4.2 Optimal sequences

In Table 4.1 the results for the above mentioned setting is given. The maximal periodicity has been chosen as $T_p = 16$ and the sampling time is $T_s = 0.01\text{s}$. The optimal sequences corresponding to the specified rise times for the first pendulum and the corresponding β^2-values shows a behaviour that could be expected from intuition. This behaviour is that more communication resources should be allocated to the controller which has the highest bandwidth requirement. Note that the results presented in Table 4.1 only concerns what schedule is the best, not how good it actually is. Take for example $T_c^{(2)} = 0.005$. As $T_s = 0.01\text{s}$ we have a time constant that is half the sampling interval. The performance with this relation between sampling and desired system speed should be very bad. The cost associated is as is seen in the table $f_p(\hat{s}) = 69.9013$ so the LQ-cost is approximately 70 times higher than the continuous. The natural question is then what happens if we take for example $T_s = 0.002\text{s}$. The results if the same set of systems are considered are given in Table 4.2 and it is clear that for this more reasonable sampling time performance is increased.

Table 4.2 Optimal sequences for the two inverted pendulums. $T_s = 0.002$. The time constant for the first pendulum is $T_c^{(1)} = 0.5\text{s}$

$T_c^{(2)}$	β^2	$\hat{s}$	$f_p(\hat{s})$
0.5	1.72	12...	1.0145
0.15	250	122...	1.0371
0.05	17000	12222222...	1.1074
0.015	2000000	122222222222222...	1.3994
0.005	170000000	1222222222222222...	2.7800

4.3 Further study

In order to study the coupling between closed loop performance, sampling time and optimal resource allocation more closely we have computed optimal sequences for a more extensive set of β-values and for different sampling times. The resource allocation corresponding to these sequences can be represented by

$$\kappa_2 = \frac{\text{number of times ctrl 2 is updated per period}}{p} \tag{39}$$

where κ_2 serves as a measure of the resource allocation corresponding to the optimal schedule. The results are displayed in Figure 4.2, where we have plotted κ_2 against $1 / T_c^{(2)}$. From the graph the following can be seen. When the desired closed loop performances for the two systems are similar to each other, then the resources are equally distributed. When the time constant for the second system is decreased over some threshold, κ_2 starts increasing rapidly and eventually levels out. The reason for why κ_2 levels out is simply that we must have $0 < \kappa_2 < 1$ for both systems to remain observable and controllable. We also note that there is a dip in the κ_2-curve for low rise times. One would expect a monotone κ_2-curve. This dip appears for smaller time constant the smaller the sampling interval T_s. For $T_s = 0.0005$ it is hardly not present for the β^2-values considered. This suggests that the phenomena has something to do

with a non-monotonic behaviour of the LQ-cost as a function of the sampling interval. As a matter of fact it was demonstrated in the thesis by Eker (1999) that the LQ-cost is monotone as a function of the sampling interval only for small sampling intervals.

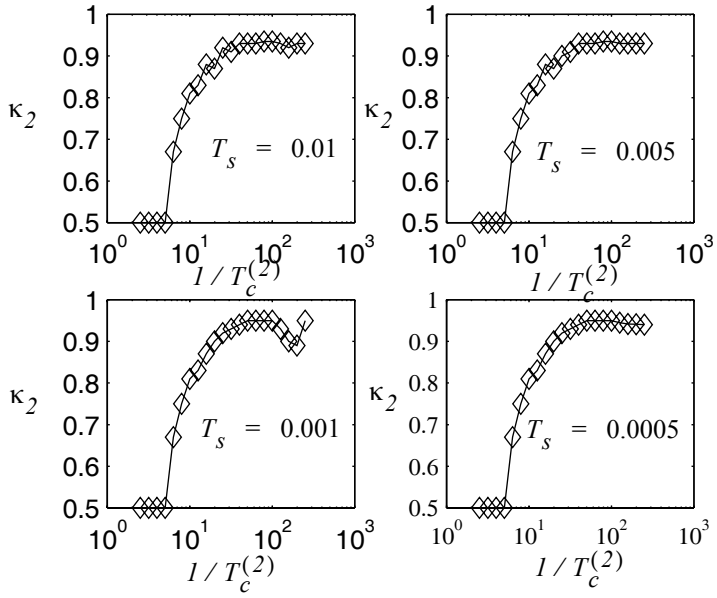


Figure 4.2 Plots of κ_2 versus $1/T_c^{(2)}$ for different sampling times T_s.

5 Discussion and summary

In this paper we have presented a method for scheduling a limited communication channel (or a limited computational resource). The result is an algorithm that, given a continuous time LQ-controller, generates the best implementation of this controller. The main feature of this implementation is that the optimal sampling pattern is irregular and that the controller is designed to compensate for this irregularity. This makes a very tight scheduling of the communication channel possible. The main problem with the periodic systems approach taken here is computations. Computing the cost for just one sequence can be very time consuming if the schedule periodicity is large. More efficient coding could probably reduce this time but the problem is inherent to the long periodicity. Further, the size of the search space is growing rapidly as a function of m and T_p.

The obvious extension of this study is to consider output feedback and include a Kalman filter as a state estimator. The simplest approach should be to consider two communication channels where one is used for control data and one for sensor data. It might then be possible to, due to the separation principle, obtain optimal performance by scheduling the sensor data channel and the control data channel separately. As Kalman filters are most naturally formulated in terms of stochastic systems it seems reasonable to consider an alternative definition of optimality where the stochastic nature of the problem is taken into account. A study of optimal control scheduling in a stochastic setting has been presented by Rehbinder and Sanfridson (2000a) but there were no sensors considered in that study. A more interesting problem formulation than the two-channel formulation is to consider only one channel which is used both for sensor and control data. Then the sensor and control signals need to be scheduled concurrently. Once a notion of optimality has been set the problem should be quite straightforward and of the same type of

combinatorial optimization as presented here. The computational complexity will unfortunately be even worse than in this paper.

The analysis underlying the algorithm presented here requires quite advanced systems theory and the computations are quite heavy. However, we would like to point out the following as a concluding remark. Once the continuous-time controller is designed, the optimal sequence and the optimal sampled-data controller can be found by simply running a computer program. Therefore, the method can be used without knowledge of the theory needed to derive it. It should also be possible to automatically generate real-time code for the optimal controller and therefore reduce the time between controller design and implementation.

6 Acknowledgement

This work was in part sponsored by the SSF through its Centre for Autonomous Systems at KTH and in part by NUTEK in the Complex Technical Systems Program, the DICOSMOS project.

7 References

K. J. Åström and B. Wittenmark, “Computer Controlled Systems: Theory and Design”, Prentice-Hall, 3rd edition, 1997.

N. Biggs, “Discrete Mathematics”, Oxford science publications, 1989.

S. Bittanti, P. Colaneri, and G. De Nicolao, “An algebraic Riccati equation for the discrete-time periodic prediction problem”, Systems & Control Letters, 14(1):71--78, Jan 1990.

S. Bittanti, A. J. Laub, and J. C. Willems (eds.),”The Riccati equation”, chapter “The periodic Riccati equation”, Springer Verlag Berlin, Heidelberg, 1991.

R. Brockett, “Stabilization of motor networks”, Proceedings of the 34th IEEE Conference on Decision & Control, pages 1484--1488, New Orleans, LA, December 1995.

R.W. Brockett, “Minimum attention control”, Proceedings of the 36th IEEE Conference on Decision & Control, San Diego, California, USA, pages 2628 -- 2632, December 1997.

T. Chen and B. Francis, “Optimal sampled-data control systems”, Springer-Verlag, London, 1995.

S. Choi, A. K. Agrawala, and L. Shi, “Intelligent temporal control”, Proceedings of Intelligent Information Systems, ISS ‘97, 1997.

J. Eker, “Flexible embedded control systems: design and implementation”, PhD thesis, Lund Institute of Technology, 1999.

J. B. Fraleigh, “A first course in abstract algebra”, Addison-Wesley, 1999.

J. Haartsen, “Bluetooth - the universal radio interface for ad hoc, wireless connectivity”, Ericsson Review, 3:110--117, 1998.

D. Hristu, “Optimal control with limited communication”, PhD thesis, The Division of Engineering and Applied Sciences, Harvard University, Cambridge, USA, 1999.

D. Hristu and K. Morgansen, “Limited communication control”, Systems and Control Letters, 37:193--205, 1999.

B. Lincoln and B. Bernhardsson, “Efficient pruning of search trees in LQR control of switched linear systems”, Proceedings of the 39th IEEE Conference on Decision & Control, Sydney, Australia, December 2000.

S. M. Melzer and B. Kuo, “Sampling period sensitivity of the optimal sampled data linear regulator”, Automatica, 7:367--370, 1971.

H. Rehbinder, “State estimation and limited communication control for nonlinear robotic systems”, PhD thesis, Royal Institute of Technology, Stockholm, Sweden, November 2001.

H. Rehbinder and M. Sanfridson, “Integration of off-line scheduling and optimal control”, Proceedings of the 12th European Conference on Real-Time Systems, Euromicro, pages 137--143, Stockholm, Sweden, 2000.

H. Rehbinder and M. Sanfridson, “Scheduling of a limited communication channel for optimal control”, Proceedings of the 39th IEEE Conference on Decision and Control, volume 1, pages 1011 -- 1016, Sydney, Australia, December 2000.

M. Ryu and S. Hong, “A period assignment algorithm for real-time system design”, Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’99), 1999.

D. Seto, J. Lehoczky, L. Sha, and K. Shin, “On task schedulability in real-time control systems”, Proceedings of the IEEE Real-Time Systems Symposium, 1996.

D. Seto, J. P. Lehoczky, and L. Sha, “Task period selection and schedulability in real-time systems”, Proceedings of the 19th Real-Time Systems Symposium, pages 188--198, 1998.

S. Tatikonda, A. Sahaim, and S. Mitter, “Control of LQG systems under communication constraints”, Proceedings of the 37th IEEE Conference on Decision and Control, Tampa, Florida, USA, pages 1165--1170, December 1998.

M. Törngren, “Fundamentals of implementing real-time control applications in distributed computer systems”, Real-Time Systems Journal, 14:219--250, 1998.

G. Walsh, “Stability analysis of networked control systems”, American Control Conference, ACC99, 1999.

B. Wittenmark, J. Nilsson, and M. Törngren, “Timing problems in real-time control systems”, American Control Conference, ACC95, 1995.

W. S. Wong and R. W. Brockett, “Systems with finite communication bandwidth constraints - part I: State estimation problems”, IEEE Transactions on Automatic Control, 42(9):1294 -- 1299, September 1997.

W. S. Wong and R. W. Brockett, “Systems with finite communication bandwidth constraints - part II: Stabilization with limited information feedback”, IEEE Transactions on Automatic Control, 44(5):1049 -- 1053, May 1999.