# Towards the Integration of Control and Real-Time Scheduling Design

Cervin, Anton

2000

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

# Towards the Integration of Control and Real-Time Scheduling Design

Anton Cervin

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

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

*Document name*
LICENTIATE THESIS

*Date of issue*
May 2000

*Document Number*
ISRN LUTFD2/TFRT--3226--SE

*Author(s)*
Anton Cervin

*Supervisor*
Karl-Erik Årzén
Bo Bernhardsson

*Sponsoring organisation*
ARTES/SSF

*Title and subtitle*
Towards the Integration of Control and Real-Time Scheduling Design

*Abstract*

The thesis deals with scheduler and controller co-design for real-time control systems. The overall goal is higher resource utilization and better control performance. One goal is to minimize the performance loss due to jitter and control delays. Another goal is to relax the nominal requirements on hard deadlines and fixed worst-case execution times. A third goal is to provide a co-simulation environment for real-time control systems.

Sub-task scheduling of the two main parts of a control algorithm is investigated. A heuristic, iterative deadline-assignment algorithm is given that attempts to minimize the computational delay for a set of control tasks.

A simulator for co-design of real-time control systems is presented. It facilitates simultaneous simulation of real-time task execution and continuous plant dynamics. The simulator makes it possible to evaluate the true, timely behavior of control algorithms, and to evaluate scheduling policies from a control performance perspective.

A feedback scheduler for control tasks with varying execution times is developed. Using a combination of feedback and feedforward, the feedback scheduler attempts to keep the CPU utilization at the desired level by manipulating the sampling periods of the controllers. A case-study with a set of hybrid control tasks for a set of double-tank processes is presented.

*Key words*
Real-Time Control, Scheduling, Co-Design, Deadlines, Performance, Simulation, Feedback

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*
0280–5316

*ISBN*

*Language*
English

*Number of pages*
152

*Recipient's notes*

*Security classification*

# Towards the Integration of Control and Real-Time Scheduling Design

Anton Cervin

Department of Automatic Control
Lund Institute of Technology
Lund, May 2000

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

E-mail: anton@control.lth.se
WWW: http://www.control.lth.se/~anton

ISSN 0280–5316
ISRN LUTFD2/TFRT--3226--SE


Printed in Sweden by Universitetstryckeriet,
Lund University, Lund 2000

# Contents

## Acknowledgments

Starting my graduate studies two years ago, I had the great advantage of joining a fresh, exciting, and well-defined research project, where several interesting problems were just waiting to be mined.

First, of course, I would like to thank my supervisor Karl-Erik Årzén. He, together with Klas Nilsson and Ola Dahl, wrote the original proposal for the project "Integrated Control and Scheduling". I admire his efficiency and his sound attitude towards research and work in general.

Johan Eker's input to this project cannot be overstated. Believe it or not, but I do enjoy working with him, and I look forward to further collaborative efforts.

The ideas on feedback scheduling were originally the brainchild of Lui Sha. I got my basic training in real-time systems from him when I spent two months at the Software Engineering Institute, Carnegie Mellon University, in the summer of 1998. I would also like to extend my gratitude to Danbing Seto.

Everybody at the department has been most helpful *and* friendly! My co-supervisor Bo Bernhardsson has provided several valuable comments. Karl Johan Åström lured me into the field of Automatic Control and encouraged me to become a graduate student.

This project is a collaboration with the Department of Computer Science, Lund Institute of Technology. It is a pleasure to work together with Patrik Persson and Klas Nilsson.

This work has been sponsored by ARTES (A network for Real-Time research and graduate Education in Sweden), which is a research programme under SSF (Swedish Foundation for Strategic Research). A travel grant from the Royal Physiographic Society is also gratefully acknowledged.

Finally, I would like to thank my wife Anna for her encouragement, support, and love.

*Anton*

# Introduction

## 1. Background and Motivation

Real-time control plays an important part in modern technology. For example, engine management systems for cars increasingly rely on real-time computations and feedback control to improve performance, reduce fuel consumption, and minimize the amount of pollutant emissions. At the same time, as the capacity of microcontrollers is increasing and the cost is decreasing, all sorts of functionality is migrating from hardware to software. A control task may thus be executing in parallel with several other tasks, including other control tasks. This puts focus on *scheduling*, i.e. the choice of which task to execute at a given time. Since the beginning of the 1970s, the academic interest in real-time scheduling has been very large. Very little of this work has, however, focused on control tasks in particular. On the other hand, digital control theory, with its origin in the 1950s, does not directly deal with the resource constraints of the computing system. Instead, it is commonly assumed that the controller executes as a simple loop in a dedicated computer.

This work aims at getting the best possible control performance from limited computing resources. To accomplish this goal, integration of the real-time scheduling design and the control design is necessary. Today, real-time control design is typically a two-step procedure; control design followed by real-time design. This leads to sub-optimal

solutions and possibly poor utilization of resources. In an integrated approach, care can be taken that both control performance and timing requirements are respected all the way through the implementation. Integrated design also allows for dynamic solutions where the controllers and the scheduler exchange information during run-time. This is referred to as *feedback scheduling*.

This thesis deals with some different aspects of the real-time control system co-design problem. One goal is to minimize the control performance loss due to jitter and delays in the real-time system. The problem is addressed by scheduling the two main parts of a control algorithm as separate tasks. Another goal is to increase the average resource utilization and the control performance by relaxing the nominal requirements on hard deadlines and fixed worst-case execution times. This problem is studied in the context of feedback scheduling of hybrid control tasks. A third goal is to provide a simulation environment for real-time control systems, where it is possible to study the interaction between the scheduler, the control tasks, and the plant under control.

## 2. Outline, Contributions, and Related Publications

The thesis consists of four papers and this introductory chapter. Below, the contributions of each paper are summarized. References to related publications are also given.

### Paper 1

Årzén, K.-E., A. Cervin, J. Eker, B. Bernhardsson, and L. Sha: "Towards the integration of control and real-time scheduling design." Submitted for journal publication.

***Contributions*** The paper gives a survey of the state of the art in the field of integrated control and scheduling. Theory from both fields is reviewed. Periodic control loops are discussed from both control and scheduling perspectives. An overview of flexible and adaptive scheduling is given, and control applications with flexible timing needs are identified. A number of research problems related to co-design of real-time control systems are outlined.

### *Related Publications*

Årzén, K.-E., B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, and L. Sha: "Integrated control and scheduling." Report ISRN LUTFD2/TFRT--7586--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, August 1999.

Årzén, K.-E., A. Cervin, J. Eker, and L. Sha: "An introduction to control and real-time scheduling co-design." Submitted to the 39th IEEE Conference on Decision and Control, Sydney, Australia, December 2000.

## Paper 2

Cervin, A.: "Improved scheduling of control tasks." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 4–10. York, UK, June 1999.

***Contributions*** Sub-task scheduling of the two main parts of a control algorithm, *Calculate Output* and *Update State*, is investigated. A deadline-assignment algorithm is given that attempts to minimize the computational delay for a set of control tasks in a priority-preemptive real-time system. The control performance improvements are evaluated by simulations of the scheduler, the controller, and the process.

## Paper 3

Eker, J. and A. Cervin: "A Matlab toolbox for real-time and control systems co-design." In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, pp. 320–327. Hong Kong, P.R. China, December 1999.

***Contributions*** A MATLAB/SIMULINK-based simulator for real-time control systems is presented. It facilitates simultaneous simulation of real-time task execution and continuous plant dynamics. The simulator makes it possible to evaluate control algorithms from a timing perspective and scheduling algorithms from a control performance perspective. It is particularly well suited for simulation of time-varying controllers and flexible schedulers.

***Related Publications*** A reference manual is available:

Cervin, A.: "The real-time control systems simulator—Reference manual." Report ISRN LUTFD2/TFRT--7592--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, April 2000.

The simulator software is available at
http://www.control.lth.se/~anton/rtkernel

### Paper 4

Cervin, A. and J. Eker: "Feedback scheduling of control tasks." Submitted to the 39th IEEE Conference on Decision and Control, Sydney, Australia, December 2000.

***Contributions*** A feedback scheduler for hybrid control tasks with large variations in their execution time is presented. The paper demonstrates that feedback and feedforward principles can be used in real-time scheduling to increase control performance and to avoid CPU overloads. The solution is based on co-design of the scheduler and the control tasks.

***Related Publications*** The execution-time properties of a hybrid controller for a double-tank process are investigated in

Persson, P., A. Cervin, and J. Eker: "Execution-time properties of a hybrid controller." Report ISRN LUTFD2/TFRT--7591--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, April 2000.

Feedback scheduling is applied to event-based PID controllers in

Årzén, K.-E. and A. Cervin: "A simple event-based PID controller." Accepted for publication in *Control Engineering Practice*.

### Outline of Chapter

The rest of this introductory chapter elaborates on some specific topics in the thesis. Section 3 discusses the notion of deadlines in real-time control systems. Section 4 presents some control and timing experiments on an inverted pendulum. Section 5 discusses future work, and Section 6 contains the references.

# 3. Deadlines in Real-Time Control Systems

This section elaborates on the notion of deadlines in real-time control systems. It is argued that most controller deadlines are not hard, and that the consequences of missed deadlines should be evaluated from a control performance perspective. An example is given that shows that control performance and scheduling performance (i.e. the ability to meet deadlines) are completely different things.

## Background

Ever since the seminal paper on real-time scheduling theory, [Liu and Layland, 1973], computer-controlled systems have been used as the primary example of *hard real-time systems*. In a hard real-time system, the computer responds to periodic or non-periodic *events* by executing *tasks* that must finish within *hard deadlines*—or else, the system will fail. This description has shaped much of the real-time systems research during the past thirty years. It is questionable, however, whether this description really fits the large majority of computer-controlled systems.

***Computer-Controlled Systems*** The basic structure of a computer-controlled system is shown in Figure 1. Typically, a controller is implemented as a task that should execute periodically in the computer. In each period, the controller should sample the process output, $y$, execute the control algorithm, and send the new control signal, $u$, to the input of the process. The controller is often designed using sampled-

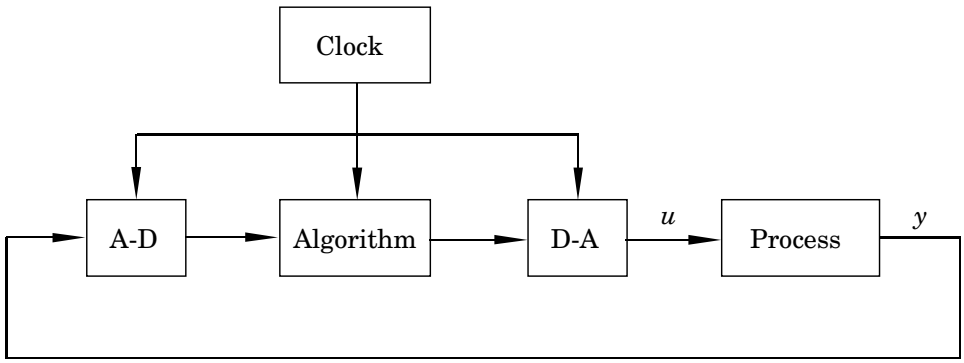



**Figure 1.** Overview of a computer-controlled system.

data control theory, see e.g. [Åström and Wittenmark, 1997], which assumes that the samples are taken at equidistant points in time.

The performance and stability of the closed-loop system depends not only on the control algorithm, but also on the actual implementation and run-time behavior of the controller in the computer system. The computing hardware, the execution-time properties of the control algorithm, the real-time operating system, the scheduling algorithm, and the possible network delays all introduce various amounts of delay and jitter in the control system. In the end, from a control perspective, it is the *actual sampling period* (including jitter) and the *actual input-output latency* (including jitter) that influence the performance and stability of the closed-loop system. Further discussion on this is found in Paper 1, Section 4; and in [Törngren, 1998].

***Real-Time Scheduling*** The purpose of real-time scheduling is to ensure that the timing requirements of all tasks in the computer are met. In the basic task model, a task is described by a period $T$, a deadline $D$, and an execution time $C$. Here, we focus on *dynamic scheduling*, where tasks are dynamically dispatched by the real-time operating system according to a scheduling algorithm. The two main approaches are fixed-priority scheduling and deadline-driven scheduling.

Schedulability analysis is used to predict, off-line, whether the timing requirements of all tasks will be met. The analysis is based on *worst-case assumptions* about the task execution times, the arrival pattern of the tasks, etc. For instance, under fixed-priority scheduling, schedulability is determined by computation of worst-case response times, $R$, of the different tasks [Joseph and Pandya, 1986]. The response-time of a task is defined as the time from its release (i.e. the beginning of the period) to its completion. All deadlines are guaranteed to be met if and only if $R \leq D$ for all tasks. A more detailed overview of real-time scheduling is given in Paper 1, Section 2.

### Where Do Deadlines Come From?

In the hard real-time scheduling literature, it is often unclear where the timing constraints—including deadlines—actually come from [Ramamritham, 1996]. The paper [Liu and Layland, 1973] is an exception to this, however. Assumption (A2) clearly states that "Deadlines consist of run-ability constraints only—i.e. each task must be completed

before the next request occurs." This implies that $D = T$ for all tasks. The connection to control theory is then made by mentioning that "Any control loops closed within the computer must be designed to allow at least an extra unit sample delay." Later research has extended the schedulability analysis to handle the cases $D < T$ and $D > T$ as well, see e.g. [Tindell *et al.*, 1994] and [Stankovic *et al.*, 1998], but the origin of the deadlines and their relation to control theory are rarely mentioned.

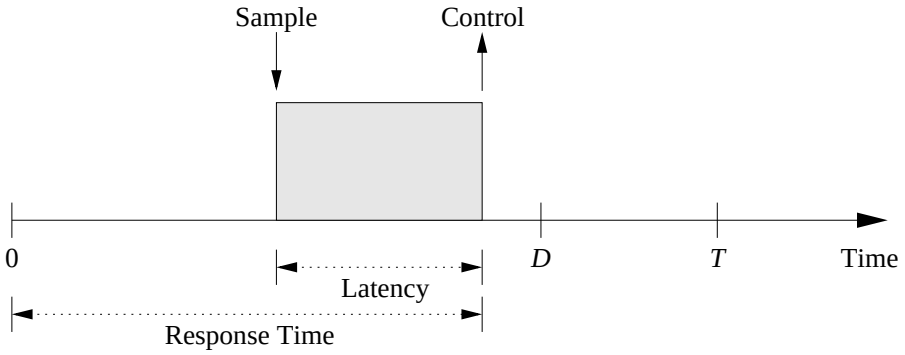



**Figure 2.** Illustration of the relationship between deadline ($D$), response time, and input-output latency for a control task. The latency is bounded by the response time, which in turn is bounded by the deadline. The task is assumed to be released at time zero.

From a control perspective, deadlines are primarily used to bound the input-output latency of the controllers. Figure 2 illustrates the relationship between deadline, response time, and input-output latency for a control task. Here, it is assumed that the A-D conversion takes place when the control task starts its execution, and that the D-A conversion takes place when the task completes its execution. A common alternative is to sample the process at the beginning of the period. Furthermore, it is common practice to divide the control algorithm into two parts—*Calculate Output* and *Update State*—and send out the control signal when the first part has completed. Further discussion on this is found in Paper 1, Section 4; and in Paper 2.

For controlled plants with unstable dynamics, bounded latency is absolutely necessary to guarantee the stability of the closed-loop system. For all plants, bounded latency is necessary to guarantee some minimum level of control performance. Also, in general, the smaller the latency can be made, the better control performance and robust-

ness can be achieved. Thus, typical deadlines assigned to control tasks ($D \leq T$) are often much tighter than what is dictated by pure stability considerations.

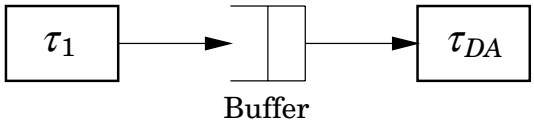



**Figure 3.** In fixed-priority scheduling with offsets, a high-priority, dedicated output task $\tau_{DA}$ can be used to minimize the output jitter of control task $\tau_1$. This enforces a deadline on $\tau_1$.

Deadlines may also be derived from other real-time constraints in the implementation. For instance, consider the case in Figure 3, where a dedicated, high-priority output task $\tau_{DA}$ is used to minimize the output jitter of control task $\tau_1$, see [Locke, 1992]. The tasks have the same period, but $\tau_{DA}$ is released with a fixed *offset* $O$ relative to the release of $\tau_1$. A deadline $D < O$ must be assigned to $\tau_1$ to ensure that fresh output data will be available in the buffer when $\tau_{DA}$ starts its execution. See [Audsley *et al.*, 1993b] for further details on offset scheduling. Different control structures, such as synchronized loops, cascaded loops, etc., may also enforce additional time constraints, see [Sandström, 1999].

Truly *hard* deadlines in real-time control systems are studied in [Shin *et al.*, 1985]. There, hard constraints on the controlled variables (e.g. physical constraints) are used to derive maximum allowable control latencies in different regions of the state-space. It is also noted that the hard deadline may be a random variable due to stochastic disturbances acting on the process. The approach is extended in [Shin and Kim, 1992], where the stability of the closed-loop system is also considered. In the examples given, the hard deadlines are typically found to be several times longer than the sampling interval. This may not be so surprising. As pointed out, the sampling period of a controller is not only chosen to satisfy Shannon's sampling theorem, but also to achieve the desired performance. Thus, a controller always has some degree of robustness against variations in the sampling interval due to, for instance, missed deadlines.

### The True Consequences of Missed Deadlines

The term "to miss a deadline" is used very frequently in the real-time literature, but the exact implications of the words are very seldom discussed. Having established that most controller deadlines are not hard, the next step is to explore the true consequences of missed deadlines.

We consider controllers implemented in real-time environments that do not explicitly support periodic processes with deadlines. Ada, occam2, and C/POSIX are examples of real-time languages that lack such support [Burns and Wellings, 1997]. There, periodic loops are implemented using an absolute-delay timing primitive. The pseudo-code for a control loop can look something like this:

```
t = CurrentTime;
LOOP
  AD-Conversion;
  ControlAlgorithm;
  DA-Conversion;
  t = t + T;
  WaitUntil(t);
END
```

The only time constraint enforced during run-time is, that the task will never be released prior to its nominal release time. If the control task does not meet its design specifications during run-time, e.g. if the execution time of the control algorithm is larger than the predicted worst-case execution time, overruns may occur. Which tasks that will suffer from the overrun, and what the impact on the control performance will be, depend on the scheduling algorithm. This is illustrated by the following example.

EXAMPLE 1—COMPUTER CONTROL OF TWO MECHANICAL SERVOS

Consider the computer-controlled system in Figure 4, where two mechanical servos are being controlled by two PD (proportional-derivative) controllers executing as tasks in a computer. Let each servo be described by the transfer function

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

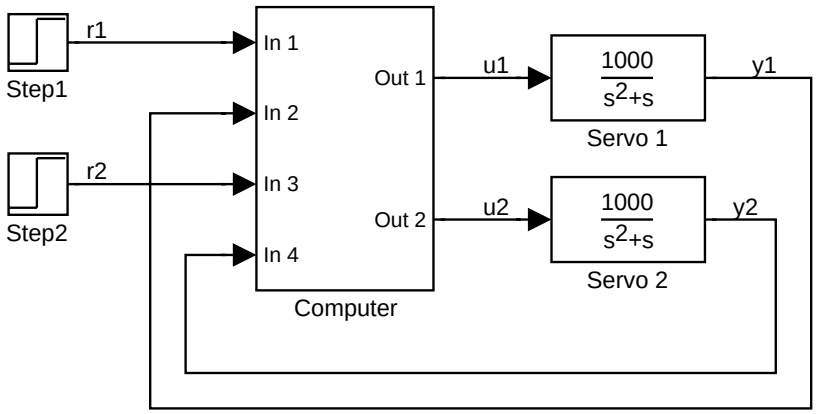



**Figure 4.** Two mechanical servos are being controlled by two PD controllers executing as tasks in the computer.

The goal of the control is to make the servo position, $y(t)$, follow the reference position, $r(t)$, as closely as possible. Every $T$ seconds, the controller should sample the servo position and the reference position and calculate a new control signal $u(t)$. The discrete-time PD control algorithm is given by

$$P(t) = K(r(t) - y(t)) \tag{2}$$

$$D(t) = \frac{T_d}{NT + T_d} D(t - T) + \frac{NKT_d}{NT + T_d}(y(t - T) - y(t)) \tag{3}$$

$$u(t) = P(t) + D(t) \tag{4}$$

where $K$ and $T_d$ are control parameters and $N = 100$ is a constant. The controllers are tuned on-line, one by one, to achieve the desired performance. Controller 1 is assigned the sampling interval $T_1 = 6$ ms and the parameters $K_1 = 1$ and $T_{d1} = 0.042$. Controller 2 has tighter specifications and is assigned the sampling interval $T_2 = 4$ ms and the parameters $K_2 = 1.5$ and $T_{d2} = 0.035$.

It is assumed that $D = T$ for both tasks. Unfortunately, the execution time of the control algorithm is longer than anticipated, $C = 3$ ms. The requested CPU utilization $U$ is

$$U = \frac{C}{T_1} + \frac{C}{T_2} = 1.25 \tag{5}$$

Since $U > 1$, the system is overloaded, and no scheduling algorithm can guarantee that all deadlines will be met.

***Rate-Monotonic Scheduling*** First, the behavior under rate-monotonic (RM) scheduling is investigated. Task 2 is given priority over Task 1 since $T_2 < T_1$. The first part of the resulting schedule, when both tasks are released at time zero, is shown in Figure 5. Because

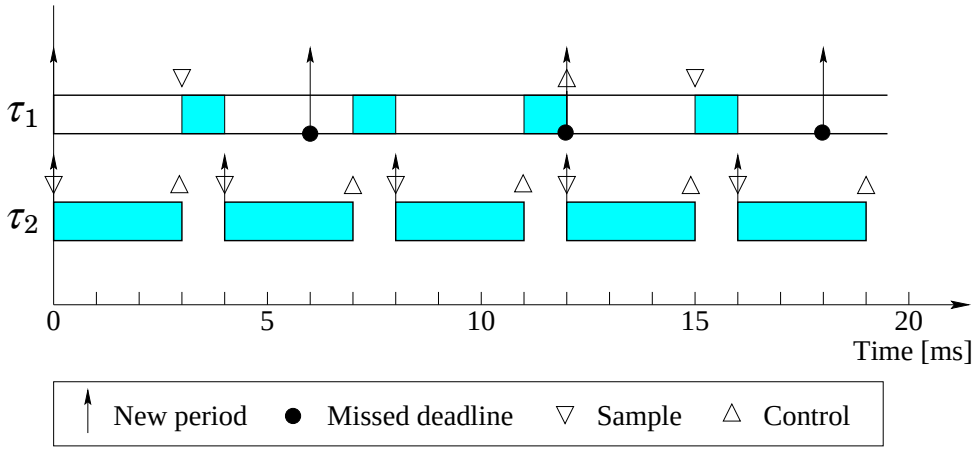



**Figure 5.** The first part of the schedule under RM scheduling. Task 1 misses all its deadlines.

of preemption, Task 1 misses all its deadlines. The regularity of the schedule makes it easy to determine the *actual sampling period*, $\overline{T}$, and the *actual input-output latency*, $\overline{L}$, of the controllers. By inspection, they are found to be constant (jitter-free) and equal to $\overline{T}_1 = 12$ ms, $\overline{L}_1 = 9$ ms, $\overline{T}_2 = 4$ ms, and $\overline{L}_1 = 3$ ms. It is these numbers that govern the closed-loop behavior of the controllers.

The complete real-time system, including the servos, is simulated using the real-time control systems simulator in Paper 3. The step responses of the closed-loop systems are displayed in Figure 6. The performance of Controller 2 is good, as expected. Controller 1, however, loses stability due to the long sampling period and the long latency, which the controller has not been designed for. This can also be shown by applying stability analysis to the resulting closed-loop system. In this case the analysis would be straight-forward since the resulting sampling periods and latencies are constant.

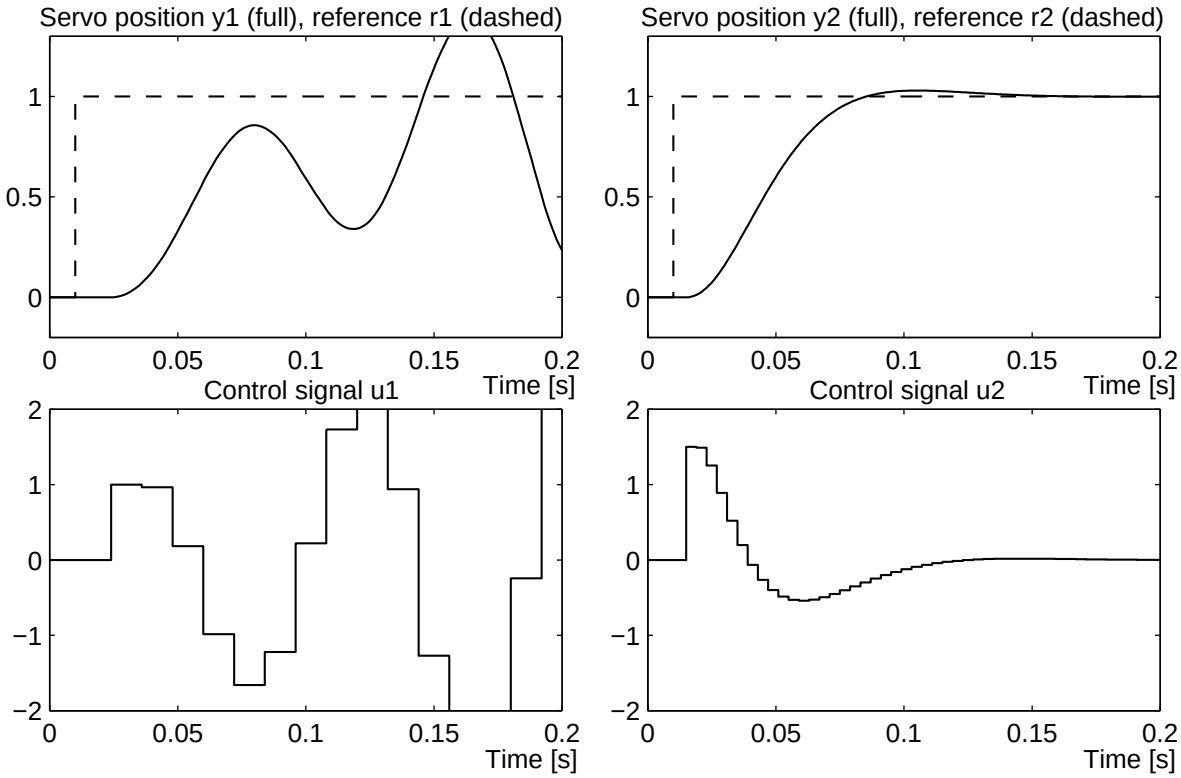



**Figure 6.** Step responses under RM scheduling for Controller 1 (left) and Controller 2 (right). Controller 1 loses stability due to the long actual sampling period and the long actual input-output latency.

***Earliest-Deadline-First Scheduling*** Next, the behavior under earliest-deadline-first (EDF) scheduling is investigated. The tasks execute in the order of their absolute deadlines. Ties can be broken arbitrarily—assume that Task 1 gets to execute before Task 2 in those cases. The first part of the resulting schedule, when both tasks are released at time zero, is shown in Figure 7. After an initial transient, *all* deadlines are missed. This is due to the well-known *domino effect*. There is no preemption in the resulting schedule so the actual input-output latency is always equal to $\overline{L} = 3$ ms for both controllers. The actual sampling periods are no longer constant, but they exhibit periodic behavior. By inspection, $\overline{T}_1$ is found to exhibit the cycle $\{6, 9\}$ ms while $\overline{T}_2$ exhibits the cycle $\{6, 6, 3\}$ ms.

The step responses of the closed-loop systems are displayed in Figure 8. The systems are stable and the performance is satisfactory for both controllers, despite the missed deadlines. The jitter due to the

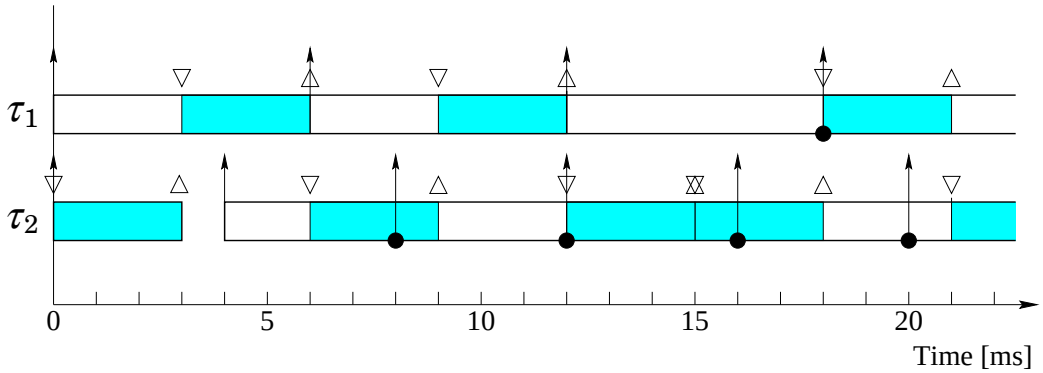



**Figure 7.** The first part of the schedule under EDF scheduling. After an initial transient, all deadlines are missed.

varying sampling periods is clearly visible in the control signals.

Again, the behavior of the closed-loop system would be possible to derive using control theory. The analysis would be more complicated in this case, however, due to the varying (but cyclic) sampling intervals. □

The example shows that control performance and scheduling performance are completely different things. It also displays that EDF scheduling distributes the available computing resources more evenly in overload situations than RM scheduling does.

The constant overload situation in the example is if course extreme. The effects of transient overruns on control performance will in general be much smaller—if at all noticeable.

## Conclusions

Relaxing the nominal requirements on hard deadlines in real-time control systems is motivated from a resource utilization viewpoint. Modern computing hardware tends to be optimized for high average-case performance rather than guaranteed worst-case performance. Also, many control algorithms display considerable variations in their execution-time demands, see Paper 1, Section 6. Taken together, scheduling based on worst-case execution times and hard deadlines may be infeasible for a large set of control applications.

Exploring the consequences of overruns is necessary in order to make the correct co-design trade-offs. For instance, it may be beneficial to decrease the average period of a controller, and allow it to miss a few

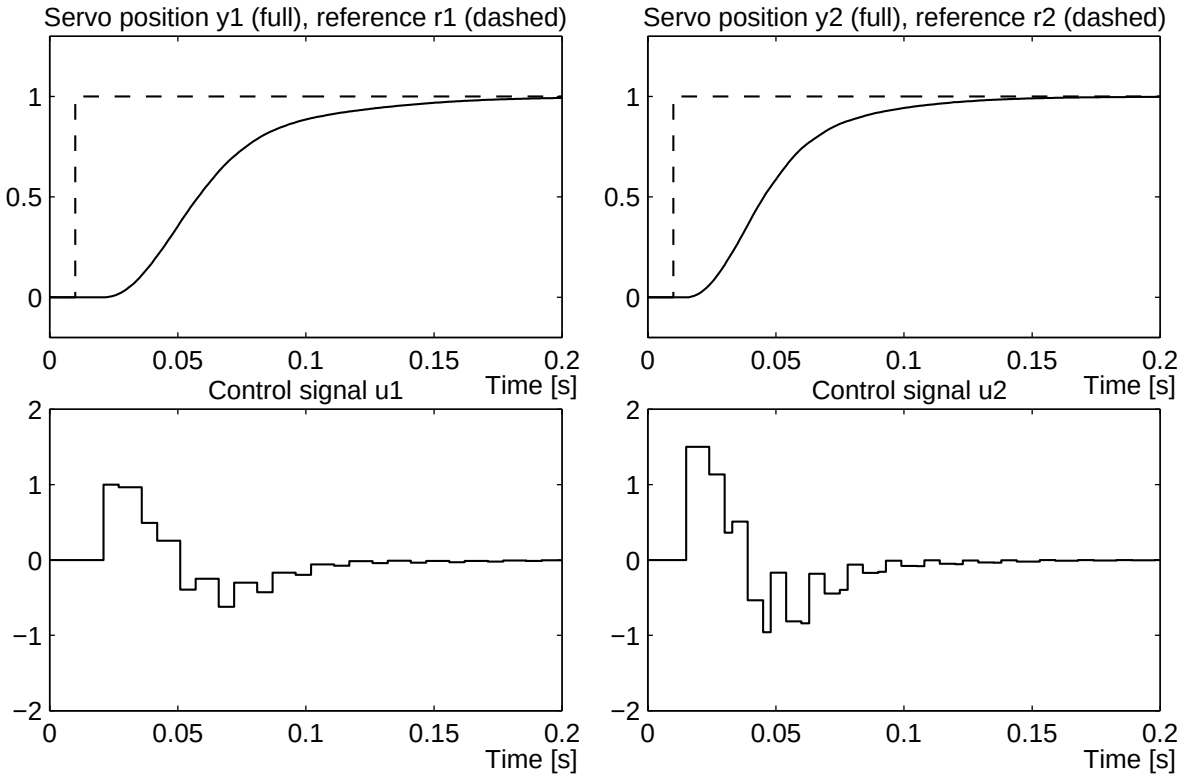



**Figure 8.** Step responses under EDF scheduling for Controller 1 (left) and Controller 2 (right). Both systems are stable despite the fact that, after an initial transient, all deadlines are missed. The sampling interval jitter is clearly visible in the control signals.

deadlines, as long as the worst-case latency does not exceed a certain bound.

In the suggested analysis, the first step is to determine what the actual sampling period and the actual input-output latency will be for the different control tasks. These quantities will be random variables in the general case, depending on the task execution-time distributions, the scheduling algorithm, the mechanisms in the real-time operating system, the controller structure, etc. The next step is to determine what the impact on the control performance will be. For this purpose, the simulator in Paper 3 can be a useful tool. Application of stochastic control theory is another alternative. In the end, however, worst-case analysis will still be necessary to guarantee that the truly hard constraints in the control system are not violated.

## 4. Inverted Pendulum Experiments

Some control and timing experiments on the Furuta pendulum, see Figure 9, were performed. One purpose was to derive some experimental performance loss functions with respect to sampling interval, input-output latency, and jitter. Such functions are essential for making scheduling trade-offs in the co-design of real-time control systems, see Section 6 in Paper 1 and the discussion on deadlines in the previous section. Another purpose was to experimentally verify the control performance improvements due to the scheduling algorithm given in Paper 2.

**Figure 9.** The Furuta pendulum.

The Furuta pendulum process consists of a pendulum attached to the end of a rotating arm. A motor is used to apply a torque to the arm, and the purpose is to swing up and stabilize the pendulum in the upright (inverted) position.

Let $\varphi$ denote the angle of the arm and $\theta$ the angle of the pendulum.

Introduce the state vector

$$x = [\,\theta \quad \dot{\theta} \quad \varphi \quad \dot{\varphi}\,]^T .$$

The full state vector is directly measurable on the process. Linearization around the upright position gives the continuous-time state-space description

$$\frac{dx}{dt} = Ax + Bu = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 31.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.588 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -71.2 \\ 0 \\ 191 \end{bmatrix} u,$$

where the control signal $u$ is proportional to the applied torque. Assuming a sampling interval $h$, a digital state-feedback controller $u(kh) = -Lx(kh)$ can be designed based on the sampled-data description

$$x(kh + h) = \Phi(h)x(kh) + \Gamma(h)u(kh),$$

where $\Phi(h) = e^{Ah}$ and $\Gamma(h) = \int_0^h e^{As}Bds$.

### Experimental Performance Loss Functions

Co-design of real-time control systems requires knowledge about the relationships between scheduling attributes and controller attributes. Since the CPU utilization of a controller is directly proportional to its sampling frequency, the relationship between the sampling interval and the control performance is of central importance. It is also important to know the effects of jitter and delay on the control performance.

Three approaches are possible. First, some loss functions may be computed analytically. Loss functions with respect to period for another pendulum model have been calculated, see Section 6 in Paper 1 and [Eker *et al.*, 2000]. Second, the loss functions can be obtained by simulation, for instance using the real-time control systems simulator in Paper 3. The third option, which is explored here, is to perform experiments on the real process. This has the advantage of capturing the effects of unmodeled dynamics and disturbances. For instance, friction can have great impact on the performance.

Controllers for the inverted pendulum were designed using pole placement. The closed-loop system is specified in terms of the continuous-time characteristic polynomial

$$(s^2 + 2\omega_1\zeta_1 s + \omega_1^2)(s^2 + 2\omega_2\zeta_2 s + \omega_2^2),$$

where $\omega_1 = \omega_2 = \omega_c = 10$ rad/s is the desired bandwidth and $\zeta_1 = \zeta_2 = 0.9$ is the desired relative damping.

The performance of the controllers were evaluated in terms of the variance of the pendulum angle $\theta$ in steady-state. In each control experiment, pendulum angle was measured for 30 s. The variance is a reasonable performance measure for the pendulum process, which is an unstable system that requires active control. It must be remembered, however, that control design is always multi-objective, and that it is hard condense the properties of the closed-loop system to a single number.

The Linux in Control [Blomdell, 1999] experimental real-time platform and MATLAB/SIMULINK with additional IO-blocks were used for the implementation. An existing SIMULINK controller implementation [Åkesson, 1999] for the Furuta pendulum was modified to allow separate timing of the sampling and control actions. The simulator in Paper 3 could unfortunately not be used for real-time implementation since it consumes too much computing resources. However, it was used to generate timing data, i.e. sampling and control instants, for the experiments.

***Performance vs Sampling Period*** The feedback gain vector $L$ was computed for ten different sampling intervals in the range $h = [5 \ldots 50]$ ms. Sampling intervals longer than that were found to be useless in preliminary experiments. This is also hinted at by the rule of thumb, see e.g. [Åström and Wittenmark, 1997], that suggests that the sampling interval should be chosen such that

$$0.15 < \omega_c h < 0.5.$$

Figure 10 shows the measured steady-state variance of the pendulum angle for different sampling periods. The loss function is monotonically increasing which seems very reasonable for an unstable process.

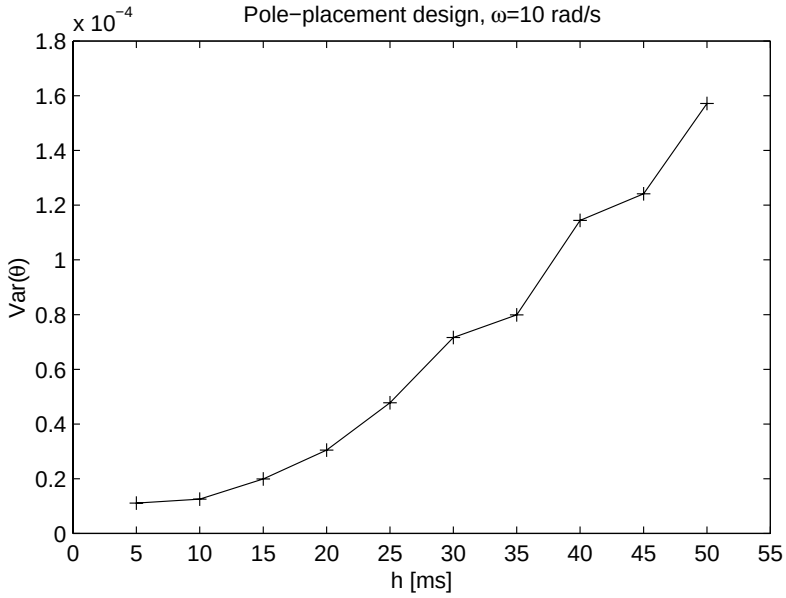



**Figure 10.** Measured steady-state variance of the pendulum angle for different sampling periods. The loss increases as the period increases.

The loss function also seems convex, which is an assumption that has been made in some papers when optimizing sampling periods for a set of control tasks [Seto *et al.*, 1996; Eker *et al.*, 2000]. The results can be compared to the analytically computed loss function for a pendulum process shown in Figure 14 (left) in Paper 1, Section 6. It should be noted, though, that those calculations assumed a second-order model, a linear-quadratic (LQ) controller, and a performance measure that included the variance of the control signal as well.

***Performance vs Input-Output Latency*** Scheduling can introduce delays in a controller that were not anticipated when the controller was designed. Figure 11 shows the results from an experiment where the pendulum controller designed for $h = 36$ ms and zero delay was subjected to fixed delays ranging from 0 to 28 ms. The performance loss naturally increases as the delay increases. When the delay exceeds 24 ms the performance deteriorates completely. Further experiments indicated that jitter in the delay didn't add significantly to the performance loss, cf. the discussion on sampling jitter below.

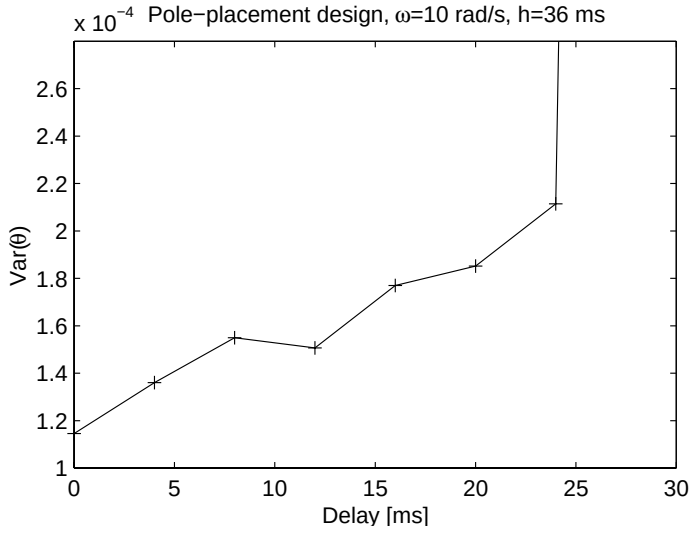



**Figure 11.** Measured steady-state variance of the pendulum angle for different amounts of input-output latency. The loss increases as the latency increases. When the latency exceeds 24 ms, the performance deteriorates completely.

***Performance vs Sampling Jitter*** Depending on how the sampling mechanism is implemented, scheduling may or may not introduce sampling jitter. Let the relative jitter be defined as the maximum difference between successive sampling intervals relative to the nominal sampling period. Figure 12 shows the results from another experiment on the $h = 36$ ms controller with relative sampling jitter in the range from 0 to 180 %. The controller seems quite insensitive to sampling jitter, as the performance loss only grows slightly when the jitter becomes very large. This can be explained by the fact that the full state vector is measurable. Thus, the controller gets correct state information regardless of when the sample is taken. A state observer would be much more sensitive to sampling jitter, since its computations rely critically on the assumptions about equidistant samples, see the discussion on sampling jitter in Paper 1, Section 4.

## Improved Scheduling Example and Experimental Evaluation

Some further inverted pendulum experiments were performed to verify the control performance improvements due to the scheduling algorithm given in Paper 2.

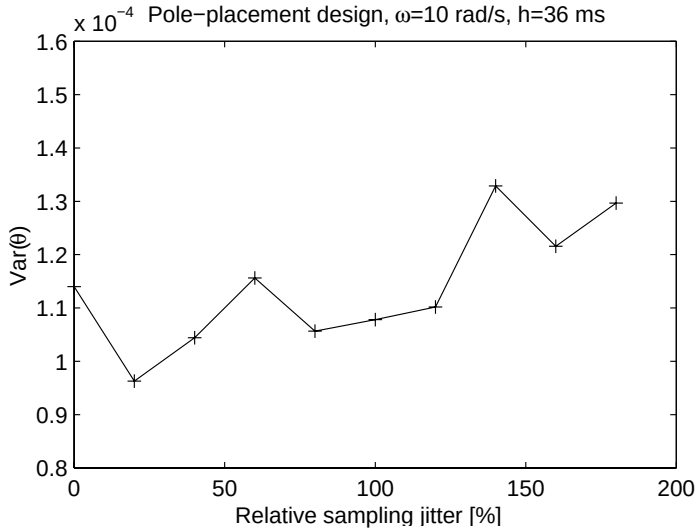



**Figure 12.** Measured steady-state variance of the pendulum angle for different amounts of sampling jitter. The loss increases only slightly as the relative jitter becomes very large.

The pseudo-code for a digital controller typically looks like this:

```
t = CurrentTime;
LOOP
  AD-Conversion;
  CalculateOutput;
  DA-Conversion;
  UpdateState;
  t := t + h;
  WaitUntil(t);
END
```

In order to minimize the input-output latency of the controller, the control algorithm is divided into two parts, *Calculate Output* and *Update State*, where the execution of Update State can be postponed until after the D-A conversion.

Paper 2 shows that, given a set of control tasks, scheduling Calculate Output and Update State as separate tasks can reduce the delay due to preemption in the controllers. The control performance improvements were verified by simulation. The reader is referred to Paper 2 for further background and references.

No dynamic real-time scheduling was actually performed in the control experiments. Instead, a set of control tasks were first *simulated* (see Paper 3) using the different scheduling algorithms from the paper. The timing data (i.e. the sampling instants and the control instants) from one of the tasks was then exported from the simulator and used as *static schedules* in the control experiments. The reason for this somewhat awkward procedure was that the original pendulum controller was implemented in SIMULINK and not as a task in a real-time operating system.

Now, consider a set of three control tasks $(\tau_1, \tau_2, \tau_3)$ with the sampling periods $(h_1, h_2, h_3) = (36, 22, 12)$ ms. Assume that the inverted pendulum is to be controlled by $\tau_1$. Furthermore, assume that the execution time of each task is $C = 6$ ms, and that each task can be divided into the subtasks Calculate Output and Update State with execution times $C_{CO} = 2$ ms and $C_{US} = 4$ ms respectively. (These are not actual execution times but only made-up numbers.) It is assumed that the A-D converters can be programmed to take samples with perfect periodicity, i.e., there is no sampling jitter. The D-A conversion is assumed to take place at the very end of the Calculate Output part.

***Rate-Monotonic Scheduling*** First, rate-monotonic scheduling, ignoring the sub-tasks, is considered. It is assumed that the deadline of a task is equal to its period, i.e. $D = T$ $(= h)$. The tasks are assigned fixed priorities $P$ according to their rates (a high priority number denotes a high priority). The task set is summarized below:

| | $T$ | $D$ | $C$ | $P$ |
|---|---|---|---|---|
| $\tau_1$ | 36 | 36 | 6 | 1 |
| $\tau_2$ | 22 | 22 | 6 | 2 |
| $\tau_3$ | 12 | 12 | 6 | 3 |

The worst-case response times, $R$, are calculated:

| | $T$ | $D$ | $C$ | $P$ | $R$ |
|---|---|---|---|---|---|
| $\tau_1$ | 36 | 36 | 6 | 1 | 36 |
| $\tau_2$ | 22 | 22 | 6 | 2 | 12 |
| $\tau_3$ | 12 | 12 | 6 | 3 | 6 |

Since $R \leq D$ for all tasks, the task set is schedulable.

From a control perspective, the input-output latencies of the controllers are important. In particular task $\tau_1$ will be preempted a lot, since it has the lowest priority. The exact distribution of the latencies depends on the execution-time distributions and the phasing of the tasks. It is assumed that the actual execution times are equal to the worst-case execution times and that all tasks are released simultaneously at time zero. Simulating the system for the duration of the least common multiple (LCM) of the periods, 396 ms, reveals the delay distribution for $\tau_1$ shown in Figure 13. The average delay is 13 ms, which gives an indication of what performance to be expected from the pendulum controller, see the loss function in Figure 11.

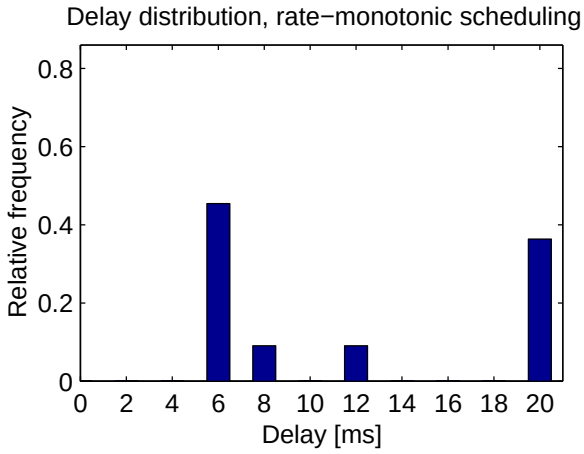



**Figure 13.** Input-output latency distribution for task $\tau_1$ under rate-monotonic scheduling. The average delay is 13 ms. An indication of what performance to expect is given by the loss function in Figure 11.

The timing data for task $\tau_1$ was exported from the simulator and used as a static schedule in the actual pendulum experiment. The controller ran for 30 s and the pendulum angle variance was found to be $1.7 \cdot 10^{-4}$.

***Improved Scheduling*** Next, the improved scheduling is considered. In the analysis, each task $\tau_i$ is divided into the subtasks $\tau_{COi}$ (Calculate Output) and $\tau_{USi}$ (Update State). The deadlines of the Update State parts are equal to the task periods. The deadlines of the Calculate Output parts are derived using the deadline assignment al-

gorithm found in Paper 2. First, the Calculate Output parts are assigned *effective deadlines* (i.e. as late deadlines as possible). Then, deadline-monotonic priorities are assigned to the sub-tasks. We have the following task set:

| | $T$ | $D$ | $C$ | $P$ |
|---|---|---|---|---|
| $\tau_{CO1}$ | 32 | 28 | 2 | 2 |
| $\tau_{US1}$ | 32 | 32 | 4 | 1 |
| $\tau_{CO2}$ | 22 | 18 | 2 | 4 |
| $\tau_{US2}$ | 22 | 22 | 4 | 3 |
| $\tau_{CO3}$ | 12 | 8 | 2 | 6 |
| $\tau_{US3}$ | 12 | 12 | 4 | 5 |

Next, new deadlines for the Calculate Output parts are derived using the deadline-assignment algorithm in Paper 2. The deadlines are iteratively decreased until the response times converge. The following table results:

| | $T$ | $D$ | $C$ | $P$ | $R$ |
|---|---|---|---|---|---|
| $\tau_{CO1}$ | 32 | 6 | 2 | 4 | 6 |
| $\tau_{US1}$ | 32 | 32 | 4 | 1 | 36 |
| $\tau_{CO2}$ | 22 | 4 | 2 | 5 | 4 |
| $\tau_{US2}$ | 22 | 22 | 4 | 2 | 20 |
| $\tau_{CO3}$ | 12 | 2 | 2 | 6 | 2 |
| $\tau_{US3}$ | 12 | 12 | 4 | 3 | 10 |

As seen, all Calculate Output parts have been given higher priorities than the Update State parts. Task $\tau_1$ will suffer from less preemption in its time-critical part and the input-output latency will be decreased. Simulating the resulting system for the duration of the LCM gives the delay distribution for $\tau_1$ shown in Figure 14. The average delay has been reduced to 4 ms, and better control performance can be expected, see Figure 11.

The timing data for task $\tau_1$ was exported and used in another pendulum experiment. Running the controller for 30 s, the pendulum angle variance was now found to be $1.2 \cdot 10^{-4}$.

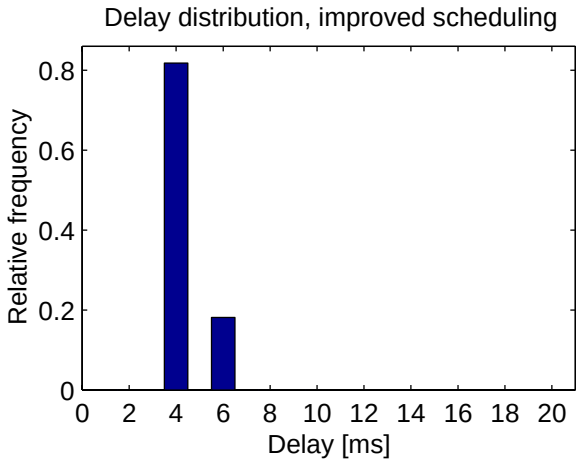



**Figure 14.** Input-output latency distribution for task $\tau_1$ under improved scheduling. The average delay is 4 ms, which is much shorter than under rate-monotonic scheduling, see Figure 13. Better performance can be expected, as indicated by the loss function in Figure 11.

***Conclusions*** Comparing the control performance measurements, it is found that improved scheduling could reduce the pendulum variance by about 30 % compared to rate-monotonic scheduling. For another controller or another process, the results would have been different. The improvements can predicted by computing the delay distributions for the task under the different scheduling policies and then examining the performance-*vs*-latency loss function for the controller.

## 5. Future Work

A number of research topics related to the field of integrated control and real-time scheduling are highlighted in Paper 1, Section 6. Some examples of possible future work are outlined here.

***Flexible Control and Scheduling*** There exist a large number of flexible and adaptive scheduling techniques, e.g. value-based scheduling and scheduling of imprecise calculations. Likewise, many controllers have flexible and variable timing needs, including model-predictive controllers and event-based controllers. Are there any direct connections? Can the scheduling techniques be tailored for control applications?

***Cost Functions and Co-Design*** In the co-design of real-time control systems, there exist fundamental trade-offs between the different task timing attributes: sampling period, input-output latency, and jitter. What are the relationships between these attributes and control performance? How can knowledge about the relationships be exploited in the control design and the scheduling design?

One approach is to represent the relationships between timing attributes and control performance by cost functions. Suitable performance criteria must be chosen and methods for computing the cost functions need to be developed.

***Feedback Scheduling*** The feedback scheduling structure suggested in Paper 4 is only one of many possible. Several other variables could be communicated between the scheduler and the control tasks. The scheduler could perform optimization of the control performance with respect to sampling periods, input-output latencies, and jitter. How the scheduler and the controllers should be implemented, and how much resources the scheduler should be allowed to consume, are some other questions.

***Switching Analysis*** A feedback scheduler may change the sampling frequency of a controller. This can be interpreted as a switch between different controllers. It is a well-known fact from hybrid system theory that switching between two dynamic systems, that each on its own is stable, may cause instability. Thus, stability analysis for controllers in feedback-scheduling systems should be investigated. A related topic could be the development of robust design methods for such systems.

***Simulation*** The work on the real-time control systems simulator in Paper 3 will continue. The next version will provide cleaner interfaces between the control algorithms, the scheduling algorithm, and the real-time operating system. We would also like to facilitate the simulation of task execution triggered by external interrupts. This would allow simulation of event-based control systems, e.g. engine controllers that are sampled against engine speed.

***Incorporation of Timing Analysis*** One half of the research project "Integrated Control and Scheduling" is devoted to interactive time and

memory analysis of software for embedded real-time systems [Persson, 2000]. Ideally, we would like to create an integrated development environment for real-time control systems, where the engineer can develop controllers and get interactive feedback on their timing properties, perform co-simulation with input from the timing analysis, and finally generate code for the target system.

***Implementation Issues*** A number of implementation issues exist. For instance, how easily can the suggested approaches be implemented in commercial real-time operating systems? We are also going to need an implementation test-bed, where we can conduct control and scheduling experiments. One possibility could be to extend Pålsjö, which is a software environment for dynamically configurable embedded control systems [Eker, 1999].

## 6. References

Åkesson, J. (1999): "Safe reference following on the inverted pendelum." Report ISRN LUTFD2/TFRT--7587--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Åström, K. J. and B. Wittenmark (1997): *Computer-Controlled Systems*, third edition. Prentice Hall.

Audsley, N., K. Tindell, and A. Burns (1993): "The end of the line for static cyclic scheduling." In *Proceedings of 5th Euromicro Workshop on Real-Time Systems*.

Blomdell, A. (1999): "Linux in control." http://www.control.lth.se.

Burns, A. and A. Wellings (1997): *Real-Time Systems and Programming Languages*. Addisson-Wesley.

Eker, J. (1999): "A tool for interactive development of embedded control systems." In *Preprints 14th World Congress of IFAC*. Beijing, P.R. China.

Eker, J., P. Hagander, and K.-E. Årzén (2000): "A feedback scheduler for real-time control tasks." *Control Engineering Practice*. To appear.

Joseph, M. and P. Pandya (1986): "Finding response times in a real-time system." *The Computer Journal*, **29:5**, pp. 390–395.

Liu, C. L. and J. W. Layland (1973): "Scheduling algorithms for multiprogramming in a hard-real-time environment." *Journal of the ACM*, **20:1**, pp. 40–61.

Locke, C. D. (1992): "Software architecture for hard real-time applications: Cyclic vs. fixed priority executives." *Real-Time Systems*, **4**, pp. 37–53.

Persson, P. (2000): *Predicting Time and Memory Demands of Object-Oriented Programs*. Licentiate thesis, Department of Computer Science, Lund Institute of Technology, Lund, Sweden.

Ramamritham, K. (1996): "Where do time constraints come from and where do they go?" *International Journal of Database Management*, **7:2**.

Sandström, K. (1999): *Modeling and Scheduling of Control Systems*. Licentiate thesis ISRN KTH/MMK/R--99/5--SE, Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden.

Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin (1996): "On task schedulability in real-time control systems." In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 13–21.

Shin, K. G. and H. Kim (1992): "Derivation and application of hard deadlines for real-time control systems." *IEEE Transactions on Systems, Man, and Cybernetics*, **22:6**, pp. 1403–1413.

Shin, K. G., C. M. Krishna, and Y.-H. Lee (1985): "A unified method for evauating real-time computer controllers and its applications." *IEEE Transactions on Automatic Control*, **30:4**, pp. 357–366.

Stankovic, J., M. Spuri, K. Ramamritham, and G. Buttazzo (1998): *Deadline Scheduling for Real-Time Systems*. Kluwer Academic Publishers.

Tindell, K., A. Burns, and A. Wellings (1994): "An extendible approach for analyzing fixed priority hard real-time tasks." *Real-Time Systems*, **6:2**, pp. 133–151.

Törngren, M. (1998): "Fundamentals of implementing real-time control applications in distributed computer systems." *Real-time systems*, **14:3**.

# Paper 1

# Towards the Integration of Control and Real-Time Scheduling Design

**Karl-Erik Årzén, Anton Cervin, Johan Eker, Bo Bernhardsson, and Lui Sha**[1]

**Abstract**

The survey presents the state of the art of the emerging field of integrated control and scheduling. Among the subtopics discussed are timing in periodic control loops, flexible and adaptive scheduling, control and scheduling co-design, and feedback scheduling.

[1]Department of Computer Science, University of Illinois at Urbana-Champaign

## 1. Introduction

Most real-time control systems are embedded systems where the computer is a component in a larger engineering system. The control system is often implemented on a microprocessor using a real-time programming language such as Ada 95 or Modula-2 with a real-time kernel or run-time system, or using a sequential programming language such as C or C++ together with a real-time operating system (RTOS). The real-time kernel or OS uses multiprogramming to multiplex the execution of the tasks on the CPU. The CPU time, hence, constitutes a shared resource which the tasks compete for. To guarantee that the time requirements and time constraints of the individual tasks are all met, it is necessary to schedule the usage of the shared resource. During the last two decades scheduling of CPU time for real-time systems has been a very active research area and a number of different scheduling models and methods have been developed.

The most common, and simplest, model used within the real-time scheduling community assumes that the tasks are periodic, or can be transformed to periodic tasks, have a fixed period, a known worst-case bound on the execution time (WCET), and a *hard deadline*. The latter implies that it is imperative that the tasks always meet their deadlines, i.e., that the response time is always less or equal to the deadline, for each invocation of the task. This is in contrast to a *soft deadline*, that may occasionally be violated.

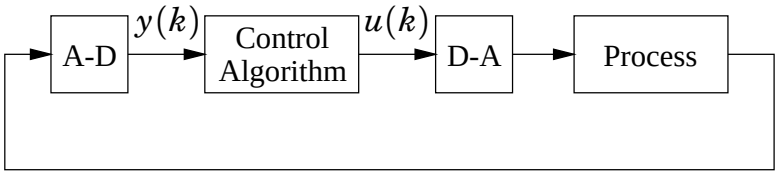



**Figure 1.** Controllers are often used as examples of periodic, hard real-time tasks. A typical control task consists of three parts: input data collection (A-D), control algorithm computation, and output signal transmission (D-A).

The most common example used by the real-time scheduling community for when this model is applicable is in computer-controlled systems, see Fig. 1. Controllers are assumed to be periodic tasks consisting of three parts: input data collection (A-D), control algorithm computation, and output signal transmission (D-A). The fixed period

assumption of the simple task model has also been widely adopted by the control community and has, e.g., resulted in the development of the sampled computer control theory with its assumption on deterministic, equi-distant sampling. Another result of the simple model is that it has provided a separation between the control community and the real-time scheduling community. The separation has allowed the control community to focus on its own problem domain without worrying about how scheduling is being done, and it has released the scheduling community from the need to understand what impact scheduling has on the stability and performance of the plant under control. From a historical perspective, the separated development of control and scheduling theories for computer-based control systems has produced many useful results and served its purpose. However, the separation has also had negative effects. The two communities have partly become alienated. This has led to a lack of mutual understanding between the fields.

Upon closer inspection it is quite clear that many of the assumptions of the simple model are too restrictive. First, the assumptions do not allow us to efficiently use low-cost general purpose hardware and off-the-shelf operating systems, which in general are not able to give any guarantees on determinism. These systems are, typically, designed to achieve good average performance rather than guaranteed worst-case performance. They often introduce significant non-determinism in task scheduling. For computation intensive high-end applications, the large variability in execution time caused by modern hardware architecture also becomes visible. The effect of this on the control loop is jitter in sampling period and control delay (input-output latency). In order to maintain good control performance it is important to compensate on-line for the variations. A requirement for this is that the necessary timing information is provided by the real-time kernel.

The assumptions of the simple model are also overly restrictive with respect to the characteristics of many control loops. Many control loops are not periodic, or they may switch between a number of different fixed sampling periods. Control loop deadlines are not always hard. On the contrary, many controllers are quite robust towards variations in sampling period and response time. Hence, it is questionable whether it is necessary to model them as hard deadline tasks. It is also in many cases possible to compensate on-line for the variations

by, e.g., recomputing the controller parameters. Obtaining an accurate value for the WCET is generally a difficult problem. Measuring WCET always implies the risk of underestimation, whereas analytical execution time analysis tools are still rare. When the WCET is much longer than the average execution time, an alternative may be to instead measure the actual execution time every task invocation and to adjust the task parameters accordingly. Finally, it is also possible to consider control systems that are able to tradeoff the available computation time, i.e., how long time the controller may spend calculating the new control signal, and the control loop performance.

There is a need for more general scheduling models that better fit the nature and needs of advanced control algorithms. The optimality of computer control is subject to the limitations of available computing resources, especially in advanced applications where we want to control fast plant dynamics and to use sophisticated state estimation and control algorithms. On the other hand, the true objective of real-time scheduling for control is to allocate limited computing resources in such a way that the state estimation and control algorithms can ensure the system's stability and optimize the system's performance. The computing resources could include CPU time and communication bandwidth. In the context of this survey we focus on CPU time. However, most of the issues brought up also apply to distributed system and scheduling of communication bandwidth.

The scheduling models and methods that we consider in this survey are based on dynamic feedback from the scheduler to the controllers and from the controllers to the scheduler. The idea of feedback has been used informally for a long time in scheduling algorithms for applications where the dynamics of the computation workload cannot be characterized accurately. For example, Internet protocols use feedback to help solve the congestion problems. Recently, under the title of quality of service (QoS), the idea of feedback has also been exploited in multi-media scheduling R&D. Given this, one might expect that the use of feedback in the scheduling of feedback control systems would have been naturally an active area. On the contrary, the scheduling research of feedback control systems are dominated by open loop analytic scheduling methods such as rate or deadline based algorithms. This is not an accident but rather the consequence of some serious theoretical challenges that require close cooperation between the control

and scheduling communities.

The aim of this survey is to present the current state of the art in the field of integrated control and scheduling and to identify some of the most important research issues.

### Visionary Goal

The development of more general scheduling models, and the complementary control theory creates a possibility for dynamic, flexible, and interactive integrated control and scheduling environments where the control design methodology takes the availability of computing resources into account during design and allows trade-offs between control performance and computing resource utilization. The system should support on-line information exchange between the on-line scheduler and the control tasks. The system should be able to adapt task parameters in overload situations in such a way that stability and an acceptable level of control performance are maintained. The requirement of known worst-case execution times should be relaxed. Instead the system should be able to guarantee stability and a certain level of control performance based only on knowledge of nominal execution times. To achieve this the system should be able to measure the actual execution time spent by the different tasks, and take appropriate actions in case of over-runs.

In order to make this possible, a lot of information needs to be provided. For example, the control tasks must be able to provide information to the on-line scheduler about the desired period of the control task, the period range for which the controller can guarantee acceptable control performance, nominal execution time, range of execution time variations, and desired deadline or deadline range. Alternatively this information can be stated as cost functions, e.g., functions for the control performance as a function of sampling period or for the control performance as a function of input-output latency.

### Outline of the Survey

A brief overview of hard real-time scheduling is given in Section 2, and an brief overview of sampled-data control theory is given in Section 3. Section 4 discusses timing in simple control loops from control and scheduling perspectives. Section 5 gives an overview of flexible

and adaptive scheduling. A number of research issues in the topic of integrated control and scheduling designs are given in Section 6.

## 2. Real-Time Scheduling Theory

In hard real-time systems it is crucial that the timing requirements always are met. Hence, it is necessary to perform an off-line analysis that guarantees that there are no cases in which deadlines are missed. In scheduling theory we assume that we have events that occur and require computations. Associated with an event is a task that executes a piece of code in response to the event. The events could be periodic, sporadic, or aperiodic. A sporadic event is non-periodic but has a minimum inter-arrival time. An aperiodic event has an unbounded arrival frequency. Each task has a required computation time, in the sequel denoted $C$. This is the worst-case CPU execution time (WCET) it takes to execute the task, in the absence of other tasks. Each task also has an associated deadline, denoted $D$. This is an upper bound on the allowed time taken to execute task, in the presence of all the other tasks on the CPU.

Here we will primarily consider scheduling of CPU time for periodic tasks. Two main alternatives exist: *static cyclic executive scheduling* and *priority-based scheduling*. Static cyclic executive scheduling is an off-line approach that uses optimization-based algorithms to generate an execution table or calendar. The execution table contains a table of the order in which the different tasks should execute and for how long they should execute. The run-time part of the scheduling is extremely simple. The drawback that makes cyclic executive scheduling unsuitable for integrated control and scheduling is its static nature. It does not support on-line admission of new tasks, and dynamic modifications of task parameters. Hence, in this report we will focus on dynamic approaches to scheduling.

In 1973, Liu and Layland proposed in their seminal paper [Liu and Layland, 1973] two optimal priority-based scheduling algorithms, *earliest deadline first scheduling* (EDF) and *rate-monotonic (RM) scheduling*. EDF is based on the principle that it is the task with the shortest remaining time to its deadline that should run. The approach is dynamic in the sense that the decision of which task to run is made

at run-time. The deadline can also be viewed as a dynamic priority, in contrast to the RM case where the priority is fixed. The latter is the reason why rate-monotonic scheduling also is referred to as fixed priority scheduling.

Formal analysis methods are available both for RM and EDF scheduling. In the simplest case the following assumptions are made:

- only periodic tasks exist,
- each task $i$ has a period $T_i$,
- each task has a worst case execution time $C_i$,
- each task has a deadline $D_i$,
- the deadline for each task is equal to the task period ($D_i = T_i$),
- there is no interprocess communication, and
- the real-time kernel is "ideal" (context switching and clock interrupt handling takes zero time).

With these assumptions the following necessary and sufficient condition holds for EDF:

THEOREM 1—EDF SCHEDULING
If the utilization $U$ of the system is not more than 100% then all deadlines will be met.

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

□

The utilization $U$ determines the CPU load. The main advantage with EDF scheduling is that the processor can be fully utilized and still all deadlines can be met. More complex analysis exists that loosens some of the assumptions above.

Rate-monotonic (RM) assignment is a scheme for assigning priorities to tasks that guarantees that timing requirements are met when preemptive fixed priority scheduling is used. The scheme is based on the simple policy that priorities are set monotonically with task rate,

i.e., a task with a shorter period is assigned a higher priority. With essentially the same assumptions as in the EDF case, a sufficient schedulability condition for RM scheduling was derived in [Liu and Layland, 1973].

THEOREM 2—RM SCHEDULING
For a system with $n$ tasks, all tasks will meet their deadlines if the total utilization of the system is below a certain bound.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

□

As $n \to \infty$, the utilization bound approaches 0.693. This has led to the simple rule-of-thumb that says that

> "If the CPU utilization is less than 69%, then all deadlines are met".

Since 1973 the analysis has evolved and many of the restrictive assumptions have been relaxed [Audsley *et al.*, 1995; Sha *et al.*, 1994]. In 1986 a sufficient and necessary condition was derived [Joseph and Pandya, 1986]. The condition is based on the notion of worst-case response time, $R_i$, for a task $i$, i.e., the maximum time it can take to execute the task. The response time of a task is given by the recursive equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{1}$$

where $hp(i)$ is the set of tasks with higher priority than task $i$ and $\lceil x \rceil$ is the ceiling function. The task set is schedulable if and only if $R_i \leq D_i$ for all tasks $i$.

The rate-monotonic priority scheme is not very good when $D_i \ll T_i$. An infrequent but urgent task will be given a very low priority. In this case the *deadline-monotonic priority scheme* is better suited. Here it is the task deadline that decides the priority rather than the period. A task with a short deadline gets high priority. This policy has been proved optimal when $D \leq T$ in the sense that if the system is

unschedulable with the deadline-monotonic priority ordering, then it is unschedulable also with *all* other orderings [Leung and Whitehead, 1982]. Equation 1 holds also for the deadline-monotonic case.

During the last decade the rate and deadline monotonic analysis have been extended in various directions [Klein *et al.*, 1993]. It has been extended to cover situations where we have processes that communicate using, e.g., a monitor or using shared data protected by a mutual exclusion semaphore. To do this it is necessary to have an upper bound on how long a high-priority process may be blocked by a low-priority process due to interprocess communication. The priority ceiling protocol for mutual exclusion synchronization gives such a worst case bound [Sha *et al.*, 1990]. The analysis has also been extended to cover release jitter, i.e., the difference between the earliest and latest release of a task relative to the invocation of the task (the arrival of the event associated with the task), nonzero context-switching times, and clock interrupts.

The analysis behind the schedulability conditions in EDF and RM is based on the notion of the *critical instant*. This is the situation when all tasks arrive simultaneously. If the task set is schedulable for this worst case it will be schedulable also for all other cases. In many cases this assumption is unnecessarily restrictive. Tasks may have precedence constraints that make it impossible for them to arrive at the same time. Alternatively, for independent tasks it is sometimes possible to shift them in time, i.e., introduce task offsets, to avoid the simultaneous arrival. If simultaneous arrivals can be avoided, the schedulability of the task set increases [Audsley *et al.*, 1993b]. Schedulability analysis, both for the case of static and dynamic task offsets, is presented in [Gutierrez and Harbour, 1998]. A number of alternative scheduling models based on serialization of task executions in different ways have been suggested recently. These include the multi-frame model, [Mok and Chen, 1997; Baruah *et al.*, 1999], the generalized multi-frame model, [Baruah *et al.*, 1999b], the recurrent task model, [Baruah, 1998a], and the serially executed subtask model, [Gonzalez Härbour *et al.*, 1994].

### Aperiodic Scheduling

Scheduling of soft aperiodic tasks in combination with hard periodic tasks is a large area where a lot of research has been performed. The simplest approach is to transform an aperiodic task to a periodic task

using polling. This may, however, increase processor utilization unnecessarily. Another approach is to use a special server for aperiodic events [Lehoczky *et al.*, 1987]. The main idea of the server approach is to have a special task, the server (usually at high priority), for scheduling the pending aperiodic work. The server has time tickets that can be used to schedule and execute the aperiodic tasks. If there is aperiodic work pending and the server has unused tickets, the aperiodic tasks execute until they finish or the available tickets are exhausted. Several servers have been proposed. The *priority-exchange server* and the *deferrable server* were proposed in [Lehoczky *et al.*, 1987]. The *sporadic server* was introduced in [Sprunt *et al.*, 1989]. The main difference between the servers concerns the way the capacity of the server is replenished and the maximum capacity of the server. In [Bernat and Burns, 1999] exact schedulability tests are presented for the sporadic and the deferred servers. It is also claimed that the deferred server is superior, since it has the same performance as the sporadic server and is easier to implement. The idea behind the *slack stealer* proposed in [Lehoczky and Ramos-Thuel, 1992] is to steal all the possible processing time from the periodic tasks, without causing their deadlines to be missed. The approach has a high overhead, but provides an optimal lower bound on the response times of aperiodic tasks. The method has also been extended to cover hard aperiodic tasks. The above servers have been developed for the fixed-priority case. Similar techniques also exist for the dynamic-priority case (EDF), see, e.g., [Spuri and Buttazzo, 1996].

The idea of stealing time from hard periodic tasks is also used in *dual priority scheduling* [Davis and Wellings, 1995]. The priority range is divided into three bands: upper, middle, and lower. Hard tasks are assigned two priorities, one in the upper range and one in the lower range. At run-time, other tasks, e.g., soft aperiodic tasks, are assigned priorities in the middle band. The hard tasks are assigned their lower priorities upon release. A fixed time interval after their release they are promoted to their upper priority. During the initial phase of a task, the soft tasks will have higher priority, and thus execute. The net effect of the approach is that the execution of the hard tasks is shifted in such a way that the end of the task execution is always close to the response time of the task.

# 3. Sampled-Data Control Theory

A computer-based control system can be designed in two different ways:

1. Discrete-time design
2. Discretization of a continuous-time design

In both cases the interface to the process consists of AD and DA converters. The AD converter acts as a sampler that returns a snapshot value of a continuous time signal, and the DA converter acts as a hold circuit that takes a discrete-time signal and converts it into a continuous-time signal. Normally zero-order hold is used, in which case the resulting continuous-time signal is piecewise constant between the DA conversions. In certain situations it is advantageous to instead use first-order hold, where the continuous-time signal is piecewise linear. In the first-order hold case, the DA conversion is implemented as a high-frequency periodic process, unless special DA hardware is used.

## Discrete-Time Design

The basic idea behind discrete or sampled control theory is to only consider the system through the values of the system inputs and outputs at the sampling instants, i.e., from the point of view of the computer according to Fig. 2.

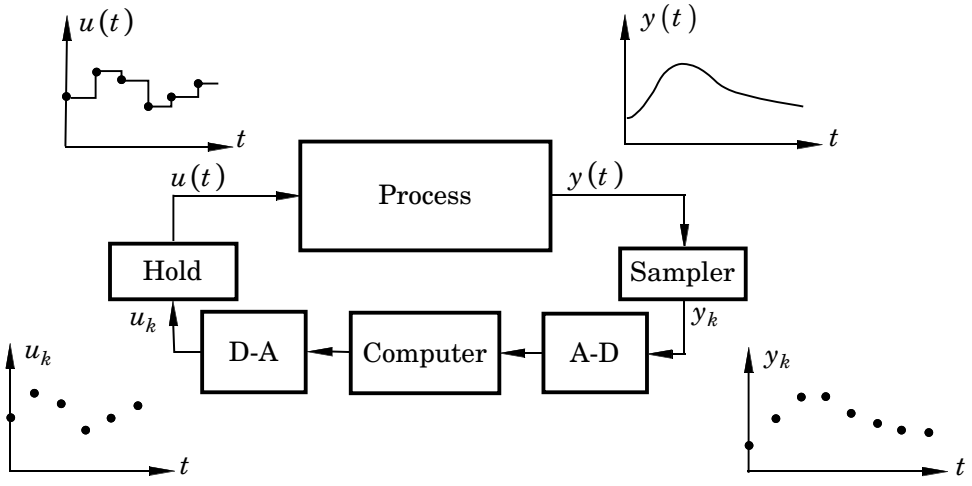



**Figure 2.** Sampled control loop.

In order to do this, a sampled version of the continuous system model is derived. This is done by letting the process inputs be piecewise constant signals and then solving the system equation by calculating step responses as in the following. Given the continuous time system description

$$\begin{aligned}\frac{dx}{dt} &= Ax(t) + Bu(t)\\ y(t) &= Cx(t) + Du(t),\end{aligned}$$

the solution to the system equation is given by

$$\begin{aligned}x(t) &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}Bu(s')\,ds'\\ &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}\,ds'\,Bu(t_k) \quad (u \text{ piecewise constant})\\ &= e^{A(t-t_k)}x(t_k) + \int_{0}^{t-t_k} e^{As}\,ds\,Bu(t_k) \quad (\text{variable change})\\ &= \Phi(t,t_k)x(t_k) + \Gamma(t,t_k)u(t_k)\end{aligned}$$

From this the values at $t = t_{k+1}$ are given by

$$\begin{aligned}x(t_{k+1}) &= \Phi(t_{k+1},t_k)x(t_k) + \Gamma(t_{k+1},t_k)u(t_k)\\ y(t_k) &= Cx(t_k) + Du(t_k)\end{aligned}$$

where

$$\begin{aligned}\Phi(t_{k+1},t_k) &= e^{A(t_{k+1}-t_k)}\\ \Gamma(t_{k+1},t_k) &= \int_{0}^{t_{k+1}-t_k} e^{As}ds\ B\end{aligned}$$

The expression above is valid both for periodic and aperiodic sampling. However, normally periodic sampling, i.e., $t_k = k \cdot h$, is assumed. This leads to the well-known discrete-time system description.

$$\begin{aligned}x(kh+h) &= \Phi x(kh) + \Gamma u(kh)\\ y(kh) &= Cx(kh) + Du(kh)\end{aligned}$$

where

$$\Phi = e^{Ah}$$

$$\Gamma = \int_0^h e^{As}\, ds\, B$$

The resulting system description is time-invariant and only describes the system at the sampling instants. Using similar techniques it is possible to also sample systems with time delays, both in the simple case when the time delay is a multiple of the sampling interval and in the case when the time delay is a fraction of the sampling interval.

A wide range of discrete-time controller design methods can then be applied, e.g. pole placement design, linear quadratic design, or model predictive control. The sampling intervals for discrete-time control designs are normally based on the desired speed of the closed loop system. A common rule-of-thumb is that one should sample 4 to 10 times per rise time $T_r$ of the closed loop system.

$$N_r = \frac{T_r}{h} \approx 4 \text{ to } 10$$

This gives relatively long sampling intervals, compared to what is used when discretization-based design is used. The long sampling interval also means that it may take long time before, e.g., load disturbances are detected by the controller. The reason for this is that the disturbances are not synchronized with the sampling.

### Discretization of Continuous-Time Design

The idea behind this design philosophy is to perform the design in the continuous time domain, and then approximate this design by a computer-based controller through fast sampling. Using this approach it is not necessary to employ any special sampled control design theory. The price that one pays for this is higher requirements on fast sampling.

Assume that a controller has been designed in continuous time and that this controller is expressed on input-output form, i.e., on Laplace transform form $G(s)$. The goal now is to approximate this design in such a way that A/D + Algorithm + D/A $\approx G(s)$ according to Fig.

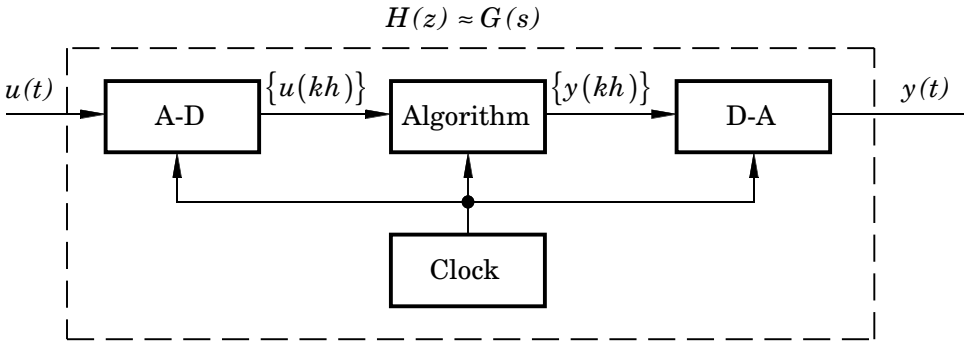



**Figure 3.** Approximation of continuous-time design.

3 This can be done in several ways. The most straightforward way is to use a simple Euler forward or backward approximation. In forward approximation a derivative is replaced by its forward approximation, i.e,

$$\frac{dx(t)}{dt} \approx \frac{x(t+h) - x(t)}{h}$$

This is equivalent to replacing the Laplace operator $s$ with $(z-1)/h$ in $G(s)$, where $z$ denotes the z-operator (can be interpreted as a shift operator). In the backward approximation the derivative is instead replaced by

$$\frac{dx(t)}{dt} \approx \frac{x(t) - x(t-h)}{h}.$$

This is equivalent to replacing $s$ with $(z-1)/zh$.

Several rules-of-thumbs exist for choosing the sampling interval for discretization based designs. A simple rule is the faster the better up to a certain limit when the limited word-length of the computer becomes a problem. One example of a rule-of-thumb is

$$h\omega_c \approx 0.15 \text{ to } 0.5$$

where $\omega_c$ is the cross-over frequency of the continuous-time system (the frequency where the gain is 1). This gives substantially smaller sampling intervals than for discrete-time design. However it also means that the resulting controller is less sensitive to relative variations in the sampling interval. A variation of 100% in the sampling interval for a discretization based design may be fairly benign whereas a similar relative variation in the sampling interval for a discrete-time design would in many cases lead to an unstable system.

# 4. Timing in Simple Control Loops

A control loop consists of three main parts: data collection, control algorithm computation, and output transmission. In the simplest case the data collection and output transmission consist of calls to an external I/O interface, e.g. AD and DA converters or a field-bus interface. In a more complex setting the input data may be received from other computational blocks, e.g., noise filters, and the output signal may be sent to other computational blocks, e.g., other control loops in the case of set-point control. The complexity of the control algorithm may range from a few lines of code implementing, e.g., a PID (Proportional-Integral-Derivative) control algorithm to the iterative solution of a quadratic optimization problem in the case of MPC (Model Predictive Control). In most cases the control is executed periodically with a constant period, or sampling interval, $T$ (often denoted $h$), that is determined by the dynamics of the process that is controlled and the requirements on the closed loop performance.

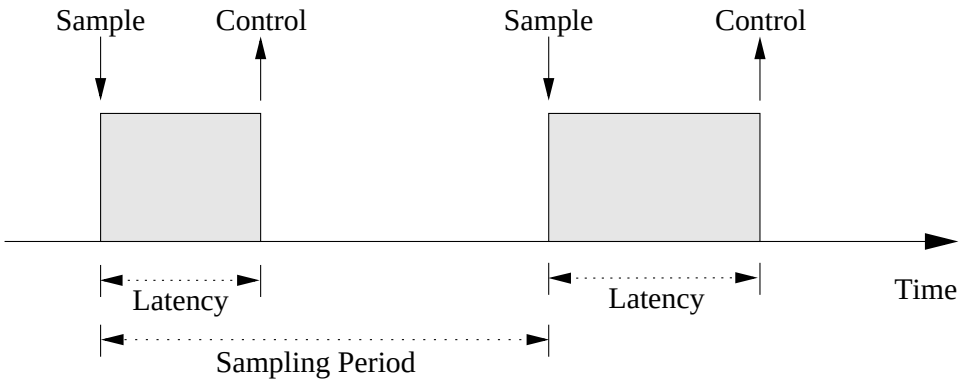



**Figure 4.** Basic timing constraints of a control loop.

The two basic timing constraints of a control loop are shown in Fig. 4. The first is the period which should be constant, i.e., without jitter. The second constraint involves the input-output latency, also known as the *control delay* or the *computational delay*. This should be as small as possible, and also without jitter. An overview of control loop timing constraints is given in [Törngren, 1998].

## Control Loop Timing and Control Theory

From a control perspective, sampling jitter and latency jitter can be interpreted as disturbances acting on the control system. The input-output latency decreases the stability margin and limits the performance of the system. If the jitter and the latency are small, they could be ignored. Otherwise, they should be accounted for in the control design.

***Sampling Jitter*** As a rule of thumb, relative variations of sampling intervals that are smaller than ten percent of the nominal sampling interval need not be compensated for. The sensitivity to sampling period variations is larger with systems that use slow sampling and for systems with small phase margins; for such systems small variations in sampling period can lead to instability.

There are several possible compensation methods, ranging from simple ad-hoc techniques to quite advanced techniques requiring extensive (off-line or on-line) calculations. The choice of compensation method depends on the range of sampling period variations, how often the changes occur and how sensitive the system is to variations. The cost of implementation is also an important factor. A related issue is intentional changes in sampling period. Most compensation schemes assume that the sampling period variations are unintentional and unknown in advance. This is the case when the variations are due to clock inaccuracy, overload or computer failure.

If the changes of sampling period are rare, or if the sampling period varies slowly, then the problem can be solved using gain-scheduling. This means that several sets of controller parameters are pre-calculated and stored in a table with the sampling rate as input parameter. When the sampling rate changes a new set of controller parameters are used. Small transients can occur when the controller parameters change and special care must be taken to minimize these mode bumps. However, the changes in sampling period often occur continuously and should not be treated this way. The simplest compensation methods are ad-hoc, but seem to work quite well. One possibility is to modify the approximation methods mentioned in Section 3. For example, the

(backward) approximation method becomes

$$\frac{dx(t)}{dt} \approx \frac{x(t_{k+1}) - x(t_k)}{h_k}$$

where $h_k = t_{k+1} - t_k$ is time-varying. This works, e.g., fine for the I- and D- parts in a PID controller. The idea must be modified for higher order controllers, see [Wittenmark and Åström, 1980], [Albertos and Salt, 1990]. If the nominal controller instead is a linear feedback with Kalman filter then a solution may be to use a time-varying Kalman filter that gives state estimates at the actual time instances (here illustrated with the Kalman filter without direct term):

$$\begin{aligned} u(t_k) &= -L\hat{x}(t_k) \\ \hat{x}(t_{k+1}) &= \Phi(h_k)\hat{x}(t_k) + \Gamma(h_k)u(t_k) + Ke(t_k) \end{aligned}$$

where $\Phi(h_k)$ and $\Gamma(h_k)$ can be pre-calculated in one-dimensional tables. More involved schemes can also be used, where also the feedback gains $L$ and $K$ depend on $h_k = t_{k+1} - t_k$.

EXAMPLE 1—SAMPLING JITTER

Consider PD control of a DC servo, see Fig. 5. The goal of the control is to make the servo position, $y(t)$, follow the reference position, $r(t)$, as closely as possible. Every $h$ seconds, the controller should sample the position of the servo and calculate a new control signal $u(t)$.

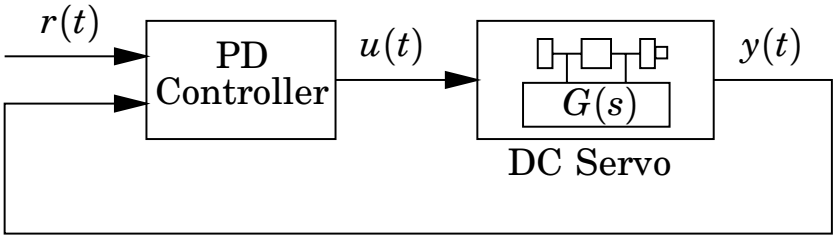



**Figure 5.** A DC Servo is being controlled by a PD controller.

Let the servo be described by the continuous-time transfer function

$$G(s) = \frac{1000}{s(s+1)}.$$

A good discrete-time implementation of the PD controller, which includes filtering of the derivative part, is

$$\begin{aligned} P(t) &= K(r(t) - y(t)), \\ D(t) &= a_d D(t-h) + b_d(y(t-h) - y(t)), \\ u(t) &= P(t) + D(t), \end{aligned}$$

where $a_d = \frac{T_d}{Nh+T_d} D(t-h)$ and $b_d = \frac{NKT_d}{Nh+T_d}$.

A nominal sampling period of $h = 10$ ms is chosen, and the PD controller is tuned to give a fast and well-damped response to set-point changes. The resulting parameters are $K = 1$, $T_d = 0.04$, and $N = 30$. The parameters $a_d$ and $b_d$ are normally precalculated, assuming that the sampling interval is constant.

A first simulation of the closed-loop system, where there is no jitter in the sampling interval, is shown in Fig. 6. The controller behaves as expected, and the performance is good.

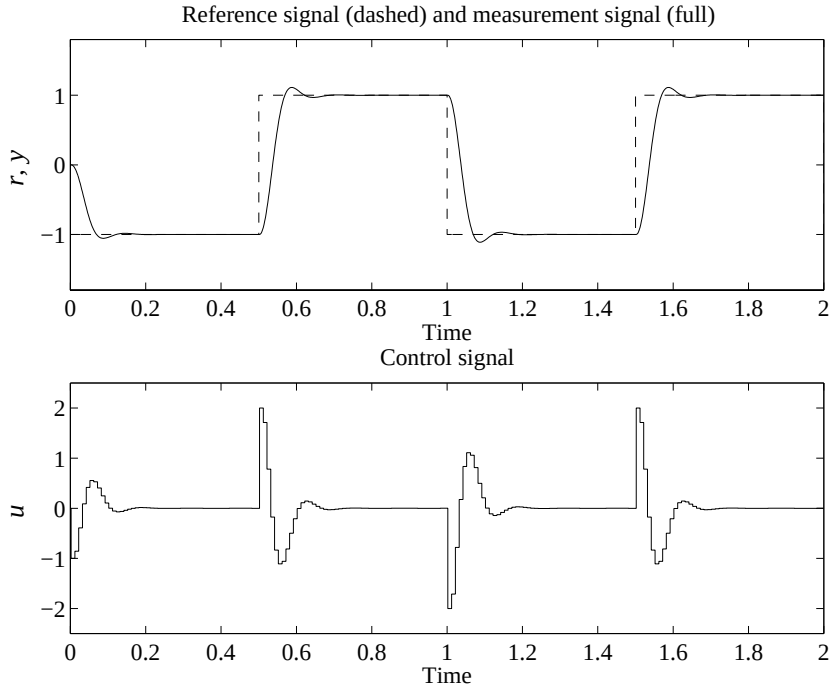



**Figure 6.** When no sampling jitter is present, the control performance is good.

A second simulation, where the actual sampling interval varies randomly between $h_{min} = 2$ ms and $h_{max} = 18$ ms, is shown Fig. 7. The

discrepancy between the nominal and the actual sampling interval causes the controller to repeatedly take either too small or too large actions. The resulting performance is quite poor. This is especially visible in the control signal.

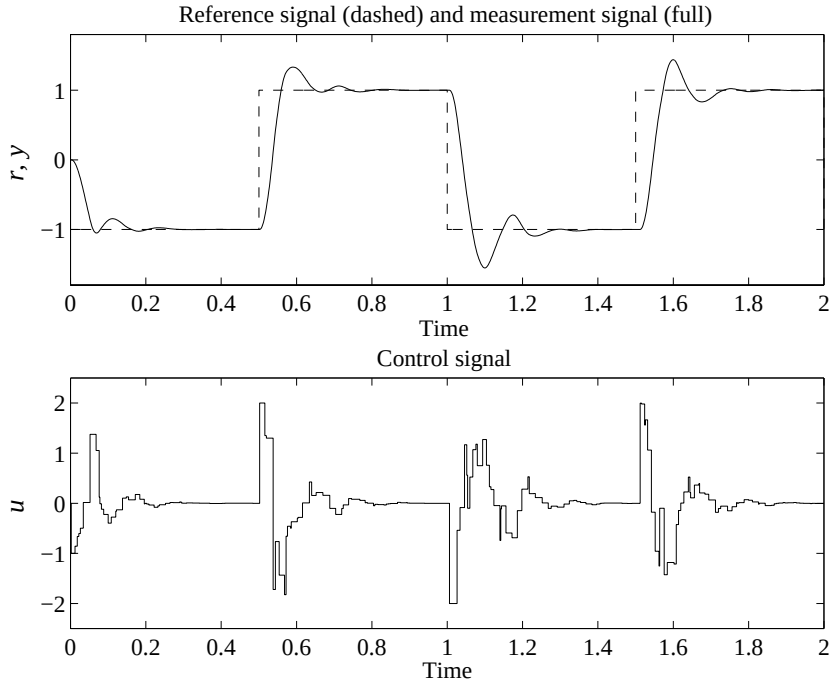



**Figure 7.** Sampling jitter causes the control performance to degrade.

Finally, in a third simulation, the controller is redesigned to compensate for the varying sampling interval. This is done by measuring the actual sampling interval and recalculating the controller parameters $a_d$ and $b_d$ at each sample. Fig. 8 shows that this version of the controller handles the sampling jitter well. The performance is almost as good as in Fig. 6 where there was no jitter at all.

□

***Input-Output Latency*** Several design approaches are possible for the input-output latency. The simplest approach is to implement the system in such a way that the delay is minimized and then ignore it in the control design. The standard way to achieve this is to separate the algorithm calculations in two parts: *Calculate Output* and *Update State*. Calculate Output contains only the parts of the algorithm that

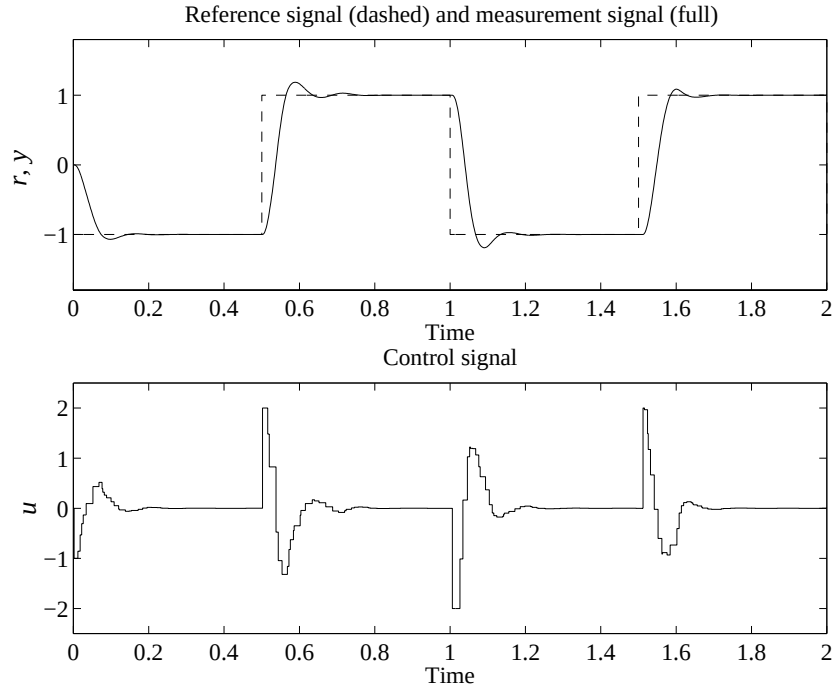



**Figure 8.** When compensating for the sampling jitter, the control performance is good again.

make use of the current sample information. Update State contains the update of the controller states and the pre-calculations that are needed to minimize the execution time of Calculate Output. Update State can therefore be executed after the output generation, hence, reducing the computational delay.

A second approach is to try to ensure that the delay is constant, i.e., jitter free, and take this delay into account in the controller design. One way of doing this is to wait with the output transmission until the beginning of the next sample. In this way the computational delay becomes close to the sampling period. If the controller is designed with discrete (sampled) control theory it is especially easy to compensate for this delay. However it is also relatively easy to compensate for delays that are shorter than the sampling period. To sample a system composed of a control delay and a continuous-time state-space system, see Figure 9, one can follow the calculation in the beginning of Section 3,

replacing the system equations with

$$\frac{dx}{dt} = Ax(t) + Bu(t-\tau)$$
$$y(t) = Cx(t) + Du(t)$$

The sampled system will now be on the form (assuming that the

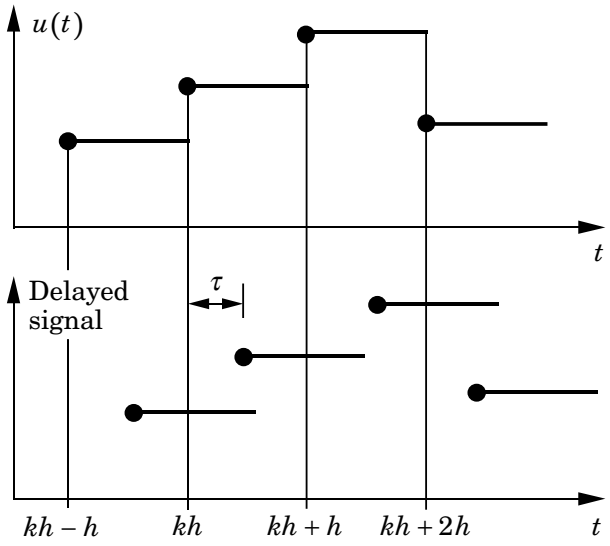



**Figure 9.** Relationship among $u(t)$ and the delayed signal $u(t-\tau)$, and the sampling instances.

control delay $\tau$ is less than the sampling interval $h$)

$$x(kh+h) = \Phi x(kh) + \Gamma_0 u(kh) + \Gamma_1 u(kh-h),$$

where

$$\Phi = e^{Ah}$$
$$\Gamma_0 = \int_0^{h-\tau} e^{As}\,ds\,B$$
$$\Gamma_1 = \int_{h-\tau}^{h} e^{As}\,ds\,B$$

A third approach is to explicitly design the controller to be robust against jitter in the computational delay, i.e., the delay is treated as a

parametric uncertainty. This approach is, however, substantially more complex than the first two. Many robust design methods can be used, such as $H_\infty$, Quantitative Feedback Theory (QFT) and $\mu$-design.

A fourth approach, finally, is based on the idea that the control algorithm in certain situations can actively, every sample, compensate for the computational delay or parts of it. This approach has been used to compensate for the computational delays obtained when a control loop is closed over a communication network [Nilsson, 1998]. The setup studied is shown in Fig. 10. The conclusion in this work is that very good performance can be obtained even under quite large variations of control delay. The case of joint variations in control delay $\tau$ and sampling period $h$ is treated in [Nilsson *et al.*, 1998].

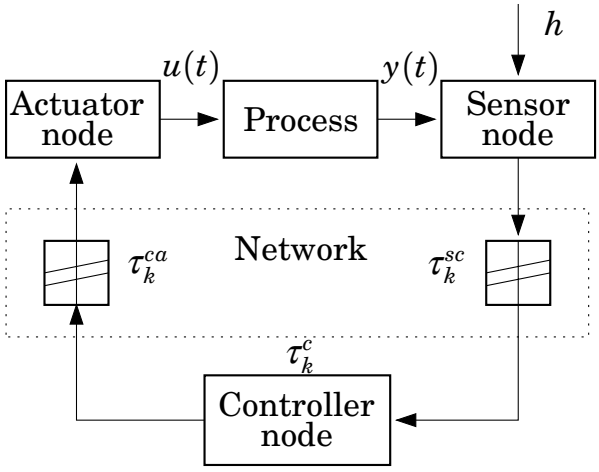



**Figure 10.** Distributed digital control system with communication delays, $\tau_k^{sc}$ and $\tau_k^{ca}$. The computational delay, $\tau_k^c$, is also indicated. The control delay equals $\tau_k^{sc} + \tau_k^c + \tau_k^{ca}$.

EXAMPLE 2—INPUT-OUTPUT LATENCY JITTER

Again, consider PD control of the DC servo from Example 1. (The sampling jitter is assumed to be zero.) A delay is now introduced from the sampling to the output action. First, the input-output latency is constant and equal to 0.007 ms. The controller is retuned assuming this delay, and the resulting parameters are $K = 1$, $T_d = 0.045$, and $N = 100$. The simulation result is shown in Fig. 11. It is not possible to get as good performance as in the previous example due to the time delay.

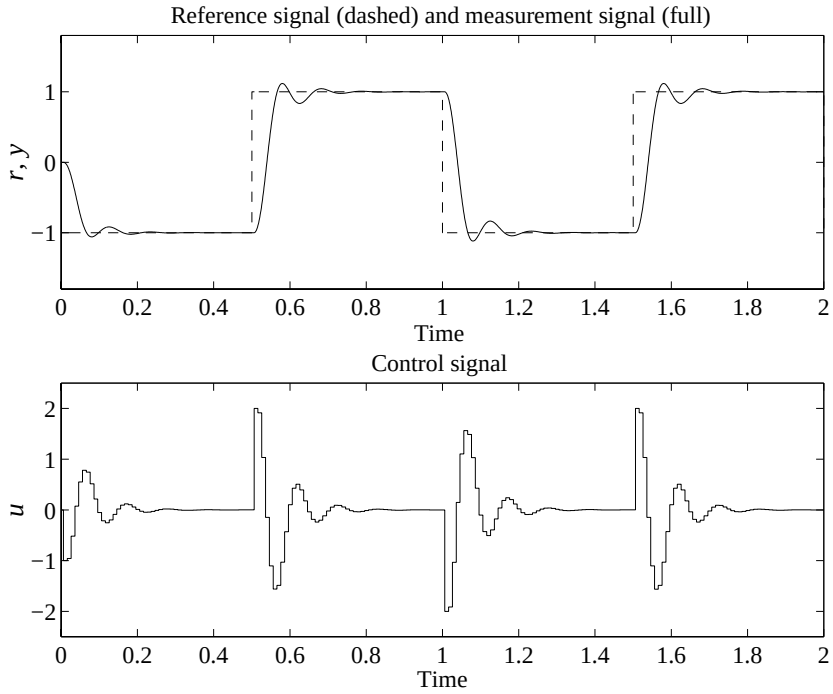



**Figure 11.** The controller can be tuned to handle a constant input-output latency.

Next, the input-output latency is randomly varying between 0.002 ms and 0.012 ms. Even though the average delay is the same as before, the performance is now worse, as shown in Fig. 12.

□

## Control Loop Timing and Scheduling

Scheduling theory can be used to analyze the time variations and delays in control loops when they are implemented as real-time tasks. Understanding the control requirements, the implementation could be made such that the resulting delay and the jitter are small.

The following example shows that a simple-minded implementation of control loops can introduce a lot of jitter and delays:

EXAMPLE 3

Three control loops with different sampling periods are implemented in a priority-preemptive real-time OS. The task code for each control loop looks like this (this would be a good implementation in a single-task system):

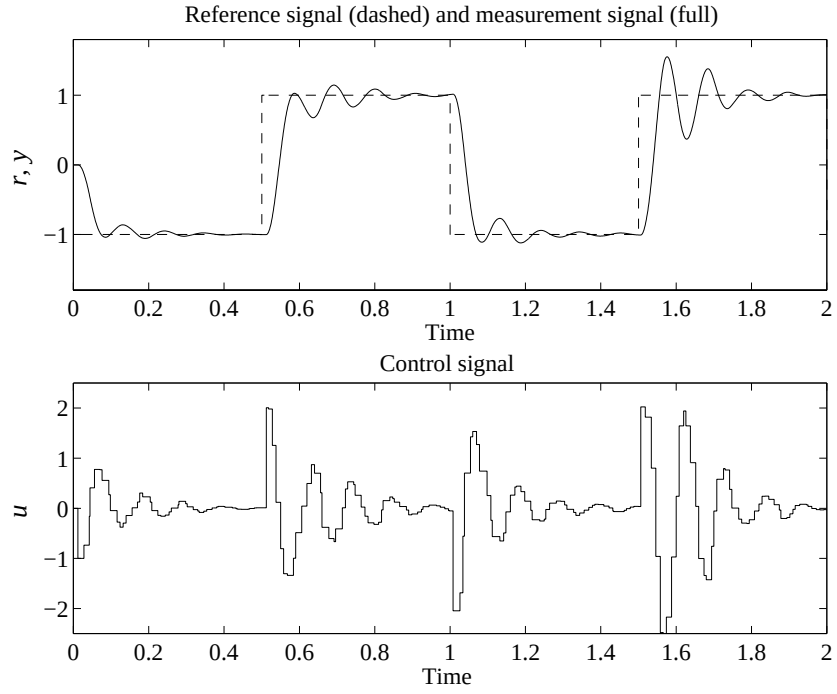



**Figure 12.** Variable input-output latency degrades the control performance.

```
t = CurrentTime;
LOOP
  AD-Conversion;
  ControlAlgorithm;
  DA-Conversion;
  t := t + h;
  WaitUntil(t);
END
```

Assume that the execution time is 2 ms for all three tasks, and that the sampling periods are $T_1 = 12$ ms, $T_2 = 8$ ms, and $T_3 = 5$ ms. Fixed priorities are assigned to the tasks according to the rate-monotonic theory. Figure 2 shows the execution graph of the three control tasks when released at time zero. Task 3 has the shortest period, thus the highest priority, and executes with perfect periodicity. Tasks 1 and 2, on the other hand, are frequently preempted. The preemption causes variations in both the sampling period and in the input-output latency. □

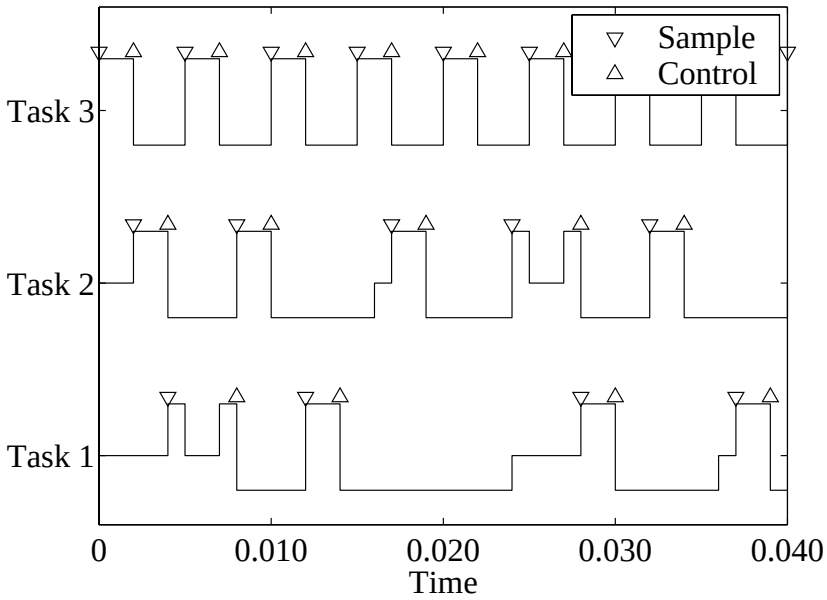



**Figure 13.** The activation graph (high=running, medium=preempted, low=sleeping) of the three control tasks in Example 3

.

***Jitter*** A simple way of implementing jitter-free sampling is to set up periodic timer interrupts and sample the process in the interrupt handler. This gives minimal jitter since interrupt handlers execute at higher priorities than the tasks, and are, hence, not affected by task scheduling. If an input-output latency of one sample is acceptable, the same technique can be used to achieve jitter-free outputs. The sending of the control signal is then also executed the interrupt handler, before reading the new sample. If such solutions are not supported by the real-time OS, then separate, high-priority application tasks for sampling and actuation may be an alternative, see [Locke, 1992]. This may, however, incur larger overheads for context switches. Under fixed-priority scheduling, the extra sampling and actuation tasks can be included in the schedulability analysis by the use of offsets [Audsley *et al.*, 1993b].

There are several jitter minimization solutions based on task scheduling. The work in [Audsley *et al.*, 1993a; Baruah *et al.*, 1997] addresses the problem of accommodating input jitter, i.e., scheduling systems of periodic tasks in which the ready-times of jobs cannot be predicted exactly a priori, but are subject to timing uncertainties at

run-time. In [Lin and Herkert, 1996] a distance-constrained task model is studied in which the difference between any two consecutive end-times of the same task is required to be bounded by a specified value. This approach attempts to minimize the output jitter. Minimization of output jitter is the topic of [Baruah *et al.*, 1999a], where two scheduling algorithms are proposed. In [Baruah and Gorinsky, 1999] an attempt is made to identify some of the properties that any jitter-minimization scheme should satisfy. The scheme is not allowed to buffer a job that is ready to execute. The scheme may not insert idle time in the execution of a job. The scheme is not allowed to buffer a job that is completed. The scheme should not require too many preemptions and the on-line scheduling overhead should be small. Within the context of these conditions two jitter-minimizing scheduling algorithms are proposed.

***Input-Output Latency*** As mentioned, the computional delay of a controller can often be reduced by splitting the control algorithm into two parts: Calculate Output, which is executed between the AD and the DA conversion, and Update State, which can be executed after the DA conversion. This idea can be further exploited in the scheduling design, by treating the two parts as separate tasks, or sub-tasks of the same task. This possibility was observed in [Gerber and Hong, 1993], and later [Gerber and Hong, 1997], where control tasks are "sliced" for the sake of increased fixed-priority schedulability.

Fixed-priority schedulability analysis for tasks where the sub-tasks have different priorities is given in [Gonzalez Härbour *et al.*, 1994]. In particular, it is noted that the deadline-monotonic priority assignment is optimal if the sub-tasks have non-increasing priorities. The simplified analysis could be applied to control tasks, where it seems reasonable that Update State does not execute at a higher priority then Calculate Output.

In [Cervin, 1999], sub-task scheduling of control tasks is explored from a control performance perspective. An algorithm is given that attempts to minimize the computational delay for a set of control tasks. The algorithm is based on the deadline-monotonic priority assignment and the calculation of response times.

# 5. Flexible and Adaptive Scheduling

In this section we will discuss different ways in which task scheduling can be made flexible and adaptive to changes and uncertainties in task parameters. A key issue for this is the ability to dynamically adjust task parameters. Reasons for the adjustments could for example be to improve the performance in overload situations or to dynamically optimize control performance. Examples of task parameters that could be modified are periods and deadlines. One could also allow the execution time for a task to be varied. In order for this to be realistic, the controllers must support dynamically changing execution times. Changes in the task period and in the execution time both have the effect of changing the utilization that the task requires.

## Overload Scheduling

An important issue in integrated control and scheduling is the relaxation of the requirement on a known worst-case task execution time. One possibility to approach this is to use on-line measurements of actual execution times. Another possibility is to simply treat the actual execution times that are longer than the worst-case bound as overload conditions and then try to use some of the scheduling techniques that have been developed for this, for example robust scheduling.

In the rate monotonic case transient overloads can easily be analyzed off-line. In [Sha and Goodenough, 1990] the following method was suggested. Each task has a nominal execution time and a worst case execution time. It is assumed that the task set is schedulable under nominal execution times. The tasks are divided into two groups: critical tasks that cannot miss any deadlines, and non-critical tasks for which deadline misses are acceptable. A check is made to find the set of all critical tasks that would miss their deadlines under worst case execution times. If this set is empty, then nothing need to be done. If the set is nonempty then period transformation is applied to all the tasks in the set. For example, a task with 10 msec execution time and a period of 100 msec is transformed into a task with 5 msec execution time and 50 msec period, where the input and output are only performed once every two periods. Period transformation can be done transparently to the application tasks using a sporadic server. After the transformation a new check is performed and, if needed, more

period transformations. This procedure is continued until no more critical tasks are found that would miss their deadlines under worst case execution times. The method works if the set of critical tasks is schedulable under worst case execution times.

The model used in overload scheduling often associates a value with the completion of a task within its deadline. For a hard task, the failure to meet the deadline is considered intolerable. For soft tasks, a positive value is associated with each task. If a soft task succeeds then the system gains its value. If a soft task fails, then the value gained by the system decreases. Sometime also the notion of a *firm task* is used. For a firm task there is no value for a task that has missed its deadline, but there is no catastrophe either. An overview of value-based scheduling is given in [Burns, 1998]. When a real-time system is overloaded, not all tasks can be completed by their deadlines. Unfortunately, in overload situations there is no optimal on-line algorithm that can maximize the cumulative value of a task set. Hence, scheduling must be made using *best-effort* algorithms. The objective is to complete the most important of the tasks by their deadline, and to avoid unwanted phenomena such as the so called *domino effect*. This happens when the first task that misses its deadline may cause all subsequent tasks to miss their deadlines. For example, dynamic-priority schemes such as EDF are especially prone to domino effects. In fixed-priority scheduling the user has more control over which tasks that will meet their deadlines also in the case of overloads.

It is normally assumed that when a task is released its value and deadline are given. The computation time may be known either precisely or within some range. The *value density* of a task is its value divided by the computation time. The *importance ratio* of a set of tasks is the ratio of the largest value density to the smallest value density. When the importance rate is 1, the task set has *uniform value density*, i.e. the value of a task equals its computation time. An on-line scheduler has a competitive factor, $r, 0 < r \leq 1$, if and only if it is guaranteed to achieve a cumulative value of at least $r$ times the cumulative value achievable for an off-line scheduler on any task set. The competitive multiplier is defined as $1/r$. In [Shasha and Koren, 1995], $D^{over}$, an optimal on-line scheduling algorithm is presented. It schedules to completion all tasks in non-overload periods and achieves at least $1/(1+\sqrt{k})^2$ of the value of an off-line scheduler, where $k$ is the

importance ratio. The method also relaxes the firm deadline assumption by giving a positive value to firm tasks even if they complete after their deadlines.

In [Buttazzo *et al.*, 1995], a comparative study is performed of four priority-assignment schemes, EDF, HVF (highest value first), HDF (highest value density first), and a mixed scheme where the priority is computed as a weighted sum of the value and the deadline. The four basic algorithms were all extended into two additional classes: a class of guaranteed algorithms, characterized by a task acceptance test, and a class of robust algorithms, characterized by a task rejection mechanism. Simulation experiments showed that the robust versions of the algorithms were the most flexible ones.

The transform-task method [Tia *et al.*, 1995] uses a threshold value to separate jobs guaranteed by rate-monotonic scheduling from those which would require additional work. The latter jobs are split into two parts. The first part is considered as a periodic job with a resource requirement equal to the threshold. The second part is considered to be a sporadic job and is scheduled via a sporadic server when the periodic part has completed.

It is difficult to improve overload performance with purely algorithmic techniques, what is needed is essentially more processing power. Several approaches have considered the use of more processing power in order to reduce the execution requirements of the tasks, e.g., [Baruah and Haritsa, 1997]. Another approach is to replicate the processor with several identical copies. In [Baruah, 1998b] a characterization is given of the relationship between overload performance and the number of processors needed.

### Quality-of-Service Resource Allocation

Much of the work on dynamic task adaptation during recent years is motivated by the requirements of multimedia applications. Activities such as voice sampling, image acquisition, sound generation, data compression, and video playing are performed periodically, but with less rigid timing requirements than those that can sometimes be found in closed-loop control systems. Missing a deadline may decrease the quality of service (QoS) but does not cause critical system faults. Depending on the requested QoS, tasks may adjust their attributes to accommodate the requirements of other concurrent activities.

On-line admission control has been used to guarantee predictability of services where request patterns are not known in advance. This concept has also been applied to resource reservation for dynamically arriving real-time tasks, e.g. in the Spring kernel [Stankovic and Ramamritham, 1991]. A main concern of this approach is predictability. Run time guarantees given to admitted tasks are never revoked, even if they result in rejecting subsequently arriving, more important requests competing for the same resources. In soft real-time systems, services are more concerned with maximizing overall utility, by serving the most important requests first, than guaranteeing reserved resources for individual requests. Priority-driven services can be categorized this way, and are supported in real-time kernels such as Mach [Tokuda *et al.*, 1990]. Under overload conditions, low priority tasks are denied service in favor of high-priority tasks. In the Rialto operating system [Jones and Leach, 1995], a resource planner attempts to dynamically maximize user-perceived utility of the entire system.

Q-RAM, a resource allocation scheme for satisfying multiple QoS dimensions in resource constrained environments was presented in [Rajkumar *et al.*, 1997]. Using the model, available system resources can be apportioned across multiple applications such that the net utility accrued to the end users of those applications could be maximized. In [Lee *et al.*, 1998], the mirror problem of apportioning multiple resources to satisfy a single QoS dimension is studied. In [Abdelzaher *et al.*, 1997] a QoS renegotiation scheme is proposed as a way to allow graceful degradation in cases of overload, failures or violation of pre-run-time violations. The mechanism permits clients to express, in their service requests, a spectrum of QoS levels they can accept from the provider and perceived utility of receiving service at each of these levels. Using this, the application designer, e.g., control engineer, is able to express acceptable tradeoffs in QoS and their relative cost/benefit. The approach is demonstrated on an automated flight-control system.

### Period Skipping

A simple task attribute adjustment is to skip an instantiation of a periodic task. This is equivalent to require that the task period should be doubled for this particular instantiation, or that the maximum allowed execution time should be zero. Scheduling of systems that allow skips is treated in [Koren and Shasha, 1995] and [Ramanathan, 1997].

The latter paper considers scheduling that guarantees that at least $k$ out of $n$ instantiations will execute. A slightly different motivation for skipping samples is presented in [Caccamo and Buttazzo, 1997]. Here the main objective is to use the obtained execution time to enhance the responsiveness of aperiodic tasks.

### Task Attribute Adjustments

In [Buttazzo *et al.*, 1998] an elastic task model for periodic tasks is presented. Each task is characterized by five parameters: computation time $C_i$, a nominal period $T_{i_0}$, a minimum period $T_{i_{min}}$, a maximum period $T_{i_{max}}$, and an elasticity coefficient $e_i \geq 0$. A task may change its period within its bounds. When this happens the periods of the other tasks are adjusted so that the overall system is kept schedulable. An analogy with a linear spring is used, where the utilization of a task is viewed as the length of a spring that has a given rigidity coefficient $(1/e_i)$ and length constraints. The elasticity coefficient is used to denote how easy or difficult it is to adjust the period of a given task (compress the string). A task with $e_i = 0$ can arbitrarily vary its period within its range, but it cannot be varied by the scheduler during load reconfiguration. The approach can be used under fixed or dynamic priority scheduling. In principal it is possible to modify the approach so that it also adjusts execution times.

Adjustment of task periods has also been suggested by others. For example, [Kuo and Mok, 1991] propose a load-scaling technique to gracefully degrade the workload of a system by adjusting the task periods. Tasks are assumed to be equally important and the objective is to minimize the number of fundamental frequencies to improve schedulability under static priority assignments. In [Nakajima and Tezuka, 1994] a system is presented that increases the period of a task whenever the deadline of the task is missed. In [Lee *et al.*, 1996] a number of policies to dynamically adjust task rates in overload conditions are presented. In [Nakajima, 1998] it is shown how a multimedia activity can adapt its requirements during transient overloads by scaling down its rate or its computational demands.

The MART scheduling algorithm [Kosugi *et al.*, 1994; Kosugi *et al.*, 1996; Kosugi and Moriai, 1997] also supports task-period adjustments. MART has also been extended to handle task execution time adjustments. The system handles changes in both the number of periodic

tasks and in the task timing attributes. Before accepting a change request the system analyzes the schedulability of all tasks. If needed it adjusts the period and/or execution time of the tasks to keep them schedulable with the rate monotonic algorithm. For the task execution time it is assumed that a minimum value exists in order for the task to guarantee a minimum level of service. For the task-period, neither minimum nor maximum are assumed to exist. The MART system is implemented on top of Real-Time Mach.

In [Shin and Meissner, 1999] the approach in [Seto *et al.*, 1996] for off-line control and scheduling co-design that will be surveyed in Section 5.2 is extended, making on-line use of the proposed method for processor utilization allocation. The approach allows task-period changes in multi-processor systems. A performance index for the control tasks is used to determine the value to the system of running a given task at a given period. The index is weighted for the task's importance to the overall system. The paper also discusses the issue of task reallocation from one processor to another, the need to consider the transient effects of task reallocations, and the question of determining a value for running a redundant shadow task as opposed to fast recovery. Two algorithms are given for task reallocation and period adjustments. An inverted pendulum control system is used as an example.

### Mode Changes

Mode changes for priority-based preemptive scheduling is an issue that has received some interest. In the basic model, the system consists of a number of tasks with task attributes. Depending on which modes the system and the tasks are in, the task attributes have different values. During a mode change, the system should switch the task attributes for a task and/or introduce or remove tasks in such a way that the overall system remains schedulable during and after the mode change.

A simple mode change protocol was suggested in [Sha *et al.*, 1989]. The protocol assumes that an on-line record of the total utilization is kept. A task may be deleted at any time, and its utilization may be reclaimed by a new task at the end of the old task's period. The new task is accepted if the resulting new task set is schedulable according to the rate-monotonic analysis. The locking of semaphores during the mode change (according to the priority ceiling protocol) is also dealt

with.

In [Tindell *et al.*, 1992], it was pointed out that the analysis of Sha *et al.* was faulty. Tasks may miss their deadlines during a mode change, even if the task set is schedulable both before and after the switch. The transient effects of a mode change can be analyzed by extending the deadline-monotonic framework. Formulas for the worst-case response times of old and new tasks across the mode change are given. New tasks may be given release offsets (relative to the mode change request) to prevent tasks from missing their deadlines. No hints are given as to how these offsets should be chosen.

The deadline-monotonic mode change analysis was both extended and modified in [Pedro and Burns, 1998]. The analysis can account for old tasks that are aborted instantly at the mode change request. Furthermore, *all* tasks present in the system after the mode change (i.e. unchanged, changed, and wholly now tasks) must be assigned release offsets relative to the mode change request. Since the offsets are constant (and assigned off-line), even unchanged tasks will experience unpredictable and unnecessary delay (jitter) during the mode change. Again, few clues are given as to how the offsets could be assigned.

It is interesting to note, that under EDF scheduling, the reasoning about the utilization from [Sha *et al.*, 1989] actually seems to hold. In [Buttazzo *et al.*, 1998], EDF scheduling of a set of tasks with deadlines equal to their periods is considered. It is shown that a task can decrease its periods at its next release, as long as the total utilization remains less than one. It is not known whether the computation time may be changed at the same time. A drawback of using EDF is that the case of deadlines less than periods is more difficult to analyze. Also, the computation of worst-case response times (if needed) is more complex than under fixed-priority scheduling. Another challenge is how to manage overloads, e.g., ensuring that the deadlines of critical tasks will not be missed.

### Feedback Scheduling

Viewing a computing system as a dynamical system or as a controller is an approach that has proved to be fruitful in many cases. For example, the step-length adjustment mechanism in numerical integration algorithms can be viewed as a PI-controller [Gustafsson, 1991]. This approach can also be adopted for real-time scheduling, i.e., it is possi-

ble to view the on-line scheduler as a controller. Important issues that then must be decided are what the right control signals, measurement signals, and set-points are, what the correct control structure should be, and which process model that may be used. The idea of using feedback in scheduling has to some extent been used previously in general purpose operating systems in the form of multi-level feedback queue scheduling [Kleinrock, 1970; Blevins and Ramamoorthy, 1976; Potier *et al.*, 1976]. However, this has mostly been done in an ad-hoc way.

So far relatively little has been done in the area of real-time feedback scheduling. In [Stankovic *et al.*, 1999] it is proposed to use a PID controller as an on-line scheduler under the notion of Feedback Control-EDF (FC-EDF). The measurement signal (the controlled variable) is the deadline miss ratio for the tasks and the control signal is the requested CPU utilization. Changes in the requested CPU utilization are effectuated by two mechanisms (actuators). An admission controller is used to control the flow of workload into the system and a service level controller is used to adjust the workload inside the system. The latter is done by changing between different versions of the tasks with different execution time demands. A more elaborate version of the same scheme is presented in [Lu *et al.*, 1999].

For multimedia applications, feedback based scheduling mechanisms that dynamically adjust the QoS level have been proposed in a few cases. In [Li and Nahrstedt, 1998] a general framework is proposed for controlling the application requests for system resources using the amount of allocated resources for feedback. It is shown that a PID controller can be used to bound tasks' resource usage in a stable and fair way. In [Abeni and Buttazzo, 1999a] task models suitable for multimedia applications are defined. Two of them, the Continuous Media (CM) model and the Event Driven (ED) task model, use PI control feedback to adjust the reserved fraction of CPU bandwidth.

Using a controller approach of the above kind it is important to be able to measure the appropriate signals on-line, for example to be able to measure the deadline miss ratio, the CPU utilization, or the task execution times. On-line measurements of these entities are not so often discussed.

### Statistical Scheduling

A slightly different approach to flexibility and uncertainty handling in scheduling is obtained by using statistical scheduling. Using this approach things like, e.g., task execution times are modeled by stochastic variables with a given distribution rather than constant variables. Instead of generating hard guarantees that the tasks will meet their deadlines or that dynamically generated tasks will be admitted, the guarantees are probabilistic.

Statistical approaches to scheduling are relatively easy to find. In [Tia *et al.*, 1995] an analysis was given for the probability that a sporadic job would meet its deadline when using the transform-task method. The Statistical Rate Monotonic Scheduling (SRMS) [Atlas and Bestavros, 1998] is a generalization of the classical RM scheduling and the semi-periodic task model of [Tia *et al.*, 1995]. A task is modeled by three attributes: the period, the probability density function for the task's periodic resource utilization requirement, and the task's requested QoS. The SRMS algorithm consists of two parts: a job admission controller and a scheduler. The scheduler is a simple preemptive fixed priority scheduler. The job admission controller is responsible for maintaining the QoS requirements of the tasks through admit/reject and priority assignment decisions. Using SRMS the probability that an arbitrary job is admitted can be calculated.

In the model proposed in [Abeni and Buttazzo, 1999b] each task is described by a pair of probability density functions: $U_i(c)$, which decides the probability that the execution time is $c$ and $V_i(t)$ which decides the probability that the minimum inter-arrival time of the task is $t$. The method guarantees that *probabilistic deadlines*, $\delta$, are respected with a given probability. A dynamic priority scheduler based on EDF is used. Each soft task is handled by a dedicated constant bandwidth server. The server assigns each job an initial deadline. The assigned deadline is postponed each time the task requests more than the reserved bandwidth.

### Imprecise Calculations

The possibility to adjust the allowed maximum execution time for a task necessitates an approach for handling tasks with imprecise execution times, particularly the case when the tasks can be described as

"any-time algorithms", i.e., algorithms that always generate a result (provide some QoS) but where the quality of the result (the QoS level) increases with the execution time of the algorithm.

The group of Liu has worked on scheduling of imprecise calculations for long time [Liu *et al.*, 1987; Chung *et al.*, 1990; Liu *et al.*, 1994]. In [Liu *et al.*, 1991] imprecise calculation methods are categorized into *milestone methods*, *sieve function methods*, and *multiple version methods*. Milestone methods use monotone algorithms that gradually refine the result, and where each intermediate result can be returned as an imprecise result. Sieve function methods can skip certain computation steps to tradeoff computation quality for time. Multiple version methods have different implementations with different cost and precision for a task.

Scheduling of monotone imprecise tasks is treated in [Chung *et al.*, 1990]. Two models are proposed. A task that may be terminated any time after it has produced an acceptable result is logically decomposed into two parts, a mandatory part that must be completed before deadline, and an optional part that further refines the result generated by the mandatory part. The refinement becomes better and better the longer the optional part is allowed to execute. The two models differ with respect to if the optional part needs to complete or not. Scheduling of tasks where it is the average error between the results from consecutive jobs that is important uses a conservative and predictable strategy for the mandatory parts and a less conservative strategy for the optional parts. A number of RM-based scheduling algorithms for these types tasks are described. The schedulability of tasks where it instead is the cumulative effect of the errors that is important is also discussed.

### Dynamic System Upgrades

An alternative view of the need for flexibility is obtained if we look upon the problem of performing safe on-line upgrades of real-time systems, in particular safety-critical real-time control systems. The Simplex group at SEI/CMU has developed a framework, Simplex, that allows these types of on-line upgrades [Seto *et al.*, 1998a; Sha, 1998]. The basic building block of Simplex is the replacement unit. A replacement unit consists of a task together with a communication template. The replacement units are organized into application units that also con-

tain communication, and task management functions. A special safety unit is responsible for basic reliable operation and operation monitoring. A typical example of a replacement unit is a controller. A common setup is that the system contains two units: a safety controller and a baseline controller. The baseline controller provides the nominal control performance and it is assumed that this controller is executing when a new upgraded version of this controller should be installed.

Simplex considers a three-dimensional fault model consisting of timing faults, system faults, and semantical faults that effect the logical control behavior of the system. RM analysis is used to ensure schedulability under fault-free conditions. If a new controller does not return an answer before the deadline, Simplex switches in the safety controller, and then eventually, when the controlled process is back in a safe state, switches back to the baseline controller. System faults generated by the new controller are trapped by the operating system. In the same way as before Simplex will switch to the safety controller and then back to the baseline controller. Semantic faults are caught by the analytical redundancy obtained by having the safety controller monitoring the operation. The safety controller is responsible for monitoring that the controlled system is in a safe state and that the performance is better then that obtained by the baseline controller. If any of these two conditions should not be fulfilled, the safety controller is switched in the same way as before.

Dynamic change management and version handling is a large area where a lot of work has been performed, e.g., [Kramer and Magee, 1990; Stewart *et al.*, 1993; Gupta and Jalote, 1996]. However, most of this is outside the area of this survey.

## 6. Research Issues

Integrated control and scheduling contains a number of interesting research problems. A few of them are highlighted here.

### Control and Scheduling Co-Design

A prerequisite for an on-line integration of control and scheduling theory is that we are able to make an integrated off-line design of

control algorithms and scheduling algorithms. Such a design process should ideally allow an incorporation of the availability of computing resources into the control design by utilizing the results of scheduling theory. This is an area where, so far, relatively little work has been performed.

One of the first references that addressed this problem was [Seto *et al.*, 1996]. An algorithm was proposed that translates a control performance index into task sampling periods considering schedulability among tasks running with preemptive priority scheduling. The sampling periods were considered as variables and the algorithm determined their values so that the overall performance was optimized subject to the schedulability constraints. Both RM and EDF scheduling were considered. The performance index was approximated by an exponential function only and the approach did not take input-output latency into account. The approach was further extended in [Seto *et al.*, 1998b].

An approach to optimization of sampling period and input-output latency subject to performance specifications and schedulability constraints is presented in [Ryu *et al.*, 1997; Ryu and Hong, 1998]. The control performance is specified in terms of steady state error, overshoot, rise time, and settling time. These performance parameters are expressed as functions of the sampling period and the input-output latency. A heuristic iterative algorithm is proposed for the optimization of these parameters subject to schedulability constraints. The algorithm is based on using the period calibration method (PCM) for determining the task attributes. A case study involving the control design for a CNC controller is presented. The tasks are scheduled using EDF and a cyclic executive is used for run-time dispatching. The same application is further explored in [Kim *et al.*, 1999] where it was revealed that the intertask communication scheme of PCM may incur large latencies, and that the absence of overload handling is a critical limitation.

### Imprecise Control Algorithms

As discussed in Section 4 imprecise calculation methods can be categorized into milestone methods, sieve function methods, and multiple version methods. Examples of all three types can be found in control. Control algorithms that are based on on-line numerical solution to an optimization problem every sample can be viewed as milestone or

"any-time" methods. The result generated from the control algorithm is gradually refined for each iteration in the optimization up to a certain bound. Examples of this type of control algorithms are found in model-predictive control. Similar situations exists for all control algorithms that can be written on iterative form, e.g., on series form. One possibility could, e.g., be controllers on polynomial forms where the terms are arranged in decreasing order of magnitude.

It is also straightforward to find examples of controllers that be cast on sieve function form. For example, in an indirect adaptive controller the updating of the adapted parameters can be skipped when time constraints are hard, or, in a LQG controller the updating of the observer can be skipped when time constraints are hard. Similarly, in a PID controller the updating of the integral part may be skipped if needed. The extreme case is of course to skip the entire execution of the controller for a certain sample. This is equivalent to temporarily doubling the sampling interval. However, in all cases it is important to guarantee that the skips are only performed temporarily, and not too frequent. Also, the system may be more or less sensitive to skips depending on external conditions. For example, during system transients due to, e.g., a set-point change or a load disturbance, the skip of a part of the control algorithm and the resulting degraded performance may be less welcome than during steady-state operation. A similar situation holds for variations in sampling interval and variations in computational delay.

Finally it is also possible to find examples of multiple version methods in control, e.g., situations with one nominal control algorithm and one backup algorithm. However, in most cases the motivation for having the backup algorithm is control performance rather then timing issues, compare the discussions about the Simplex algorithm in Section 7.

***Model Predictive Control*** Model predictive control (MPC) is a control method that is based on the explicit usage of a model to predict the process output at future discrete time instants, over a prediction horizon. A sequence of future control actions are calculated over a control horizon by minimizing a given objective function, such that the predicted process output is as close as possible to a desired reference signal. At each sample the first control action in the sequence is ap-

plied to the controlled process, and the optimization is repeated. The most often used objective functions are modifications of the following quadratic function [Clarke *et al.*, 1987]:

$$J = \sum_{i=1}^{H_p} \alpha_i \left(r(k+i) - \hat{y}(k+i)\right)^2 + \sum_{i=1}^{H_c} \beta_i \Delta u(k+i-1)^2 \qquad (2)$$

The first term accounts for minimizing the variance of the process output from the reference, while the second term represents a penalty on the control effort (related for instance to energy). The latter term can also be expressed by using $u$ itself or other filtered forms of $u$, depending on the problem. The vectors $\alpha$ and $\beta$ define the weighting of the output error and the control effort with respect to each other and with respect to the prediction step. Constraints, e.g., level and rate constraints of the control input or other process variables can be specified as a part of the optimization problem. Generally, any other suitable cost function can be used, but for a quadratic cost function, a linear, time-invariant model, and in the absence of constraints, an explicit analytic solution of the above optimization problem can be obtained. Otherwise, numerical (usually iterative) optimization methods must be used. In the more industrially relevant case of a quadratic cost function, a linear, time-invariant model and inequality constraints on $u, \Delta u$, and $y$, the resulting optimization problem is quadratic and must be solved every sample.

MPC fits nicely into the general scheme of integrated control and scheduling. The optimization technique employed is based on sequential unconstrained minimization techniques. The calculations performed in every sample are organized as a sequence of iterations (outer iterations). In every outer iteration a number of inner, Newton iterations are performed. An interesting possibility is to calculate bounds on how much the objective function decreases for each new iteration. MPC can also be used in other ways. In every sample the control signals for the full control horizon are calculated. Normally only the first one is used, i.e., applied to the process, and the rest are discarded. However, another possibility is to instead store the future values of the control signals and to use them if the execution time is limited and/or the measured values of $y$ correspond well with the ones

that were used in optimization step. In this case it is essentially possible to use the MPC in open loop if there is not enough time to even start a new optimization.

### Event-Based Sampling

Although fixed sampling periods are adequate for many simple control loops, there are a lot of control problems where it is more natural to use varying sampling intervals. One such example is combustion engines that are sampled against engine speed, another example is in manufacturing systems, such as paper machines, where sampling is related to production rate. Another reason for using non equidistant sampling is if there is a large cost related to sampling, or if the rate of sampling is governed by unpredictable events. Such examples occur for example in biology and in economics. One motivation for using clever sampling is that the communication resources are limited and no unnecessary signals should be sent in the network.

There are several alternatives to periodic sampling. A basic prerequisite for the implementation of event-triggered sampling is that 'significant events' are detected. The traditional techniques to do this are to use fast polling (sampling) or event-triggered sensors that generates an event whenever the signal passes a certain limit. A typical situation is that the sensor sends a signal whenever the scalar signal $y(t)$ passes any of a certain pre-specified levels in the sampling set $Y = \{y_1, \ldots, y_N\}$. The times $t_k$ when this happens will be called *sampling times* and are defined by

$$y(t_k) = y_{i_k}$$

where $i_k \in \{1, \ldots, N\}$ determine the *sampling level* at sampling time $t_k$. Such a scheme has many conceptual advantages. Control is not executed unless it is required, control by exception, see [Kopetz, 1993]. This type of sampling is natural when using many digital sensors such as encoders. A disadvantage is that analysis and design are complicated.

Event-based sampling occurs naturally in many contexts. A common case is in motion control where angles and positions are sensed by encoders that give a pulse whenever a position or an angle has changed by a specific amount. Event based sampling is also a natural approach

when actuators with on-off characteristic are used. Satellite control by thrusters is a typical example, [Dodds, 1981]. Systems with pulse frequency modulation, [Polak, 1968], [Pavlidis and Jury, 1965], [Pavlidis, 1966],
[Skoog, 1968], [Noges and Frank, 1975], [Skoog and Blankenship, 1970], [Frank, 1979], [Sira-Ramirez, 1989] and [Sira-Ramirez and Lischinsky-Arenas, 1990] are other examples. In this case the control signal is restricted to be a positive or negative pulse of given size. The control actions decide when the pulses should be applied and what sign they should have. Other examples are analog or real neurons whose outputs are pulse trains, see [Mead, 1989] and [DeWeerth *et al.*, 1990]. Analysis of systems with event based sampling are related to general work on discontinuous systems, [Utkin, 1981], [Utkin, 1987], [Tsypkin, 1984] and to work on impulse control, see [Bensoussan and Lions, 1984]. It is also relevant in situations where control complexity has to be weighted against execution time.

Much work on systems of this type was done in the period 1960–1980. Analysis of event-based sampled systems is harder than for time-based sampled systems. This is due to the fact that sampling is no longer a linear operation. There are several papers that treat special system setups, such as observers for linear system with quantized outputs, [Sur, 1996], [Delchamps, 1989] many of which use classical ideas from Kalman observer design. In [Åström and Bernhardsson, 1999] it is shown that event-based sampling is more efficient than equidistant sampling. For example, an integrator system driven by white noise must be sampled 3–5 times faster using equidistant sampling than using event-based sampling to achieve the same output variance.

Just as event-triggered sampling can be used for deciding suitable measurement times, one can use event-triggering instead of time-triggering for choice of actuator signal. The motivation can be that it costs energy, bandwidth, etc, to change the control signal, so such changes should be avoided. The standard linear-quadratic control theory where quadratic weights are put on states and control signals has been extended also with discontinuous costs on changes in control signal. Such a cost can be either fixed or dependent on the size of the control change.

### Execution-Time Measurement

A prerequisite for integrated control and scheduling is on-line measurements of the actual task execution times. Several open issues exist. One possibility is to let the task dispatcher be responsible for execution time measurements. Another alternative is to let the tasks themselves perform the time measurements by calling appropriate kernel primitives at vital points in the code. One possibility is that the time measurement code is automatically added to the application task by the programming environment.

### Mode Changes

Mode changes are frequent in adaptive real-time control systems. For instance, every adjustment of task timing attributes in a feedback-scheduling system results in a mode change. A key question is whether the transient effects of mode changes in real-time control systems can be ignored or not. Ignoring the effects gives a simpler mode-change protocol and probably better average-case performance (no unnecessary delays and less overhead). Still, it is probably necessary to have a fall-back strategy and some guarantees about the worst-case behavior. This could perhaps include a calculation of the worst-case delay and an estimate of the resulting loss in control performance.

### Cost Functions

Control performance optimization in the context of task attribute adjustments needs cost functions that relate the sampling period with the control performance for each control loop, cost functions that relate the input/output latency with the control performance and, perhaps also, cost functions that relate the execution time for the controller (e.g., the number of iterations in a MPC) with the control performance. In addition to this it would also be beneficial if the control design methodology could provide cost functions for the jitter in sampling period and input output latency.

An interesting question is the general nature of these cost functions. For example, it is not always so that faster sampling gives better performance. For example, consider the cost function for a pendulum,

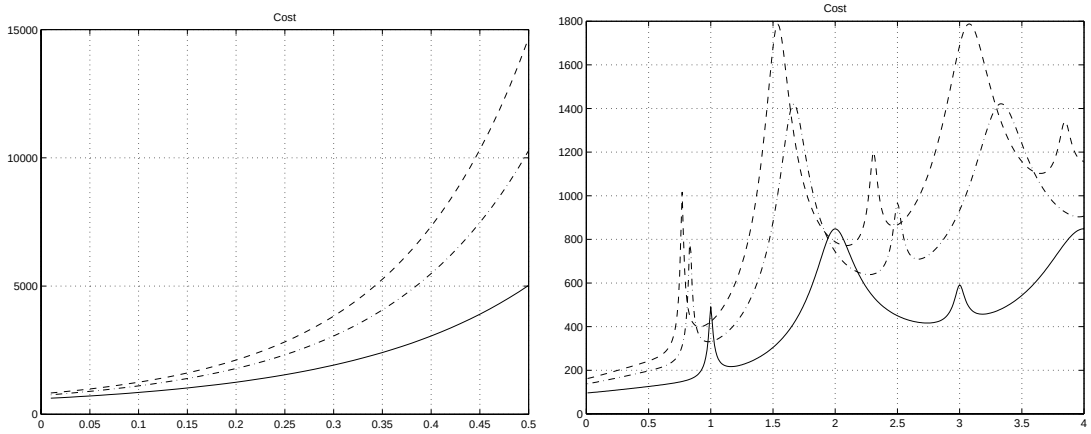



**Figure 14.** The cost $J_i(h)$ as a function of the sampling interval for the inverted pendulum (left) and for the normal pendulum (right). The plots shows the graphs for $\omega_0 = 3.1416$(full), 3.7699(dot-dashed), and 4.0841(dashed). Note that the cost does not depend monotonously on the sampling period.

described by the following equations:

$$dx = \begin{bmatrix} 0 & 1 \\ \alpha \cdot \omega_0^2 & -d \end{bmatrix} x dt + \begin{bmatrix} 0 \\ \alpha \cdot b \end{bmatrix} u dt + dv_c$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x, \qquad R_{1c} = \begin{bmatrix} 0 & 0 \\ 0 & \omega_0^4 \end{bmatrix}$$

The natural frequency is $\omega_0$, the damping $d = 2\zeta\omega_0$, and $b = \omega_0/9.81$. If $\alpha = 1$ the equations describe the pendulum in the upright position (the inverted pendulum), and with $\alpha = -1$ they describe the pendulum in the downward position. The incremental covariance of $v_c$ is $R_{1c}$, which corresponds to a disturbance on the control signal. The cost for the system is described by a linear quadratic function:

$$J(h) = \frac{1}{h} \int_0^h \begin{bmatrix} x^T(t) & u^T(t) \end{bmatrix} Q \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} dt$$

The cost function, $J$, for the inverted pendulum as a function of the sampling interval $h$ is shown in Figure 14 (left). The corresponding function for the stable pendulum is shown in Figure 14 (right). Figure 14 (right) clearly demonstrates that faster sampling not necessarily gives better control performance.

Cost functions may be used as criteria for choosing sampling interval. In a setup where several control loops compete for the CPU resources, cost functions can be used as an instrument for portioning resources. To optimize the resource sharing using cost functions, we need to investigate how they depend on the resource, e.g. we need ways of calculating the cost function derivatives.

### Task Attribute Adjustments

Most work on task attribute adjustments assume that the cost functions at hand have certain properties, e.g., are continuous and monotone. In the control context this is most likely not true. For example, concerning adjustments of the task period it may from a control point of view only be possible to adjust the period in discrete steps. An interesting question is whether it is possible to state the problem as a global optimization problem, and in that case, what the nature of this optimization is, e.g., convex or non-convex. If it is possible to state the problem as an optimization problem it is also important to find ways of solving this or an approximation of it on-line.

In a given situation it may be possible to reduce the CPU utilization of tasks in different ways, e.g., by increasing the period, reducing the allowed execution time, skipping a sample. An interesting question is to decide which way to use. Also, the possibility of a control task to handle task attribute adjustments is most likely dependent on external conditions, e.g., transient set-point changes, load disturbances, etc. During a set-point change the possibility to reduce the sampling time may be rather limited. However, in many cases the system has the possibility to postpone the set-point change for a short while, and thereby schedule it in a way that affects the other tasks as little as possible.

### Simulation

Much of the work performed by both control engineers and real-time systems engineer are verified by simulation. There are tools available both for simulating different scheduling protocols and other tools for simulating different controller designs. However, the problem of how the implemented controller will actually interact with the plant has been studied very sparsely. A tool that allows the user to verify both the scheduling of the control loops and the control performance is needed. How will, for example, jitter caused by the fact that several control

loops share CPU affect the performance of the controllers? A simulator of this type is described in [Eker and Cervin, 1999].

## 7. Summary

For more demanding control applications requiring higher degrees of flexibility and for situations where computing resources are limited it is motivated to study more general models and methods that allow an integrated approach to control and scheduling design. The aim of this survey is to present the emerging field of integrated control and scheduling, and to give a state-of-the-art survey of the methods and techniques from the fields of real-time scheduling and control that are of relevance to this. Issues that have been discussed include scheduling of control tasks, dynamic task attribute adjustments, feedback scheduling, probabilistic scheduling, and compensation for sampling period variations.

## 8. References

Abdelzaher, T., E. Atkins, and K. Shin (1997): "QoS negotiation in real-time systems, and its application to flight control." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Abeni, L. and G. Buttazzo (1999a): "Adaptive bandwidth reservation for multimedia computing." In *Proceedings of IEEE Real Time Computing Systems and Applications, Hong Kong*.

Abeni, L. and G. Buttazzo (1999b): "QoS guarantee using probabilistic deadlines." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Albertos, P. and J. Salt (1990): "Digital regulators redesign with irregular sampling." In *IFAC 11th World Congress*, vol. IV of *IFAC*, pp. 465–469. Tallinn, Estonia.

Åström, K. J. and B. Bernhardsson (1999): "Comparison of periodic and event based sampling for first-order stochastic systems." In *Proceedings of the 14th IFAC World Congress*. Beijing, P.R. China.

Atlas, A. and A. Bestavros (1998): "Statistical rate monotonic scheduling." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Audsley, N., A. Burns, R. Davis, K. Tindell, and A. Wellings (1995): "Fixed priority pre-emptive scheduling: An historical perspective." *Journal of Real-Time Systems*, **8**, pp. 173–198.

Audsley, N., A. Burns, M. Richardson, K. Tindell, and A. Wellings (1993a): "Applying new scheduling theory to static preemptive scheduling." *Software Engineering Journal*, **8:5**, pp. 285–292.

Audsley, N., K. Tindell, and A. Burns (1993b): "The end of the line for static cyclic scheduling." In *Proceedings of 5th Euromicro Workshop on Real-Time Systems*.

Baruah, S. (1998a): "A general model for recurring real-time tasks." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Baruah, S. (1998b): "Overload tolerance for single-processor workloads." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society Press.

Baruah, S., G. Buttazzo, S. Gorinsky, and G. Lipari (1999a): "Scheduling periodic task systems to minimize output jitter." Proc. 6th International IEEE Conference on Real-Time Computing Systems and Applications, Hong-Kong, China.

Baruah, S., D. Chen, S. Gorinsky, and A. Mok (1999b): "Generalized multi-frame tasks." Submitted for publication. Available from `www.emba.uvm.edu/~sanjoy`.

Baruah, S., D. Chen, and A. Mok (1997): "Jitter concerns in periodic task systems." In *Proceedings of the 18th Real-Time Systems Symposium*.

Baruah, S., D. Chen, and A. Mok (1999): "Static-priority scheduling of multi-frame tasks." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Baruah, S. and S. Gorinsky (1999): "Scheduling periodic task systems to minimize output jitter." Submitted for publication. Available from `www.emba.uvm.edu/~sanjoy`.

Baruah, S. and J. Haritsa (1997): "Scheduling for overload in real-time systems." *IEEE Trans Computers*, **46:9**, pp. 1034–1039.

Bensoussan, A. and J.-L. Lions (1984): *Impulse control and quasi-variational inequalities*. Gauthier-Villars, Paris.

Bernat, G. and A. Burns (1999): "Exact schedulability analysis of aperiodic servers." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Blevins, P. and C. Ramamoorthy (1976): "Aspects of a dynamically adaptive operating system." *IEEE Trans Computers*, **25:7**.

Burns, A. (1998): "The meaning and role of value in scheduling flexible real-time systems." vol. Proceedings of the IEEE Real-Time Computing Systems Application Conference (RTCSA), Hiroshima, Japan.

Buttazzo, G., G. Lipari, and L. Abeni (1998): "Elastic task model for adaptive rate control." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Buttazzo, G., M. Spuri, and F. Sensini (1995): "Value vs deadline scheduling in overload conditions." vol. Proceedings of the 16th IEEE Real-Time Systems Symposium.

Caccamo, M. and G. Buttazzo (1997): "Exploiting skips in periodic tasks for enhancing aperiodic responsiveness." In *Proceedings of the 18th IEEE Real-Time System Symposium*.

Cervin, A. (1999): "Improved scheduling of control tasks." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 4–10. York, England.

Chung, J.-Y., J. Liu, and K.-J. Lin (1990): "Scheduling periodic jobs that allow imprecise results." *IEEE Trans on Computers*, **39:9**.

Clarke, D., C. Mohtadi, and P. Tuffs (1987): "Generalised predictive control. Part 1: The basic algorithm. Part 2: Extensions and interpretations." *Automatica*, **23:2**, pp. 137–160.

Davis, R. and A. Wellings (1995): "Dual priority scheduling." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Delchamps, D. (1989): "Extracting state information from a quantized output record." *Systems and Control Letter*, **13**, pp. 365–372.

DeWeerth, S., L. Nielsen, C. Mead, and K. J. Åström (1990): "A neuron-based pulse servo for motion control." In *IEEE Int. Conference on Robotics and Automation*. Cincinnati, Ohio.

Dodds, S. J. (1981): "Adaptive, high precision, satellite attitude control for microprocessor implementation." *Automatica*, **17:4**, pp. 563–573.

Eker, J. and A. Cervin (1999): "A MATLAB toolbox for real-time and control systems co-design." Proc. 6th International IEEE Conference on Real-Time Computing Systems and Applications, Hong-Kong, China.

Frank, P. M. (1979): "A continuous-time model for a pfm-controller." *IEEE Trans. of Automat. Control*, **AC-25:5**, pp. 782–784.

Gerber, R. and S. Hong (1993): "Semantics-based compiler transformations for enhanced schedulability." In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, pp. 232–242.

Gerber, R. and S. Hong (1997): "Slicing real-time programs for enhanced schedulabilty." *ACM Transactions on Programming Languages and Systems*, **19:3**, pp. 525–555.

Gonzalez Härbour, M., M. H. Klein, and J. P. Lehoczky (1994): "Timing analysis for fixed-priority scheduling of hard real-time systems." *IEEE Transactions on Software Engineering*, **20:1**, pp. 13–28.

Gupta, D. and P. Jalote (1996): "A formal framework for on-line software version change." *IEEE Trans Software Engineering*, **22:2**.

Gustafsson, K. (1991): "Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods." *ACM Transactions on Mathematical Software*, **17:4**, pp. 533–554.

Gutierrez, J. and M. Harbour (1998): "Schedulability analysis for tasks with static and dynamic offsets." In *Proceedings of 19th IEEE Real-Time Systems Symposium, Madrid*.

Jones, M. and P. Leach (1995): "Modular real-time resource management in the Rialto operating system." In *Proceedings of the of the Fifth Workshop on Hot Topics in Operating Systems*.

Joseph, M. and P. Pandya (1986): "Finding response times in a real-time system." *The Computer Journal*, **29:5**, pp. 390–395.

Kim, N., M. Ryu, S. Hong, and H. Shin (1999): "Experimental assessment of the period calibration method: A case study." *Journal of Real-Time Systems*. Accepted for publication.

Klein, M. H., T. Ralya, B. Pollak, R. Obenza, and M. Gonzalez Härbour (1993): *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publisher.

Kleinrock, L. (1970): "A continuum of time-sharing scheduling algorithms." In *Proc. of AFIPS, SJCC*.

Kopetz, H. (1993): "Should responsive systems be event triggered or time triggered?" *IEICE Trans. on Information and Systems*, **E76-D:10**, pp. 1525–1532.

Koren, G. and D. Shasha (1995): "Skip-over: Algorithms and complexity for overloaded systems that allow skips." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Kosugi, N., A. Mitsuzawa, and M. Tokoro (1996): "Importance-based scheduling for predictable real-time systems using MART." In *Proceedings of the 4th Int. Workshop on Parallel and Distributed Systems*, pp. 95–100. IEEE Computer Society.

Kosugi, N. and S. Moriai (1997): "Dynamic scheduling for real-time threads by period adjustment." In *Proceedings of the World Congress on Systems Simulation*, pp. 402–406.

Kosugi, N., K. Takashio, and M. Tokoro (1994): "Modification and adjustment of real-time tasks with rate monotonic scheduling algorithm." In *Proceedings of the 2nd Workshop on Parallel and Distributed Systems*, pp. 98–103.

Kramer, J. and J. Magee (1990): "The evolving philosophers problem: Dynamic change management." *IEEE Trans Software Engineering*, **16:11**.

Kuo, T.-W. and A. Mok (1991): "Load adjustment in adaptive real-time systems." In *Proceedings of the 12th IEEE Real-Time Systems Symposium*.

Lee, C., R. Rajkumar, J. Lehoczky, and D. Siewiorek (1998): "Practical solutions for QoS-based resource allocation." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Lee, C., R. Rajkumar, and C. Mercer (1996): "Experiences with processor reservation and dynamic QoS in real-time Mach." In *Proceedings of Multimedia Japan 96*.

Lehoczky, J. and S. Ramos-Thuel (1992): "An optimal algorithm for scheduling soft aperiodic tasks in fixed-priority preemptive systems." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Lehoczky, J., L. Sha, and J. Strosnider (1987): "Enhanced apriodic responsiveness in hard real-time environment." In *Proceedings of the 8th IEEE Real-Time Systems Symposium*.

Leung, J. Y. T. and J. Whitehead (1982): "On the complexity of fixed-priority scheduling of periodic, real-time tasks." *Performance Evaluation*, **2:4**, pp. 237–250.

Li, B. and K. Nahrstedt (1998): "A control theoretic model for quality of service adaptations." In *Proceedings of Sixth International Workshop on Quality of Service*.

Lin, K. and A. Herkert (1996): "Jitter control in time-triggered systems." In *Proceedings of the 29th Hawaii International Conference on System Sciences*.

Liu, C. L. and J. W. Layland (1973): "Scheduling algorithms for multiprogramming in a hard-real-time environment." *Journal of the ACM*, **20:1**, pp. 40–61.

Liu, J., K.-J. Lin, and S. Natarajan (1987): "Scheduling real-time, periodic jobs using imprecise results." In *Proceedings of the IEEE Real-Time System Symposium*, pp. 252–260.

Liu, J., K.-J. Lin, W.-K. Shih, A. Yu, J.-Y. Chung, and W. Zhao (1991): "Algorithms for scheduling imprecise computations." *IEEE Trans on Computers*.

Liu, J., W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung (1994): "Imprecise computations." *Proceedings of the IEEE*, Jan, pp. 83–93.

Locke, C. D. (1992): "Software architecture for hard real-time applications: Cyclic vs. fixed priority executives." *Real-Time Systems*, **4**, pp. 37–53.

Lu, C., J. Stankovic, and G. Tao (1999): "Design and evaluation of a feedback control EDF scheduling algorithm." In *Proc. of the Real-Time Systems Symposium, Phoenix*.

Mead, C. A. (1989): *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, Massachusetts.

Mok, A. and D. Chen (1997): "A multi-frame model for real-time tasks." *IEEE Transactions on Software Engineering*, **23:10**.

Nakajima, T. (1998): "Resource reservation for adaptive QoS mapping in real-time Mach." In *Proceedings of the Sixth International Workshop on Parallel and Distributed Real-Time Systems*.

Nakajima, T. and H. Tezuka (1994): "A continuous media application supporting dynamic QoS control on real-time Mach." In *Proceedings of ACM Multimedia'94*.

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

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1998): "Some topics in real-time control." In *American Control Conference*. Philadelphia.

Noges, E. and P. M. Frank (1975): *Pulsfrequenzmodulierte Regelungssysteme*. R. Oldenbourg, München.

Pavlidis, T. (1966): "Optimal control of pulse frequency modulated systems." *IEEE Trans. of Automat. Control*, **AC-11:4**, pp. 35–43.

Pavlidis, T. and E. J. Jury (1965): "Analysis of a new class of pulse frequency modulated control systems." *IEEE Trans. of Automat. Control*, **AC-10**, pp. 35–43.

Pedro, P. and A. Burns (1998): "Schedulability analysis for mode changes in flexible real-time systems." In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*.

Polak, E. (1968): "Stability and graphical analysis of first order of pulse-width-modulated sampled-data regulator systems." *IRE Trans. Automatic Control*, **AC-6:3**, pp. 276–282.

Potier, D., E. Gelenbe, and J. Lenfant (1976): "Adaptive allocation of central processing unit quanta." *Journal of ACM*, **23:1**.

Rajkumar, R., C. Lee, J. Lehoczky, and D. Siewiorek (1997): "A resources allocation model for QoS management." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Ramanathan, P. (1997): "Graceful degradation in real-time control application using (m,k)-firm guarantee." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Ryu, M. and S. Hong (1998): "Toward automatic synthesis of schedulable real-time controllers." *Integrated Computer-Aided Engineering*, **5:3**, pp. 261–277.

Ryu, M., S. Hong, and M. Saksena (1997): "Streamlining real-time controller design: From performance specifications to end-to-end timing constraints." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Seto, D., B. Krogh, L. Sha, and A. Chutinan (1998a): "Dynamic control system upgrade using the Simplex architecture." *IEEE Control Systems*, August.

Seto, D., J. Lehoczky, and L. Sha (1998b): "Task period selection and schedulability in real-time systems." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Seto, D., J. Lehoczky, L. Sha, and K. Shin (1996): "On task schedulability in real-time control systems." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Sha, L. (1998): "Dependable system upgrade." In *Proceedings of IEEE Real Time Systems Symposium*.

Sha, L. and J. Goodenough (1990): "Real-time scheduling theory and ada." *IEEE Computer*, **23:4**, pp. 53–62.

Sha, L., R. Rajkumar, and J. Lehoczky (1989): "Mode change protocols for priority-driven pre-emptive scheduling." *Real-Time Systems*, **1:3**, pp. 244–264.

Sha, L., R. Rajkumar, and J. Lehoczy (1990): "Priority inheritance protocols: An approach to real-time synchronization." *IEEE Trans on Computers*, **39:9**.

Sha, L., R. Rajkumar, and S. Sathaye (1994): "Generalized rate-monotonic scheduling theory: A framework for developing real-time systems." *Proceedings of the IEEE*, **82:1**.

Shasha, D. and G. Koren (1995): "D-over: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems." *Siam Journal of Computing*, **24:2**, pp. 318–339.

Shin, K. G. and C. L. Meissner (1999): "Adaptation of control system performance by task reallocation and period modification." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 29–36.

Sira-Ramirez, H. (1989): "A geometric approach to pulse-width modulated control in nonlinear dynamical systems." *IEEE Trans. of Automat. Control*, **AC-34:2**, pp. 184–187.

Sira-Ramirez, H. and P. Lischinsky-Arenas (1990): "Dynamic discontinuous feedback control of nonlinear systems." *IEEE Trans. of Automat. Control*, **AC-35:12**, pp. 1373–1378.

Skoog, R. A. (1968): "On the stability of pulse-width-modulated feedback systems." *IEEE Trans. of Automat. Control*, **AC-13:5**, pp. 532–538.

Skoog, R. A. and G. L. Blankenship (1970): "Generalized pulse-modulated feedback systems: Norms, gains, lipschitz constants and stability." *IEEE Trans. of Automat. Control*, **AC-15:3**, pp. 300–315.

Sprunt, B., L. Sha, and J. Lehoczky (1989): "Aperiodic task scheduling for hard real-time systems." *The Journal of Real-Time Systems*.

Spuri, M. and G. Buttazzo (1996): "Scheduling aperiodic tasks in dynamic priority systems." **10:2**, pp. 179–210.

Stankovic, J. and K. Ramamritham (1991): "The Spring kernel: A new paradigm for real-time systems." *IEEE Software*, **8**.

Stankovic, J. A., C. Lu, S. H. Son, and G. Tao (1999): "The case for feedback control real-time scheduling." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 11–20.

Stewart, D., R. Volpe, and P. Khosla (1993): "Design of dynamically reconfigurable real-time software using port-based objects." Technical Report CMU-R1-TR-93-11. CMU.

Sur, J. (1996): *State Observers for Linear Systems with Quantized Outputs*. PhD thesis, University of Santa Barbara.

Tia, T.-S., Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu (1995): "Probabilistic performance guarantee for real-time tasks with varying computation times." vol. Proceedings of the IEEE Real-Time Technology and Applications Symposium. Chicago, IL.

Tindell, K., A. Burns, and A. J. Wellings (1992): "Mode changes in priority preemptively scheduled systems." In *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pp. 100–109.

Tokuda, H., T. Nakajima, and P. Rao (1990): "Real-time Mach: Towards a predictable real-time kernel." In *Proceedings of USENIX Mach Workshop*.

Törngren, M. (1998): "Fundamentals of implementing real-time control applications in distributed computer systems." *Real-time systems*, **14:3**.

Tsypkin, Ya. Z. (1984): *Relay Control Systems*. Cambridge University Press, Cambridge, UK.

Utkin, V. (1981): *Sliding modes and their applications in variable structure systems*. MIR, Moscow.

Utkin, V. I. (1987): "Discontinuous control systems: State of the art in theory and applications." In *Preprints 10th IFAC World Congress*. Munich, Germany.

Wittenmark, B. and K. J. Åström (1980): "Simple self-tuning controllers." In Unbehauen, Ed., *Methods and Applications in Adaptive Control*, number 24 in Lecture Notes in Control and Information Sciences, pp. 21–29. Springer-Verlag, Berlin, FRG.

# Paper 2

# Improved Scheduling of Control Tasks

**Anton Cervin**

**Abstract**

The paper considers the implementation of digital controllers as real-time tasks in priority-preemptive systems. The performance of a digital feedback control system depends critically on the timing of its sampling and control actions. It is desirable to minimize the computational delay in the controller, as well as the sampling jitter and the control jitter. It is shown that by scheduling the two main parts of a control algorithm as separate tasks, the computational delay can often be reduced significantly. A heuristic method for assigning deadlines to the parts is presented. Further modifications are given to reduce the jitter and to facilitate delay compensation. The result is improved control performance under maintained schedulability.

# 1. Introduction

Digital control systems constitute a large part of all real-time systems. Despite of this, surprisingly little effort has gone into studying their timely behavior when implemented as periodic tasks in a computer. In this paper, we concentrate on the implementation of digital controllers as real-time tasks in priority-preemptive systems.

An overview of a digital control system is shown in Figure 1. At a

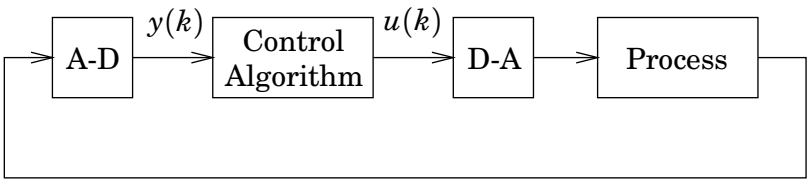



**Figure 1.** Overview of a digital control system.

fixed frequency, the controller requests A-D conversion to obtain a measurement sample, $y(k)$, from the physical process, computes a control signal, $u(k)$, and requests D-A conversion, sending the control signal to the process. Timing is very critical to the stability and performance of the closed-loop control system. Jitter in the sampling times and output times can be viewed as disturbances acting on the process. A computational delay between the A-D and the D-A conversions decreases the stability margin and the performance of the control system.

There are two common ways of synchronizing the inputs and the outputs [Åström and Wittenmark, 1997]. In the first approach, which we refer to as Textbook Implementation A, the control signal is sent out as soon as it has been calculated. To minimize the computational delay, the control algorithm is split into two parts. The first part, called Calculate Output, contains only the operations necessary to produce a control signal. The rest of the calculations, called Update State, are postponed until after the D-A conversion.

Detailed in source code, Textbook Implementation A could look like this (the source code is a variant of Modula-2, with support for real-time primitives):

```
LOOP
   Wait(ClockInterrupt);
```

```
   A_D_Conversion;
   CalculateOutput;
   D_A_Conversion;
   UpdateState;
END
```

For example, consider the common control strategy of using state feedback in conjunction with a state observer (this includes the popular LQG controller). The control algorithm can be structured on the following form [Gustafsson and Hagander, 1991]:

$$\varepsilon(k) = y(k) - \hat{y}(k \mid k-1) \tag{1}$$

$$u(k) = \hat{u}(k \mid k-1) - M\varepsilon(k) \tag{2}$$

$$\hat{x}(k+1 \mid k) = A\hat{x}(k \mid k-1) + Bu(k) + K\varepsilon(k) \tag{3}$$

$$\hat{y}(k+1 \mid k) = C\hat{x}(k+1 \mid k) \tag{4}$$

$$\hat{u}(k+1 \mid k) = -L\hat{x}(k+1 \mid k) \tag{5}$$

Here, the A-D conversion is implicit in the use of the measurement variable $y(k)$ in (1), and the D-A conversion is implicit in the calculation of the control variable $u(k)$ in (2). Calculate Output contains only a few scalar operations in (1) and (2), while all the matrix multiplications have been moved to the Update State part in (3)–(5).

A second common way of synchronizing the inputs and the outputs is to send out the control signal at the beginning of the next period. In this approach, which we refer to as Textbook Implementation B, the computational delay is always approximately equal to one period. The delay is more deterministic, but also longer.

At a first glance, digital controllers seem to fit right into the rate-monotonic framework—each controller could be described as a periodic task $\tau_i$ having a period $T_i$ and a worst-case computation time $C_i$. Well known tests [Liu and Layland, 1973] [Joseph and Pandya, 1986] can be applied to check for schedulability. But with that kind of thinking, the specific timing needs of digital controllers are ignored. Even if a set of control tasks are schedulable using rate-monotonic scheduling, the controllers can suffer from significant sampling jitter, computational delay, and output jitter. Lower-priority tasks can be preempted by higher-priority tasks at any point in the code. With a more detailed task model, where each control task is decomposed into subtasks, these

issues can be dealt with. Decomposition of control tasks has been suggested before [Gerber and Hong, 1993] [Burns *et al.*, 1994] [Gerber and Hong, 1997], but only for the sake of increased schedulability. The key problems that we address are:

1. Derivation of a more detailed task model which captures the specific timing needs of control tasks.
2. Assignment of task attributes (priorities, deadlines, offsets, etc.) to optimize the control performance, subject to the schedulability constraints.

Previous work on task attribute assignment with respect to control performance [Seto *et al.*, 1996] [Kim, 1998] have focused on task period selection for single-task models of controllers. The detailed timely behavior has not been addressed.

The rest of this paper is outlined as follows. Section 2 deals with periodic sampling. In Section 3, the problems of deriving a task model and assigning task attributes are treated, and corresponding schedulability analysis is reviewed. Section 4 discusses different strategies for delay compensation. Section 5 gives an example, where the theory in the paper is applied to a control example with three inverted pendulums. Simulations of processes, controllers, and real-time kernel together show that a more detailed scheduling can reduce jitter and computational delay, and thus give better control performance.

## 2. Periodic Sampling

Digital control theory assumes that measurement samples are taken periodically. As for the textbook implementations discussed in Section 1 however, a control task may very well be preempted by higher-priority tasks when it is time to request an A-D conversion. This can lead to serious sampling jitter for lower-priority control tasks.

The code segment found in Section 1 also assumes that the A-D conversion works like a function that returns a value. This must not be true today, however, when most A-D converters can be treated as asynchronous input devices. Some A-D converters can be programmed to automatically sample at a given rate. Others must be periodically

requested to start the conversion. This could be done by a dedicated high-priority, low-cost task. Whichever way the conversion is initialized, the A-D converter will give a hardware interrupt when it has finished. An interrupt handler can then retrieve the value and signal to the control task that a new sample is available. This effectively solves the problem of sampling jitter.

Now assuming that a semaphore NewSample is signaled each time a new sample is available to the control task, we modify our code to

```
LOOP
   Wait(NewSample);
   GetSample;
   CalculateOutput;
   D_A_Conversion;
   UpdateState;
END
```

## 3. Scheduling

In this section, we investigate the possibility of scheduling the parts of the control algorithm as separate tasks. Basic scheduling analysis and subtask scheduling analysis is reviewed, and the problem of deadline assignment for the subtasks is treated.

### Basic scheduling analysis

Disregarding the different parts of the control algorithm, a digital controller can be described as a period task $\tau_i$ having period $T_i$, deadline $D_i$, worst-case execution time $C_i$, and priority $P_i$. If it is assumed that $D_i = T_i$, the rate-monotonic priority assignment is optimal (in the schedulability sense) [Liu and Layland, 1973]. The worst-case response time $R_i$ of a task can be calculated from the equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{6}$$

where $hp(i)$ is the set of tasks with higher priority than $\tau_i$ [Joseph and Pandya, 1986]. The task set is schedulable if $R_i \leq D_i$ for all tasks.

The rate-monotonic model is sufficient for controller implementations where the control signal is sent out at the beginning of the next period. For implementations where the control signal is sent out as soon as possible, however, the scheduling model does not reflect the fact that the Calculate Output part should finish as soon as possible. The result is unnecessarily large delays and output jitter for lower-priority control tasks.

We could allow $D_i \leq T_i$, in which case the deadline-monotonic priority assignment is optimal [Leung and Whitehead, 1982]. Eq. (6) holds for this case also. The deadlines could be used to improve the response time (and thus the computational delay) of a few selected tasks. Decreasing the deadlines of all tasks could render the task set unschedulable.

### Subtask scheduling analysis

For simplicity, it is assumed that the requests for A-D and D-A conversions can be neglected in the analysis. Let each control task $\tau_i$ consist of two subtasks, $\tau_{COi}$ (Calculate Output) and $\tau_{USi}$ (Update State). The worst-case execution time of the subtasks are assumed to be known and equal to $C_{COi}$ and $C_{USi}$ respectively.

We first look at the timing analysis developed by Härbour *et al.* [Gonzalez Härbour *et al.*, 1994]. In their model, each subtask is assigned a fixed priority and a deadline, and the subtasks are executed serially. For control tasks, Update State has a natural deadline $D_{USi} = T_i$. The deadline for Calculate Output must at least be constrained by

$$C_{COi} \leq D_{COi} \leq T_i - C_{USi} \tag{7}$$

Since Calculate Output is more time-critical than Update State, it is natural to enforce a higher priority on it. For this special case, the deadline-monotonic priority assignment is optimal [Gonzalez Härbour *et al.*, 1994].

### Deadline assignment

The scheduling model above assumes that deadlines have been assigned to all Calculate Output subtasks. A key question is, how should this be done?

To maximize control performance, the deadlines (and thus computational delays) should be minimized. This could be stated as an optimization problem—for instance to minimize a weighted sum of the deadlines

$$f = \sum_i \frac{D_{CO_i}}{T_i} \tag{8}$$

under the schedulability constraint. A first try would be to let all the Calculate Output parts have higher priorities than all the Update States parts. Unfortunately this might render the task set unschedulable.

To find the optimal deadline assignment in the general case, an exhaustive search among the different priority orderings must be carried out. With $n$ tasks, there are $1 \cdot 3 \cdot 5 \cdots (2n-1)$ possible subtask priority assignments!

For cases where exact minimization is unrealistic, some heuristic deadline assignment method must be used. For *soft* real-time systems, several such methods exists, for instance the *equal flexibility* deadline assignment [Kao and Garcia-Molina, 1993]. They are not applicable here, since they cannot guarantee that all deadlines are met.

For control tasks, we present the following heuristic which attempts to minimize the deadlines of the Calculate Output parts while maintaining schedulability:

1. Start by assigning effective deadlines to the Calculate Output parts, i.e. set $D_{CO_i} := T_i - C_{US_i}$.

2. Assign deadline-monotonic priorities.

3. Calculate response times according to (6).

4. Decrease deadlines by assigning $D_{CO_i} := R_{CO_i}$.

5. Repeat from 2 until no further improvement is given (for instance by the criterion in Eq. (8)).

The heuristic works because of the optimality of the deadline-monotonic priority assignment. The task set must be schedulable after each improvement—at least by the previous priority ordering.

### Offset scheduling

Another scheduling model that could be applied to the parts of a control algorithm is *offset scheduling* [Audsley *et al.*, 1993]. The subtasks are not serially executed—rather, the subtask $\tau_{USi}$ is released with a fixed offset $O_{USi}$ compared to the release of $\tau_{COi}$. The offset must be chosen somewhere in the interval

$$D_{COi} \leq O_{USi} \leq T_i - C_{USi} \tag{9}$$

and $D_{COi}$ must be chosen in the interval

$$C_{COi} \leq D_{COi} \leq O_{USi} \tag{10}$$

Figure 2 shows the execution of the two subtasks in isolation.

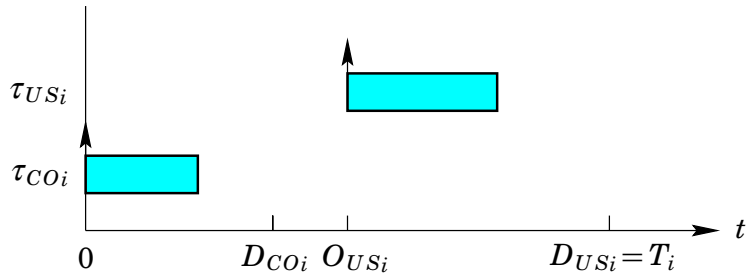



**Figure 2.** The parts $\tau_{COi}$ and $\tau_{USi}$ are scheduled using an offset.

This task model is more general than the priority-constrained model and thus provides a higher degree of schedulability. The price for this improvement is a more complex optimization problem. We have to choose both deadlines and offsets. The deadline monotonic priority assignment is no longer optimal, the response time calculations are more complicated, and the simple heuristic presented before cannot be used. Still, with the right set of computer tools, this approach could very well produce better results than the priority-constrained approach.

### Implementation

Even though we have modeled Calculate Output and Update State as two separate tasks in the schedulability analysis, this does not imply that we have to implement them as such. If we have used the priority-constrained approach, and if the real-time operating system allows

priorities to be changed dynamically, we can simply insert ChangePriority commands into our existing code:

```
LOOP
   ChangePriority(P_CO);
   Wait(NewSample);
   GetSample;
   CalculateOutput;
   D_A_Conversion;
   ChangePriority(P_US);
   UpdateState;
END
```

# 4. Delay Compensation

After scheduling the parts of the control algorithm as separate tasks, we should have been able to reduce the worst-case computational delay of the controller significantly. If the remaining delay is very small, it can be neglected altogether. Otherwise we have the option of redesigning the controller to compensate for the delay.

### Compensation assuming a fixed delay

Compensating for a fixed computational delay is straight forward [Åström and Wittenmark, 1997]. Essentially, it is just a matter of introducing an extra state in the controller. This causes a slight increase in the computation time of the control algorithm. The task set could become unschedulable, in which case we would respond by increasing the sampling period of some controllers. If the performance gain due to the delay compensation is greater than the performance loss due to the slower sampling, the compensation pays off.

The implementation must be modified once again, this time to ensure that the delay between the A-D and D-A conversions really is constant. It becomes necessary to have *time-stamped* samples, i.e. the GetSample function now returns both the sample time and the sample itself. After Calculate Output, we delay the control task until the deadline $D_{CO}$. We also raise the priority momentarily when requesting the D-A conversion, so that the output jitter is kept small (we assume that the request can be neglected in the schedulability analysis):

```
LOOP
   ChangePriority(P_CO);
   Wait(NewSample);
   GetSample(t, value);
   CalculateOutput;
   ChangePriority(High);
   WaitUntil(t+D_CO);
   D_A_Conversion;
   ChangePriority(P_US);
   UpdateState;
END
```

### Compensation assuming a random delay

The computational delay for a controller is generally not constant. Because of variations in execution time and interference from higher-priority tasks, the delay will be of stochastic nature. If the distribution of the delay is known, it is possible to derive a compensating controller that performs better than its fixed-delay counterpart [Nilsson *et al.*, 1996]. The increase in computation time will be larger though. It is an open question, under what conditions the different compensation strategies really pay off.

## 5. An Example

As an example, the suggested improvements from the previous sections are applied to a control problem, step by step. Simulations show that control performance can be improved significantly.

### The control problem

The control problem is to stabilize three identical inverted pendulums, see Figure 3. The measurement signal is the angle $y$, and the control signal is the acceleration $u$ of the pivot point. The process can be described by the transfer function

$$Y(s) = \frac{1}{s^2 - 1} U(s) \tag{11}$$

The pendulums are affected by input disturbances and measurement noise, both modeled as sequences of white noise. Furthermore, it is

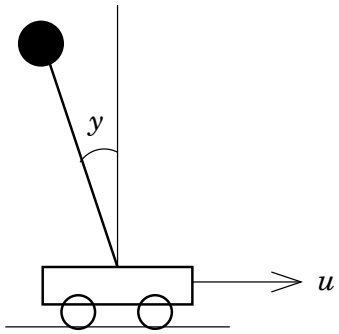



**Figure 3.** An inverted pendulum. The pendulum can be stabilized in the upright position $y = 0$ by controlling the acceleration $u$ of the pivot point.

assumed that the desired closed-loop behavior of the different processes is given by

$$s^2 + 2\zeta\omega_i s + \omega_i^2 = 0 \tag{12}$$

where $\zeta = \sqrt{3}/2 = 0.886$, $\omega_1 = 3$ rad/s, $\omega_2 = 5$ rad/s, and $\omega_3 = 7$ rad/s.

### Controller design

For each process, we design a digital controller with state feedback and Kalman filtering using pole placement, see for instance [Åström and Wittenmark, 1997]. The observer poles are chosen to have the same damping and twice the speed of the desired closed-loop behavior.

The sampling interval $T_i$ of each controller can be chosen according to the rule of thumb

$$0.1 < \omega_i T_i < 0.6 \tag{13}$$

Knowing that we have limited computing resources, we tend toward the upper bound and choose $T_1 = 167$ ms, $T_2 = 100$ ms, and $T_3 = 71$ ms.

### Performance evaluation

In the following, several different implementations of the controllers are evaluated by simulations. To capture the detailed timing behavior, the simulations include models of the process, the digital controllers, and the real-time kernel.

The three controllers are released simultaneously at time zero and then simulated for a time $T_{sim} = 1000$ s. Every simulation uses the same sequences for process noise and measurement noise. For each controller, we record the *performance loss*

$$J_i = \int_0^{T_{sim}} y_i^2(t)dt \tag{14}$$

As a reference, the controllers are first evaluated in a simulation where the execution times, the sampling jitter, and the control jitter are all assumed to be zero. The reference values obtained are:

| | Ref. |
|---|---|
| $J_1$ | 2.40 |
| $J_2$ | 1.35 |
| $J_3$ | 1.16 |

In the rest of the simulations, it is assumed that the execution time of the entire control algorithm is constant $C_i = 28$ ms, and that the execution times of the parts are constant $C_{CO\,i} = 10$ ms and $C_{US\,i} = 18$ ms.

### Implementation 1—Textbook Implementation A

Not caring about the different parts of the control algorithm, we model the controllers as the three periodic tasks $\tau_1$, $\tau_2$, and $\tau_3$. Having no further information, we assume $D_i = T_i$ and assign rate-monotonic priorities. We check that the task set is schedulable by calculating response times:

| | $T$ | $D$ | $C$ | $P$ | $R$ |
|---|---|---|---|---|---|
| $\tau_1$ | 167 | 167 | 28 | 1 | 140 |
| $\tau_2$ | 100 | 100 | 28 | 2 | 56 |
| $\tau_3$ | 71 | 71 | 28 | 3 | 28 |

In Textbook Implementation A (see Section 1), the A-D conversion takes place at the very beginning of Calculate Output, and the D-A conversion takes place at the very end of Calculate Output. The lower-priority tasks suffer from a lot of interference, resulting in poor control performance for Controller 1 and 2:

| | Ref. | Impl. 1 |
|---|---|---|
| $J_1$ | 2.40 | 4.90 |
| $J_2$ | 1.35 | 4.27 |
| $J_3$ | 1.16 | 1.28 |

A close-up of the behavior of Controller 1 is shown in Figure 4. It is clearly seen that the interference causes both the samples and the control actions to occur at irregular times.

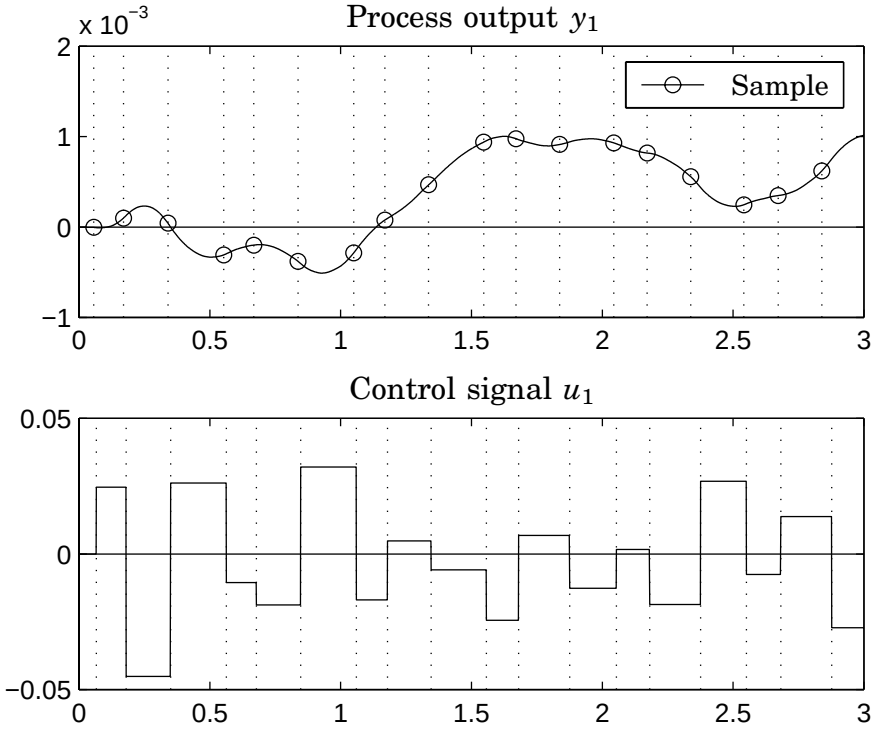



**Figure 4.** Close-up behavior of Controller 1 when Textbook Implementation A (Implementation 1) is used. Notice the large jitter in the sampling actions and the control actions.

## Implementation 2—Textbook Implementation B

We now evaluate the implementation where the control signal is sent out at the beginning of the next period, i.e. Textbook Implementation B. The computational delay is always equal to one period, and the controllers are easily redesigned to compensate for this, see Section 4. To keep the example simple, we assume that the compensating control

algorithm has the same execution time as the non-compensating one. New simulations give the following results:

| | Ref. | Impl. 1 | Impl. 2 |
|---|---|---|---|
| $J_1$ | 2.40 | 4.90 | 4.16 |
| $J_2$ | 1.35 | 4.27 | 1.96 |
| $J_3$ | 1.16 | 1.28 | 1.45 |

This implementation is sometimes better and sometimes worse than Textbook Implementation A—the output jitter is smaller, but the computational delay is larger.

### Implementation 3—Improved scheduling

Referring to the procedure detailed in Section 3, we now let each control task $\tau_i$ consist of the subtasks $\tau_{CO_i}$ and $\tau_{US_i}$. The optimal set of deadlines $D_{CO_i}$—by for instance the criterion in Eq. (8)—can easily be found using an exhaustive search over the possible priority orderings. But instead we shall illustrate the use of our heuristic for choosing deadlines. The task set is

| | $T$ | $C$ |
|---|---|---|
| $\tau_{CO_1}$ | 167 | 10 |
| $\tau_{US_1}$ | 167 | 18 |
| $\tau_{CO_2}$ | 100 | 10 |
| $\tau_{US_2}$ | 100 | 18 |
| $\tau_{US_3}$ | 71 | 10 |
| $\tau_{US_3}$ | 71 | 18 |

The deadlines of the Update State parts are equal to their periods. The deadlines of the Calculate Output parts are initialized to $D_{CO_i} := T_i - C_{US_i}$. Assigning deadline-monotonic priorities and calculating response-times we get

| | $T$ | $D$ | $C$ | $P$ | $R$ |
|---|---|---|---|---|---|
| $\tau_{CO1}$ | 167 | 149 | 10 | 2 | 66 |
| $\tau_{US1}$ | 167 | 167 | 18 | 1 | 140 |
| $\tau_{CO2}$ | 100 | 82 | 10 | 4 | 38 |
| $\tau_{US2}$ | 100 | 100 | 18 | 3 | 56 |
| $\tau_{CO3}$ | 71 | 53 | 10 | 6 | 10 |
| $\tau_{US3}$ | 71 | 71 | 18 | 5 | 28 |

We set $D_{COi} := R_{COi}$, assign new deadline-monotonic priorities, and repeat the calculations:

| | $T$ | $D$ | $C$ | $P$ | $R$ |
|---|---|---|---|---|---|
| $\tau_{CO1}$ | 167 | 66 | 10 | 4 | 30 |
| $\tau_{US1}$ | 167 | 167 | 18 | 1 | 140 |
| $\tau_{CO2}$ | 100 | 38 | 10 | 5 | 20 |
| $\tau_{US2}$ | 100 | 100 | 18 | 2 | 66 |
| $\tau_{CO3}$ | 71 | 10 | 10 | 6 | 10 |
| $\tau_{US3}$ | 71 | 71 | 18 | 3 | 48 |

Repeating the procedure once more, we get no further improvements of the response times:

| | $T$ | $D$ | $C$ | $P$ | $R$ |
|---|---|---|---|---|---|
| $\tau_{CO1}$ | 167 | 30 | 10 | 4 | 30 |
| $\tau_{US1}$ | 167 | 167 | 18 | 1 | 140 |
| $\tau_{CO2}$ | 100 | 20 | 10 | 5 | 20 |
| $\tau_{US2}$ | 100 | 100 | 18 | 2 | 66 |
| $\tau_{CO3}$ | 71 | 10 | 10 | 6 | 10 |
| $\tau_{US3}$ | 71 | 71 | 18 | 3 | 48 |

The suggested choices of deadlines are thus $D_{CO1} = 30$, $D_{CO2} = 20$, and $D_{CO3} = 10$. Those deadlines actually minimize the criterion in Eq. (8), so we should be quite happy about the result. Running a new simulation reveals a significant improvement in performance for the lower-priority controllers:

| | Ref. | Impl. 1 | Impl. 2 | Impl. 3 |
|---|---|---|---|---|
| $J_1$ | 2.40 | 4.90 | 4.16 | 2.74 |
| $J_2$ | 1.35 | 4.27 | 1.96 | 1.71 |
| $J_3$ | 1.16 | 1.28 | 1.45 | 1.28 |

The improvement is also clearly visible in the close-up of the behavior of Controller 1 in Figure 5.

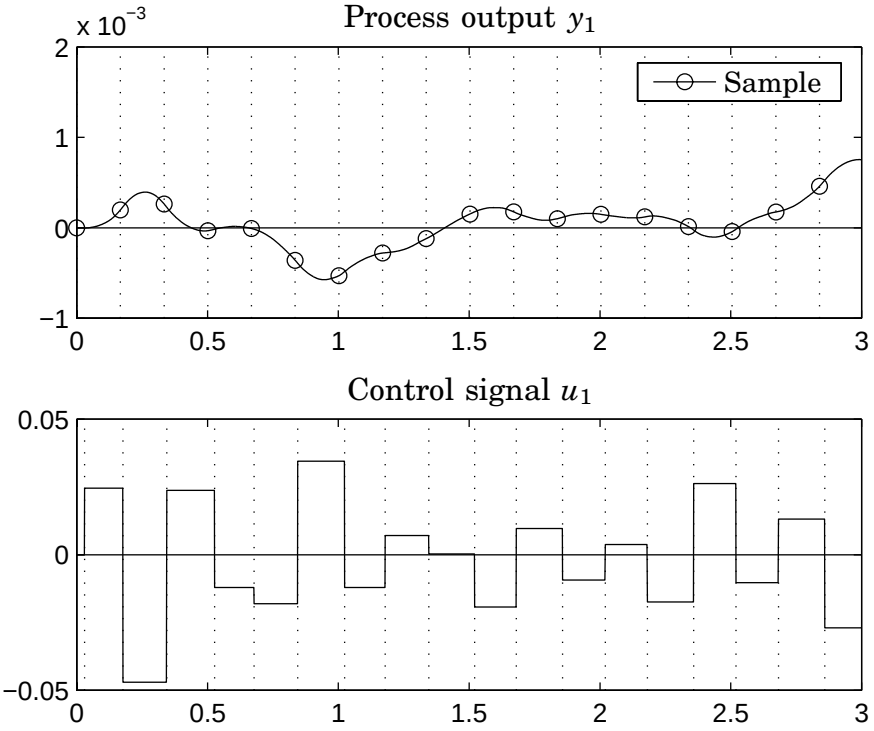



**Figure 5.** Close-up behavior of Controller 1 when improved scheduling is used (Implementation 3). Notice that the jitter is much smaller than in Figure 4.

## Implementation 4—Improved scheduling and fixed-delay compensation

Our last improvement consists of redesigning the controllers to compensate for the remaining computational delays. Again, we assume that the computation times remain the same. A final simulation shows that the performance has been improved even further:

| | Ref. | Impl. 1 | Impl. 2 | Impl. 3 | Impl. 4 |
|---|---|---|---|---|---|
| $J_1$ | 2.40 | 4.90 | 4.16 | 2.74 | 2.66 |
| $J_2$ | 1.35 | 4.27 | 1.96 | 1.71 | 1.46 |
| $J_3$ | 1.16 | 1.28 | 1.45 | 1.28 | 1.21 |

It can be noted that with improved scheduling and fixed-delay compensation, the performance of the three controllers all come quite close to the reference performance.

## 6. Conclusions

It has been shown that it is possible to improve the performance of digital controllers by using more detailed timing analysis. By treating the main parts of a control algorithm as two subtasks, and by scheduling them appropriately, it is often possible to reduce the computational delay significantly. The remaining delay could be fixated, allowing for fixed-delay compensation to be used.

The results tell us that digital controllers should be designed with the implementation as periodic tasks in mind. The selection of task timing attributes, such as periods and deadlines, affect both control performance and schedulability. It is also necessary to have good estimates of the worst-case execution times of the different parts of the algorithm. It would be useful to have a design tool for digital controllers that took all of these considerations into question.

The need for more elaborate simulation tools for real-time control systems is also evident. In order to capture the true behavior of the such systems, the simulation software must include models of the physical processes, the controllers, and the real-time kernel.

## 7. Acknowledgments

This work has been performed as a part of the ARTES project “Integrated Control and Scheduling”.

## 8. References

Åström, K. J. and B. Wittenmark (1997): *Computer-Controlled Systems*, third edition. Prentice Hall.

Audsley, N., K. Tindell, and A. Burns (1993): "The end of the line for static cyclic scheduling?" In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, pp. 36–41.

Burns, A., K. Tindell, and A. J. Wellings (1994): "Fixed priority scheduling with deadlines prior to completion." In *Proceedings of the 6th Euromicro Workshop on Real-Time Systems*, pp. 138–142.

Gerber, R. and S. Hong (1993): "Semantics-based compiler transformations for enhanced schedulability." In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, pp. 232–242.

Gerber, R. and S. Hong (1997): "Slicing real-time programs for enhanced schedulabilty." *ACM Transactions on Programming Languages and Systems*, **19:3**, pp. 525–555.

Gonzalez Härbour, M., M. H. Klein, and J. P. Lehoczky (1994): "Timing analysis for fixed-priority scheduling of hard real-time systems." *IEEE Transactions on Software Engineering*, **20:1**, pp. 13–28.

Gustafsson, K. and P. Hagander (1991): "Discrete-time LQG with cross-terms in the loss function and the noise description." Report TFRT-7475.

Joseph, M. and P. Pandya (1986): "Finding response times in a real-time system." *The Computer Journal*, **29:5**, pp. 390–395.

Kao, B. and H. Garcia-Molina (1993): "Deadline assignment in a distributed soft real-time system." In *Proceedings of the 13th International Conference on Distributed Computing Systems*.

Kim, B. K. (1998): "Task scheduling with feedback latency for real-time control systems." In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*, pp. 37–41.

Leung, J. Y. T. and J. Whitehead (1982): "On the complexity of fixed-priority scheduling of periodic, real-time tasks." *Performance Evaluation*, **2:4**, pp. 237–250.

Liu, C. L. and J. W. Layland (1973): "Scheduling algorithms for multiprogramming in a hard real-time environment." *Journal of the ACM*, **20:1**, pp. 40–61.

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1996): "Stochastic analysis and control of real-time systems with random time delays." In *IFAC'96, Preprints 13th World Congress of IFAC*. San Francisco, California.

Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin (1996): "On task schedulability in real-time control systems." In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 13–21.

# Paper 3

# A Matlab Toolbox for Real-Time and Control Systems Co-Design

**Johan Eker and Anton Cervin**

**Abstract**

The paper presents a Matlab toolbox for simulation of real-time control systems. The basic idea is to simulate a real-time kernel in parallel with continuous plant dynamics. The toolbox allows the user to explore the timely behavior of control algorithms, and to study the interaction between the control tasks and the scheduler. From a research perspective, it also becomes possible to experiment with more flexible approaches to real-time control systems, such as feedback scheduling. The importance of a more unified approach for the design of real-time control systems is discussed. The implementation is described in some detail and a number of examples are given.

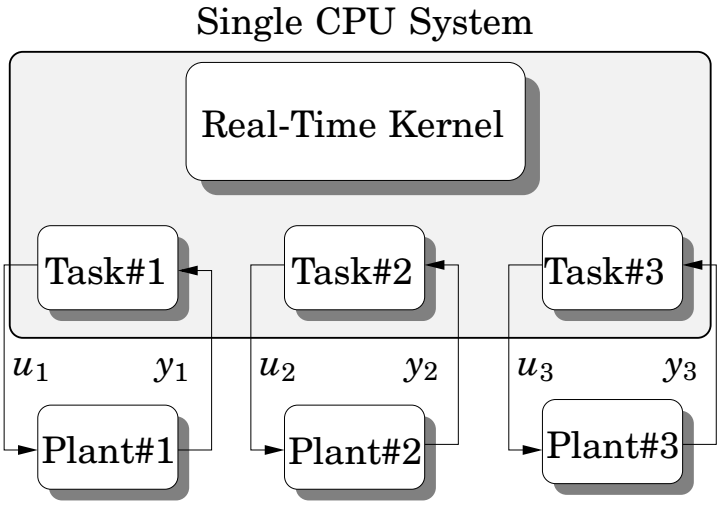



**Figure 1.** Several control loops execute concurrently on one CPU. The interaction between the control tasks will affect the control performance.

## 1. Introduction

Real-time control systems are traditionally designed jointly by two different types of engineers. The control engineer develops a model for the plant to be controlled, designs a control law and tests it in simulation. The real-time systems engineer is given a control algorithm to implement, and configures the real-time system by assigning priorities, deadlines, etc.

The real-time systems engineer usually regards control systems as hard real-time systems, i.e. deadlines should never be missed. The control engineer on the other hand expects the computing platform to be predictive and support equidistant sampling. In reality none of the assumptions are necessarily true. This is even more obvious in the case where several control loops are running on the same hardware unit. The controllers will interact with each other since they are sharing resources such as CPU, network, analog/digital converters, etc. see Figure 1.

A new interdisciplinary approach is currently emerging where control and real-time issues are discussed at all design levels. One of the first papers that dealt with co-design of control and real-time systems was [Seto *et al.*, 1996], where the sampling rates for a set of con-

trollers sharing the same CPU are calculated using standard control performance metrics. Control and scheduling co-design is also found in [Ryu *et al.*, 1997], where the control performance is specified in terms of steady state error, overshoot, rise time, and settling time. These performance parameters are expressed as functions of the sampling period and the input-output latency. A heuristic iterative algorithm is proposed for the optimization of these parameters subject to schedulability constraints.

Good interaction between control theory and real-time systems theory opens up for a unified approach and more integrated algorithms. Scheduling parameters could for example be adjusted automatically on-line by a kernel supervisor. Such a setup would allow much more flexible real-time control systems than those available today. Ideas on adaption of scheduling parameters are for example found in [Abdelzaher *et al.*, 1997] and [Stankovic *et al.*, 1999].

The development of algorithms for co-design of control and real-time systems requires new theory and new tools. This paper presents a novel simulation environment for co-design of control systems and real-time systems within the Matlab/Simulink environment. The advantages of using Matlab for this purpose are many. Matlab/Simulink is commonly used by control engineers to model physical plants, to design control systems, and to evaluate their performance by simulations. A missing piece in the simulations, however, has been the actual execution of the controllers when implemented as tasks in a real-time system. On the other hand, most of the existing tools for task simulations, for instance STRESS [Audsley *et al.*, 1994], DRTSS [Storch and Liu, 1996], and the simulator in [Ancilotti *et al.*, 1998], give no support for the simulation of continuous dynamics. Not much work has previously been done on mixed simulations of both process dynamics, control tasks, and the underlying real-time kernel. An exception is [Liu, 1998], where a single control task and a continuous plant was simulated within the Ptolemy II framework.

The simulator proposed in this paper is designed for simultaneous simulation of continuous plant dynamics, real-time tasks, and network traffic, in order to study the effects of the task interaction on the control performance.

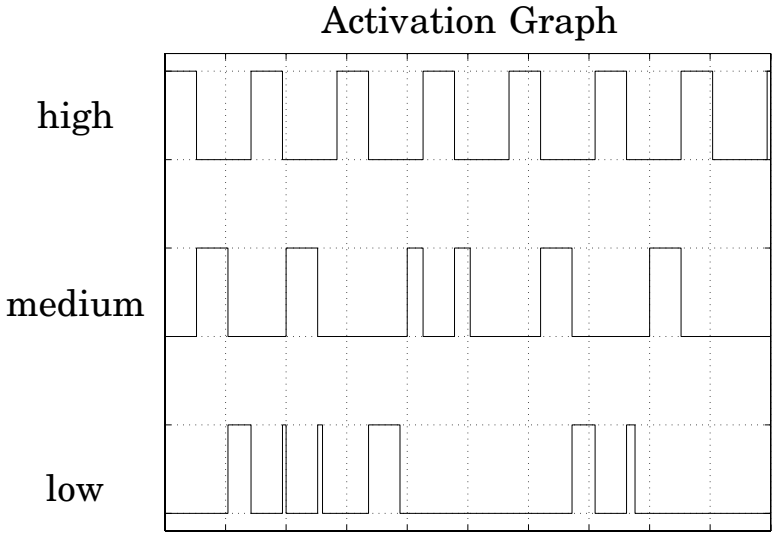



**Figure 2.** The activation graph for three control tasks, with fixed priorities (high, medium, low), running in a pre-emptive kernel. The execution times are the same for all three processes.

## 2. The Basic Idea

The interaction between control tasks executing on the same CPU is usually neglected by the control engineer. It is however the case that having a set of control tasks competing for the computing resources will lead to various amounts of delay and jitter for different tasks. Figure 2 shows an example where three control tasks with the same execution times but different periods are scheduled using rate-monotonic priorities. In this case the schedule does not tell the whole story. In the example, the actual control delay (the delay from reading the input signal until writing a new output signal) for the low priority task varies from one to three times the execution time. Intuitively, this delay will affect the control performance, but how much, and how can we investigate this?

To study how the execution of tasks affects the control performance we must simulate the whole system, i.e. both the continuous dynamics of the controlled plant and the execution of the controllers in the CPU. We need not simulate the execution of the controller code on instruction

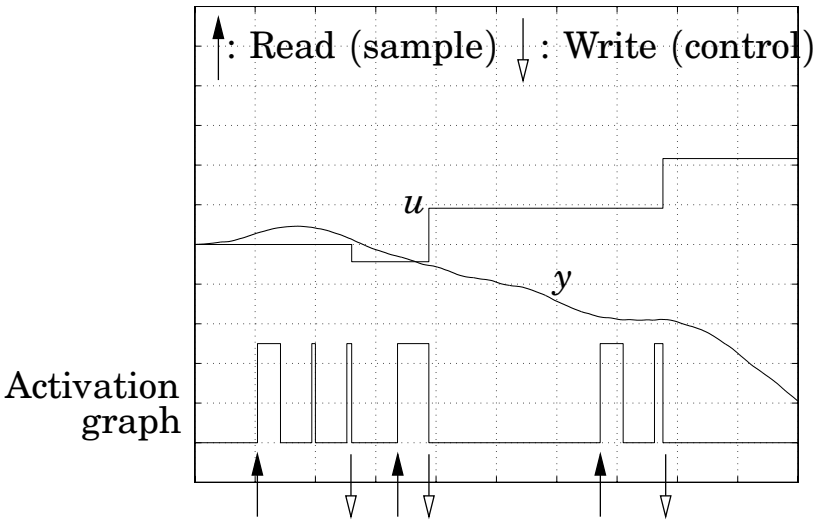



**Figure 3.** This is how the low priority task from Figure 2 interacts with its plant. ($u$ is the control signal, $y$ is the measurement signal.)

level. In fact, it is enough to model the timely aspects of the code that are of relevance to other tasks and to the controlled plant. This includes computational phases, input and output actions, and blocking of common resources (other than the CPU).

Figure 3 shows the activation graph for the low priority task from Figure 2 and how it interacts with the continuous plant. The controller samples the continuous measurement signal from the plant ($y$) and writes new control outputs ($u$).

Figure 4 provides a schematic view of how we simulate the system. A model of a real-time kernel handles the scheduling of the control tasks and is also responsible for properly interfacing the tasks with the physical environment. The outputs from the kernel model, i.e. the control signals, are piecewise constant. The plant dynamics and the plant outputs, i.e. the measurement signals, are continuous.

## 3. The Simulation Model

The heart of the toolbox is a Simulink block (an S-function) that simulates a tick-driven preemptive real-time kernel. The kernel maintains a

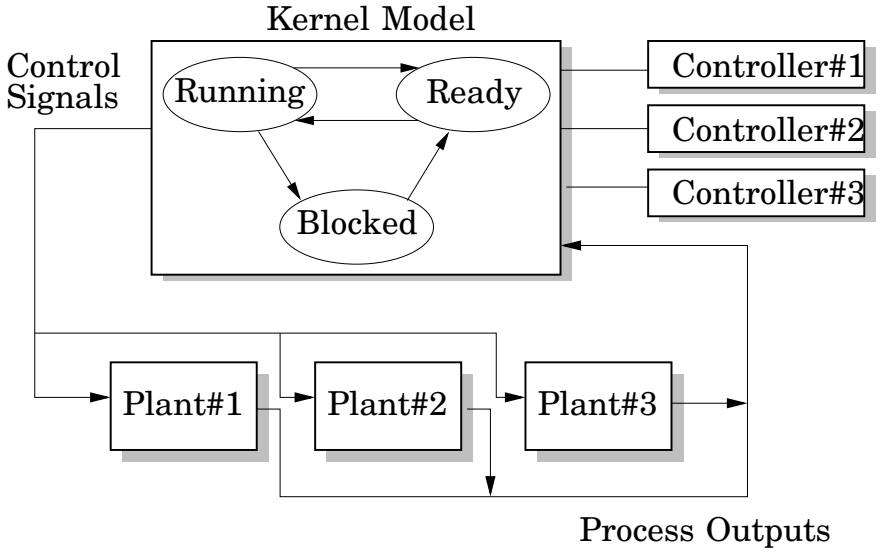



**Figure 4.** Schematic view of the simulation setup. The controllers are tasks executing in a simulated pre-emptive kernel. The controllers and the control signals are discrete while the plant dynamics and the plant output are continuous. The continuous signals from the plants are sampled by the control tasks.

number of data structures that are commonly found in a real-time kernel: a set of task records, a ready queue, a time queue, etc. At each clock tick, the kernel is responsible for letting the highest-priority ready task, i.e. the running task, execute in a virtual CPU. The scheduling policy used is determined by a priority function, which is a function of the attributes of a task. For instance, a priority function that returns the period of a task implements rate-monotonic scheduling, while a function that returns the absolute deadline of a task implements earliest-deadline-first scheduling. There currently exist predefined priority functions for rate-monotonic (RM), deadline-monotonic (DM), arbitrary fixed-priority (FP), and earliest-deadline-first (EDF) scheduling. The user may also write his own priority function that implements an arbitrary scheduling policy.

The execution model used is similar to the *live task model* described in [Storch and Liu, 1996]. During a simulation, the kernel executes user-defined code, i.e. Matlab functions, that have been associated with the different tasks. A code function returns an execution time estimate, and the task is not allowed to resume execution until the same amount of time has been consumed by the task in the virtual CPU.

### The Task

Each task in the kernel has a set of basic attributes: A name, a list of code segments to execute, a period, a release time, a relative deadline, and the remaining execution time to be consumed in the virtual CPU. Some of the attributes, such as the release time and the remaining execution time, are constantly updated by the kernel during a simulation. The other attributes, such as the period and the relative deadline, remain constant unless they are explicitly changed by kernel function calls from the user code.

### The Code

The local memory of a task is represented by two local, user-defined data structures `states` and `parameters`. The states may be changed by the user code, while the parameters remain constant throughout the execution.

To capture the timely behavior of a task, the associated code is divided into one or several code segments, see Figure 5. The execution

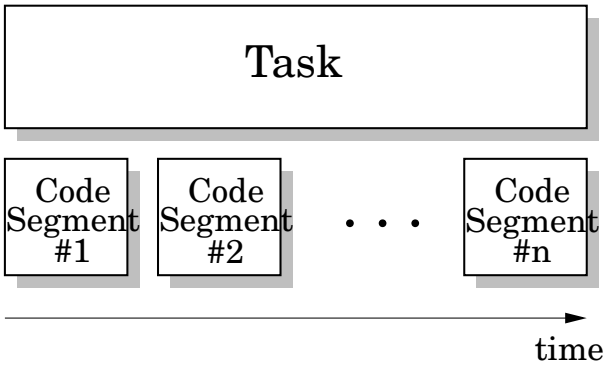



**Figure 5.** The execution structure of a task. The flexible structure supports data dependent execution times and advanced scheduling techniques.

time of a segment is determined dynamically at its invocation. Normally, the segments are executed in order, but this may be changed by kernel function calls from the user code.

On a finer level, actual execution of statements in a code segment can only occur at two points: at the very beginning of the code segment (in the `enterCode` part) or at the very end of the code segment (in the `exitCode` part), see Figure 6. Typically, reading of input signals,

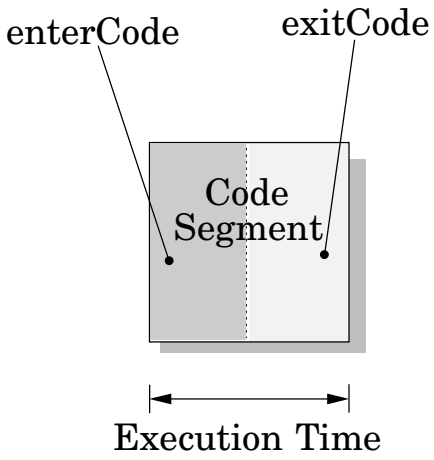
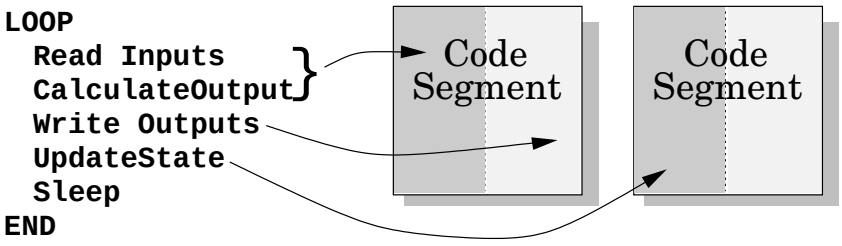



**Figure 6.** A code segment is divided in two parts: the `enterCode` part and the `exitCode` part.

locking of resources, and calculations are performed in the `enterCode` part. Writing of output signals, unlocking of resources, and other kernel function calls are typically performed in the `exitCode` part. The following examples illustrate how code segments can model real-time tasks.

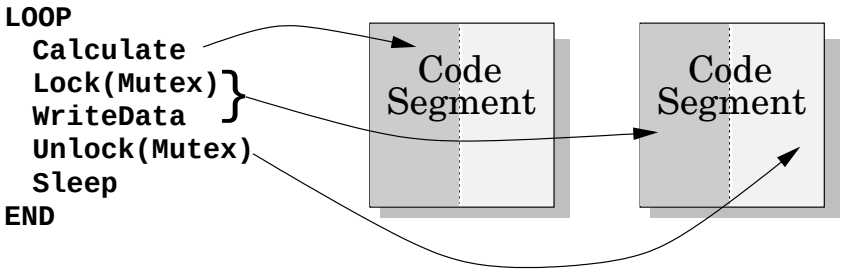
EXAMPLE 1
A task implementing a control loop can often be divided into two parts: one that calculates a new control signal and one that updates the controller states. The first part, called Calculate Output, has a hard timing constraint and should finish as fast as possible. The timing requirement for the second part, Update State, is that it must finish before the next invocation of the task. Two code segments are appropriate to model the task:




The `enterCode` of the first segment contains the reading of the measurement signals, and the calculation of a new control signal. In the same segment, in `exitCode`, the control signal is sent to the actuator.

The control delay of the controller is thus equal to the execution time of the first segment. The `enterCode` of the second code segment contains Update State. When the last segment has completed, the task is suspended until the next period by the kernel. □

EXAMPLE 2
The structure of a periodic task that first calculates some data and then writes to a common resource could look like this:




Again, two code segments can capture the timely behavior. The first code segment contains the `Calculate` statement, located in the `enterCode` part. The `enterCode` part of the second code segment contains the `Lock(Mutex)` and `WriteData` statements, while `exitCode` contains the `Unlock(Mutex)` statement. When the last segment has completed, the task is suspended until the next period by the kernel. □

## 4. Using the Simulator

From the user's perspective, the toolbox offers a Simulink block that models a computer with a real-time kernel. Connecting the Computer block's inputs and outputs (representing for instance A-D and D-A converters) to the plant, a complete computer-controlled system is formed, see Figure 7.

The plant dynamics may have to be controlled by several digital controllers, each implemented as a periodic control task in the computer. Besides the controllers, other tasks could be executing in the computer, for instance planning tasks, supervision tasks, and user communication tasks.

Opening up the Computer block, the user may study detailed information about the execution of the different tasks, see Figure 8. It is

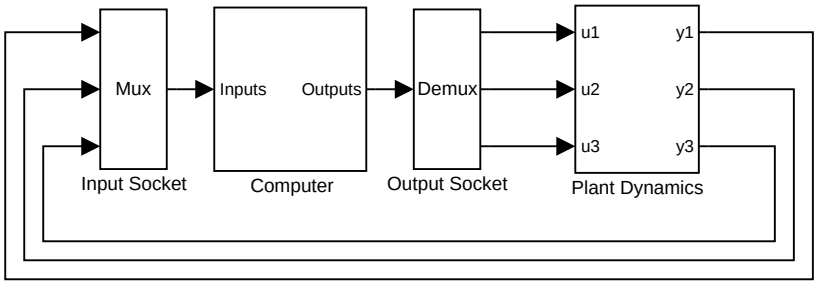



**Figure 7.** The simulation environment offers a Simulink Computer block that can be connected to the model of the plant dynamics.

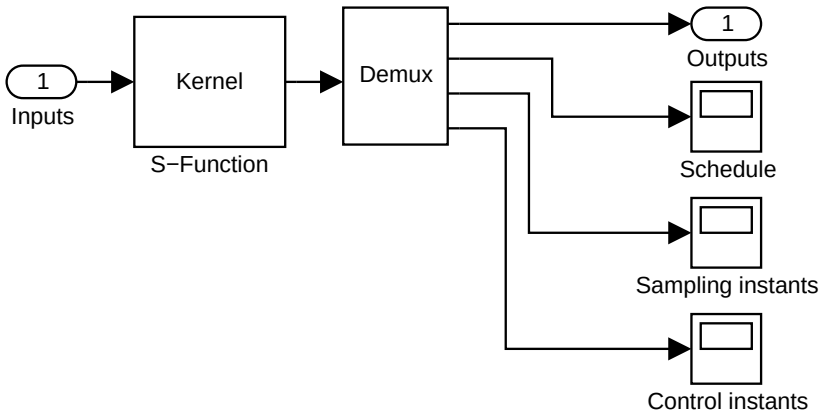



**Figure 8.** Inside the Computer block, it is possible to study details about the execution of different tasks.

for instance possible to study the schedule, i.e. a plot that shows when different tasks are executing, at run-time. Further statistics about the execution is stored in the workspace and may be analyzed when the simulation has stopped.

## Controller Implementation

It is highly desirable that the design of the kernel is flexible and allows components to be reused and replaced. Much effort has been put into writing control algorithms in Matlab, and these algorithms should be straightforward to reuse. In the toolbox, a control algorithm can be implemented as a code segment with the following format:

```
function [exectime,states] = ...
```

```
     myController(flag,states,params)
switch flag,
 case 1, % enterCode
   y = analogIn(params.inChan);
   states.u = <place control law here>
   exectime = 0.002;
  case 2, % exitCode
   analogOut(params.outChan,states.u)
end
```

The input variables to `myController` are the state variables `states`, and the controller parameters `params`. The `flag` is used to indicate whether the `enterCode` or the `exitCode` part should be executed. If the `enterCode` part is executed, the function returns the execution time estimate `exectime` and the new state variables. The control signal is sent to the plant in the `exitCode` part.

***Remark*** The output signal $u$ is not normally regarded as a state variable in a controller. In this example, however, we need to store the value of $u$ between two invocations of the `myController` function.

### Configuration

Before a simulation can start, the user must define what tasks that should exist in the system, what scheduling policy should be used, whether any common resources exist, etc. The initialization is performed in a Matlab script.

EXAMPLE 3
Three dummy tasks are initialized in the script below. The tick-size of the kernel is set to 0.001 s and the scheduling type is set to rate monotonic. The dummy code segment `empty` models a task that computes nothing for a certain amount a time. Each task is assigned a period, and a deadline which is equal to the period.

```
function rtsys = rtsys_init

% 1 = RM, 2 = DM, 3 = Arbitrary FP, 4 = EDF
rtsys.st = 1;
rtsys.tick_size = 0.001;
```

```
T = [0.10 0.08 0.06];  % Task Periods
D = [0.10 0.08 0.06];  % Deadlines
C = [0.02 0.02 0.02];  % Computation times

rtsys.Tasks = {}
code1 = code('empty',[],C(1))
code2 = code('empty',[],C(2))
code3 = code('empty',[],C(3))

rtsys.Tasks{1}=task('Task1',code1,T(1),D(1));
rtsys.Tasks{2}=task('Task2',code2,T(2),D(2));
rtsys.Tasks{3}=task('Task3',code3,T(3),D(3));
```

The initialization script is given as a parameter to a Computer block in a Simulink model. Simulating the model for one second produces, among other things, the schedule plot shown in Figure 9. □

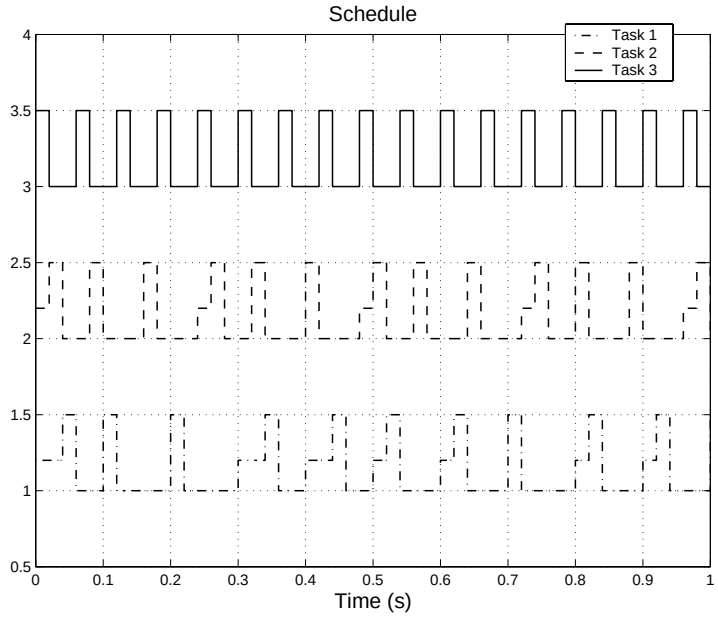



**Figure 9.** The schedule resulting from the simulation in Example 3. The bottom graph shows when Task 1 is running (high), ready (medium) or blocked (low). The other two graphs represent Task 2 and Task 3.

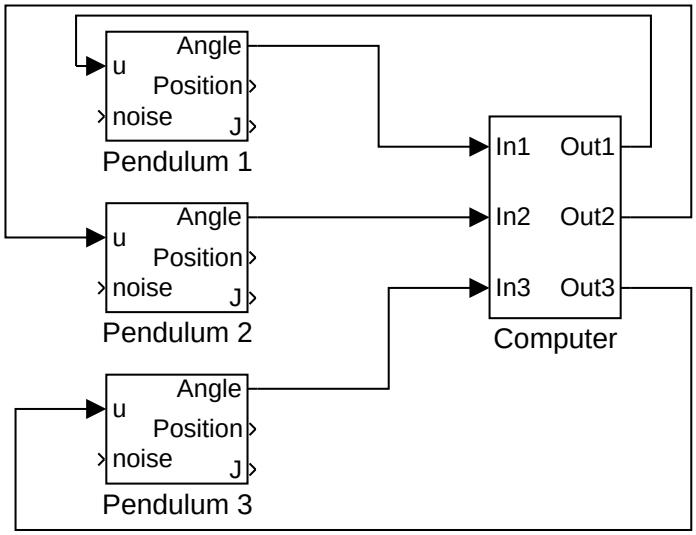



**Figure 10.** A Simulink diagram where three continuous pendulum models are connected with the real-time kernel. The simulation result from this system is both the activation graph and the output from the continuous plants.

## Connecting a Continuous Plant

Figure 10 shows a Simulink diagram where a Computer block is connected to three pendulum models. The continuous plant models are described by other Simulink blocks.

A real-time system with three control loops are created in Example 4. One code segment named `myController` is associated with each task.

EXAMPLE 4

```
function rtsys = rtsys_init
% Scheduling type, 1=RM, 2=DM, 3=FP, 4=EDF
rtsys.st=1;
rtsys.tick_size=0.001;

% Desired bandwidths
omega=[3 5 7];
% Sampling periods
T=[0.167 0.100 0.071];
for i=1:3
  % Design controller
  params=ctrl_design(omega(i),T(i));
```

```
  % Initialize control code
  states.xhat=[0 0]';
  % The controller reads from input i
  params.inChan=i;
  % The controller writes to output i
  params.outChan=i;
  sfbcode=code('myController',states,params);
  % Create task
  tasks{i}=task(['Task 'num2str(i)],...
                sfbcode, T(i), T(i));
end
rtsys.tasks=tasks;
```

□

The outputs from a simulation of this system are a set of continuous signals from the plants together with an activation graph. It is hence possible to evaluate the performance of the real-time systems both from a control design point of view and from a scheduling point of view.

## 5. A Co-Design Example

Using the simulator, it is possible to evaluate different scheduling policies and their effect on the control performance. Again consider the problem of controlling three inverted pendulums using only one CPU, see Figure 11. The inverted pendulum may be approximated by the following linear differential equation

$$\ddot{\theta} = \omega_0^2 \theta + \omega_0^2 u/g,$$

where $\omega_0 = \sqrt{g/l}$ is the natural frequency for a pendulum with length $l$. The goal is to minimize the angles, so for each pendulum we want to minimize the accumulated quadratic loss function

$$J_i(t) = \int_0^t \theta_i^2(s) ds. \tag{1}$$

Three discrete-time controllers with state feedback and observers are designed. Sampling periods for the controllers are chosen according to

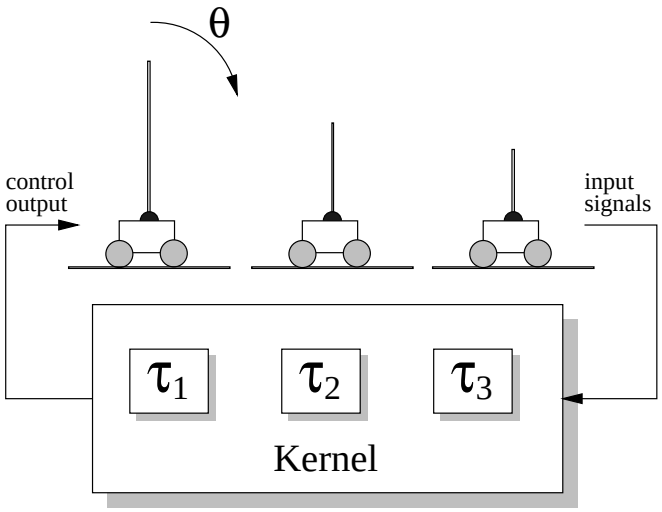



**Figure 11.** The setup described in Section 5. Three inverted pendulums with different lengths are controlled by three control tasks running on the same CPU.

the desired bandwidths (3, 5 and 7 rad/s respectively) and the CPU resources available. The execution times of the control tasks, $\tau_i$, are all 28 ms, and the periods are $T_1 = 167$ ms, $T_2 = 100$ ms, and $T_3 = 71$ ms.

Task objects are created according to Example 4. Also, similar to Example 1, the control algorithm is divided into two code segments, Calculate Output and Update State, with execution times of 10 and 18 ms respectively.

In a first simulation, the control tasks are assigned constant priorities according to the rate-monotonic schema, and the two code segments execute at the same priority. In a second simulation, the Calculate Output code segments are assigned higher priorities than the Update State segments, according to iterative priority/deadline assignment algorithm suggested in [Cervin, 1999]. The accumulated loss function for the slow pendulum ($T_1 = 167$ ms) is easily recorded in the Simulink model, and the results from both simulations are shown in Figure 12.

A close-up inspection of the schedule produced in the second simulation is shown in Figure 13. It can be seen that the faster tasks sometimes allow the slower tasks to execute, and in this way the control delays in the slower controllers are minimized. The result is a smaller accumulated loss, and thus, better control performance.

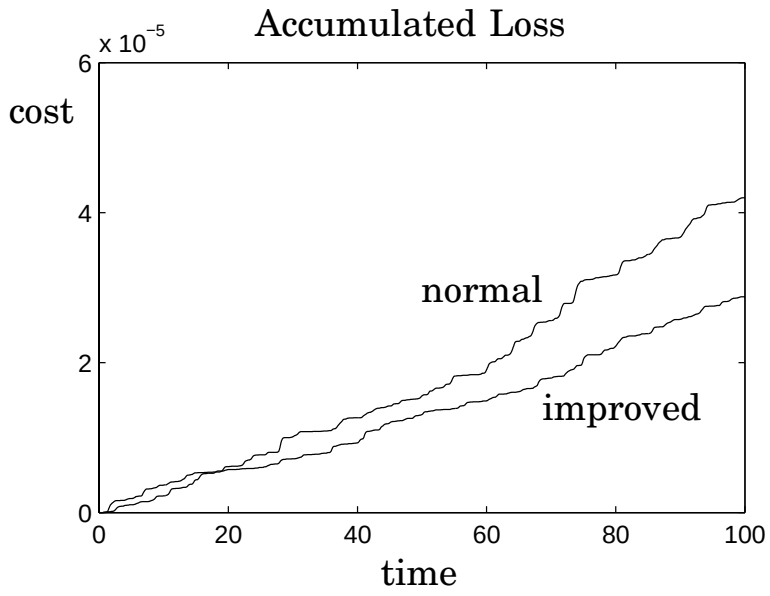



**Figure 12.** The accumulated loss, see Equation (1), for the low priority controller using normal and improved scheduling. The cost is substantially reduced under the improved scheduling.

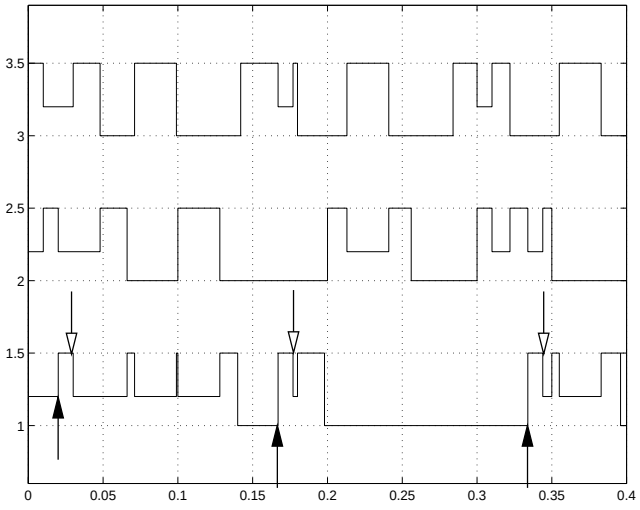



**Figure 13.** The activation graph when the improved scheduling strategy is used. Note that the control delay for the low priority task is approximately the same as for the other tasks.

# 6. Simulation Features

Further features of the toolbox are the support for common real-time primitives like mutual exclusion locks, events (also known as condition variables), and network communication blocks.

## Locks and Events

The control tasks do not only interact with each through the use of same CPU, but also due to sharing other user-defined resources. The kernel allows the user to define monitors and events, for implementing complex dependencies between the task. The syntax and semantics of the `mutex` and `event` primitives are demonstrated by a small example. Two tasks `Regul` and `OpCom` are sharing a variable called `data`. To ensure mutual exclusion the variable is protected by the mutex variable `M1`. Associated with `M1` is a monitor event called `E1`. The `Regul`-task consists of two code segments called `rseg1` and `rseg2`, that are shown in Example 5. Each time the `Regul`-task is released it tries to lock the monitor variable `M1`. Once the monitor is locked it may access the shared data. If the value of the `data`-variable is less than two, it waits for the event `E1` to occur.

EXAMPLE 5

```
function [exectime, states] = ...
            rseg1(flag,states,params)
 switch flag,
  case 1,   % enterCode
    if lock('M1')
      data = readData('M1');
      if data < 2
        await('E1');
        exectime = 0;
      else
        exectime = 0.003;
      end
    else
      exectime = 0;
    end
  case 2,   % exitCode
    unlock('M1');
```

```
 end

function [exectime,states] = ...
            rseg2(flag,states,params)
 switch flag,
  case 1,   % enterCode
    y = analogIn(params.inChan);
    states.u = -50*y;
    exectime = 0.003;
  case 2,   % exitCode
    analogOut(params.outChan,states.u)
 end
```

□

The locks and the events are designed similarly to how monitors and events are implemented in a standard real-time kernel, i.e. using queues associated with the monitor for storing tasks blocking on locks or events. The execution time used for trying, but failing to lock, is in the example above zero.

### Network Blocks

It is possible to include more than one Computer block in a Simulink model, and this opens up the possibility to simulate much more complex systems than the ones previously discussed. Distributed control systems may be investigated. Furthermore, fault-tolerant systems, where, for redundancy, several computers are used for control, could also be simulated. In order to simulate different communication protocols in such systems, communication blocks for sending data between the different Computer blocks are needed. Figure 14 shows a simulation setup for a simple distributed system where the controller, and the actuator and sensor, are located at different places. Besides the kernel blocks there is a network block for communication. The network block is event driven, and each time any of the input signals change, the network is notified. The user needs to implement the network protocol, since the blocks simply provides the mechanisms for sending data between kernels.

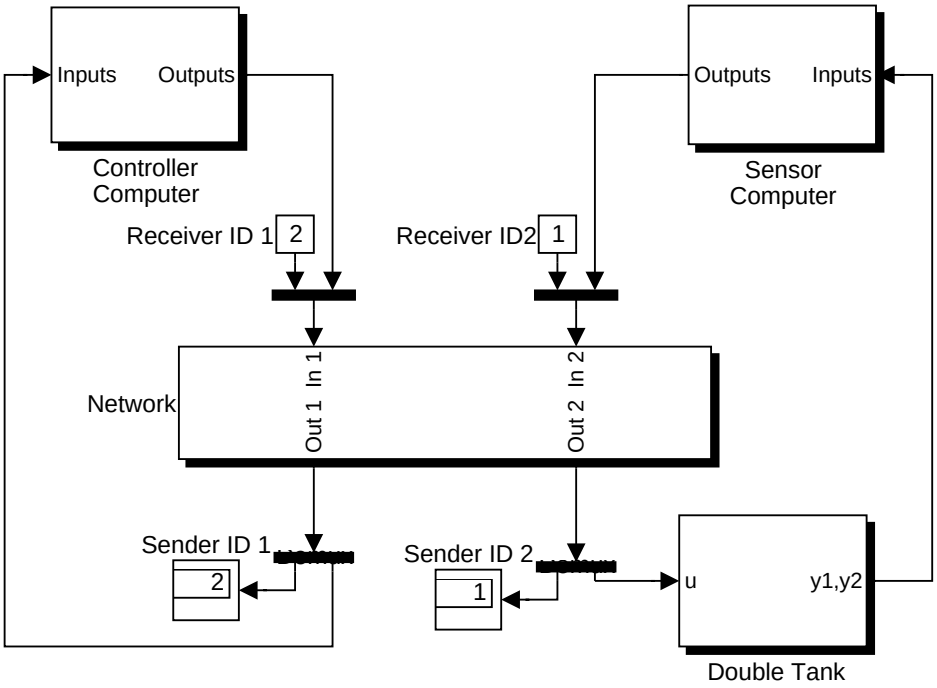



**Figure 14.** A distributed control system where the sensor and the CPU are dislocated. The controller and the sensor are implemented as periodic tasks running on separate CPUs.

## High Level Task Communication

One of the main reasons for designing the kernel and the network blocks was to facilitate the simulation of flexible embedded control system, i.e. systems where the task set is allowed to change dynamically and the underlying real-time system must compensate for this. From a control theory perspective we might say that we want to design a feedback connection between the control tasks and the scheduler, see Figure 15. To support the simulation of feedback scheduling, there must be ways for the tasks and the task scheduler to communicate. Therefore the kernel also supports system-level message passing between tasks.

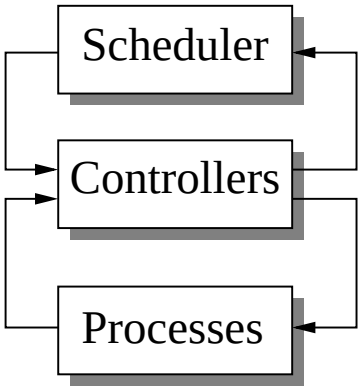



**Figure 15.** The control tasks and the task scheduler are connected in a feedback loop.

## 7. Conclusions

This paper presented a novel simulator for the co-design of real-time systems and control systems. The main objective is to investigate the consequences on control performance of task interaction on kernel level. This way, scheduling algorithms may be evaluated from a control design perspective. We believe that this is an issue of increasing importance. There are many more things to be implemented and improved before this block set will become a truly useful tool. Currently the kernel is tick-based, and has little support for external interrupts. The next version of the kernel block will probably be event-based in order to better support interrupts and event-based sampling. To make the simulations more realistic, the scheduler itself could also be modeled as a task that consumes CPU time. This would also enhance the possibilities for the user to implement new scheduling strategies for control tasks.

## 8. Acknowledgments

This work was sponsored by the Swedish national board of technical development (NUTEK) and by the Swedish network for real-time research and education (ARTES).

## 9. References

Abdelzaher, T., E. Atkins, and K. Shin (1997): "QoS negotiation in real-time systems, and its application to flight control." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Ancilotti, P., G. Buttazzo, M. D. Natale, and M. Spuri (1998): "Design and programming tools for time critical applications." *Real-Time Systems*, **14:3**, pp. 251–269.

Audsley, N., A. Burns, M. Richardson, and A. Wellings (1994): "STRESS—A simulator for hard real-time systems." *Software—Practice and Experience*, **24:6**, pp. 543–564.

Cervin, A. (1999): "Improved scheduling of control tasks." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 4–10. York, England.

Liu, J. (1998): "Continuous time and mixed-signal simulation in Ptolemy II." Technical Report UCB/ERL Memorandum M98/74. Department of Electrical Engineering and Computer Science, University of California, Berkeley.

Ryu, M., S. Hong, and M. Saksena (1997): "Streamlining real-time controller design: From performance specifications to end-to-end timing constraints." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Seto, D., J. Lehoczky, L. Sha, and K. Shin (1996): "On task schedulability in real-time control systems." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Stankovic, J. A., C. Lu, S. H. Son, and G. Tao (1999): "The case for feedback control real-time scheduling." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 11–20.

Storch, M. F. and J. W.-S. Liu (1996): "DRTSS: A simulation framework for complex real-time systems." In *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium*, pp. 160–169.

# Paper 4

# Feedback Scheduling of Control Tasks

**Anton Cervin and Johan Eker**

**Abstract**

The paper presents a feedback scheduling mechanism in the context of co-design of the scheduler and the control tasks. We are particularly interested in controllers where the execution time may change abruptly between different modes, such as in hybrid controllers. The proposed solution attempts to keep the CPU utilization at a high level, avoid overload, and distribute the computing resources evenly among the tasks. The feedback scheduler is implemented as a periodic or sporadic task that assigns sampling periods to the controllers based on execution-time measurements. The controllers may also communicate feedforward mode-change information to the scheduler. As an example, we consider hybrid control of a set of double-tank processes. The system is evaluated, from both scheduling and control performance perspectives, by co-simulation of controllers, scheduler, and tanks.

## 1. Introduction

There is currently a trend towards more flexible real-time control systems. By combining scheduling theory and control theory, it is possible to achieve higher resource utilization and better control performance. To achieve the best results, co-design of the scheduler and the controllers is necessary. This research area is only beginning to emerge, and there is still a lot of theoretical and practical work to be done, both in the control community and in the real-time community.

Control tasks are generally viewed by the scheduling community as *hard* real-time tasks with *fixed* sampling periods and *known* WCETs. Upon closer inspection, neither of these assumptions need necessarily be true. For instance, many control algorithms are quite robust against variations in sampling period and response time. Controllers can be designed to switch between different modes with different execution times and perhaps also different sampling intervals. It is also possible to consider control systems that are able to do a trade-off between the available computation time and the control loop performance.

As an example throughout this paper, we study the problem of scheduling a set of hybrid-control tasks. Such tasks are good examples of tasks that do not really meet the assumptions commonly made in the scheduling theory. A hybrid controller switches between different modes, which may have very different execution-time characteristics. Utilizing only worst-case execution-time (WCET) estimates in the scheduling design can result in very low resource utilization, slow sampling, and low control performance. On the other hand, if instead, average-case execution-time estimates are used in the scheduling design, the CPU may experience transient overloads during run-time. This, again, can result in low control performance, and even temporary shut-down of the controllers.

In this work, we present a feedback scheduler for control tasks that attempts to keep the CPU utilization at a high level, avoid overload, and distribute the computing resources evenly among the tasks. While we want to keep the number of missed deadlines as low as possible, control performance is our primary objective. Thus, control tasks, in our view, fall in a category somewhere between hard and soft real-time tasks. The known-WCET assumption is relaxed by the use of feedback from execution-time measurements. We also introduce feedforward to

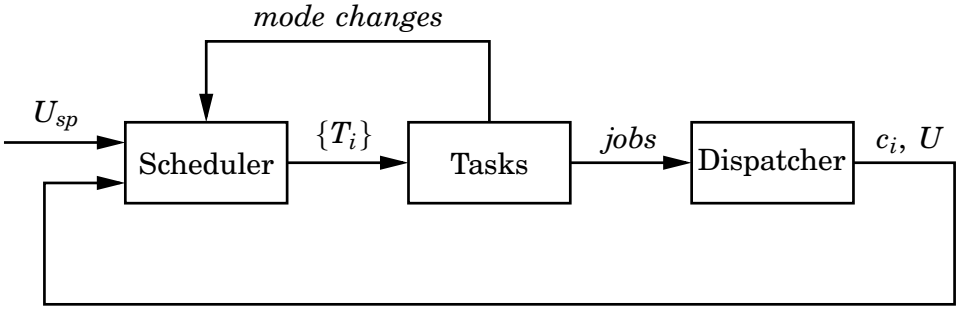



**Figure 1.** The feedback scheduling structure.

further improve the regulation of the utilization.

The structure of the feedback scheduler is shown in Figure 1. A set of control tasks generate jobs that are fed to the run-time dispatcher. The scheduler gets feedback information about the actual execution time, $c_i$, of the jobs (it is assumed that this information can be provided by the real-time operating system). It also gets feedforward information from control tasks that are about to switch mode. This way, the scheduler can proact rather than react to sudden changes in the workload. The scheduler tries to keep the utilization, $U$, as close as possible to the utilization setpoint, $U_{sp}$. This is done by manipulating the sampling periods, $\{T_i\}$. The choice of utilization setpoint depends on the scheduling policy of the dispatcher, and on the sensitivity of the controllers to missed deadlines. Notice that the well-known, guaranteed utilization bounds of 100% for earliest-deadline-first (EDF) scheduling and 69% for fixed-priority scheduling [Liu and Layland, 1973] are not valid in this context, since the assumptions about known, fixed WCETs and fixed periods are violated.

The calculated task periods should reflect the relative importance of the different control tasks. One possibility is to assign nominal sampling periods to the controllers off-line. The feedback scheduler can then do linear rescaling of the task periods to achieve the desired utilization. The controllers are informed of the new sampling periods and may adjust their parameters if necessary. Other possibilities could include on-line optimization of a control performance criterion over the task periods, subject to the utilization constraint. The feedback scheduler is in the end also implemented as a periodic or sporadic task that consumes computing resources. There is a fundamental trade-off be-

tween the time that should be spent doing scheduling, and the time left over for control computations.

### Related Work

The related work falls into three categories. The first one is the field of integrated control system and real-time system design. In [Seto *et al.*, 1996], sampling period selection for a set of control tasks is considered. The performance of a task is given as a function of its sampling frequency, and an optimization problem is solved to find a set of optimal task periods. Co-design of real-time control systems is also considered in [Ryu *et al.*, 1997], where the performance parameters are expressed as functions of the sampling periods and the input-output latencies. [Shin and Meissner, 1999] deals with on-line rescaling and relocation of control tasks in a multi-processor system. A simulator for co-design is introduced in [Eker and Cervin, 1999]. It facilitates co-simulation of control task execution, scheduling, and continuous plant dynamics.

The second area of related work is on quality-of-service (QoS) aware real-time software, where a system's resource allocation is adjusted on-line in order to maximize the performance in some respect. In [Li and Nahrstedt, 1998] a general framework is proposed for controlling the application requests for system resources using the amount of allocated resources for feedback. It is shown that a PID controller can be used to bound the resource usage in a stable and fair way. A resource allocation scheme called Q-RAM is presented in [Rajkumar *et al.*, 1997]. Several tasks are competing for finite resources, and each task is associated with a utility value, which is a function of the assigned resources. The system distributes the resources between the tasks to maximize the total utility of the system. In [Abdelzaher *et al.*, 1997] a QoS renegotiation scheme is proposed as a way to allow graceful degradation in cases of overload, failures or violation of pre-run-time assumptions. The mechanism permits clients to express, in their service requests, a range of QoS levels they can accept from the provider, and the perceived utility of receiving service at each of these levels.

The third area relates to the wealth of flexible scheduling algorithms available. An interesting alternative to linear task rescaling is given in [Buttazzo *et al.*, 1998], where an elastic task model for periodic tasks is presented. The relative sensitivity of tasks to rescaling are expressed in terms of elasticity coefficients. Schedulability anal-

ysis of the system under EDF scheduling is given. Closely related to our work, [Stankovic *et al.*, 1999] presents a scheduling algorithm that explicitly uses feedback. A PID controller regulates the deadline miss-ratio for a set of soft real-time tasks with varying execution times, by adjusting their requested CPU utilization. It is assumed that tasks can change their CPU consumption by executing different versions of the same algorithm. An admission controller is used to accommodate larger changes in the workload.

### Outline

The rest of the paper is outlined as follows. Section 2 describes a hybrid controller for a double-tank process. Section 3 applies the feedback scheduling principle to a system with three hybrid controllers. The design and implementation are discussed and simulation results are given. Finally, Section 4 contains the conclusions.

## 2. A Hybrid Controller

A hybrid controller for the double-tank process, see Figure 2, is described. The controller was designed and implemented in [Eker and Malmborg, 1999]. The goal is to control the level of the lower tank to a desired setpoint. The measurement signals are the levels of both tanks, and the control signal is the inflow to the upper tank. Choosing

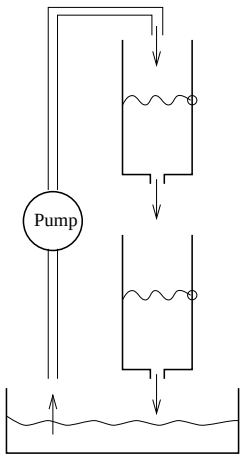



**Figure 2.** The double-tank process.

state variables $x_1(t)$ for the upper tank level and $x_2(t)$ for the lower tank level, we get the nonlinear state-space description

$$\frac{dx}{dt} = \begin{bmatrix} -\alpha\sqrt{x_1(t)} + \beta u(t) \\ \alpha\sqrt{x_1(t)} - \alpha\sqrt{x_2(t)} \end{bmatrix} \tag{1}$$

The process constants $\alpha$ and $\beta$ depend on the cross-sections of the tanks, the outlet areas, and the capacity of the pump. The control signal $u(t)$ is limited to the interval [0,1].

Traditionally there is a trade-off in design objectives when choosing controller parameters. It is usually hard to achieve the desired step-change response and at the same time get the wanted steady-state behavior. An example of contradictory design criteria is tuning a PID controller to achieve both fast response to setpoint changes, fast disturbance rejection, and no or little overshoot. In process control it is common practice to use PI control for steady state regulation and to use manual control for large setpoint changes. One solution to this problem is to use a hybrid controller consisting of two sub-controllers, one PID controller and one time-optimal controller, together with a switching scheme. The time-optimal controller is used when the states are far away from the reference point. Coming close, the PID controller will automatically be switched in to replace the time optimal controller.

The sub-controller designs are based on a linearization of Equation (1).

$$\frac{dx}{dt} = \begin{bmatrix} -a & 0 \\ a & -a \end{bmatrix} x(t) + \begin{bmatrix} b \\ 0 \end{bmatrix} u(t) \tag{2}$$

The new process parameters $a$ and $b$ are functions of $\alpha$, $\beta$ and the current linearization level.

### PID Controller

The PID parameters $(K, T_i, T_d)$ are calculated to give the closed-loop characteristic polynomial

$$(s + \omega_0)(s^2 + 2\zeta\omega_0 s + \omega_0^2) \tag{3}$$

where $(\omega_0, \zeta) = (6, 0.7)$ are chosen to give good rejection of load disturbances. The following discrete-time implementation, which includes

low-pass filtering of the derivative part ($N = 10$), is used:

$$P(t) = K(y_{sp}(t) - y(t)) \tag{4}$$

$$I(t) = I(t-h) + \frac{Kh}{T_i}(y_{sp}(t) - y(t)) \tag{5}$$

$$D(t) = \frac{T_d}{Nh+T_d}D(t-h) + \frac{NKT_d}{Nh+T_d}(y(t-h) - y(t)) \tag{6}$$

$$u(t) = P(t) + I(t) + D(t) \tag{7}$$

### Time-Optimal Controller

The time-optimal control signal is of bang-bang type. For the linearized process it is possible to derive the switching curve

$$x_2(x_1) = \frac{1}{a}((ax_1 - b\overline{u})(1 + \ln(\frac{ax_1^R - b\overline{u}}{ax_1 - b\overline{u}})) + b\overline{u}) \tag{8}$$

where $\overline{u}$ takes values in $\{0, 1\}$, and $x_1^R$ is the target state for $x_1$. The control signal is $u = 0$ above the switching curve and $u = 1$ below. A closeness criterion on the form

$$V_{close} = \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix}^T P(\theta, \gamma) \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix} \tag{9}$$

where $P(\theta, \gamma)$ is positive definite matrix, is evaluated at each sample, to determine whether the controller should switch to PID mode.

### Implementation

The controller implementation is outlined below.

```
y = analogIn(yChan);
ysp = getSetpoint();
if (getMode() == PID) {
  if (ysp != ysp_old) {
    setMode(OPT);
    signal(FBS_sem); /* feedforward, see Section 3.3 */
    u = calculateOPT();
  } else {
    u = calculatePID();
  }
} else { /* OPT */
```

```
  Vclose = computeVclose();
  if (Vclose < Vregion) {
    setMode(PID);
    u = calculatePID();
  } else {
    u = calculateOPT();
  }
}
analogOut(uChan,u);
```

### Real-Time Properties

The execution-time properties of the hybrid controller were investigated in [Persson *et al.*, 2000]. It was found that the optimal-control mode had considerable longer execution time than the PID mode. In each mode, the execution time was close to the best case most of the time, but it also exhibited random bursts. For purposes of illustration, assume that the execution-time characteristics in the different modes can be described by $C_{PID} = 1.8 + 0.2\varepsilon_i^2$ ms and $C_{Opt} = 9.5 + 0.5\varepsilon_i^2$ ms, where $\{\varepsilon_i\}$ is unit-variance Gaussian white noise.

The nominal sampling interval is chosen to be one tenth of the rise time, $T_r$, of the closed-loop system. Our first example process has $T_{r1} = 210$ ms which gives $h_{nom\,1} = 21$ ms. A simulation of the computer-controlled system is found in Figure 3. The controller displays very good set-point response and steady-state regulation. It is seen that the requested CPU utilization is very low in PID mode, on average $\overline{U} = \overline{C}_{PID}/h_{nom\,1} = 9\%$. In Optimal mode, it is significantly higher, on average $\overline{U} = \overline{C}_{Opt}/h_{nom\,1} = 45\%$.

## 3. Feedback Scheduling Example

Now assume that two additional hybrid double-tank controllers should execute on the same CPU as the first one. The tanks have slightly different process parameters. Based on the rise-times, $(T_{r2}, T_{r3}) = (180, 150)$ ms, they are assigned the nominal sampling intervals $(h_{nom\,2}, h_{nom\,3}) = (18, 15)$ ms. To consider scheduling, some assumptions about the real-time operating system must be made. Throughout this example, we assume a fixed-priority real-time kernel with the pos-

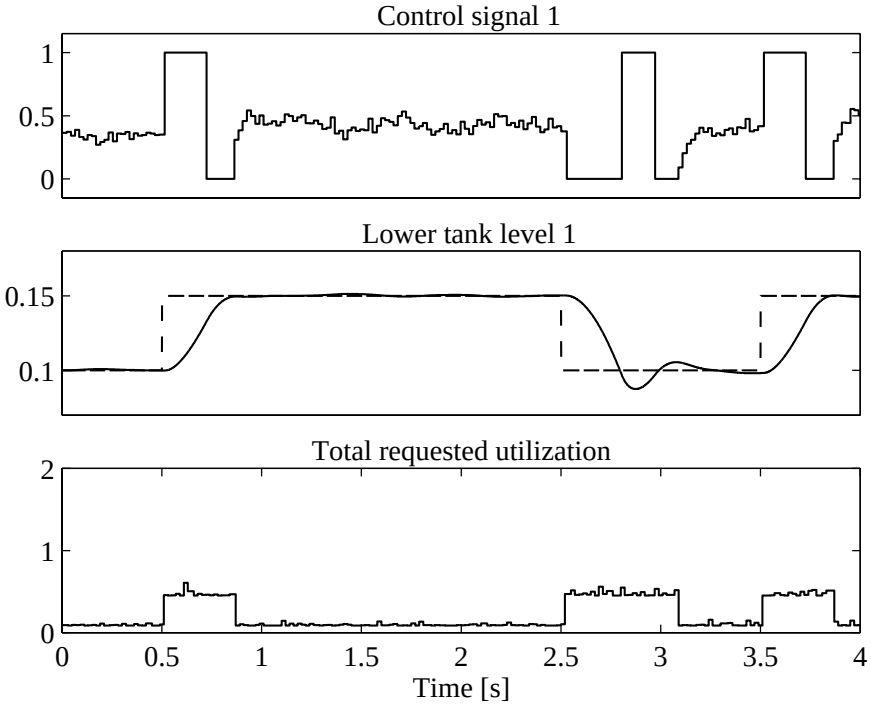



**Figure 3.** Performance of Controller 1 when running in isolation. The controller displays very good set-point response and steady-state regulation. (The undershoot at $t = 2.9$ s is due to a load disturbance.) The CPU never becomes overloaded.

sibility to measure task execution time. The tasks are assigned rate-monotonic priorities, i.e., the task with the shortest period gets the highest priority.

First, open-loop scheduling is attempted. Then, a feedback scheduler is added to the system. Finally, feedforward is introduced in the scheduler. The systems are evaluated by co-simulation of the real-time kernel and the plant dynamics [Eker and Cervin, 1999]. A 4-second simulation cycle is constructed as follows. At time $t = 0$, all controllers start in the PID mode. At $t = 0.5$ s, the worst-case scenario appears: all controllers receive new setpoints and should switch to Optimal mode. Following this, the controllers get new setpoints pairwise, and then one by one. For each simulation, the behavior of Controller 1, now having the lowest priority, is plotted. Also plotted is the total requested utilization, $\sum_i c_i/h_i$, where $c_i$ is the current actual execution time of task $i$, and $h_i$ is the current period of task $i$. Notice that the total requested utilization cannot be directly measured and used for feedback,

but must be estimated.

### Open-Loop Scheduling

We first consider open-loop scheduling, where the controllers are implemented as tasks with fixed periods equal to their nominal sampling intervals. The simulation results are shown in Figures 4 and 5.

The system easily becomes overloaded, since in the worst case, $\overline{U} = \sum_i \overline{C}_{Opt}/h_{nom\,i} = 170$ %. Controller 1 is for instance temporarily turned off in the intervals $t = [0.5, 0.8]$ s and $t = [1.5, 1.8]$ s because of preemption. The result is low control performance.

### Feedback Scheduling

Next, a feedback scheduler is introduced. In its first version, it is implemented as a high-priority task with a period $T_{\mathrm{FBS}} = 100$ ms. The utilization setpoint is set to $U_{sp} = 80$ %. At each invocation, the feedback scheduler estimates the current total requested utilization of the tasks by computing $\hat{U} = \sum_i \hat{C}_i/h_i$. The estimate $\hat{C}_i$ is obtained from filtered execution-time measurements,

$$\hat{C}_i(k) = \lambda \hat{C}_i(k-1) + (1-\lambda)c_i \tag{10}$$

where $\lambda$ is a forgetting factor. Setting $\lambda$ close to 1 results in a smooth, but slow estimate. In this case, $\lambda = 0$, which gives fast detection of overloads, was preferred. Finally, new task periods are assigned according to the linear rescaling

$$h_i = h_{nom\,i}\hat{U}/U_{sp} \tag{11}$$

The execution time of the feedback scheduler is assumed to be 2 ms. The simulation results are shown in Figures 6 and 7. The scheduler tries to keep the workload close to 80 %. However, there is a delay from a change in the requested utilization until it is detected by the feedback scheduler. This results in overload peaks at some of the mode change instants. For instance, Controller 1 is preempted in the interval $t = [0.5, 0.6]$ s. The result is slightly degraded control performance.

### Feedback and Feedforward Scheduling

A feedforward mechanism is added to the scheduler. The basic period of the scheduler is kept at $T_{\mathrm{FBS}} = 100$ ms. However, when a task in

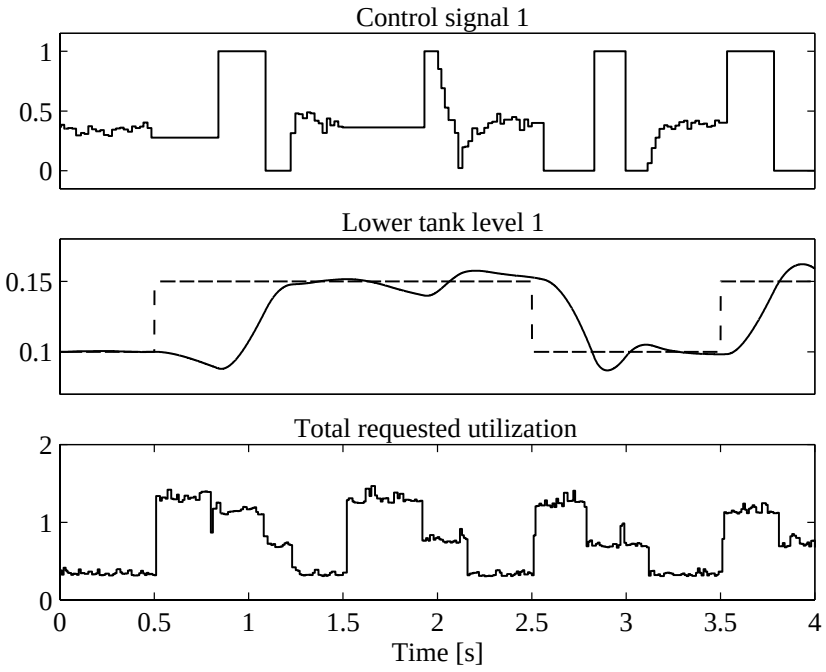



**Figure 4.** Performance of Controller 1 under open-loop scheduling. The CPU is overloaded during long intervals, and the controller cannot update its control signal very often. The result is low control performance.

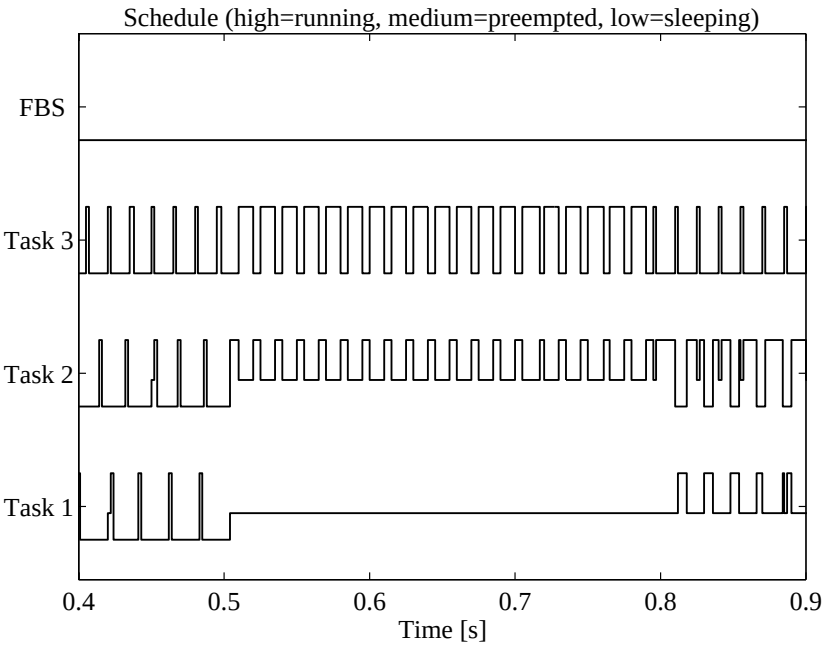



**Figure 5.** Close-up of the schedule under open-loop scheduling. At $t = 0.5$ s, Task 2 and 3 switch to Optimal mode, and the CPU gets overloaded. As a result, Task 1 is preempted in a long interval.

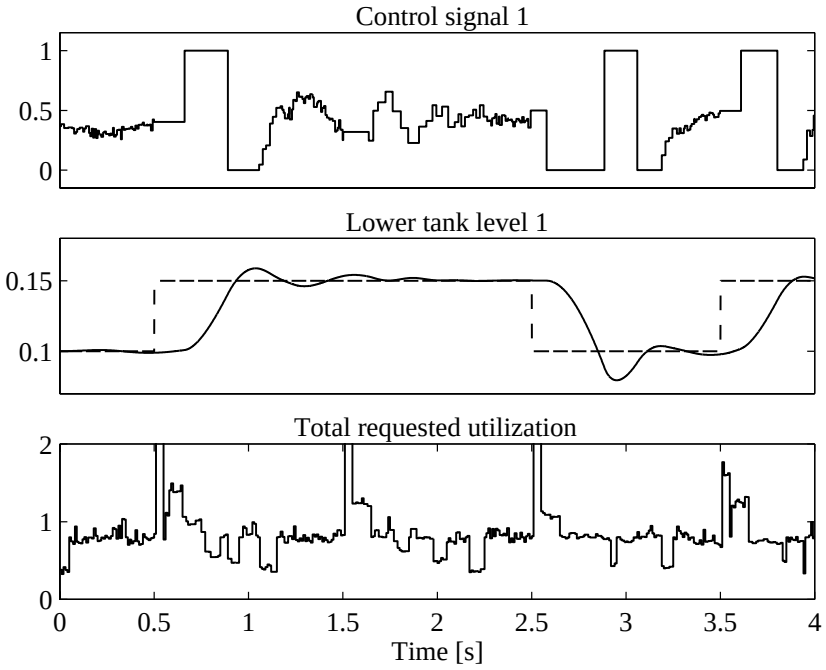



**Figure 6.** Performance of Controller 1 under feedback scheduling. The CPU is overloaded in shorter intervals and the performance is better than under open-loop scheduling, cf. Figure 4.

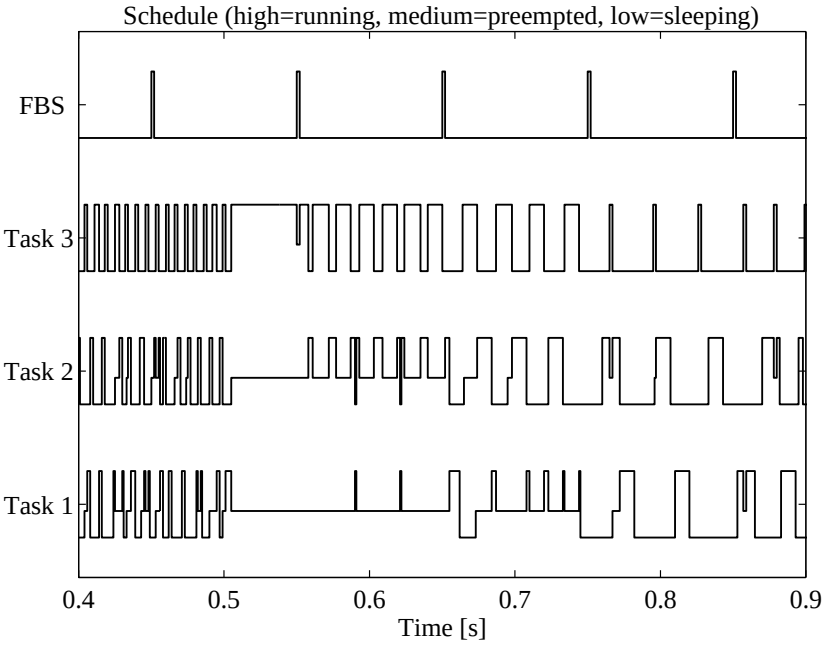



**Figure 7.** Close-up of the schedule under feedback scheduling. At $t = 0.5$, Task 3 switches to Optimal mode, and the CPU gets overloaded. At $t = 0.55$, the feedback scheduler rescales the task periods. But this allows Task 2 to switch to Optimal mode, and the CPU gets overloaded again.

PID mode detects a new setpoint, it notifies the feedback scheduler, which is released immediately. The task periods are adjusted before the notifying task can continue to execute in the Optimal mode. The execution-time estimation can also benefit from the mode-change information, by running separate estimators in the different modes. A forgetting factor of $\lambda = 0.9$ was chosen to give smooth estimates in both modes. The result is a more responsive and accurate feedback scheduler. The simulation results are shown in Figures 8 and 9. It is seen that the delay for Controller 1 at $t = 0.5$ s has been reduced, and that the control performance is slightly better.

### Performance Evaluation and Summary

The performance of the controllers under different scheduling policies are evaluated using the criterion

$$V_i(t) = \int_0^t (y_{nom\,i}(\tau) - y_{act\,i}(\tau))^2 d\tau \tag{12}$$

where $y_{nom\,i}$ is the process output when Controller $i$ is running unpreempted at its nominal sampling interval, and $y_{act\,i}$ is the actual process output when Controller $i$ is running in the multitasking real-time system. The function $V$ is referred to as the *additional loss due to scheduling*. Twenty-five simulation cycles (100 s) are simulated and the final losses for the controllers are summarized below:

| Scheduling | $V_1(100)$ | $V_2(100)$ | $V_3(100)$ |
|---|---|---|---|
| Open-loop | $42.4 \cdot 10^{-3}$ | $2.0 \cdot 10^{-3}$ | 0 |
| Feedback | $5.0 \cdot 10^{-3}$ | $3.2 \cdot 10^{-3}$ | $1.0 \cdot 10^{-3}$ |
| Feedback and feedforward | $1.5 \cdot 10^{-3}$ | $1.1 \cdot 10^{-3}$ | $1.0 \cdot 10^{-3}$ |

Under open-loop scheduling, Controller 3 has zero additional loss. This is because Task 3 has the highest priority and thus executes unpreempted at its nominal sampling period. Controller 2 suffers from some preemption from Task 3 which gives a small loss, while Controller 1 is preempted during long intervals which gives a very large loss.

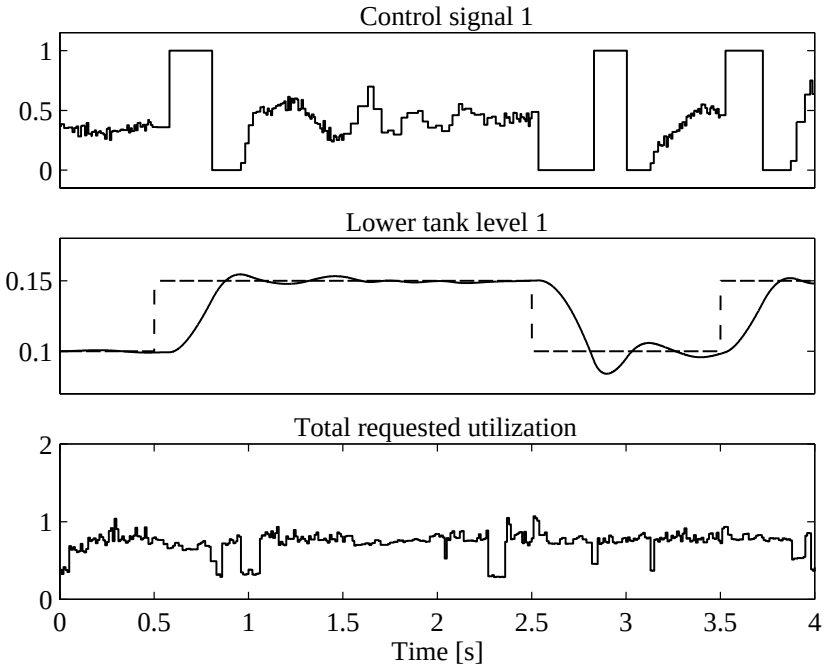



**Figure 8.** Performance of Controller 1 under feedback and feedforward scheduling. The CPU is almost never overloaded, which results in better control performance, cf. Figures 4 and 6. The performance is not as good as in Figure 3 though, since the controller must sometimes execute at a lower rate.

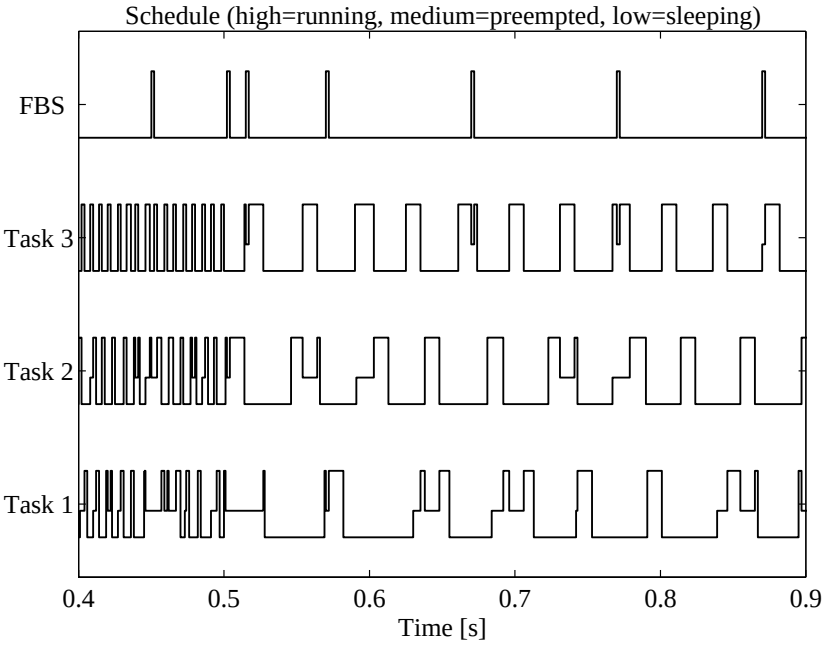



**Figure 9.** Close-up of the schedule under feedback and feedforward scheduling. The periods are rescaled by the feedback scheduler as each controller switches to Optimal mode. As a result, the CPU is almost never overloaded.

Under feedback scheduling, the loss is much smaller for Controller 1, due to the drastically reduced amount of preemption from Task 2 and 3. Because of the period rescaling, however, Controller 2 and 3 increase their losses.

Under feedback and feedforward scheduling, Controller 1 and 2 decrease their losses, since the CPU overloads are almost completely avoided. The total loss is small, and it is evenly distributed among the controllers.

The evolution of the additional loss for Controller 1 is shown in Figure 10. There is a very large improvement when introducing feedback, and the addition of the feedforward mechanism gives even further reduction of the loss.

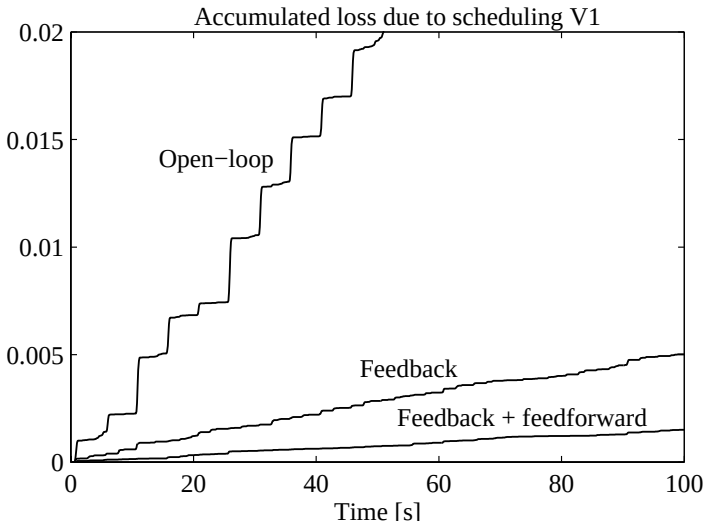



**Figure 10.** The accumulated additional loss due to scheduling for Controller 1, $V_1(t)$. The introduction of feedback scheduling gives a large reduction in the loss. Adding the feedforward mechanism reduces the loss even further.

## 4. Conclusions

The feedback scheduler presented improves the control performance and relaxes the requirement on known execution times for multitasking control systems. The controllers are allowed to miss an occasional deadline, and are hence not treated as hard real-time tasks. In case of

an overload, the scheduler calculates new sampling periods for all control tasks. The estimate of the current workload is based on execution-time measurements. The new sampling periods are given by simple linear rescaling of the nominal sampling periods, i.e., the relative importance order of the controllers is preserved. A more elaborate rescaling procedure would most likely give better control performance but also require more computational power. The feedback scheduler itself is implemented as a task, and its period is an important design parameter.

## 5. References

Abdelzaher, T., E. Atkins, and K. Shin (1997): "QoS negotiation in real-time systems, and its application to flight control." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Buttazzo, G., G. Lipari, and L. Abeni (1998): "Elastic task model for adaptive rate control." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Eker, J. and A. Cervin (1999): "A Matlab toolbox for real-time and control systems co-design." In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, pp. 320–327. Hong Kong, P.R. China.

Eker, J. and J. Malmborg (1999): "Design and implementation of a hybrid control strategy." *IEEE Control Systems Magazine*, **19:4**.

Li, B. and K. Nahrstedt (1998): "A control theoretic model for quality of service adaptations." In *Proceedings of Sixth International Workshop on Quality of Service*.

Liu, C. L. and J. W. Layland (1973): "Scheduling algorithms for multiprogramming in a hard-real-time environment." *Journal of the ACM*, **20:1**, pp. 40–61.

Persson, P., A. Cervin, and J. Eker (2000): "Execution-time properties of a hybrid controller." Report ISRN LUTFD2/TFRT--7591--SE. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Rajkumar, R., C. Lee, J. Lehoczky, and D. Siewiorek (1997): "A resources allocation model for QoS management." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Ryu, M., S. Hong, and M. Saksena (1997): "Streamlining real-time controller design: From performance specifications to end-to-end timing constraints." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin (1996): "On task schedulability in real-time control systems." In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 13–21.

Shin, K. and C. Meissner (1999): "Adaptation of control system performance by task reallocation and period modification." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 29–36.

Stankovic, J. A., C. Lu, S. H. Son, and G. Tao (1999): "The case for feedback control real-time scheduling." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 11–20.