Improved Scheduling of Control Tasks *

Anton Cervin
Department of Automatic Control
Lund Institute of Technology
Box 118, SE 221 00 Lund, Sweden
E-mail: anton@control.lth.se

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 Fig.1. At a fixed frequency, the

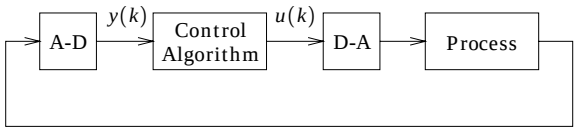

Figure 1 Overview of a digital control system.

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 [1]. 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

*This paper will be presented at the 11th Euromicro Conference on Real-Time Systems, York, England, June 1999.

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 [7]:

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

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

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

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

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

Here, the A-D conversion is implicated by the use of the measurement variable $y(k)$ in (1), and the D-A conversion is implicated by 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 τ_i having a period T_i and a worst-case computation time C_i. Well known tests [12] [8] 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 [4] [3] [5], 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 [14] [10] 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 τ_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) [12]. 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 τ_i [8]. 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 [11]. 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 τ_i consist of two subtasks, $\tau_{CO\,i}$ (Calculate Output) and $\tau_{US\,i}$ (Update State). The worst-case execution time of the subtasks are assumed to be known and equal to $C_{CO\,i}$ and $C_{US\,i}$ respectively.

We first look at the timing analysis developed by Härbour *et al.* [6]. 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 $D_{US\,i} = T_i$. The deadline for Calculate Output must at least be constrained by

$$C_{CO\,i} \leq D_{CO\,i} \leq T_i - C_{US\,i} \tag{7}$$

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 [6].

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 [9]. 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* [2]. The subtasks are not serially executed—rather, the subtask τ_{USi} is released with a fixed offset O_{USi} compared to the release of τ_{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

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

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

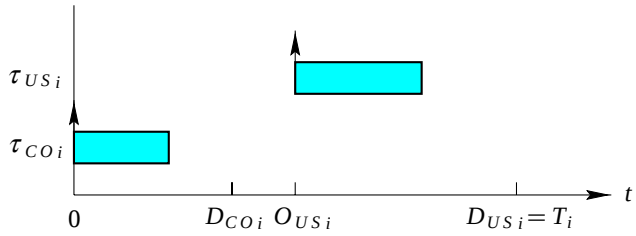

Figure 2 The parts τ_{COi} and τ_{USi} could be scheduled using an offset.

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 [1]. 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 [13]. 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 Fig. 3. The

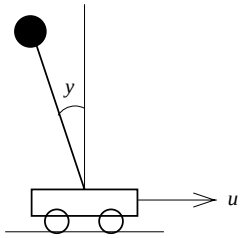

Figure 3 An inverted pendulum. The pendulum can be stabilized in the upright position $y = 0$ by controlling the acceleration u of the pivot point.

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 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 [1]. 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 τ_1, τ_2, and τ_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
τ_1	167	167	28	1	140
τ_2	100	100	28	2	56
τ_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 Fig. 4. It is clearly seen that the interference from Controller 2 and 3 causes the control actions to occur at irregular times.

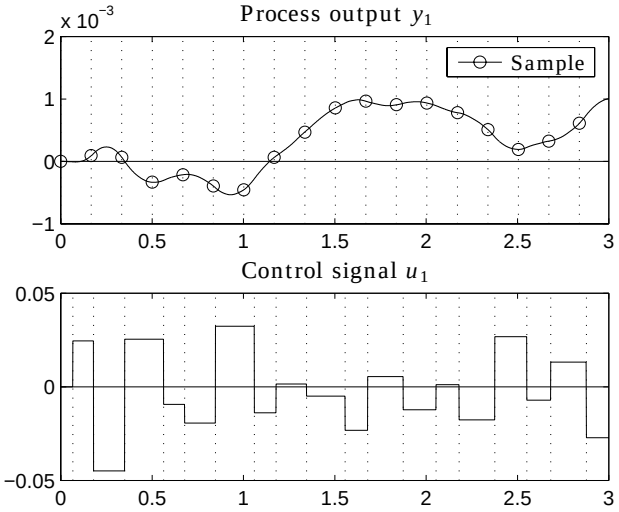

Figure 4 Close-up behavior of Controller 1 when Textbook Implementation A (Implementation 1) is used. Notice the large jitter in the control actions.

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 τ_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
$\tau_{CO\,1}$	167	149	10	2	66
$\tau_{US\,1}$	167	167	18	1	140
$\tau_{CO\,2}$	100	82	10	4	38
$\tau_{US\,2}$	100	100	18	3	56
$\tau_{CO\,3}$	71	53	10	6	10
$\tau_{US\,3}$	71	71	18	5	28

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

	T	D	C	P	R
$\tau_{CO\,1}$	167	66	10	4	30
$\tau_{US\,1}$	167	167	18	1	140
$\tau_{CO\,2}$	100	38	10	5	20
$\tau_{US\,2}$	100	100	18	2	66
$\tau_{CO\,3}$	71	10	10	6	10
$\tau_{US\,3}$	71	71	18	3	48

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

	T	D	C	P	R
$\tau_{CO\,1}$	167	30	10	4	30
$\tau_{US\,1}$	167	167	18	1	140
$\tau_{CO\,2}$	100	20	10	5	20
$\tau_{US\,2}$	100	100	18	2	66
$\tau_{CO\,3}$	71	10	10	6	10
$\tau_{US\,3}$	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 Fig. 5. The control jitter is not visible anymore.

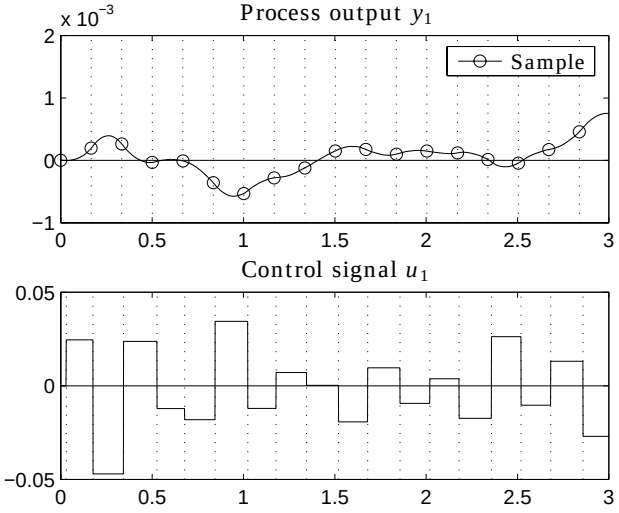

Figure 5 Close-up behavior of Controller 1 when improved scheduling is used (Implementation 3). Notice that the control jitter is much smaller than in Fig. 4.

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.

Bibliography

[1] K. J. ÅSTRÖM and B. WITTENMARK. *Computer-Controlled Systems*. Prentice Hall, third edition, 1997.

[2] N. AUDSLEY, K. TINDELL, and A. BURNS. "The end of the road for static cyclic scheduling." In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems, Oulu, Finland*, pp. 36–41, 1993.

[3] A. BURNS, K. TINDELL, and A. J. WELLINGS. "Fixed priority scheduling with deadlines prior to completion." In *Proceedings of the 6th Euromicro Workshop on Real-Time Systems, Västerås, Sweden*, pp. 138–142, 1994.

[4] R. GERBER and S. HONG. "Semantics-based compiler transformations for enhanced schedulability." In *Proc. of the 14th IEEE Real-Time Systems Symposium*, pp. 232–242, December 1993.

[5] R. GERBER and S. HONG. "Slicing real-time programs for enhanced schedulabilty." *ACM Transactions on Programming Languages and Systems*, **19:3**, pp. 525–555, May 1997.

[6] M. GONZALEZ HÄRBOUR, M. H. KLEIN, and J. P. LEHOCZKY. "Timing analysis for fixed-priority scheduling of hard real-time systems." *IEEE Transactions on Software Engineering*, **20:1**, pp. 13–28, January 1994.

[7] K. GUSTAFSSON and P. HAGANDER. "Discrete-time LQG with cross-terms in the loss function and the noise description." Technical Report CODEN: LUTFD2/(TFRT-7475)/1–15/(1991), Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1991.

[8] M. JOSEPH and P. PANDYA. "Finding response-times in a real-time system." *BCS Computer Journal*, **29:5**, pp. 390–395, 1986.

[9] B. KAO and H. GARCIA-MOLINA. "Deadline assignment in a distributed soft real-time system." In *Proc. of the 13th International Conference on Distributed Computing Systems*, 1993.

[10] B. K. KIM. "Task scheduling with feedback latency for real-time control systems." In *Proc. of the 5th International Conference on Real-Time Computing Systems and Applications*, pp. 37–41, 1998.

[11] J. Y. T. LEUNG and J. WHITEHEAD. "On the complexity of fixed-priority scheduling of periodic, real-time tasks." *Performance Evaluation*, **2:4**, pp. 237–250, 1982.

[12] C. L. LIU and J. W. LAYLAND. "Scheduling algorithms for multiprogramming in a hard real-time environment." *Journal of the ACM*, **20:1**, pp. 40–61, 1973.

[13] J. NILSSON, B. BERNHARDSSON, and B. WITTENMARK. "Stochastic analysis and control of real-time systems with random time delays." In *IFAC'96, Preprints 13th World Congress of IFAC*, San Francisco, California, 1996.

[14] D. SETO, J. P. LEHOCZKY, L. SHA, and K. G. SHIN. "On task schedulability in real-time control systems." In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 13–21, Los Alamitos, CA, USA, 1996.