Feedback Scheduling of Control Tasks

Cervin, Anton; Eker, Johan

Published in:
Proceedings of the 39th IEEE Conference on Decision and Control, 2000.

DOI:
10.1109/CDC.2001.914702

2000

Proceedings of the 39th IEEE
Conference on Decision and Control
Sydney, Australia • December, 2000

Feedback Scheduling of Control Tasks

Anton Cervin **Johan Eker**

Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund
Sweden
{anton,johane}@control.lth.se

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 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

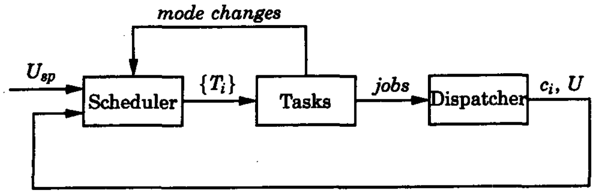

Figure 1 The feedback scheduling structure.

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

1.1 Related Work

A survey of related work in the area of control and CPU scheduling is found in [Årzén *et al.*, 2000]. Further references can be found in the theses [Cervin, 2000] and [Eker, 1999].

Closely related to our work, [Stankovic *et al.*, 1999] presents a general 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 the tasks can change their CPU consumption by executing different versions of the same algorithm. An admission controller is used to accommodate large changes in the workload.

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 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

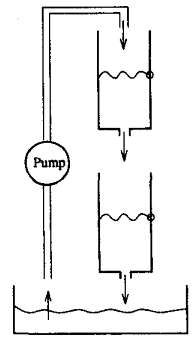

Figure 2 The double-tank process.

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

The process constants α and β 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 α, β and the current linearization level.

2.1 PID Controller

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

$$(s+\omega_0)(s^2+2\zeta\omega_0 s+\omega_0^2), \quad (3)$$

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:

$$P(t) = K(y_{sp}(t) - y(t)), \quad (4)$$
$$I(t) = I(t-h) + \tfrac{Kh}{T_i}(y_{sp}(t) - y(t)), \quad (5)$$
$$D(t) = \tfrac{T_d}{Nh+T_d}D(t-h) + \tfrac{NKT_d}{Nh+T_d}(y(t-h) - y(t)), \quad (6)$$
$$u(t) = P(t) + I(t) + D(t). \quad (7)$$

2.2 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

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

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

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

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

2.3 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 Sec 3.3 */
    u = calculateOPT();
  } else {
    u = calculatePID();
  }
} else { /* OPT */
  Vclose = computeVclose();
  if (Vclose < Vregion) {
    setMode(PID);
    u = calculatePID();
  } else {
    u = calculateOPT();
  }
}
analogOut(uChan,u);
```

2.4 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 $h_{nom1} = 21$ ms. A simulation of the computer-controlled system is found in Figure 3. The controller

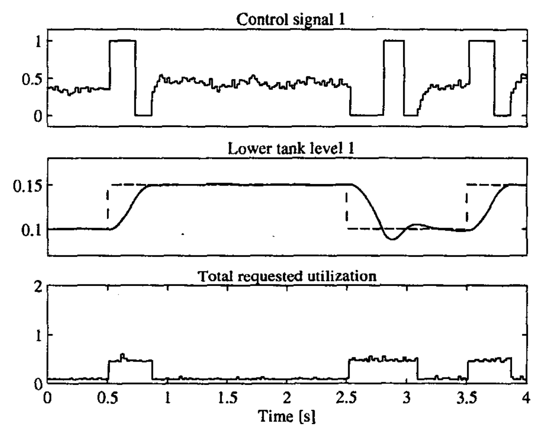

Figure 3 Performance of Controller 1 when running in isolation. The controller displays very good set-point response and steady-state regulation. (The undershoot at $t = 2.9$ s is due to a load disturbance.) The CPU never becomes overloaded.

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 $\overline{U} = \overline{C}_{PID}/h_{nom1} = 9\%$. In Optimal mode, it is significantly higher, on average $\overline{U} = \overline{C}_{Opt}/h_{nom1} = 45\%$.

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 $(h_{nom2}, h_{nom3}) = (18, 15)$ 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 possibility 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 occurs: 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.

3.1 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.

3.2 Feedback Scheduling

Next, a feedback scheduler is introduced. In its first version, it is implemented as a high-priority task with a period $T_{\text{FBS}} = 100$ 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,

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

where λ is a forgetting factor. Setting λ 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

$$h_i = h_{nom\,i}\hat{U}/U_{sp}. \qquad (11)$$

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

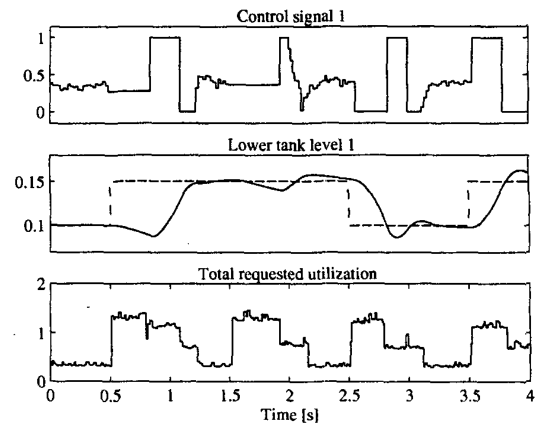

Figure 4 Performance of Controller 1 under open-loop scheduling. The CPU is overloaded during long intervals, and the controller cannot update its control signal very often. The result is low control performance.

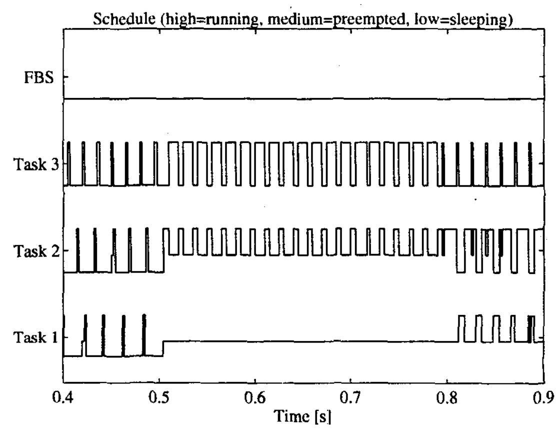

Figure 5 Close-up of the schedule under open-loop scheduling. At $t = 0.5$ s, Task 2 and 3 switch to Optimal mode, and the CPU gets overloaded. As a result, Task 1 is preempted in a long interval.

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.

3.3 Feedback and Feedforward Scheduling

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

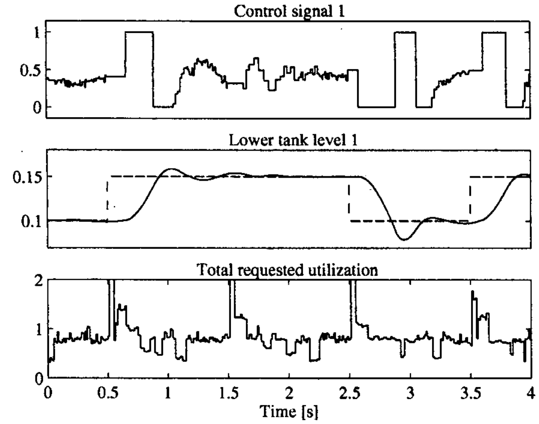

Figure 6 Performance of Controller 1 under feedback scheduling. The CPU is overloaded in shorter intervals and the performance is better than under open-loop scheduling, cf. Figure 4.

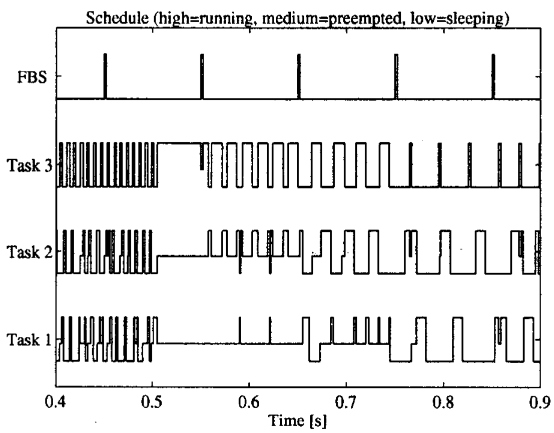

Figure 7 Close-up of the schedule under feedback scheduling. At $t = 0.5$, Task 3 switches to Optimal mode, and the CPU gets overloaded. At $t = 0.55$, the feedback scheduler rescales the task periods. But this allows Task 2 to switch to Optimal mode, and the CPU gets overloaded again.

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.

3.4 Performance Evaluation and Summary

The performance of the controllers under different

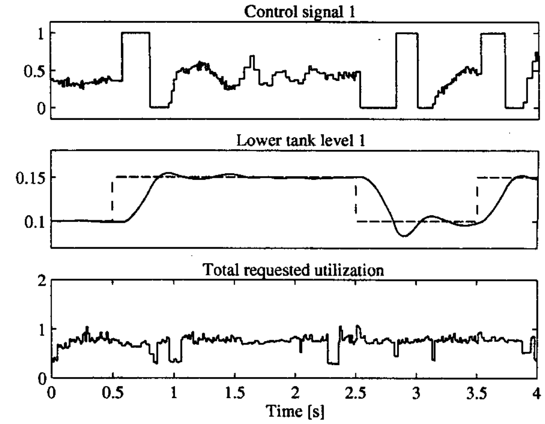

Figure 8 Performance of Controller 1 under feedback and feedforward scheduling. The CPU is almost never overloaded, which results in better control performance, cf. Figures 4 and 6. The performance is not as good as in Figure 3 though, since the controller must sometimes execute at a lower rate.

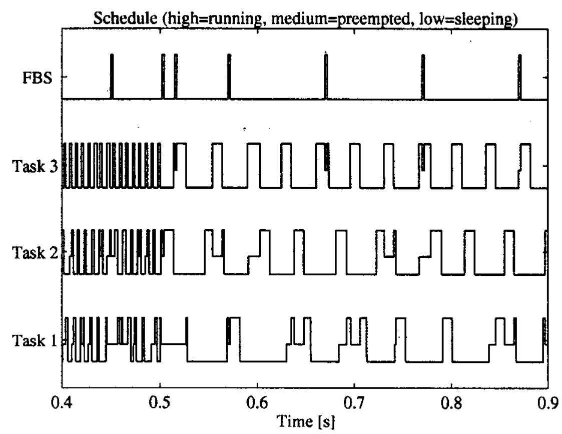

Figure 9 Close-up of the schedule under feedback and feedforward scheduling. The periods are rescaled by the feedback scheduler as each controller switches to Optimal mode. As a result, the CPU is almost never overloaded.

scheduling policies are evaluated using the criterion

$$V_i(t) = \int_0^t (y_{nom_i}(\tau) - y_{act_i}(\tau))^2 d\tau, \qquad (12)$$

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.

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.

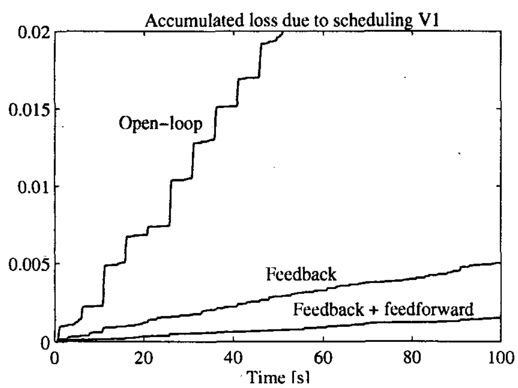

Figure 10 The accumulated additional loss due to scheduling for Controller 1, $V_1(t)$. The introduction of feedback scheduling gives a large reduction in the loss. Adding the feedforward mechanism reduces the loss even further.

4. Conclusions

The presented feedback scheduler 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 the 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

Årzén, K.-E., A. Cervin, J. Eker, and L. Sha (2000): "An introduction to control and scheduling co-design." In *Proceedings of the 39th IEEE Conference on Decision and Control.* Sydney, Australia.

Cervin, A. (2000): *Towards the Integration of Control and Real-Time Scheduling Design.* Licentiate thesis ISRN LUTFD2/TFRT--3226--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Eker, J. (1999): *Flexible Embedded Control Systems. Design and Implementation.* PhD thesis ISRN LUTFD2/TFRT--1055--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

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**.

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.

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. York, UK.