# Delay-Aware Period Assignment in Control Systems

Bini, Enrico; Cervin, Anton

2008

# Delay-Aware Period Assignment in Control Systems*

**Enrico Bini**
*Scuola Superiore Sant'Anna*
*Pisa, Italy*
e.bini@sssup.it

**Anton Cervin**
*Automatic Control, Lund University*
*Lund, Sweden*
anton@control.lth.se

## Abstract

*We consider the problem of optimal static period assignment for multiple independent control tasks executing on the same CPU. Previous works have assumed that the control performance can be expressed as a function of the sampling rate only. Arguing that the control delay has a large impact on the control performance, in this work we include the control delay in the cost function. The delay is estimated using an approximate response-time analysis. Assuming linear cost functions for the controllers then allows us to solve the optimal period assignment problem analytically. The performance improvements over previous methods are verified in evaluations on synthetic task sets as well as detailed co-simulations of the controllers, the plants, and the scheduler.*

## 1 Introduction

In the design of control systems, the law for controlling a plant is often developed starting from a continuous-time model. Based on the control performance specifications, the sampling rate is decided and the controller is synthesized. Finally, the controller is implemented as a periodic task that is assigned to the scheduler. However, as pointed out by several authors [18, 17, 19, 8, 7, 15], it can happen that the controller misbehaves with respect to the theoretical behavior foreseen in the design phase. A reason of possible misbehavior is that the interactions between the task implementing the controller and the other activities running on the processor may be incompatible with the stringent constraints on the controller schedule implicitly assumed in the design phase, such as no jitter and no delay between reading from the sensors and writing to the actuators. Unfortunately, these hypotheses cannot be guaranteed by any task schedulers. It follows that, during the design phase of the controller, as many aspects as possible of the scheduling environment should be taken into account. However, since the general control–scheduling co-design problem is highly nonlinear, any design procedure will typically have to be iterative and involve several model simplifications as well as additional (artificial) design constraints. The key to developing a successful design methodology is finding the right simplifications and constraints. The mathematical models should be simple enough to facilitate efficient synthesis of the control laws and the scheduling parameters, but at the same time they should capture the properties that matter the most to the application performance.

To be able to do the proper trade-offs in the design phase, it is important to understand how the quality of the control is affected by the scheduling parameters. The control performance is typically measured using a cost function. The primary timing attributes that we will consider in this paper are the activation rate of the controller and the delay between the controller start time (when the sensors are read) and its finish time (when the actuation occurs). Other attributes include jitter in the sensing and actuation instants. In order to limit the complexity, we will not consider jitter in the design state, but only in final control performance evaluation. A previous case study [6] has indicated that delay is indeed the more important attribute to consider at the design stage.

A basic assumption in the paper is that the achievable cost for a given plant is a monotonically decreasing function of the activation rate and a monotonically increasing function of the control delay. The best case is hence a short sampling period and a short delay. This is true for all reasonable choices of activation rates (i.e. all cases but extremely slow sampling, where the sampling frequency is lower than the speed of the fastest pole of the plant). As the sampling rate approaches infinity and the delay approaches zero, the cost approaches that of an ideal, analog controller.

### 1.1 Contributions

The main contribution of the paper is the development of an approximate response-time analysis under fixed-priority

*This work was partially supported by ArtistDesign NoE IST-2008-214373.

1052-8725/08 $25.00 © 2008 IEEE
DOI 10.1109/RTSS.2008.45

scheduling that can be used to roughly estimate the control delay. We further show, assuming cost functions that are linear in the task periods and in the delays, how the response-time approximation allows the optimal period assignment problem for a set of controllers to be solved analytically. Being very efficient, the period assignment method can be used in an iterative co-design procedure, where the real (nonlinear) cost functions are linearized around the current solution in each step. Since the delay has a quite significant impact on the control performance, taking the delay into account at design time can significantly reduce the implementation-related performance degradation. In both theoretical and simulation-based evaluations, we show that our co-design method overall provides a lower cost than previously proposed period assignment schemes for multiple controllers.

### 1.2 Related Work

In 1996, Seto et al. [17] found the optimal task rate assignment such that a given performance index is maximized and the available computational resources are not overloaded. In this work, which is a milestone in the literature of real-time control co-design, however the delay between sensing and actuation is still assumed to have no impact in the performance of the system. This was pointed out by Kim [12], who proposed to extended the cost function to include also the control delay. No method for assigning periods based on the modified cost function was proposed however. Another limitation of [17] was that they assumed as feasibility constraint the utilization upper bound [14]. However the utilization upper bound is only a sufficient condition when a fixed priority scheduler is used. Bini and Di Natale [5] proposed an algorithm that finds the optimal period assignment of control tasks scheduled by fixed priority. In their work, the delay is guaranteed not to exceed the period for all tasks. Since the optimal method requires a time-consuming branch-and-bound algorithm to be executed, they also proposed a faster algorithm to find a sub-optimal period assignment taking advantage of some geometrical considerations in the space of feasible activation rates. The simpler method will be compared with assignment method proposed in this paper in Section 5.

In recent years, there has been a growing interest in online control task period assignment, where the purpose is to reallocate more resources to the control loop currently in need. However, neither Martí et al. [16] nor Henriksson and Cervin [9] considered the control delay in their resource allocation schemes.

### 1.3 Outline

The remainder of this paper is outlined as follows. In Section 2, we state the system model, including the scheduling and controller parameters. In Section 3, the approximate response-time analysis is developed, followed by the analytical solution of the optimal period assignment problem. Section 4 very briefly discusses the problem of also assigning optimal priorities. The performance evaluation is given in Section 5, where the quality of the approximate analysis and the optimization is first tested on synthetic task sets and then in simulated multi-loop control systems. The conclusion and some discussion on future work are given in Section 6.

## 2 Application Model

An application consists of a set of $n$ independent control tasks, each one controlling a plant. A task $\tau_i$ is characterized by two sets of parameters: the real-time parameters, which describe how the task interacts with the scheduler and the other tasks, and the control parameters, which describe the plant, the controller, and the quality of the control.

### 2.1 Real-Time Parameters

The tasks are scheduled by a preemptive fixed-priority (FP) scheduler. A task $\tau_i$ is characterized by the following attributes:

- The worst-case computation time $C_i$ is the maximum execution requirement that a task can require. We highlight that the computation time of a linear controller is often quite static and predictable, since the code typically has not many conditional branches.
- The period of the task activations $T_i$ is the time separation between two consecutive activations. Equivalently, we will sometimes use the frequency of the activations, $f_i = \frac{1}{T_i}$.
- The task utilization $U_i$ is equal to $\frac{C_i}{T_i}$, and it measures the worst-case amount of computational resources required by the controller. If $\sum_i U_i > 1$, then we say that we are in *overload* conditions.
- The task priority is used by the FP scheduler to schedule the tasks for execution. Without loss of generality we assume that the priority is implicitly assigned by the task ordering, such that $\tau_i$ has higher priority than $\tau_{i+1}$.
- The worst-case response time $R_i$ is the maximum time that may elapse from the task activation to its finishing time.

The periods $T_i$ and the priorities are independent parameters, in the sense that they can be freely specified by the system designer. All the other parameters can be derived once these parameters are set.

### 2.2 Control Parameters

For the control parameters, we adopt a linear-quadratic Gaussian (LQG) framework [1], where each plant is described by a linear system

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= Ax(t) + Bu(t) + v_c(t) \\ y(t_k) &= Cx(t_k) + e(t_k) \end{aligned} \tag{1}$$

where $x$ is the plant state, $u$ is the controlled input, and $v_c$ is a continuous-time Gaussian white noise process with intensity $R_{1c}$. The output $y$ is measured at discrete time instants $t_k$, with measurement noise $e$ described by a discrete-time Gaussian white noise process with variance $R_2$. $A$, $B$, and $C$ are matrices of appropriate size. The control performance is measured by a standard quadratic cost function

$$J = \lim_{t\to\infty} \frac{1}{t} \mathrm{E}\left\{ \int_0^t \left( y^2(\tau) + \rho u^2(t) \right) \mathrm{d}\tau \right\} \tag{2}$$

where $\rho$ is a weighting factor that describes how much large control signals should be penalized compared to large plant outputs. (The $\mathrm{E}\{\cdot\}$ operator denotes expected value.)

For a given sampling period $T$ and a constant control delay $\Delta$ (from reading $y$ to updating $u$), it is straightforward to synthesize an optimal discrete-time LQG controller that minimizes (2). Further, it is possible to evaluate how the cost will change if $\Delta$ is changed from its nominal value. The controller and the corresponding cost can be calculated using MATLAB and the Jitterbug toolbox [13].

In general, the cost $J$ is a nonlinear function of the sampling period $T$ and the delay $\Delta$. For reasonably short periods however (abiding to common rules of thumb for sampling period selection [1]), the cost is usually a near-linear function of $T$ and $\Delta$. Three examples with scalar plants ($A = \{-1,\, 0,\, 1\}$, $B = 1$, $C = 1$, $R_{1c} = 1$, $R_2 = 0$, $\rho = 0$) are shown in Figure 1. For the case $A = 0$ (an integrator plant), the cost is indeed an exact linear function,

$$J = \tfrac{3+\sqrt{3}}{6}\, T + \Delta$$

In the general case, the cost function can at least locally be approximated by a linear function (possibly with a constant offset that will not matter for the design problem at hand). For each control task $\tau_i$, we hence model the relation between period, delay, and cost as

$$J_i = \alpha_i T_i + \beta_i \Delta_i = \frac{\alpha_i}{f_i} + \beta_i\, \Delta_i \tag{3}$$

Typically, the parameters $\alpha_i$ and $\beta_i$ are found to be within the same order of magnitude. This is not surprising, since the sample-and-hold operation can itself be interpreted as a delay. A common rule of thumb says that sampling with the interval $T$ is roughly equivalent to a control delay of $T/2$ [1].

## 3 The Period Assignment

In our design problem we assume that the task execution times are known. In this section we also assume that the priorities are given. In Section 4 we briefly discuss the priority assignment problem.

The goal of the design is to assign task periods such that the overall system cost is minimized. We define the overall cost $J$ as

$$J = \sum_{i=1}^{n} J_i \tag{4}$$

Notice that the controller cost function (3) also allows the designer to assign different weights $w_i$ to the controllers, which can be taken into account by simply multiplying both $\alpha_i$ and $\beta_i$ by $w_i$.

### 3.1 Modeling the Delay

As argued in Section 2.2, a reasonable model of the cost of a controller should take the delay into account. The concept of *job delay* is quite clear: it is the time that elapses from the sampling instant (that can be assumed coincident with the job activation or the job start time) to the actuation instant (that can be assumed equal to the job completion time). Unfortunately the job delay can vary from job to job. An exact approach would have to investigate how the controller cost is affected by the full schedule of the task, taking into account all the different delays experienced by the jobs. However, current scheduling theory does not allow us to express all the job delays as a function of the task periods. Hence the following problem arises: what should we consider as the *control delay* $\Delta_i$ such that (3) is a proper model of the controller cost?

A first attempt is to set the control delay equal to the maximum job delay. Hence, we can set the delay $\Delta_i$ equal to the task response time $R_i$. In fact, the response time is the maximum time that can elapse from the task activation to its completion. However, the response time is not a continuous function of the task periods [11, 2]. It then becomes unclear what optimization method we can use to perform the minimization of the overall cost.

A second option can be to use an approximation of the response time that has some smoothness properties that allows us to solve the problem of minimizing $J$. Recently,

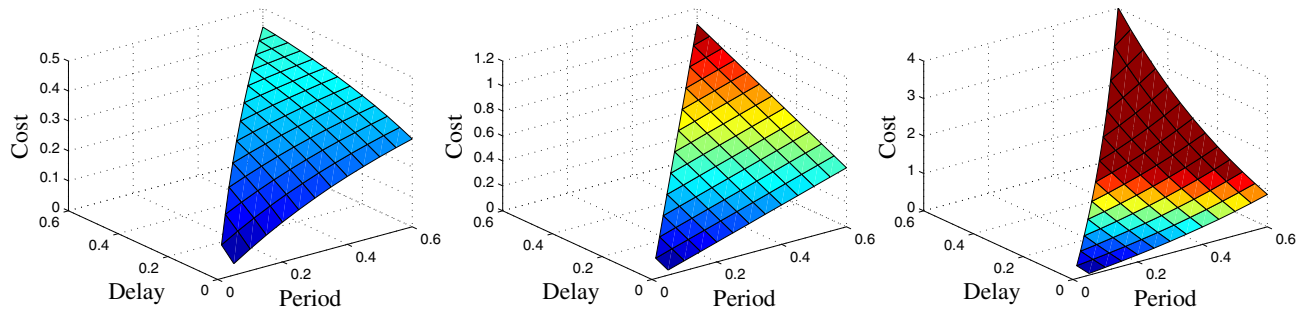


**Figure 1. Typical cost functions under LQG control. Left: stable plant, middle: marginally stable plant, right: unstable plant. In the plots, it is assumed that the delay never exceeds the period.**

Bini and Baruah [3] proposed a continuous and differentiable upper bound of the response time. They proved that

$$R_i \leq R_i^{\text{ub}} = \frac{C_i + \sum_{j=1}^{i-1} C_j(1-U_j)}{1-\sum_{j=1}^{i-1} U_j} \tag{5}$$

Thanks to the smoothness of this upper bound, we can imagine finding the task periods that minimize the cost (4) by replacing all delays $\Delta_i$ by the expression (5). However, this approach presents two drawbacks:

1. Since $R_i^{\text{ub}}$ is not convex, numerical solvers are not guaranteed to find a period assignment that guarantees a global minimum cost $J$.

2. Since the response time $R_i$ is the maximum response time of all the jobs activated by $\tau_i$ and $R_i^{\text{ub}} \geq R_i$, if we set $\Delta_i = R_i^{\text{ub}}$ we would overestimate the impact of the delay in the controller.

Instead we would prefer to approximate the delay $\Delta_i$ by the average of all the job response times over the task schedule. To follow this intuition we set

$$\Delta_i = R_i^{\text{approx}} = \frac{C_i}{1-\sum_{j=1}^{i-1} U_j} \tag{6}$$

that is the response time that the task $\tau_i$ would experience in a processor that provides a fluid share of $1-\sum_{j=1}^{i-1} U_j$ of the available bandwidth. This would be the exact response time if the higher priority tasks $\tau_1, \ldots, \tau_{i-1}$ received a constant share of the processor equal to their utilization.

Let's now examine some properties of $R_i^{\text{approx}}$. First, its convexity allows to assert that a period assignment that is a local minimum is also the global minimum. Second, $R_i^{\text{approx}}$ is no longer an upper bound of the task response time $R_i$. Instead it is a lower bound on the (worst-case) response time $R_i$. In fact,

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \geq C_i + \sum_{j=1}^{i-1} \frac{R_i}{T_j} C_j$$

$$R_i(1 - \sum_{j=1}^{i-1} U_j) \geq C_i$$

$$R_i \geq \frac{C_i}{(1-\sum_{j=1}^{i-1} U_j)} = R_i^{\text{approx}}$$

This property makes $R_i^{\text{approx}}$ suitable for our purposes, since our goal is to find an approximation of the average response time that is smaller than or equal to the task response time $R_i$. In Section 5.1 we will show that the average response time of the jobs does not deviate significantly from $R_i^{\text{approx}}$.

Finally, from a design point of view, an excellent property of $R_i^{\text{approx}}$ is that, if we assume $\Delta_i = R_i^{\text{approx}}$ we are able to find a closed solution of the optimal periods that minimizes the system cost $J$.

### 3.2 Solving the Minimum Cost Problem

If we now assume $\Delta_i = R_i^{\text{approx}}$ then the cost of a single controller becomes

$$J_i = \alpha_i \frac{1}{f_i} + \beta_i \frac{C_i}{1-\sum_{j=1}^{i-1} U_j} \tag{7}$$

To have more compact expressions we will use the controller utilizations as optimization variables instead of the task periods. Hence, by introducing

$$a_i = \alpha_i C_i \qquad b_i = \beta_i C_i \tag{8}$$

the task cost $J_i$ becomes

$$J_i = \frac{a_i}{U_i} + \frac{b_i}{1-\sum_{j=1}^{i-1} U_j} \tag{9}$$

Clearly the utilization assignment must satisfy the necessary feasibility constraint on the available bandwidth,

$$\sum_{i=1}^{n} U_i \leq 1 \tag{10}$$

It is well known that the constraint (10) is not sufficient to guarantee that the jobs complete not later than the activation of the next one [14]. However this condition is not required in our context. In fact there may be controllers poorly sensitive to the delay that work fine even if the jobs are not completed before the activation of the next one. Hence, in our problem we allow $R_i > T_i$ and we use the condition (10) as unique feasibility constraint.

It follows that the problem that we are solving is

$$\begin{aligned} &\text{minimize } J = \sum_{i=1}^{n} \left( \frac{a_i}{U_i} + \frac{b_i}{1 - \sum_{j=1}^{i-1} U_j} \right) \\ &\text{subject to } \sum_{i=1}^{n} U_i \leq 1, \quad U_i \geq 0 \end{aligned} \tag{11}$$

Now we start computing the partial derivatives of the task costs $J_i$ with respect to the utilizations $U_k$. First of all we realize that the utilization $U_n$ only affects the cost $J_n$, because the lowest priority task $\tau_n$ does not interfere with any other task. We have

$$\frac{\partial J_n}{\partial U_n} = -\frac{a_n}{U_n^2} \tag{12}$$

Since $a_n \geq 0$, $J_n$ is a decreasing function of $U_n$. Then it follows that the best value for $U_n$ is the largest possible. Hence the bandwidth constraint of Equation (10) is always adherent (it holds with the equal sign), and we have

$$U_n = 1 - \sum_{j=1}^{n-1} U_j \tag{13}$$

This also has an intuitive explanation: since $\tau_n$ is the lowest priority task, it does not interfere with any other task. Hence, increasing $U_n$ can only benefit $\tau_n$ without causing any damage to the other tasks. From (13), the cost $J_n$ becomes

$$J_n = \frac{a_n}{U_n} + \frac{b_n}{1 - \sum_{j=1}^{n-1} U_j} = \frac{a_n + b_n}{1 - \sum_{j=1}^{n-1} U_j} \tag{14}$$

The minimization problem (11) is then performed on the first $n-1$ utilizations only, because $U_n$ is implicitly assigned by (13). It can be rewritten as

$$\begin{aligned} &\text{minimize } J = \sum_{i=1}^{n-1} \left( \frac{a_i}{U_i} + \frac{b_i}{1 - \sum_{j=1}^{i-1} U_j} \right) + \frac{a_n + b_n}{1 - \sum_{j=1}^{n-1} U_j} \\ &\text{subject to } \sum_{i=1}^{n-1} U_i \leq 1, \quad U_i \geq 0 \end{aligned} \tag{15}$$

We observe that $J$ goes to $+\infty$ at the boundary of the domain of the feasible utilizations,

$$D = \left\{ (U_1, \ldots, U_{n-1}) \in \mathbb{R}^{n-1} \colon U_i \geq 0, \sum_{i=1}^{n-1} U_i \leq 1 \right\} \tag{16}$$

In fact, when some $U_i \to 0$ then the corresponding $J_i \to +\infty$ because it has a term $\frac{a_i}{U_i}$ in it. When $\sum_{i=1}^{n-1} U_i \to 1$ then $J_n \to +\infty$. Hence the minimum cost utilization assignment must occur in the interior $\mathring{D}$. This observation allows us to disregard all the constraints of the minimization problem, and the problem can then be solved by unconstrained minimization.

Since $J$ is differentiable, the minimum must satisfy the null gradient condition

$$k = 1, \ldots, n-1 \qquad \frac{\partial J}{\partial U_k} = \sum_{i=1}^{n} \frac{\partial J_i}{\partial U_k} = 0 \tag{17}$$

If we differentiate $J_n$ with respect to all the utilizations $U_1, \ldots, U_{n-1}$ we find

$$k = 1, \ldots, n-1 \qquad \frac{\partial J_n}{\partial U_k} = \frac{a_n + b_n}{(1 - \sum_{j=1}^{n-1} U_j)^2} \tag{18}$$

Now we differentiate the costs $J_1, \ldots, J_{n-1}$ with respect to the controller utilizations. Since the value of $U_k$ does not affect the tasks $\tau_i$ with higher priority, we have

$$k = 1, \ldots, n-1,\ k > i \qquad \frac{\partial J_i}{\partial U_k} = 0 \tag{19}$$

Moreover we simply have

$$k = 1, \ldots, n-1 \qquad \frac{\partial J_k}{\partial U_k} = -\frac{a_k}{U_k^2} \tag{20}$$

Finally when $k < i$ we have

$$k = 1, \ldots, n-1,\ k < i \leq n-1 \qquad \frac{\partial J_i}{\partial U_k} = \frac{b_i}{(1 - \sum_{j=1}^{i-1} U_j)^2} \tag{21}$$

Now we write the null gradient condition of (17) for all its components, for $k$ from $n-1$ down to 1:

$$\frac{\partial J}{\partial U_{n-1}} = 0 \;\Rightarrow\; \frac{a_{n-1}}{U_{n-1}^2} = \frac{a_n + b_n}{(1 - \sum_{j=1}^{n-1} U_j)^2} \tag{22}$$

$$\frac{a_{n-2}}{U_{n-2}^2} = \frac{b_{n-1}}{(1 - \sum_{j=1}^{n-2} U_j)^2} + \frac{a_n + b_n}{(1 - \sum_{j=1}^{n-1} U_j)^2} \tag{23}$$

$$\ldots$$

$$\frac{a_k}{U_k^2} = \sum_{i=k+1}^{n-1} \frac{b_i}{(1 - \sum_{j=1}^{i-1} U_j)^2} + \frac{a_n + b_n}{(1 - \sum_{j=1}^{n-1} U_j)^2} \tag{24}$$

To find the closed solution for the optimum it is necessary to explore the properties of the relationship

$$\frac{\mu_k^2}{U_k^2} = \frac{\lambda_k^2}{(1-\sum_{j=1}^{k} U_j)^2} \tag{25}$$

for any $\mu_k$ and $\lambda_k$, since the (22)–(24) can be written in a similar way to (25).

From (25) it follows that

$$\frac{\mu_k}{U_k} = \frac{\lambda_k}{1-\sum_{j=1}^{k} U_j}$$

$$U_k = \frac{\mu_k}{\lambda_k}(1-\sum_{j=1}^{k} U_j) = \frac{\mu_k}{\lambda_k}(1-\sum_{j=1}^{k-1} U_j) - \frac{\mu_k}{\lambda_k}U_k \tag{26}$$

$$(1+\frac{\mu_k}{\lambda_k})U_k = \frac{\mu_k}{\lambda_k}(1-\sum_{j=1}^{k-1} U_j) \tag{27}$$

$$U_k = \frac{\mu_k}{\lambda_k+\mu_k}(1-\sum_{j=1}^{k-1} U_j) \tag{28}$$

$$1-\sum_{j=1}^{k} U_j = \frac{\lambda_k}{\mu_k}U_k = \frac{\lambda_k}{\lambda_k+\mu_k}(1-\sum_{j=1}^{k-1} U_j) \tag{29}$$

$$\frac{1}{1-\sum_{j=1}^{k} U_j} = \frac{\lambda_k+\mu_k}{\lambda_k}\,\frac{1}{1-\sum_{j=1}^{k-1} U_j} \tag{30}$$

Equations (28) and (29) also provide an interesting interpretation of the coefficients $\mu_k$ and $\lambda_k$. Basically $\mu_k + \lambda_k$ is proportional to the bandwidth $1-\sum_{j=1}^{k-1} U_j$ that is available to the task $\tau_k$. Then the task $\tau_k$ uses a bandwidth $U_k$ that is proportional to $\mu_k$ and leaves $\lambda_k$ for the lower priority tasks.

Thanks to the found relationship it is possible to find a solution of (22)–(24). We rewrite (22), introducing $\mu_{n-1}$ and $\lambda_{n-1}$,

$$\frac{a_{n-1}}{U_{n-1}^2} = \frac{a_n+b_n}{(1-\sum_{j=1}^{n-1} U_j)^2}$$

$$\Rightarrow \quad \frac{\mu_{n-1}^2}{U_{n-1}^2} = \frac{\lambda_{n-1}^2}{(1-\sum_{j=1}^{n-1} U_j)^2} \tag{31}$$

For all the other equations we have

$$\frac{a_k}{U_k^2} = \sum_{i=k+1}^{n-1} \frac{b_i}{(1-\sum_{j=1}^{i-1} U_j)^2} + \frac{a_n+b_n}{(1-\sum_{j=1}^{n-1} U_j)^2}$$

$$\Rightarrow \quad \frac{\mu_k^2}{U_k^2} = \frac{\lambda_k^2}{(1-\sum_{j=1}^{k} U_j)^2} \tag{32}$$

$$\frac{a_{k-1}}{U_{k-1}^2} = \frac{\mu_{k-1}^2}{U_{k-1}^2} = \frac{b_k}{(1-\sum_{j=1}^{k-1} U_j)^2} + \frac{\lambda_k^2}{(1-\sum_{j=1}^{k} U_j)^2}$$

$$= \frac{\lambda_{k-1}^2}{(1-\sum_{j=1}^{k-1} U_j)^2} \tag{33}$$

Recalling (30), the following recursive definition of $\mu_k$ and $\lambda_k$ holds:

$$\begin{cases} \mu_k = \sqrt{a_k} & k = 1,\ldots,n-1 \\ \lambda_{n-1} = \sqrt{a_n+b_n} & \\ \lambda_{k-1} = \sqrt{b_k+(\lambda_k+\mu_k)^2} & k = 2,\ldots,n-1 \end{cases} \tag{34}$$

Notice that $\mu_k$ and $\lambda_k$ are functions of only the input parameters $a_1,\ldots,a_n,b_1,\ldots,b_n$.

The values of $\mu_k$ and $\lambda_k$ can be used to express the solution of the problem. From (28) we can find the optimal utilization $U_1$ of task $\tau_1$,

$$U_1 = \frac{\mu_1}{\lambda_1+\mu_1} \tag{35}$$

and, in general, from (28) and (26) we have

$$U_k = \frac{\mu_k}{\lambda_k+\mu_k}(1-\sum_{j=1}^{k-1} U_j) = \frac{\mu_k}{\lambda_k+\mu_k}\frac{\lambda_{k-1}}{\mu_{k-1}}U_{k-1} \tag{36}$$

from which we finally have the closed solution

$$\begin{aligned} U_k &= U_1\frac{\mu_k}{\mu_1}\prod_{j=1}^{k-1}\frac{\lambda_j}{\lambda_{j+1}+\mu_{j+1}} \\ \frac{U_k}{\mu_k} &= \frac{U_1}{\mu_1}\prod_{j=1}^{k-1}\frac{\lambda_j}{\lambda_{j+1}+\mu_{j+1}} \end{aligned} \tag{37}$$

### 3.3 Iterative Period Assignment

The closed solution above gives the optimum task periods in the case that all the cost functions are exactly linear in $T$ and $\Delta$. In general, however, the cost of each control loop may be given by a nonlinear but smooth function $J_i(T,\Delta)$. The period assignment can then be performed iteratively as follows.

Based on the control specifications, nominal task periods $T_i^0$ are assigned and nominal task delays $\Delta_i^0$ are computed. For $k = 1, 2, \ldots$, the following steps are then performed:

1. Evaluate $\alpha_i^k = \frac{\partial J_i(T_i^k,\Delta_i^k)}{\partial T}$ and $\beta_i^k = \frac{\partial J_i(T_i^k,\Delta_i^k)}{\partial \Delta}$ numerically for all tasks using e.g. finite difference approximations and the Jitterbug toolbox [13].

2. Assign new periods $T_i^k$ using (37) based on the linearized cost functions $J_i^k(T,\Delta) = \alpha_i^k T + \beta_i^k \Delta$.

3. Evaluate $\Delta_i^k$ and redesign the controllers according to $T_i^k$ and $\Delta_i^k$.

To guarantee the convergence of the algorithm, further assumptions on the cost functions $J_i(T, \Delta)$ must be made. This however lies outside the scope of the current paper.

# 4 The Priority Assignment

In the previous section we assumed that the priorities of the control tasks are given. Here we expose two possible strategies for assigning priorities.

**Seto96RM.** One technique to assign the static priorities to the control tasks is to assign rate-monotonic (RM) priorities based on the periods returned by the algorithm in Seto's classical paper [17]. The advantage of this technique is its low run-time. However, this assignment is delay-insensitive, since it does not take the cost parameter $\beta_i$ into account.

**BruteForce.** Since the optimal period assignment for a given priority assignment is extremely simple (it has complexity $\mathcal{O}(n)$), we could try to test all the possible $n!$ priority assignments and then select the one that gives the smallest cost. Even though this method is brute force and is clearly not scalable with $n$, it finishes in a few minutes for a dozen tasks.

Other suboptimal but more efficient methods for assigning priorities to control tasks will be investigated in future work.

# 5 Performance Evaluation

In this section we evaluate the proposed period assignment method against other existing methods in the literature. The methods that we will consider are the following.

**RiApprox.** The method proposed in this paper, minimizing $\sum_i \alpha_i T_i + \beta_i \Delta_i$ assuming the delay $\Delta_i$ equal to the response time approximation $R_i^{\text{approx}}$ of (6).

**Seto96.** The method proposed in Seto et al. [17], minimizing $\sum_i \alpha_i T_i$ over the constraint $\sum_i \frac{C_i}{T_i} \leq 1$ or $\sum_i \frac{C_i}{T_i} \leq U_{\text{RM}}^{\text{lub}}$. This period assignment method neglects the effect of delay.

**FirstVertex.** The suboptimal method proposed in Bini and Di Natale [5], minimizing $\sum_i \alpha_i T_i$ subject to the FP schedulability constraint. The suboptimal method is much faster than the true optimal solution that requires integer linear programs to be solved but performs nearly as well. This period assignment method also neglects the effect of delay, although thanks to the FP schedulability constraint, it guarantees that the delay $\Delta_i$ will never exceed the period $T_i$.

In Section 5.1 we estimate the amount of bias introduced by the response time approximation $R_i^{\text{approx}}$. In Section 5.2 we evaluate the cost on synthetic task sets. Finally, in Section 5.3 we evaluate the overall control cost in detailed co-simulations.

## 5.1 Quality of the Delay Approximation

In Section 3.1 we set the control delay $\Delta_i$ equal to the response time approximation $R_i^{\text{approx}}$. In this section we are going to evaluate the amount of bias introduced by this delay estimate.

In this experiment we assumed $n \in \{3, 7, 19\}$. The computation times are uniformly distributed in $(0.01/n, 0.1/n)$. The cost coefficients $\alpha_i$ and $\beta_i$ are extracted using a probability density that reproduces a posteriori the same statistical distribution observed in the experiments made in Section 5.3. For each task number $n$ we generated 1000 task sets. For each task set we have proceeded as follows:

1. We assigned the priorities according to Seto96RM.
2. We computed the optimal periods $T_i$, using RiApprox.
3. Starting from the computation times $C_i$ and the periods $T_i$ we estimated the average task response time $R_i^{\text{avg}}$. This value is extracted by making the average of the first 50 jobs of the task $\tau_i$. Notice that the periods $T_i$ returned by RiApprox are real-valued, hence it is not possible to compute the response times of all the jobs within the hyperperiod, because the hyperperiod does not exist.

In Figure 2 we report the results. For each number of tasks $n$ we report the statistics of $\frac{R_i^{\text{approx}}}{R_i^{\text{avg}}}$ that we call "approximation bias" (reported on the $y$-axis). The tasks are sorted along the $x$-axis from the highest to the lowest priority. A thick line marks the average value for all the 1000 experiments, and a dot is placed in correspondence of a task. Also the stripe of the average value plus/minus the standard deviation is drawn.

In can be noted that $R_i^{\text{approx}}$ is, on the average, some amount above the average response time $R_i^{\text{avg}}$. This over-estimation is small for high priority tasks and it increases as the priority becomes smaller. A good property of $R_i^{\text{approx}}$ seems to be that the factor of over-estimation does not scale with the number of tasks.

For the lowest priority task we notice an unexpected behavior, since in this case $R_i^{\text{approx}}$ becomes smaller than $R_i^{\text{avg}}$. A partial explanation of this behavior can be found in the experiment methodology. The periods returned by $T_i$

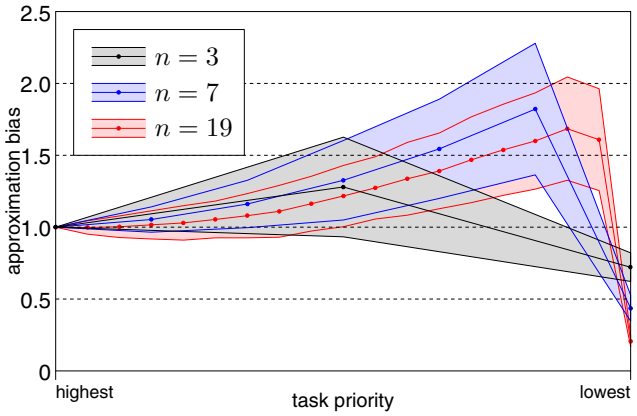


**Figure 2. Quality of the response time approximation.**

returned by the method RiApprox are such that the processor is fully utilized (see Eq. (13)). Hence the first 50 jobs, used to compute $R_i^{\text{avg}}$ are probably experiencing more interference than the average. This could partially motivate the fact that the real average response time is lower than the value $R_i^{\text{avg}}$ measured in our experiment.

We can conclude that the usage of $R_i^{\text{approx}}$ to estimate the delay introduces an overestimation of the delay. This negative aspect however does not compromise the validity of the presented results. First of all, it is not clear whether the delay $\Delta_i$ experienced by the task is well modeled by $R_i^{\text{avg}}$. Moreover this bias can be corrected by adjusting the values of $\beta_i$, as shown in the pseudo-code (in Matlab) reported in Figure 3.

The function RiApproxIter starts by invoking the efficient algorithm RiApprox. Using the returned vector of periods $\mathbf{T}$ and the computation times $\mathbf{C}$, it estimates the vector of delays $\mathbf{\Delta}$ experienced by each task using, for example, the average response time of some initial jobs. Then, it compares the delays with the response time approximation $\mathbf{R}^{\text{approx}}$ (that was assumed by RiApprox to be the delay for finding $\mathbf{T}$), storing the bias factor. The function continues invoking RiApprox until the bias factor has become stable.

## 5.2 Evaluation on Synthetic Tasks

In this experiment we have investigated the cost reduction that is possible to achieve by the presented methods. In this context we compare the three period assignment methods FirstVertex, RiApprox, and RiApproxIter (the iterative method that was described in Section 5.1). The cost achieved by the three methods is divided by the cost achieved by Seto96.

The number of tasks $n$ is set to 5. The computation times $C_i$ and the cost coefficient $\alpha_i$ are generated as in the experiment of Section 5.1. The coefficient $\beta_i$ is extracted uniformly in $[0, 2s\,\beta_i^{\text{ref}}]$, where $\beta_i^{\text{ref}}$ is extracted in the same way as in was extracted $\beta_i$ in Sec. 5.1, and $s$ indicates the *sensitivity to the delay*. In fact, if $s = 0$ then all the $\beta_i$ will be set equal to zero. As $s$ increases the values of $\beta_i$ will increase as well. For each method the cost is evaluated assuming as delay the average response time of the first 10 jobs $R_i^{\text{avg}}$.

```
function T = periodRiApprox(C, α, β)
% Returns a vector of optimal periods

function Δ = delayEstimate(C, T);
% Returns a vector of delay estimates. It can be
% done, for example, by simulating the schedule
% and returning the average of the response times

function T = periodRiApproxIter(C, α, β)
err = 1;
bias = ones(1,n);
while (err > η),   % stop when err below η
  % invoke the period assignment
  T = periodRiApprox(C, α, β./bias);
  % estimate the delay
  Δ = delayEstimate(C, T);
  % evaluate the bias
  oldBias = bias;
  bias = R^approx./Δ;
  err = norm((oldBias−bias)./bias);
end
```

**Figure 3. Pseudo-code of** RiApproxIter

In Figure 4 we show the results. We plot the cost of the three methods normalized with the cost by Seto96, as a function of the sensitivity to the delay $s$. It can be noticed that when $s = 0$ the methods RiApprox and RiApproxIter gives the same cost as Seto96. This is quite evident, since when all $\beta_i = 0$ the minimization problem of Eq. (11) is exactly the same as Seto's. FirstVertex is worse, because it bounds the delay by the period even though $\beta_i = 0$, which is clearly suboptimal.

As the sensitivity to delay $s$ increases all the three methods improve. When $s \approx 0.75$, FirstVertex returns solutions with lower cost than Seto96. As expected both RiApprox and RiApproxIter reduce the overall cost, as the system becomes more sensitive to the delay.

Finally it can be noticed that the benefit of RiApproxIter w.r.t. the simpler RiApprox is quite negligible. Hence we can affirm that even if the response time approximation $R_i^{\text{approx}}$ can deviate significantly than the average response time $R_i^{\text{avg}}$, the effect of this difference is negligible in terms of the system cost of the final solution. Hence we drop the

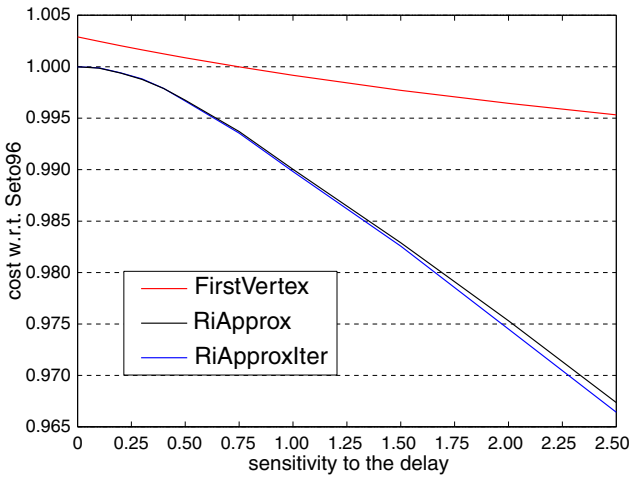


**Figure 4.**

method RiApproxIter in the final experiment, since its long execution time does not pay off,

## 5.3 Evaluation of the Control Cost

For the control evaluation, we considered a set of random tasks controlling a set of randomly generated plants.

### 5.3.1 Random Task Generation

For the control evaluation, we considered $n = \{2, 5, 10, 20\}$ control tasks. Nominal task utilizations $U_i^{\text{nom}}$ were generated using an $n$-dimensional uniform distribution with total utilization 1 [4]. The execution time was given by $C_i \in U(0.01, 0.1)/n$. Nominal task periods were given by $T_i^0 = C_i/U_i^{\text{nom}}$ and priorities were assigned based on the periods returned by Seto96.

### 5.3.2 Random Plant Generation

In order to vary the sensitivity towards delay, random plants were generated in three different families.

In Family I, all plants had two stable poles, with each plant with equal probability drawn from

$$P_1(s) = \frac{1}{(s+a_1)(s+a_2)}, \quad P_2(s) = \frac{1}{s^2 + 2\zeta\omega s + \omega^2}$$

with $a_{1,2} \in U(0,1), \omega \in U(0,1), \zeta \in U(0,1)$.

In Family II, all plants had two stable and/or unstable poles, with each plant with equal probability drawn from

$$P_3(s) = \frac{1}{(s+a_1)(s+a_2)}, \quad P_4(s) = \frac{1}{s^2 + 2\zeta\omega s + \omega^2}$$

with $a_{1,2} \in U(-1,1), \omega \in U(0,1), \zeta \in U(-1,1)$.

In Family III, all plants had three stable and/or unstable poles, with each plant with equal probability drawn from

$$P_5(s) = \frac{1}{(s+a_1)(s+a_2)(s+a_3)},$$
$$P_6(s) = \frac{1}{(s^2 + 2\zeta\omega s + \omega^2)(s+a_3)}$$

with $a_{1,2,3} \in U(-1,1), \omega \in U(0,1), \zeta \in U(-1,1)$.

Unstable plants are more sensitive to delay and jitter, and we would hence expect to see a larger performance difference for Families II and III than for Family I.

For the control design we throughout assumed the parameters $\rho = 0.01$, $R_{1c} = BB^T$, and $R_2 = 0.01\,\text{tr}\{R1_c\}$. (Here, $\text{tr}\{\cdot\}$ denotes trace.) Optimal LQG controllers were then designed assuming the assigned period $T$ and expected control delay $\Delta$.

### 5.3.3 Simulation Results

The various period assignment methods were evaluated in Monte Carlo simulations, where the plants (including the random disturbances), the controllers, and the scheduler were simulated in parallel using the TrueTime simulation toolbox [10]. From each family of plants, 20 random plants and controllers were generated and simulated for 100 s, the total cost $J$ being recorded. The cost under Seto96 (with $U = U_{\text{RM}}^{\text{lub}}$), FirstVertex, and RiApprox were compared to an ideal case where each controller had its own CPU and could execute at its maximum possible rate, $T_i = C_i$. From the simulation results, the average performance degradation in percent and the standard deviation was computed.

For Family I the results were as follows:

| | $n=2$ | $n=5$ | $n=10$ | $n=20$ |
|---|---|---|---|---|
| Seto96 | $20(^{+8}_{-9})$ | $39 \pm 3$ | $49 \pm 7$ | $67 \pm 3$ |
| FirstVertex | $16(^{+9}_{-9})$ | $35 \pm 3$ | $50 \pm 12$ | $59 \pm 5$ |
| RiApprox | $16(^{+6}_{-8})$ | $33 \pm 3$ | $42 \pm 6$ | $55 \pm 4$ |

For Family II the results were:

| | $n=2$ | $n=5$ | $n=10$ | $n=20$ |
|---|---|---|---|---|
| Seto96 | $24 \pm 6$ | $51 \pm 9$ | $60 \pm 11$ | $87 \pm 14$ |
| FirstVertex | $22 \pm 6$ | $42 \pm 9$ | $57 \pm 6$ | $76 \pm 10$ |
| RiApprox | $19 \pm 5$ | $40 \pm 6$ | $48 \pm 8$ | $66 \pm 4$ |

Finally, for Family III the results were:

| | $n=2$ | $n=5$ | $n=10$ | $n=20$ |
|---|---|---|---|---|
| Seto96 | $27 \pm 3$ | $81 \pm 24$ | $121 \pm 13$ | $155 \pm 9$ |
| FirstVertex | $27 \pm 8$ | $61 \pm 14$ | $108 \pm 11$ | $146 \pm 9$ |
| RiApprox | $24 \pm 4$ | $102 \pm 79$ | $100 \pm 12$ | $128 \pm 7$ |

It is seen that all period assignment schemes introduce some performance degradation. Overall RiApprox outperforms FirstVertex which in turn outperforms Seto96. The

former achieve a utilization close to 1 while Seto96 stays at the RM least upper bound. As expected the performance degradation is the worst for Family III, and there the period assignment scheme matters the most. Also, the choice of method seems to be more important the larger the task set.

## 6 Conclusion

The paper has tackled the control–scheduling co-design problem of optimal period assignment for multi-loop control systems. The key improvement compared to previous work has been the closed-form formulas for the approximate control delay and optimal periods, which enabled delay-aware period assignment under fixed-priority scheduling. Detailed co-simulations showed that delay-aware period assignment could indeed significantly the reduce the implementation-induced performance degradation, especially for large task sets and delay-sensitive plants.

For future work, it would be interesting to investigate how to assign optimal priorities to the controllers. An extension to the EDF case would also be of interest. From the control perspective, it would be useful to develop approximate formulas also for the jitter and include its effect in the cost function.

## References


[1] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems. Theory and Design*. Prentice Hall, third edition, 1997.

[2] N. C. Audsley, A. Burns, M. Richardson, K. W. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sept. 1993.

[3] E. Bini and S. K. Baruah. Efficient computation of response time bounds under fixed-priority scheduling. In *Proceedings of the* $15^{th}$ *conference on Real-Time and Network Systems*, pages 95–104, Nancy, France, Mar. 2007.

[4] E. Bini and G. C. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the* $16^{th}$ *Euromicro Conference on Real-Time Systems*, pages 196–203, Catania, Italy, June 2004.

[5] E. Bini and M. Di Natale. Optimal task rate selection in fixed priority systems. In *Proceedings of the* $26^{th}$ *IEEE Real-Time Systems Symposium*, pages 399–409, Miami (FL), U.S.A., Dec. 2005.

[6] G. Buttazzo and A. Cervin. Comparative assessment and evaluation of jitter control methods. In *Proceedings of the* $15^{th}$ *conference on Real-Time and Network Systems*, pages 163–172, Nancy, France, Mar. 2007.

[7] A. Cervin. Improved scheduling of control tasks. In *Proceedings of the* $11^{th}$ *Euromicro Conference on Real-Time Systems*, pages 4–10, York, UK, June 1999.

[8] A. Crespo, I. Ripoll, and P. Albertos. Reducing delays in RT control: the control action interval. In *Proceedings of the* $14^{th}$ *IFAC World Congress*, pages 257–262, Beijing, China, July 1999.

[9] D. Henriksson and A. Cervin. Optimal on-line sampling period assignment for real-time control tasks based on plant state information. In *Proceedings of the* $44^{th}$ *IEEE Conference on Decision and Control and European Control Conference*, Seville, Spain, Dec. 2005.

[10] D. Henriksson, A. Cervin, and K.-E. Årzén. TrueTime: Simulation of control loops under shared computer resources. In *Proceedings of the* $15^{th}$ *IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.

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

[12] B. K. Kim. Task scheduling with feedback latency for real-time control systems. In *Proceedings of the* $5^{th}$ *International Workshop on Real-Time Computing Systems and Applications*, pages 37–41, Hiroshima, Japan, Oct. 1998.

[13] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proceedings of the* $41^{st}$ *IEEE Conference on Decision and Control*, Las Vegas, NV U.S.A., Dec. 2002.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[15] P. Martí, J. M. Fuertes, K. Ramamritham, and G. Fohler. Jitter compensation for real-time control systems. In *Proceedings of the* $22^{nd}$ *IEEE Real-Time System Symposium*, pages 39–48, London, UK, Dec. 2001.

[16] P. Martí, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes. Optimal state feedback based resource allocation for resource-constrained control tasks. In *Proceedings of the* $25^{th}$ *IEEE Real-Time Systems Symposium*, pages 161–172, Lisbon, Portugal, Dec. 2004.

[17] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proceedings of the* $17^{th}$ *IEEE Real-Time Systems Symposium*, pages 13–21, Washington (DC), U.S.A., Dec. 1996.

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

[19] M. Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-Time Systems*, 14(3):219–250, 1998.