Minimum deadline calculation for periodic real-time tasks in dynamic priority systems

Patricia Balbastre, Ismael Ripoll and Alfons Crespo
Department of Computer Engineering
Technical University of Valencia, Spain
{patricia, iripoll, alfons}@disca.upv.es

***Abstract*—Real-time systems are often designed using a set of periodic tasks. Task periods are usually set by the system requirements, but deadlines and computation times can be modified in order to improve system performance. Sensitivity analysis in real-time systems has focused on changes in task computation times, using fixed priority analysis. Only a few studies deal with the modification of deadlines in dynamic-priority scheduling. The aim of this work is to provide a sensitivity analysis for task deadlines in the context of dynamic-priority, pre-emptive, uniprocessor scheduling. In this paper, we present a deadline minimisation method that computes the shortest deadline of a periodic task. As undertaken in other studies concerning computation times, we also define and calculate the critical scaling factor for task deadlines. Our proposal is evaluated and compared with other works. The deadline minimisation proposed strongly reduces jitter and response time of control tasks, which can lead to a significant improvement of the system performance.**

***Index Terms*—D.4.7.e Real-time systems, periodic task systems, D.4.1.e Scheduling, earliest deadline first.**

I. Introduction

A. Motivation

REAL-TIME systems are often designed using a set of periodic activities running on top of a real-time operating system. For instance, in control systems, the correct behaviour of the closed-loop controller requires that the system meet timing constraints such as periods and latencies, which are expressed as deadlines. Widely established scheduling policies, such as Rate [1] or Deadline Monotonic [2] for fixed priorities and EDF for dynamic priorities assign priorities according to timing parameters, sampling periods and deadlines respectively.

Both timing parameters, periods and deadlines, are susceptible to adjustment during the design phase of the control system. While modifications in the periods may require the redesigning of the controller in order to adjust it to the new rate, deadline adjustments do not. A modification in the deadline of a control activity can, for instance, increase or decrease the task priority if Deadline Monotonic criteria is used in a fixed priority scheme. When using the EDF scheduling policy, a deadline modification does not have a direct effect on the task priority as in the case of fixed priority scheduling. The effect is acknowledged in a variation in the task response time throughout the task activations. In both cases, this effect can be measured through the output jitter of the task.

In control systems, the influence of jitter on performance quality is not always easy to analyse. From a control perspective, sampling jitter and latency jitter can be interpreted as disturbances acting upon the control system. The input-output latency decreases the stability margin and limits the performance of the system. If the jitter and the latency are small, then these can be overlooked. Otherwise, they should be accounted for in the control design or, if possible, compensated for at run-time. Further information regarding these issues is detailed in [3]. In general, output jitter has a negative effect on the control performance. This can be significant depending on the term known as control effort introduced by Albertos and Olivares [4]. The control effort measures how sensitive a control task is to time delays. The reduction of the output jitter and the subsequent improvement in the control performance, is directly associated with the reduction of task deadlines.

Multimedia applications are also sensitive to jitter. In these applications, jitter measures the variability of delay of packets in the given stream. Ideally, packets should be delivery in a perfectly periodic fashion [5] but, due to variable queuing and propagation delays, packets arrive at the destination with a wide range of inter-arrival times. Delay jitter can be eliminated by buffering at the receiver [6]. Specifically, the maximum expected delay can be translated to the maximum buffer size needed at the destination. If jitter is not bounded the buffer space can be significant. The problem is that not all receiving systems can have enough memory for buffering. Thus, in multimedia applications jitter reduction may lead to significant buffer space savings.

The output jitter reduction can be regarded as a more general question related to deadline minimisation. Questions such as: ”By how much can a task deadline be reduced?”, or, ”What is the maximum deadline reduction factor needed to maintain the schedulability of a set of tasks?, can be of interest for a set of applications. For example, in real-time databases, temporal freshness of data is assured and workload minimised by assigning a deadline lower than the transaction’s period, such as in the *More-Less* model proposed in [7] by Xiong

This work was partially granted by the Spanish Government Research Office (CICYT) under grant DPI2005-09327-C02-02

Digital Object Indentifier 10.1109/TC.2007.70787

and Ramamritham. Most of the work on feasibility analysis in real-time systems provides a yes or no answer, in other words, whether the system is feasible or not. However, given a schedulable set of tasks, is it possible to determine the smallest deadline or period, or the largest computation time, which will still render the system schedulable? This kind of analysis is referred to in the literature as *sensitivity analysis*.

B. Related work

Sensitivity analysis has focused on permissible changes on tasks WCET, mainly because this has been applied to fault tolerance design. In this sense, Lehoczky, Vestal and Punnekkat[8], [9], [10] define the *Critical scaling factor*, as the largest possible scaling factor for task computation times while the task set to remain schedulable. These works assume Rate Monotonic priority assignment and deadlines equal to periods.

Sensitivity analysis for computation times using EDF scheduling has been performed by Balbastre *et* al. in [11]. Moreover, the *Optional Computation Window* (OCW) is defined as the possibility that a task can execute in n activations in a window of m consecutive invocations with an increased computation time than the initially assigned. Deadlines less than or equal to periods are assumed.

Sensitivity analysis for task periods was performed by Buttazzo *et* al. [12]. Requested variations in task execution rates or overload conditions are managed by changing the rates based on the spring elastic coefficients. This work assumes deadlines equal to periods and EDF scheduling. This technique is very useful in multimedia applications, since in control systems a modification in the period requires to redesign the controller.

Regarding deadline sensitivity analysis, there are some papers which assign new deadlines to periodic tasks in order to achieve a secondary objective. Cervin *et* al. [13] calculate new deadlines for control tasks in order to guarantee close-loop stability of real-time control systems. Baruah *et* al. [14] developed two methods to minimise output jitter of tasks by assigning shorter relative deadlines. The first method does not achieve the minimum deadline but it has polynomial time complexity. The second method proposed by the authors starts from the value obtained in the first method. This value is decreased until the system is not feasible. This requires to apply EDF feasibility test every time the deadline is decreased, so it has pseudo-polynomial time complexity. Regarding aperiodic tasks, the minimum deadline assignment is presented by Ripoll *et* al. in [15], [16]. Finding the minimum deadline of a periodic task has been also independently addressed by Hoang *et* al. in [17], using a similar approach. However, as it will be shown in Section VI, our algorithm is faster.

Since reducing task periods and deadlines is equivalent to modify the operating frequency of the processor, sensitivity analysis can be used in the context of power-aware applications. Pillai and Shin [18] derive the maximum speed that can make a task set schedulable under EDF, assuming that deadlines are equal to periods. This assumption is removed in [19] by Jejurikar and Gupta, where two methods for computing constant and variable slowdown factors were provided. Our proposal differs from Jejurikar and Gupta's method in the sense that we do not change the deadlines proportionally to periods but we consider that periods are fixed. Jejurikar and Gupta's approach can be applied to shortening only deadlines (while periods remain fixed) but the resulting deadline is not the minimum. In order to reduce jitter in control applications, periods and deadlines can be shortened at once using Jejurikar's approach but, as shown in Section VI the slowdown factor obtained is greater and the jitter reduction is lower than the obtained with our proposal.

Regarding output jitter, several studies use scheduling solutions in order to reduce this. In [20] the difference between two consecutive finishing times of the same task is required to be bounded by a specific value. Natale and Stankovic [21] presented a new scheduling approach to minimise jitter in distributed real-time systems. In [22], [23], high priority tasks are used to reduce the jitter. Kim *et* al. [24] assign new deadlines to tasks scheduled under EDF, using an integer linear program. It can be noted that, in all these studies dealing with deadline reduction except for the work developed by Hoang *et* al., this reduction is not optimal in the sense that deadlines can be minimised any further. In order to minimise control jitter, other strategies have been used. Crespo [25], Cervin [26] and Balbastre [27] propose a subtask partition, using dynamic priorities. Furthermore, in [25] and [27] control jitter is both reduced and determined off-line the variation range, enabling control designers to take decisions before executing the control system in order to achieve a better performance. In these works, task splitting is combined with deadline reduction, so higher priority tasks are assigned a shorter deadline than the rest of the tasks. New deadlines are calculated following an iterative algorithm, where the worst case response time of the task is assigned as a new deadline. When no further improvement in the response time is obtained, the algorithm stops. The drawback of this method is that worst case response times in EDF are not easy to obtain, implying a high computational cost.

C. Contributions

The aim of this work is to provide a sensitivity analysis for task deadlines in the context of dynamic-priority, preemptive, uniprocessor scheduling. Specifically, this paper integrates and extends some preliminary results presented by Balbastre *et* al. in [28]. Three different kinds of analysis have been carried out:

- Finding the minimum deadline of a periodic task. This analysis is made off-line. Note that the maximum reduction implies that a lower deadline makes the system not feasible. We will demonstrate that, in this case, worst case response times of tasks are known off-line. Therefore, there is no need to calculate them.
- Finding the minimum deadline of any task job. This algorithm is applied on-line, whenever is convenient to reduce some task jobs during the execution of the system.
- Finding the maximum reduction for all task deadlines, that is, the *critical scaling factor*. The critical scaling factor is the lowest number by which the deadline of all

tasks in a set can be multiplied without rendering the task set infeasible.

This deadline reduction can be used subsequently to reduce output jitter, control jitter, response times of tasks and, in general, to improve system responsiveness. It also can be used to replace the iterative algorithm used by Cervin [26] and Balbastre *et* al. [27], so there is no need to calculate worst case response times in EDF.

D. Outline

The remainder of this paper is organised as follows: Section II introduces the computational model and the assumptions used. Section III describes the general deadline minimisation algorithm for periodic tasks. In section IV the method is extended to cope with more general situations. An example to illustrate how the task deadline minimisation works is presented in Section V. We have implemented the deadline minimisation algorithm, and the critical scaling factor calculation, and we have conducted some experiments in order to assess the validity and effectiveness of our proposals. The results and the comparison with other related works are presented in Section VI. Finally, in Section VII some conclusions are highlighted and future lines of research are discussed.

II. System model and notation

Let $\tau = \{T_1, ..., T_n\}$ be a periodic task system scheduled under EDF. Each task $T_i \in \tau$ has the following temporal parameters $T_i = (C_i, D_i, P_i)$. Where P_i is the period; D_i is the deadline relative to the start time; and C_i is the worst case execution time, $C_i \leq D_i$. In this work, we consider that $D_i \leq P_i$. Each task T_i produces a sequence of jobs $J_{i,j} = (r_{i,j}, C_i, d_{i,j})$, where $r_{i,j}$ denotes the arrival time (or request time) of the j^{th} job of the task T_i, and $d_{i,j} = r_{i,j} + P_i$ is the absolute deadline. The sequence of jobs $J_{i,j}$ are arranged by absolute deadline in ascending order. Subsequently, a task set can be seen as an infinite number of activations.

Definition 1: Let $G_\tau(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{P_i} \right\rceil$. It denotes the maximum cumulative execution time requested by activations of τ whose arrival times are before time t.

Note that $G_\tau(t)$ function does not depend on task deadlines. Changing the deadlines of the tasks have no impact on the value of this function.

Definition 2: Baruah *et* al.[29] Let $H_\tau(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{t+P_i-D_i}{P_i} \right\rfloor$. It denotes the maximum cumulative execution time requested by jobs of τ whose absolute deadlines are less than or equal to time t.

Feasibility test for EDF consists of checking that $\forall t \leq \mathcal{R}$ the inequality $H_\tau(t) \leq t$ holds, where $[0, \mathcal{R})$ is the ICI of the task set τ. ICI stands for *Initial Critical Interval* and it is the first interval where the processor is not idle. $\mathcal{R}$ is the smallest $t > 0$ such that $t \leq G_\tau(t)$. This bound was proposed by Ripoll et al. in [30]. In fact, there is no need to confirm that $H_\tau(t) \leq t$ applies for all $t \leq \mathcal{R}$ to determine whether τ is feasible, except in the designated scheduling points [8].

Definition 3: The set $\mathcal{S}_\tau$ of scheduling points for τ is defined by:

$$\mathcal{S}_\tau = \left\{ d_{i,j} \quad / \quad 1 \leq i \leq n \ , \quad 1 \leq j \leq k_i \ \wedge \ d_{i,j} \leq \mathcal{R} \right\}$$

where $k_i = \left\lceil \frac{\mathcal{R}}{P_i} \right\rceil$ denotes the number of activations of T_i in $[0, \mathcal{R})$.

Definition 4: Let $\alpha_i = 1 - \frac{D'_i}{D_i}$ be the deadline reduction factor of T_i where $C_i \leq D'_i \leq D_i$

Definition 5: Let $\alpha_i^{max} = 1 - \frac{C_i}{D_i}$ be the maximum deadline reduction factor of T_i

Definition 6: Let Δ^* be the *critical scaling factor* of τ. It denotes the smallest scaling factor for task deadlines, above which the task set remains schedulable $\forall i \quad / \quad T_i = (C_i, \Delta D_i, P_i)$, where $\Delta \geq \Delta^*$.

If τ is a schedulable task set, then $\Delta^* \leq 1$, otherwise $\Delta^* > 1$. Note that $\alpha_i = 1 - \Delta^*$.

III. Minimum deadline of a periodic task

This section formally presents the theory and proposes an algorithm to reduce a task deadline to its minimum feasible value. In other words if we continue to reduce it, the system becomes unfeasible. We will assume firstly that the task system is feasible, the analysis is made off-line, and the deadline minimisation is applied only to one task, or with a pre-defined order. All these restrictions will be removed in Section IV.

The theorems presented in this section are based on the $H_\tau(t)$ function throughout the ICI. Note that if a task deadline is modified, the ICI does not change (this depends on task periods and computation times) and, whenever $H_\tau(t) < t$ in $[0, \mathcal{R})$, system feasibility is assured. Moreover, while calculating the minimum deadline of a task there is no need to re-calculate $H_\tau(t)$ function, unlike the pseudopolinomial method of Baruah *et* al. in [14].

First of all, we will calculate the minimum deadline for the first job of a task and next, the minimum deadline of the task itself.

Theorem 1: Let τ= $\{T_1, ...T_i, ...T_n\}$ be a feasible set of n periodic tasks. Let $J_{i,1}$ be the first job of T_i. If there exists a time $t_1 \in \mathcal{S}_\tau$ such that $C_i \leq t_1 \leq D_i$ and $H_\tau(t_1) = H_\tau(D_i - \varepsilon) > t_1 - C_i$ then the sequence of jobs $J_{1,1}, .., J'_{i,1}, .., J_{n,k_n}$ with $J'_{i,1} = \{0, C_i, H_\tau(t_1) + C_i\}$ is schedulable. Where ε is assumed to be an arbitrarily small positive number.

Proof:

Let $d'_{i,1} = H_\tau(t_1) + C_i$ be the absolute deadline of $J'_{i,1}$. We need to look at in which scheduling points $H_\tau(t)$ changes. In Figure 1 t_1, D_i and $d'_{i,1}$ are represented for a better understanding of the theorem.

There are no scheduling points in the time interval $[t_1, D_i)$ since $H(t)$ is a non-decreasing step function that changes its value only at scheduling points, and $H_\tau(t_1) = H_\tau(D_i - \varepsilon)$. Both $H_{\tau'}(t)$ (the new $H_\tau(t)$ function with $J'_{i,1}$ instead of $J_{i,1}$) and $H_\tau(t)$ are the same function for $t \leq t_1$ and $t > D_i$ due to the fact that the set of scheduling points remains the same except for job $J'_{i,1}$ which changes from D_i to $d'_{i,1}$.

Then, the system is feasible in $[0, t_1] \cup [D_i, \mathcal{R}]$. It is also feasible in $[t_1, d'_{i,1}) \cup (d'_{i,1}, D_i]$ since the condition $H_\tau(t_1) = H_\tau(D_i - \varepsilon)$ guarantees that there are no more scheduling points. It only remains to prove that $J'_{i,1}$ is schedulable in $t = d'_{i,1}$.

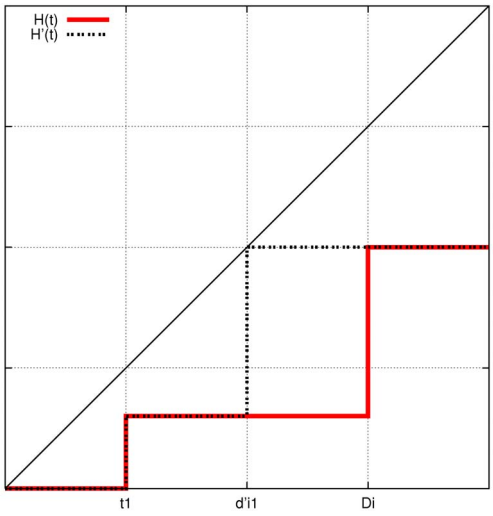


Fig. 1. Graphical representation of theorem 1

The new H_τ function in $d'_{i,1}$ will be

$$H'_\tau(d'_{i,1}) = H_\tau(t_1) + C_i$$

Since $d'_{i,1} = H_\tau(t_1) + C_i$, it follows that

$$\begin{aligned} H'_\tau(d'_{i,1}) &= d'_{i,1} + C_i - C_i \\ &= d'_{i,1} \end{aligned}$$

■

Corollary 1: If it does not exists such t_1 then $d'_{i,1} = C_i$ and the systems is schedulable.

Proof: This situation can occur when:

- There are not scheduling points in $[0, D_i)$. In this case, clearly C_i is a valid deadline for $J_{i,1}$.
- Or if there exists a scheduling point with enough slack, that is $\exists t_2 \quad / \quad H_\tau(t_2) = H_\tau(D_i - \varepsilon) \leq t_2 - C_i$ In this case, τ is schedulable in t_2 since $H'_\tau(t_2) = H_\tau(t_2) + C_i \leq t_2$. It is also schedulable in C_i since there are no scheduling points in $[0, C_i)$ and $H_\tau(C_i) = C_i$.

■

Corollary 2: $d'_{i,1}$ is the minimum deadline for job $J_{i,1}$.

Proof: Clearly, if $d'_{i,1} = C_i$, this is the minimum deadline for $J_{i,1}$. Else if $d'_{i,1} = H_\tau(t_1) + C_i$, then $d'_{i,1} - \varepsilon$ will render the set of jobs unfeasible. We need to look at schedulability in $t_{min} = d'_{i,1} - \varepsilon$.

$$H'_\tau(t_{min}) = H_\tau(t_1) + C_i$$

Since $t_{min} = d'_{i,1} - \varepsilon = H_\tau(t_2) + C_i - \varepsilon$ and substituting $H_\tau(t_1)$ in the previous equation:

$$\begin{aligned} H'_\tau(t_{min}) &= t_{min} + C_i - C_i + \varepsilon \\ &= t_{min} + \varepsilon \end{aligned}$$

and then, τ is not schedulable in t_{min}. ■

Theorem 1 shows how to calculate the minimum deadline for the first job of a task. However, it does not imply that all the job deadlines can be shortened the same amount. The next definition extends the calculation of $d'_{i,1}$ for any job $J_{i,j}$.

Definition 7: Let $d'_{i,j} = H_\tau(t_1) + C_i$ where $r_{i,j} + C_i \leq t_1 \leq d_{ij}$ and $H_\tau(t_1) = H_\tau(d_{ij} - \varepsilon) > t_1 - C_i$. If such t_1 does not exists, then $d'_{i,j} = C_i$

It is important to note that $d'_{i,j}$ (when $1 < j \leq k_i$) may not be the minimum deadline of $J_{i,j}$ (it will be calculated in section IV-D). Although Theorem 1 is valid in the sense that $H_\tau(t) \leq t$, schedulability of τ is not guaranteed. This is due to the fact that $H(t)$ is defined for periodic tasks, whose job deadlines have the same value in each activation. However, by Theorem 1 (extended for a job $J'_{i,j} = \{r_{i,j}, C_i, d'_{i,j}\}$ instead of $J'_{i,1}$), we can assure that $H_\tau(d'_{i,j}) = d'_{i,j}$. Next theorem will be based on this result in order to calculate the minimum deadline of a task.

Corollary 2: Let $\{J'_{i,1}, .., J'_{i,k_i}\}$ be a deadline ordered sequence of jobs of T_i in $[0, \mathcal{R})$, where $J'_{i,j} = \{r_{i,j}, C_i, d'_{i,j}\}$. Then, $\tau' = \{T_1, .., T'_i, .., T_n\}$ with $T'_i = \{C_i, max_{j=1}^{k_i}(d'_{i,j} - r_{i,j}), P_i\}$ is schedulable.

Proof:

Suppose that jobs belonging to T_i are arranged in ascending order by its relative deadline $(d'_{i,j} - r_{i,j})$. This results in $max_{j=1}^{k_i}(d'_{i,j} - r_{i,j}) = d'_{i,k_i} - r_{i,k_i}$. As job deadlines are calculated in accordance with Theorem 1, then in every $t_j = d'_{i,j}$, it can be assured that $H_\tau(t_j) = t_j$.

Clearly, if for any $J'_{i,j}$ a lower deadline than $d'_{i,j}$ is chosen, then at the scheduling point of $J'_{i,j}$ it holds that $H_\tau(t_j) > t_j$, which means that $J'_{i,j}$ would miss its deadline.

As a consequence, the maximum relative deadline for jobs, calculated as described in Theorem 1, has to be chosen, otherwise the system will not be schedulable.

■

Corollary 3: Let $WCRT_i$ be the worst case response time of task T_i. Then,

$$WCRT_i = D_i^{min}$$

where $D_i^{min} = max_{j=1}^{k_i}(d'_{i,j} - r_{i,j})$.

Proof:

We know that $\exists k \quad / \quad t = kP_i + D_i^{min} H_\tau(t) = t$. Then, we can assure that in time t, the job $J_{i,k}$ will finish its execution at its deadline. The only case in which this does not occur is when two or more tasks have their deadline arrival at t, that is, when several scheduling events take place at the same time. Let us see that if $D_i^{min} = max_{j=1}^{k_i}(d'_{i,j} - r_{i,j})$, then no other scheduling points in t are present.

Let t_j be a scheduling point of T_j, and let us suppose that in t_j there is enough slack time for T_i, so $t_j - H_\tau(t_j) = C_i$. Let t_3 be a scheduling point such that $t_3 - H_\tau(t_3) \leq C_i$, $t_3 \leq t_j$ and $t_3 \leq t$. By Theorem 1, $t = H_\tau(t_3) + C_i$, since t_3 is the immediate backwards scheduling point of t.

If t_3 and t_j are two consecutive scheduling points then:

$$H_\tau(t_j) - H_\tau(t_3) \geq C_i$$

Therefore, $H_\tau(t_3) < H_\tau(t_j)$, since $C_i \neq 0$. Then

$$\begin{aligned} t_j &\geq H_\tau(t_j) + C_i \\ t_j &> H_\tau(t_3) + C_i \\ t_j &> t \end{aligned}$$

Hence, it is not possible that the resulting D_i^{min} coincides with another existing scheduling point. This means that, as in

D_i^{min} there is no slack time, the worst case response time of T_i is known and equal to the minimum deadline of T_i.

■

Under fixed priority scheduling (without offsets), the problem of finding the worst case response time of a task is equivalent to the problem of calculating the response time of that task of the first activation. This is not longer true in EDF. Instead, the worst case response time calculation requires the analysis of several different scenarios, which has pseudo-polynomial time complexity [31]. Specifically, the response times of the a activations within the first busy period ($\mathcal{R}$) must be calculated, applying different offsets O_i (from 0 to task period) to T_i. Specifically, the iterative expression that calculates the $WCRT_i$ in EDF is:

$$WCRT_i = \sum_{j=1}^{n} C_j \left(\left\lceil \frac{WCRT_i - O_j}{P_j} \right\rceil - \left[\left(\left\lceil \frac{WCRT_i}{P_j} \right\rceil - 1 \right) P_j + b_j > aP_i + b_i \right] \right)_0$$

being

$$a = 0, 1, .., \left\lceil \frac{\mathcal{R}}{P_i} \right\rceil - 1$$

and

$$b_k = D_k + O_k$$

where $(x)_0 = x$ if $x > 0$, $(x)_0 = 0$ if $x \leq 0$ and $[condition] = 1$ if the condition is true, otherwise $[condition] = 0$.

The range of offsets O_i is:

$$O_i = \left(\bigcup_{j=1}^{n} (kP_j + D_j - D_i : k \geq 0) \right) \bigcap [0, \mathcal{R})$$

As it can be observed, this computation is quite complex. Using our minimisation algorithm, $WCRT_i = D_i$. For this reason, we want to remark the importance of these results. Our proposal algorithm not only calculates the minimum deadline of a task, but it permits to obtain the WCRT under EDF in an efficient and simple way.

Based on the previous theorems, an algorithm to minimise the deadline of a periodic task T_i (Listing 1) proposed. We will refer to it as the *Deadlinemin* algorithm. The algorithm starts at time $t = jP_i + D_i$ and computes the slack time in the immediate previous scheduling point of t. If the slack is greater than C_i, the algorithm continues backwards to the next scheduling point; else $d'_{i,j}$ of $J_{i,j}$ is calculated. The procedure is repeated for all jobs of T_i whose deadline is in the interval $[0, \mathcal{R})$. Once the *Deadlinemin* algorithm is applied to a task T_i, its deadline reduction factor is:

$$\alpha_i = 1 - \frac{D_i^{min}}{D_i}$$

This algorithm has pseudo-polynomial time complexity in the size of the problem instance. However, as stated in [14], this bound is quite reasonable in practice whenever the system has a relative small utilization. Moreover, minor changes on some periods may reduce drastically the complexity to polynomial time [32].

Listing 1. *Deadlinemin* algorithm

```
1  Input: τ, T_i;
2  R := Calculate_ICI(τ);
3  deadline := C_i;
4  k := ⌈R / P_i⌉;
5  D_i^min := 0;
6  for s in 0.. (k−1) loop
7      t := min{(sP_i)+D_i, R};
8      deadline = C_i;
9      while t > sP_i + C_i loop
10         if ∃j i ≤ j ≤ n / t=(⌈t/P_j⌉ − 1)P_j + D_j then
11             if t − H_τ(t) < C_i then
12                 deadline := H_τ(t) + C_i − sP_i;
13                 break while;
14             end if;
15         end if;
16         t := t − 1;
17     end while;
18     D_i^min := max(D_i^min, deadline);
19 end for;
20 return D_i^min;
```

The *Deadlinemin* algorithm shows how to minimise the deadline of one task in the set. When the goal is to minimise the deadline of more than one task, two alternatives are proposed:

- When all the tasks have the same importance, deadlines of all tasks will be reduced with the same ratio. Thus, we must calculate the *critical scaling factor*, which is the central issue discussed in detail in section IV-C.
- When there are some tasks more important than others, these tasks are selected for reducing their deadlines.

In the second case, the solution is to apply the *Deadlinemin* algorithm to the most important task, update its deadline to the minimum value obtained and repeat the previous steps to the next important task. The order in which tasks are chosen determines the amount of deadline reduction of each one. Clearly, if a task is minimised in second place, its deadline reduction (α_i) may be not that large as if it were minimised in first place. The task selection criteria should follow some semantic approach. In [4], a criteria for determining the importance of a control task based on a the control effort is used. The control effort measures the effort required for the controller to fulfil the control specification. A task with higher control effort is more sensible to the output jitter and can produce a performance degradation of the control response. In this case, the criteria consists on the selection of the task with higher control effort first.

A. Output jitter analysis

As stated in [14], *output jitter* refers to the variation between the inter-completion times of successive jobs of the same task. This section presents the output jitter analysis of a deadline minimised task. This analysis has the advantage that can be calculated off-line, due to the property $WCRT_i = D_i^{min}$. The minimum and maximum separation between successive completions of T_i are:

$$p_i^{max} \leq P_i + D_i^{min} - C_i$$
$$p_i^{min} = P_i$$

Hence, the *absolute jitter* of T_i is:

$$\begin{aligned} AbsJitter(T_i) &= max(p_i^{max} - P_i, P_i - p_i^{min}) \\ &\leq max(D_i^{min} - C_i, 0) \\ &\leq D_i^{min} - C_i \end{aligned}$$

When the *Deadlinemin* algorithm achieves the maximum reduction factor ($\alpha_i = \alpha_i^{max}$) for T_i, we say that task T_i is *jitter-free*. The *relative jitter* of T_i can be obtained as a fraction of its period:

$$RelJitter(T_i) = \frac{AbsJitter(T_i)}{P_i} \leq \frac{D_i^{min} - C_i}{P_i}$$

IV. Extensions

The *Deadlinemin* algorithm illustrated in Listing 1 computes the minimum deadline of a periodic task $T_i \in \tau$, assuming that the original task set is feasible. In this section, we relax the assumptions made in Section III to cope with more practical situations. Specifically the following problems will be solved:

A. Minimum deadline of a non-feasible task.
B. Minimum deadline of a new incoming task.
C. Critical scaling factor calculation.
D. Minimum deadline of a single job of a periodic task.

A. Minimum deadline of a non-feasible task

The *Deadlinemin* algorithm of a task T_i presented in the previous section assumes that the task set is EDF-feasible. This means that the new deadline is less than or equal to the original deadline D_i. This section presents a modification of the deadline minimisation algorithm when the task set is not feasible. Thus, we are interesting in increasing the deadline of T_i until the task set τ is feasible. We will assume thatthe total utilization factor $U_\tau = \sum \frac{C_i}{P_i} \leq 1$ and the set τ is non feasible, but the set $\tau - T_i$ is feasible.

Assuming these conditions, one or more jobs of T_i will miss one or more deadlines in $[0, \mathcal{R})$. The *Deadlinemin* algorithm presented in the previous section searches backwards the available slack in each scheduling point, and when there is not enough slack, Theorem 1 is applied to calculate $d'_{i,j} = H_\tau(t_1) + C_i$. In this case, we know that in $r_{i,j} + D_i$ for every non-feasible job there is negative slack, since in these points there is a missed deadline. Then, the non-feasibility of these jobs makes even faster the *Deadlinemin* algorithm, since there is no need to search backwards the scheduling point where to calculate the value $d'_{i,j} = H_\tau(t_1) + C_i$. This point is $r_{i,j} + D_i$. Therefore, the *Deadlinemin* algorithm for a non-feasible task computes the $d'_{i,j}$ of the non-feasible jobs of T_i. Then, the minimum deadline D_i^{min} of T_i is:

$$D_i^{min} = max_j(d'_{i,j} - r_{i,j}) \quad \forall j \quad / \quad H_\tau(r_{i,j} + D_i) > r_{i,j} + D_i$$

The algorithm to calculate the minimum EDF-feasible deadline of a non-feasible task is presented in Listing 2. This algorithm assumes that we know that T_i is not feasible. If we do not know if the task is feasible or not, then lines 8, 9 and 10 of Listing 2 can be easily inserted on Listing 1 so the algorithm calculates the minimum deadline of T_i reducing it if the task set is feasible, or increasing it if not.

Listing 2. *Deadlinemin* algorithm of a non-feasible task

```
1   Input: τ, T_i;
2   R := Calculate_ICI(τ);
3   H_τ(t) := Σ_{j=1}^{n} C_j ⌊(t+P_j−D_j)/P_j⌋;
4   deadline := C_i;
5   k_i := ⌈R/P_i⌉;
6   D_i^min := 0;
7   for s in 0.. (k_i − 1) loop
8       if H_τ(sP_i + D_i) > sP_i + D_i then
9           deadline := H_τ(sP_i + D_i) + C_i − sP_i;
10      end if;
11      D_i^min := max(D_i^min, deadline);
12  end for;
13  D_i := D_i^min;
```

B. Arrival of a new periodic task

The *Deadlinemin* algorithm can also be used to calculate the minimum deadline of a new periodic task T_{n+1} that enters the system at time t, while the system is running. The analysis is made considering that T_{n+1} were released at time 0, just like the rest of the tasks. As D_{n+1}^{min} is calculated for the first busy period of a synchronous system (all tasks release at time 0), by [30] we know that the task set will be feasible in any other busy period.

However, it is possible that the calculated D_{n+1}^{min} is not the minimum deadline. This can occur when there is no busy period 'similar' to the first busy period (busy periods with a great number of activations) in the whole execution history. However, the algorithm finds the minimum deadline when all the periods of the tasks are co-prime. In this case, we can find a time point with a common release for all tasks [30].

C. Critical scaling factor for periodic task sets

If we want to minimise all task deadlines, but we do not have any preference regarding which task deadline requires the greatest level of reduction, then we have to calculate the critical scaling factor Δ^* for the whole system. From now on we will refer to the CSF algorithm. This way the task set $\tau = \{T_1, ..., T_n\}$ with $T_i = (C_i, D_i, P_i)$ becomes $\tau = \{T'_1, ..., T'_n\}$ with $T'_i = (C_i, \Delta^* * D_i, P_i)$.

The method consists of calculating the point with the minimum slack. This slack is the maximum amount that a task deadline can be reduced. However, the same amount for all task deadlines will lead to different ratios. Hence, as a first approach Δ^* is chosen as the minimum ratio (maximum Δ^*) when new task deadlines are $D'_i = D_i - slack$. These steps are repeated until there is no slack available. Listing 3 shows the CSF algorithm.

Listing 3. CSF algorithm

```
1  Input: τ;
2  R := Calculate_ICI(τ);
3  Δ* := 1;
4  while min_{t=0}^{t=R}(t − H_τ(t)) ≠ 0 loop
5      Δ* := max_{i=1}^{i=n}(1 − min_{t=0}^{t=R}(t−H_τ(t)) / D_i);
6      Let D_i := Δ* D_i ∀i = 1..n;
7  end loop;
8  return Δ*;
```

It is important to note that now the minimisation is not optimal, in the sense that some tasks may reduce its deadline even more, but they are limited by other tasks. This way, the worst case response time of the tasks may not be equal to $\Delta^* D_i$. Hence, the output jitter analysis presented in the previous section is not applicable here.

D. On-line minimum deadline of a single job

As it has been explained in section III, the minimum deadline D_i^{min} of a task is the maximum over all the d'_{ij} calculated as stated in Theorem 1, but d'_{ij} may not be the tightest deadline of the $j-th$ job (except for the first job).

The following example illustrates the problem. Figure 2 shows the execution chronogram of the task set τ={$T_1(2,6,6)$, $T_2(2,2,7)$} and Figure 3 the corresponding H(t) function. For T_1, according to the deadline minimisation method $d'_{1,1} = 4$ and $d'_{1,2} = 8$, which implies relative deadlines of 4 and 2 for $J_{1,1}$ and $J_{1,2}$, respectively. If we pay attention to the second job of T_1, that is, $J_{1,2}$, we see in the chronogram how a relative deadline of 2 units ($d'_{1,2} - r_{1,2}$) makes T_2 to miss its deadline in the second activation. While the calculation of $d'_{i,j}$ seems to indicate that this value is the minimum deadline of $J_{i,j}$, we see that it can make the system unfeasible.

Although d'_{ij} is not the minimum deadline of the $j-th$ job, intuitively we know that some jobs can have a lower deadline that the minimum calculated D_i^{min}. For example, in Figure 2 we see that $J_{1,3}$ can have a relative deadline of 2 units while the minimum deadline D_1^{min} is 4. In this section the minimum deadline of a job $J_{i,j}$ of task T_i will be calculated.

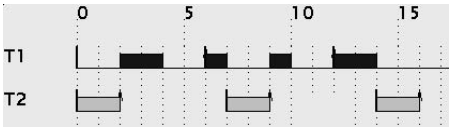


Fig. 2. Execution chronogram of $\tau = \{T_1(2,6,6), T_2(2,2,7)\}$

$H(t)$ function can not be used to calculate the minimum deadline of any job $J_{i,j}$ because it keeps track of the amount of execution time required from tasks whose deadline is smaller than or equal to time t in the interval $[0,t]$. We only need the execution time requirement for jobs in the interval $[r_{i,j}, r_{i,j} + d_{i,j})$. Our exposition requires the use of the following function:

Definition 8: [15] The maximum cumulative execution time requested by jobs of τ whose absolute deadlines are

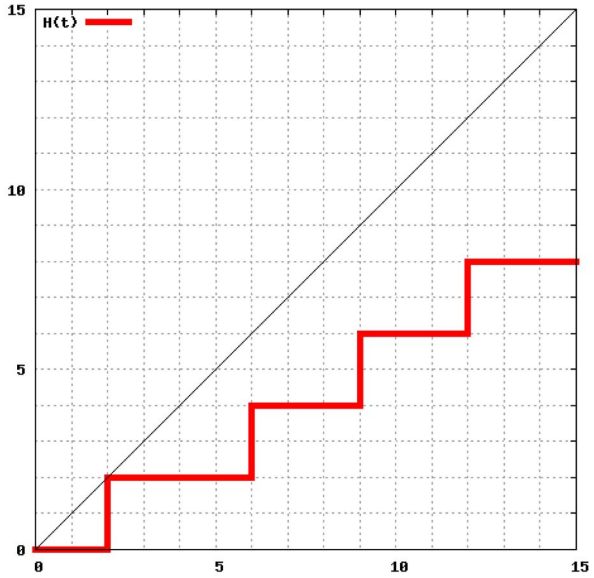


Fig. 3. H_τ of $\tau = \{T_1(2,6,6), T_2(2,2,7)\}$

smaller than or equal to time t in the interval $[t_0, t_0 + t)$ is:

$$H_\tau^{t_0}(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{\left(t + P_i - D_i - \lceil \frac{t_0}{P_i} \rceil P_i + t_0\right)_0}{P_i} \right\rfloor + \\ + \sum_{i=1}^{n} c_{i,t_0} \left\lfloor \frac{\left(2t - \lfloor \frac{t_0 + P_i - D_i}{P_i} \rfloor P_i - D_i\right)_0}{P_i} \right\rfloor$$

where c_{i,t_0} is the remaining computation time of the active invocation of task T_i and $(x)_0 = max(0,x)$.

This function was introduced in [15] to calculate on-line the minimum deadline of an aperiodic task. $H_\tau^{t_0}(t)$ represents the minimum amount of time that must be completed at time t in order to successfully schedule active tasks between t_0 and t. This is a generalization of the $H(t)$ function defined in [29] for $t_0 \neq 0$. Any interval $[t_0, t)$ is feasible if $H_\tau^{t_0}(t) \leq t - t_0$.

Then, Theorem 1 can be rewritten in the following way:

Corollary 3: Let τ= $\{T_1, ...T_i, ...T_n\}$ be a feasible set of n periodic tasks. Let $J_{i,j}$. If there exists a time t_2 such that $r_{i,j} + C_i \leq t_2 \leq d_{i,j}$ and $H_\tau^{r_{i,j}}(t_2) = H_\tau^{r_{i,j}}(d_{i,j} - \varepsilon) > t_2 - r_{i,j} - C_i$ then the sequence of jobs $J_{1,1}, .., J'_{i,j}, .., J_{n,k_n}$ with $J'_{ij} = \{r_{i,j}, C_i, d'_{i,j},\}$ where $d'_{i,j} = r_{i,j} + H_\tau^{r_{i,j}}(t_2) + C_i$ is schedulable.

Proof:

Following the same reasoning that in Theorem 1, we only have to prove that the sequence of jobs is schedulable in $t_3 = d'_{i,j} = H_\tau^{r_{i,j}}(t_2) + C_i + r_{i,j}$.

The new $H_\tau^{r_{i,j}}$ function in t_3 will be:

$$H_{\tau'}^{r_{i,j}}(t_3) = H_\tau^{r_{i,j}}(t_2) + C_i$$

Since $t_3 = d'_{i,j} = H_\tau^{r_{i,j}}(t_2) + C_i + r_{i,j}$, it follows that

$$H_{\tau'}^{r_{i,j}}(t_3) = t_3 + C_i - C_i - r_{i,j} \\ = t_3 - r_{i,j}$$

■

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Corollary 4: $d'_{i,j} = H_\tau^{r_{i,j}}(t_2) + C_i + r_{i,j}$ is the minimum deadline for a job $J_{i,j}$.

Proof: If $d'_{i,j}$ is the minimum deadline, then $d'_{i,j} - \varepsilon$ will render the set of jobs unfeasible. We need to look at schedulability in $t_{min} = d'_{i,j} - \varepsilon$.

$$H_{\tau'}^{r_{i,j}}(t_{min}) = H_\tau^{r_{i,j}}(t_2) + C_i$$

Since $t_{min} = d'_{i,j} - \varepsilon = H_\tau^{r_{i,j}}(t_2) + C_i + r_{i,j} - \varepsilon$ and substituting $H_\tau^{r_{i,j}}(t_2)$ in the previous equation:

$$\begin{aligned} H_{\tau'}^{r_{i,j}}(t_{min}) &= t_{min} + C_i - C_i - r_{i,j} + \varepsilon \\ &= t_{min} - r_{i,j} + \varepsilon \end{aligned}$$

and then, the sequence of jobs are not schedulable in t_{min}. ■

Then, using theorem 3 to calculate $d'_{i,j}$, it is true that $d'_{i,j}$ is the minimum deadline of job $J_{i,j}$.

Let us suppose that we want to apply the minimum possible deadline to job $J_{i,j}$ of task T_i. Listing 4 shows the algorithm that calculates it.

Listing 4. On-line job deadline minimisation algorithm

```
1  t := r_{i,j} + D_i;
2  d'_{i,j} = C_i;
3  while t > r_{i,j} + C_i loop
4    if t=(⌈t/P_z⌉ − 1)P_z + D_z then --sched point
5      slack := t − H_τ^{r_{i,j}}(t) − r_{i,j}
6      if slack < C_i then
7        d'_{i,j} := r_{i,j} + H_τ^{r_{i,j}}(t_2) + C_i;
8        exit ;
9      end if ;
10   end if ;
11   t := t−1;
12 end while;
```

This algorithm is calculated on-line, whenever we want to reduce the deadline of a specific job. If the reduction is to be applied to a window of jobs $(J_{i,j}, ..., J_{i,j+s})$, the previous algorithm is executed whenever a new release arrives or, if there is enough processor time in $t = r_{i,j}$, the s minimum deadlines can be calculated at once.

V. Example

Now, we are going to illustrate the *Deadlinemin* algorithm presented in section III by means of an example. Consider a system of three tasks: $\tau = T_1 = (1,7,7)$, $T_2 = (3,10,10)$, $T_3 = (5,20,20)$. The minimisation sequence will be T_2, T_1, T_3, since we are interested on reducing T_2 deadline more that the rest of the tasks. As explained in Section I this is be motivated by the application itself, such as in control applications to reduce latency and jitter. The ICI of this system is $[0,10)$.

In order to apply the *Deadlinemin* algorithm to T_2, we have to calculate the number of jobs of T_2 within $[0,10)$, that is, the k_2 parameter. For T_2, $k_2 = 1$. Then, from $t = D_2 = 10$, the algorithm searches backwards the next scheduling point. This point is $t = 7$. The slack at time $t = 7$ is 6. Since $C_2 = 3 \leq 6$, we should continue searching backwards the next scheduling point, but the search ends because no more scheduling points before time 7 exists. Then, we can assign $D_2^{min} = C_2 = 3$.

Now, the new task system becomes $\tau' = \{T_1 = (1,7,7), T'_2 = (3,3,10), T_3 = (5,20,20)\}$, and the next task to minimise deadline is T_1. For T_1, $k_1 = 2$. Let us apply the deadline minimisation algorithm to each job:

- J_{11}: Algorithm starts at $t = 7$. The only scheduling point in $[0,7)$ is in $t = 3$. Slack time in $t = 3$ is 0. Then $d'_{11} = H\tau'(3) + C_1 = 3 + 1 = 4$.
- J_{21}: Algorithm starts at $t = 14$. We have to search scheduling points from $t = 14$ to $t = P_1 + C_1 = 8$. As the only scheduling point in $[8,14)$ is in $t = 13$ and in this time there is sufficient slack time, then $d'_{21} = 1$.

Finally, $D_1^{min} = max(d'_{11}, d'_{21}) = 4$. With the minimisation of T_1 deadline, the new task set is $\tau'' = T'_1 = (1,4,7)$, $T'_2 = (3,3,10)$, $T_3 = (5,20,20)$. For T_3, $k_3 = 1$, subsequently we have to find the scheduling points in $[5,20)$. Table I shows the scheduling points in this interval and the slack time associated. According to Theorem 1, $D_3^{min} = d'_{3,1} = H''_\tau(4) + C_3 = 9$.

TABLE I
Slack time in $[5,20)$

t	18	13	11	4
Slack = $t - H''_\tau(t)$	9	5	6	0

Therefore, the minimised deadline task set for order (T_2, T_1, T_3) is $\tau''' = T'_1 = (1,4,7)$, $T'_2 = (3,3,10)$, $T'_3 = (5,9,20)$. Table II shows the initial and final deadlines and the deadline reduction factors for the three tasks. This table also shows the calculation of Δ^*, and the deadlines associated with this parameter (rounded to the nearest upper integer).

TABLE II
Results of *Deadlinemin* algorithm for order T_2, T_1, T_3

	D_i	D_i^{min}	α_i	α_i^{max}	Δ^*	$\Delta^* D_i$
T1	7	4	0.43	0.85	0.5	4
T2	10	3	0.70	0.70	0.5	5
T3	20	9	0.55	0.75	0.5	10

VI. Experimental evaluation

In this section a series of simulations and comparisons with other works are presented to evaluate our deadline minimisation proposal. A number of tests have been run, specifically, 1000 task sets have been generated for each utilisation. Each task was generated by randomly choosing the task period as an integer between 5 and 100, the task utilisation between 0 and 1, and then selecting the task computation in such a way that the total system utilisation is approximately equal to the desired load. The deadline is set to be equal to the period, except for subsection VI-B2 where deadlines were randomly decreased by 40% to 90% of the task period. The generated task is added to the task set only if the set is schedulable. The simulations have been run for 3,5 and 10 tasks in each set.

The experiments are focused upon evaluating the level of the deadline reduction (α_i) with *Deadlinemin* algorithm and CSF algorithm. As the deadline minimisation algorithm can be used to minimise jitter a second group of experiments is focused

on the comparison with other similar works. Moreover, to quantify the benefits of our proposal, we have simulated a control system application. The system performance of our model will also be evaluated.

A. Reducing the deadline

Regarding the minimisation of one task deadline, experiments have been run with 10 tasks in each set. The *Deadlinemin* algorithm has been executed to minimise the deadline of one task in the set. As can be seen in Figures 4 and 5, minimisation rises to 90% for 10 tasks and the maximum deadline reduction is accomplished for all utilizations (reduced deadlines are equal or very close to task WCET). With 5 tasks, a smaller reduction is achieved, and it moves away from the maximum as the system utilisation increases. The main reason of this lower reduction is the experiment design. As the experiment generator tries to obtain a set of tasks for a specific utilisation, in general, the computation time of the tasks is greater than if the number of tasks is lower for the same periods. As a consequence, in a set of 5 tasks, a task is preempted for a shorter time period.

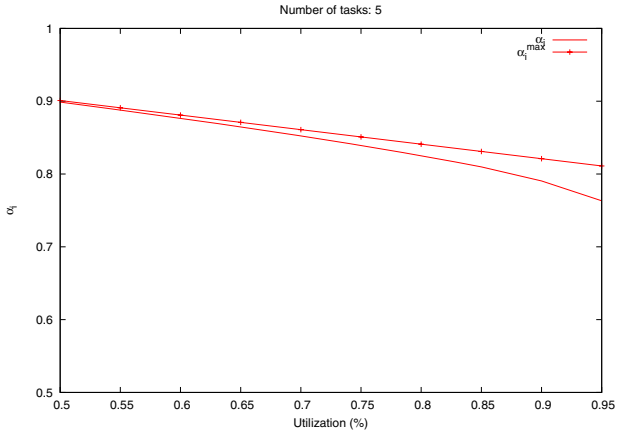


Fig. 4. Deadline reduction factor (5 tasks)

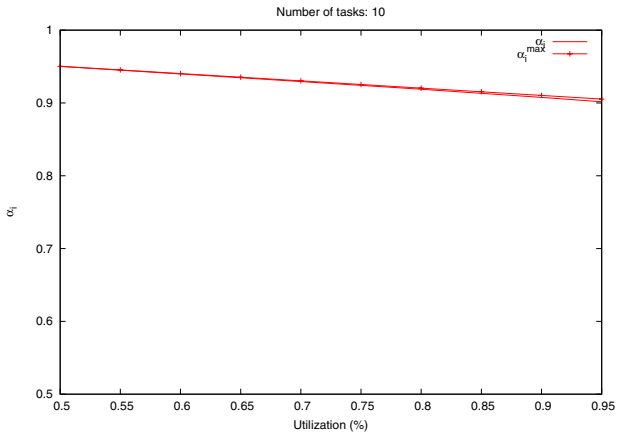


Fig. 5. Deadline reduction factor (10 tasks)

Figure 6 depicts the simulations obtained when CSF algorithm is used to calculate Δ^* for tasks sets with 3, 5 and 10 tasks. As system load and number of tasks increase, the deadline reduction decreases (Δ^* approaches to 1).

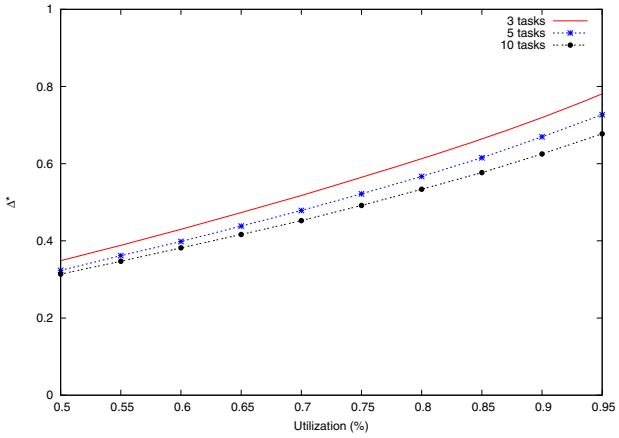


Fig. 6. CSF algorithm

B. Comparison with other works

This section describes the comparison with relevant works that deal with deadline and jitter reduction. Specifically, we compare the *Deadlinemin* algorithm with the `minD` algorithm proposed by Hoang *et* al. ([17]) and the CSF algorithm with the bisection method proposed by Jejurikar and Gupta [19]. We also compare the jitter reduction and the system performance obtained by *Deadlinemin* algorithm with other proposals to reduce jitter in control applications.

1) `minD` *and* Deadlinemin*:* Next simulations present the comparison with the work developed by Hoang *et* al. ([17]), which developed the `minD` algorithm to calculate the minimum deadline of a periodic task. The approach is based on assigning the minimum deadline of a task as its computation time ($D_i = C_i$) and then check feasibility in each scheduling point in the window $[0, \min\{L_a, \mathcal{R}\}]$, where:

$$L_a = \frac{U_T}{1-U_t} \max_{i=1}^{n}(P_i - D_i)$$

being U_T the total utilisation factor of the task set.

As not all the tasks can have such a low deadline, the algorithm calculates a new deadline whenever the system is not feasible and continues checking that the condition $H(t) \leq t$ holds in the remaining scheduling points.

Two considerations about this approach and our proposal:

- The bound L_a depends on task deadlines. Therefore, if a new deadline is assigned to task T_i, the number of scheduling points increases. Moreover, the deadline assigned in the first iteration ($D_i = C_i$) is a such lower value that the number of scheduling points can be considerable. As *Deadlinemin* algorithm only uses the first busy period, we do not need to re-calculate the scheduling points.
- The `minD` algorithm requires to check that the condition $H(t) \leq t$ holds for all the scheduling points in $[0, \min\{L_a, \mathcal{R}\}]$, even if the first iteration makes the system feasible. The reason is that we can not assure that the system will be feasible in each iteration unless the

feasibility test is completed for each scheduling point. Our proposal stops checking feasibility (or available slack) once the minimum deadline has been found. The reason is that going backwards and not forwards (like `minD`), we are sure that the deadline obtained does not jeopardise feasibility in the remaining scheduling points.

Listing 5. minD algorithm

```
1  Input: τ, T_i;
2  R := Calculate_ICI(τ);
3  L_a := U_T/(1−U_t) max_{i=1}^{n}(P_i − D_i)
4  t_max := min{L_a, R};
5  D_i^min := C_i;
6  t := 0;
7  while (t ≤ t_max) loop
8      if ∃j i ≤ j ≤ n / t=(⌈t/P_j⌉ − 1)P_j + D_j then
9          H_τ(t) := Σ_{j=1}^{n} C_j ⌊(t+P_j−D_j)/P_j⌋;
10         if H_τ(t) > t then
11             K = ⌈(H_τ(t)−t)/C_i⌉
12             r = ⌊t/P_i⌋P_i
13             D_i^min = H_τ(t) + (K−1)(P_i − C_i) − r
14             L_a := U_T/(1−U_t) max_{i=1}^{n}(P_i − D_i)
15             t_max := min{L_a, R};
16         end if;
17     end if;
18     t := t+1;
19 end while;
20 return D_i^min;
```

To understand and compare `minD` with *Deadlinemin* Table 5 presents the pseudo-code of the `minD` algorithm.

Figure 7 represents the number of steps needed by `minD` and *Deadlinemin* to calculate the minimum deadline of one task. Here, the number of steps is equivalent to the number of scheduling points in which the algorithm has to check feasibility in the case of the `minD` algorithm, or compute the available slack in the case of *Deadlinemin* algorithm. In [17], the number of steps are the iterations until a feasible deadline is found. We can not compare *Deadlinemin* and `minD` using the number of iterations, because *Deadlinemin* does not iterate with the deadline value, but with the number of scheduling points.

The number of processor cycles needed by the algorithms has been measured on an AMD Athlon 64 2 GHz (Dual-Core). In order to avoid some external interferences interrupts were disabled during the algorithm execution. The runtime distribution for 100000 executions of *Deadlinemin* and `minD` algorithms is depicted in Figure 8(b), whereas maximum and average values for the same executions are represented in Figure 8(a). As shown in the figures `minD` has a great dispersion and greater execution times than *Deadlinemin*.

One way to measure the algorithm independently of the execution platform is to count the number of instructions required by the algorithm. In order to measure it, the test program has been instrumented using the *ptrace* system call. This system call allows a parent process to control the execution of the child (test) process. The single step mode permits to the parent process be notified each time a child instruction is executed.

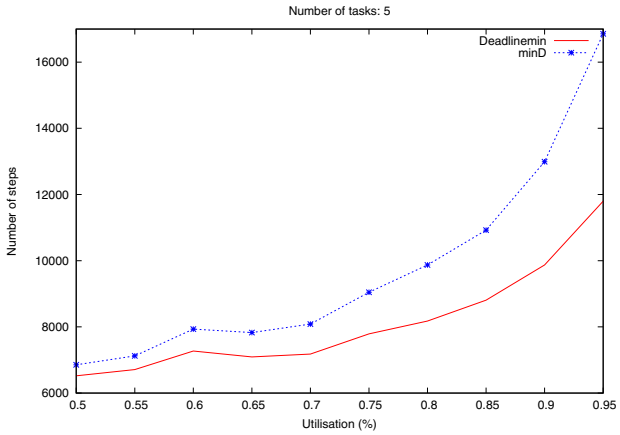
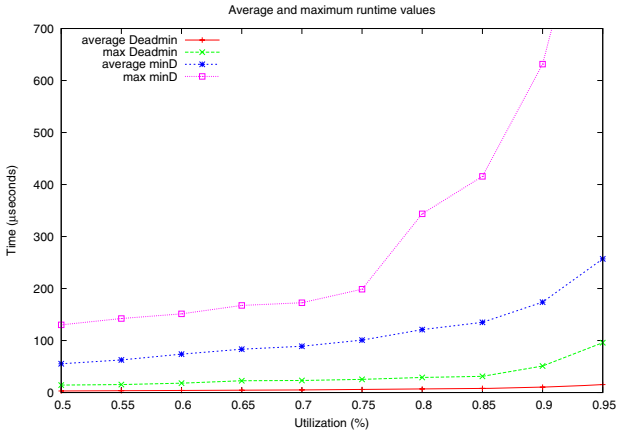
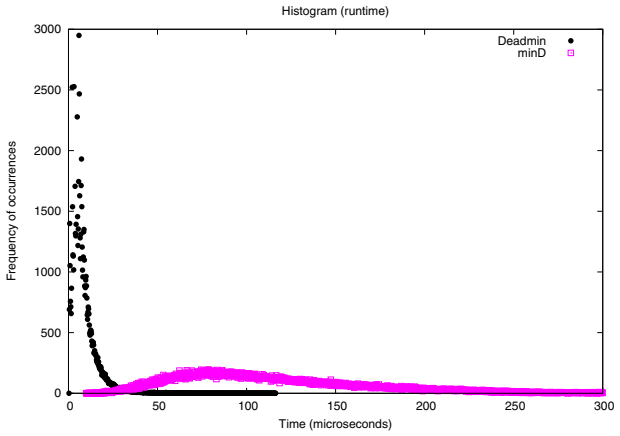


Fig. 7. `minD` and *Deadlinemin* number of steps



(a) Average and maximum values



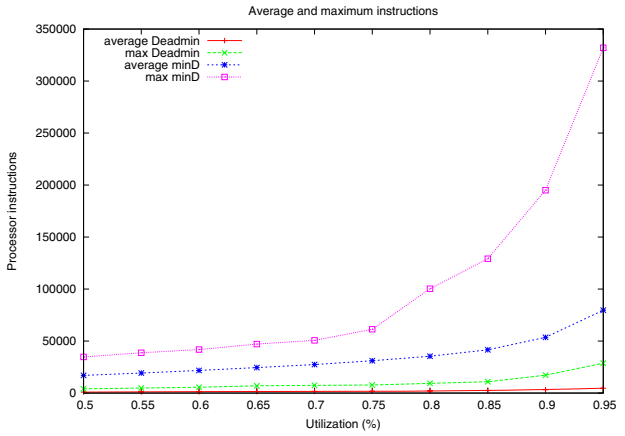
(b) Histogram

Fig. 8. `minD` and *Deadlinemin* runtime comparison

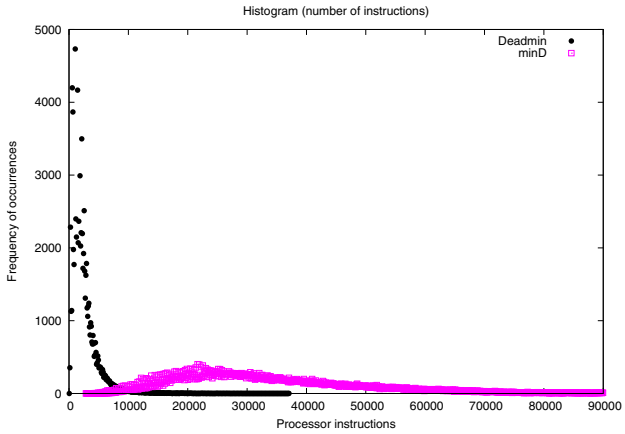
In this case both algorithms have been compiled with the same compiler version (gcc 4.1) and without any optimisation options. Results can slightly change depending on the compiler version and its optimisation level options used. Figures 9(b) and 9(a) present the comparison of both algorithms as far as the number of processor instructions is concerned. Again, the improvement of *Deadlinemin* over `minD` is meaningful.

Although both algorithms are optimal, in the sense that they

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



(a) Average and maximum values



(b) Histogram

Fig. 9. `minD` and *Deadlinemin* processor instructions comparison

obtain the minimum deadline, and run in pseudo-polynomial time complexity, the previous experiments demonstrate that *Deadlinemin* is much faster than `minD`. The reason is that `minD` even finding short deadlines immediately, it has to check all the scheduling points in the set and recalculate L_a, while *Deadlinemin* stops iterating when $H_\tau(t) > t - C_i$, avoiding some scheduling points. As the total utilisation increases, the probability of C_i being a valid deadline for T_i decreases, which implies to make a lot of calculations in each scheduling point in `minD` algorithm.

If we take the deadline minimization of T_2 in the example of section V, we see how with *Deadlinemin* we only have to check one scheduling point in time $t = 7$, while the `minD` method starts assigning a initial deadline of $D_i^{min} = 3$ and it has to check feasibility in the remaining five scheduling points, even being a feasible deadline for T_2.

2) CSF and Jejurikar's bisection method: We also have compare the CSF algorithm with the work developed by Jejurikar and Gupta in [19]. In this work, the proposed bisection algorithm makes a binary search of the minimum η, which expresses the ratio between the current and the maximum frequency of the processor. If all periods and deadlines are multiplied by this factor η (equivalent to $\Delta*$ obtained by the CSF algorithm) the task set is also schedulable.

The difference between the bisection method in [19] and our work is that we do not change the periods proportionally to deadlines, but they are fixed. Consequently, it is expected that CSF achieves lower scaling factors for the same task set. This not implies that our method is better than the bisection method, it is only other contribution, since in [19] due to the schedulability condition used, periods must change accordingly to deadlines. The comparison is depicted in Figure 10.

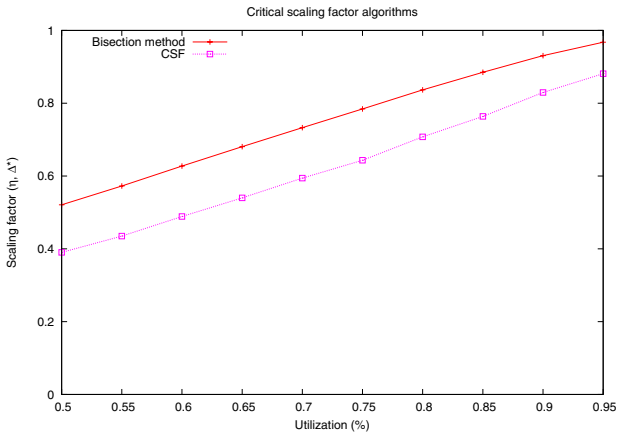


Fig. 10. CSF and bisection method comparison

3) Control jitter reduction: The deadline minimisation method can be used to reduce the jitter of control tasks in real-time control applications. *Control jitter* is defined as the variation in the response time of the task that sends the control action to the target system. Note the difference with *output jitter*, which is defined as the variation between successive periods, rather than over the lifetime of the system. We have compared our method with the task models presented in [26] and [25]. Figure 11 shows the average control jitter (relative jitter of the task that sends the control action to the target system) of the following task models:

- Standard model: This is the original task set consisting of 3 control periodic tasks, where task deadlines are equal to periods.
- COUS model: This model was proposed in [26]. Each control task is split into two subtasks: *Calculate Output* subtask (CO) and *Update State* subtask (US). New deadlines are calculated for each subtask. Control jitter is associated to CO subtask. This model produces a task set with 6 tasks.
- IMF model: This is the model presented in [25] and it splits the system in three subtasks (Initial, Mandatory and Final). As in the COUS model, new deadlines are obtained for initial and final subtasks, and a fixed offset is assigned to final tasks. Here, control jitter is associated to Final task. The original task set of 3 tasks, becomes 9 tasks.
- *Deadlinemin*: This is the standard model to which the deadline minimisation method has been applied where tasks have been sorted by period.

As figure 11 illustrates, the worst jitter is, as it is expected, for the standard model. The best results are obtained with the

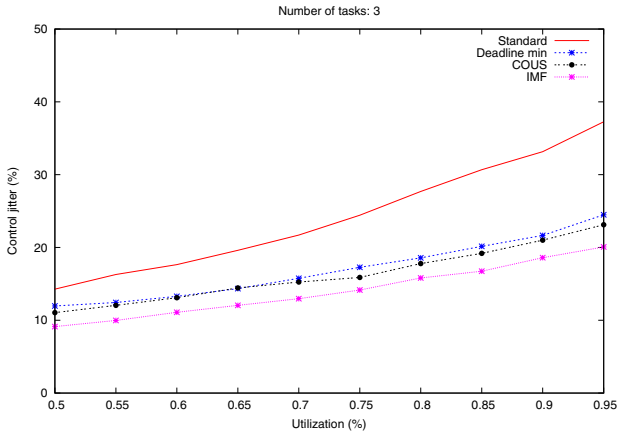


Fig. 11. Control jitter comparison

IMF model, whereas the COUS and the deadline minimisation model are very similar in terms of control jitter. This is also expected, because the more partitioning is made, the lesser computation time for final tasks, and hence, the lower jitter. However, some important remarks:

- IMF and COUS models make a task partitioning which implies more context switches than in a non partitioning task model. As Deadline minimisation model and COUS have similar jitter, it is better to choose our proposal that has less tasks.
- IMF applies a fixed delay to the final task. This means that the control action is delayed to achieve a very small jitter. This may cause a bad performance depending on the controlled system. For this reason, it is sometimes better to have a slightly bigger jitter with no fixed delay. In these situations, we would choose the deadline minimisation model.

As explained in the introduction, our deadline minimization method can be used jointly with CO_US and IMF models to calculate the minimum deadlines of CO and F subtasks respectively.

4) Improvement in the system performance: The algorithms developed in this work can be used in real-time control applications to improve the system performance. In this section we will compare the system output of a real-time process when the traditional task model (STM model in section VI-B3) and the *Deadlinemin* model is executed, under EDF scheduling. The chosen process is the three inverted pendulum system described and implemented for the STM model in [33]. The control of each pendulum is associated to a control task (task T_1 refers to pendulum 1, etc ...). The different pendulum lengths motivate different sampling periods, in such a way that $P_1 < P_2 < P3$. The two task models to be compared are presented in Table III.

The STM model considers $D_i = P_i$, while the *Deadlinemin* model is originally the STM model to which the deadline minimisation method explained in section III has been applied. The minimisation order is T_3, T_1, T_2. We have chosen an order inversely proportional to task periods (although not exactly, since order T_3, T_2, T_1 is not feasible). The reason is that, in

TABLE III
TASK MODELS

	STM			Deadline min		
	C	D	P	C	D	P
T_1	7	20	20	7	14	20
T_2	7	29	29	7	21	29
T_3	7	35	35	7	7	35

the STM model, the task with the greatest jitter is the task with the greatest period (although this is much more accused in rate monotonic than in EDF) and minimising D_3 more than the rest of the deadlines, we expect a greater improvement. Anyway, as it depends on the process, control designers must select the best order in accordance to their needs. Figures 12,13 and 14 show the system's response to a step input of the three pendulums when they are scheduled under EDF using the STM and the *Deadlinemin* model. As it is expected, pendulum 3 presents the best performance, since it presents less oscillation. Pendulums 1 and 2 have a similar performance in both models (*Deadlinemin* is slightly better than STM) since jitter reduction of pendulum 3 is at the expense of, above all, pendulum 1.

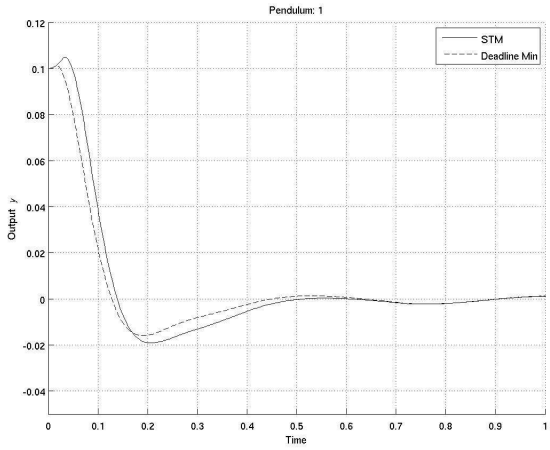


Fig. 12. Discrete-time response of inverted pendulum 1

VII. CONCLUSIONS AND FUTURE WORK

Task scheduling is a fundamental issue in real-time control algorithm implementation to obtain the desired performance of the control system. Task parameters, such as computation times, periods and deadlines, are susceptible to adjustment during the design phase of the control system. Modifications in the periods may require the redesign of the controller in order to adjust it to the new rate. As a consequence, sensitivity analysis has mainly focused upon changes in task computation times. Although there are some works that deal with deadline modification, we have not found any sensitivity analysis of task deadlines, in other words, to find the minimum deadline of a task without jeopardising system feasibility, either for fixed or dynamic priority scheduling. This paper proposes a sensitivity analysis for periodic task deadlines, using EDF scheduling. The deadline minimisation algorithm reduces a

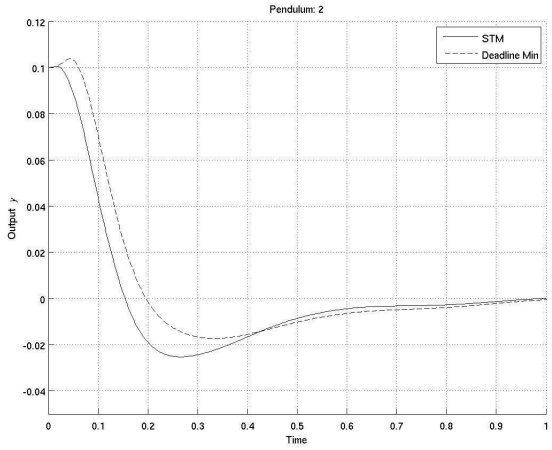


Fig. 13. Discrete-time response of inverted pendulum 2

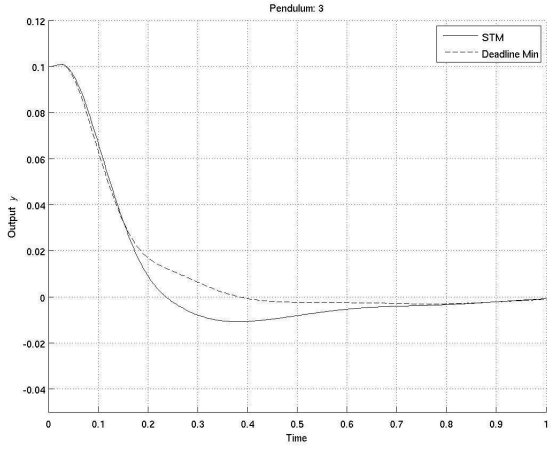


Fig. 14. Discrete-time response of inverted pendulum 3

task deadline to the limit, while the critical scaling factor calculation establishes the same reduction ratio for all task deadlines within the system. The first method should be applied when it is important to reduce one task deadline in particular, this being more important than the others. While the second method is more useful when all tasks in the system have the same significance. Moreover, the minimum deadline for a single job has been also presented. The sensitivity analysis presented in this paper has several advantages. As the simulations have shown, output jitter is strongly reduced. Moreover, in the case of deadline minimisation, the deadlines obtained are equal to the worst case response times of a task. Note that the calculation of WCRT in EDF is quite complex. Using our algorithm, there is no need to calculate the WCRT of a task. The deadline minimisation algorithm can be use to calculate the minimum deadline of control task models based on task splitting. Our method improves the existing one based on iteratively calculating the WCRT in EDF.

As future lines of research, we can combine the deadline minimisation method with the control effort concept. The control effort of a task expresses the sensitivity of the task to time delays, i.e, to output jitter. In order to improve system performance, tasks with higher control effort are prone to greater reductions in their deadlines than the rest of the tasks.

REFERENCES

[1] C. Liu and J.W.Layland., "Scheduling algorithms for multiprogramming in a hard real-time environment," *JACM*, vol. 23, pp. 46–68, 1973.

[2] J. Leung and J. Whitehead, "On the complexity of fixed-priority schedulings of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[3] A. Crespo, P. Albertos, K. Arzen, A. Cervin, M. Torgren, and Z. Hanzalek, "Artist2 roadmap on real-time techniques in control system implementation," Control for Embedded Systems Cluster. EU/IST IST-004527, Tech. Rep., 2005.

[4] P. Albertos, A. Crespo, I. Ripoll, M. Vallés, and P. Balbastre, "Rt control scheduling to reduce control performance degrading," in *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000.

[5] Y. Mansour and B. Patt-Shamir, "Jitter control in qos networks," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 492–502, 2001.

[6] D. C. Verma, H. Zhang, and D. Ferrari, "Delay jitter control for real-time communication in a packet switching network," in *Proceedings of the IEEE Conference on Communications Software*, 1991, pp. 35–43.

[7] M. Xiong and K. Ramamritham, "Deriving deadlines and periods for real-time update transactions," in *IEEE Real-Time Systems Symposium*, 1999, pp. 32–43.

[8] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour," in *IEEE Real-Time Systems Symposium*, 1989, pp. 166–171.

[9] S. Vestal, "Fixed-priority sensitivity analysis for linear compute time models," *IEEE Transactions on Software Engineering*, vol. 20, no. 4, pp. 308–317, April 1994.

[10] S. Punnekkat, R. Davis, and A. Burns, "Sensitivity analysis of real-time task sets," in *Proceedings of the Conference of Advances in Computing Science*, 1997, pp. 72–82.

[11] P. Balbastre, I. Ripoll, and A. Crespo, "Schedulability analysis of window-constrained execution time tasks for real-time control," in *14th Euromicro Conference on Real-Time Systems.*, 2002.

[12] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *IEEE Real-Time Systems Symposium*, December 1998, pp. 286–295.

[13] A. Cervin, B. Lincoln, J. Eker, K. Arzen, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Proceedings of RTCSA*, 2004.

[14] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling periodic task systems to minimize output jitter," in *Sixth Conference on Real-Time Computing Systems and Applications*, 1999, pp. 62–69.

[15] I. Ripoll, A. Crespo, and A. García-Fornes, "An optimal algorithm for scheduling soft aperiodic tasks in dynamic-priority preemptive systems," *IEEE Transactions on Software Engineering*, vol. 23, no. 6, pp. 388–400, 1995.

[16] G. C. Buttazzo and F. Sensini, "Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments," *IEEE Trans. Comput.*, vol. 48, no. 10, pp. 1035–1052, 1999.

[17] H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson, "Computing the minimum edf feasible deadline in periodic systems," in *Proceedings of RTCSA*, August, 2006.

[18] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems." in *ACM Symposium on Operating Systems Principles*, 2001, pp. 89–102.

[19] R. Jejurikar and R. Gupta, "Optimized slowdown in real-time task systems," in *16th Euromicro Conference on Real-Time Systems*, 2004.

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

[21] M. DiNatale and J. A. Stankovic, "Scheduling distributed real-time tasks with minimum jitter." *IEEE Trans. Computers*, vol. 49, no. 4, pp. 303–316, 2000.

[22] C. Locke, "Software architecture for hard real-time applications: cyclic executives vs. priority executives," *Journal of Real-Time Systems*, vol. 4, no. 1, pp. 37–53, 1992.

[23] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[24] T. Kim, H. Shin, and N. Chang, "Deadline assignment to reduce output jitter of real-time tasks," in *16th IFAC Workshop on Distributed Computer Control Systems*, 2000.
[25] A. Crespo, I. Ripoll, and P. Albertos, "Reducing delays in rt control: the control action interval," *IFAC World Congress Beijing*, 1999.
[26] A. Cervin, "Improved scheduling of control tasks," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, 1999.
[27] P. Balbastre, I. Ripoll, J. Vidal, and A. Crespo, "A task model to reduce control delays," *Journal of Real-Time Systems*, vol. 27, no. 3, pp. 215–236, 2004.
[28] P. Balbastre, I. Ripoll, and A. Crespo, "Optimal deadline assignment for periodic real-time tasks in dynamic priority systems," in *18th Euromicro Conference on Real-Time Systems.* IEEE Computer Society Press, July, 2006.
[29] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard real-time sporadic tasks on one processor," in *IEEE Real-Time Systems Symposium*, 1990, pp. 182–190.
[30] I. Ripoll, A. Crespo, and A. Mok, "Improvement in feasibility testing for real-time tasks," *Journal of Real-Time Systems*, vol. 11, pp. 19–40, 1996.
[31] M. Spuri, "Analysis of deadline scheduled real-time systems," *Rapport de Recherche RR-2772, INRIA, Le Chesnay Cede, France*, 1996.
[32] M. Garey and D. Johnson, *Computer and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
[33] A. Cervin, "Integrated control and real-time scheduling," Ph.D. dissertation, Department of Automatic Control. Lund Institute of Technology, May 2000.

Patricia Balbastre is an assistant professor of Computer Engineering. She graduated in Electronic Engineering at the Technical University of Valencia, Spain, in 1998. And the Ph.D. degree in Computer Science at the same university in 2002. Her main research interests include real-time operating systems, dynamic scheduling algorithms and real-time control.

Ismael Ripoll received the B.S. degree from the Polytechnic University of Valencia, Spain, in 1992; the Ph.D. degree in Computer Science at the Polytechnic University of Valencia, Spain, in 1996. Currently he is Professor in the DISCA Department of the same University. His research interests include embedded and real-time operating systems.

Alfons Crespo is Professor of the Department of Computer Engineering of the Technical University of Valencia. He received the PhD in Computer Science from the Technical University of Valencia, Spain, in 1984. He held the position of Associate professor in 1986 and full Professor in 1991. He leads the group of Industrial Informatics and has been the responsible of several European and Spanish research projects. His main research interest include different aspects of the real-time systems (scheduling, hardware support, scheduling and control integration, ...). He has published more than 60 papers in specialised journals and conferences in the area of real-time systems.