Quality-of-Control Management in Overloaded Real-Time Systems

Giorgio Buttazzo, *Senior Member, IEEE*, Manel Velasco, and Pau Martí, *Member, IEEE*

Abstract—Transient overload conditions may cause unpredictable performance degradations in computer controlled systems if not properly handled. To prevent such problems, a common technique adopted in periodic task systems is to reduce the workload by enlarging activation periods. In a digital controller, however, the variation applied on the task period also affects the control law, which needs to be recomputed for the new activation rate. If computing a new control law requires too much time to be performed at runtime, a set of controllers has to be designed offline for different rates and the system has to switch to the proper controller in the presence of an overload condition. In this paper, we present a method for reducing the number of controllers to be designed offline, while still guaranteeing a given control performance. The proposed approach has been integrated with the elastic scheduling theory to promptly react to overload conditions. The effectiveness of the proposed approach has been verified through extensive simulation experiments performed on an inverted pendulum. In addition, the method has been implemented on a real inverted pendulum. Experimental results and implementation issues are reported and discussed.

Index Terms—Quality-of-control, overload management, real-time control.

1 Introduction

Digital control systems are designed to optimize performance under a given set of constraints. When the controller is executed in a computer together with several concurrent activities (e.g., sensory acquisition, system monitoring, filtering, and graphics animation), timing constraints become a crucial issue to ensure the expected control performance. Timing constraints are usually guaranteed offline by analyzing the feasibility of the schedule produced by a scheduling algorithm on a given set of tasks. Schedulability analysis is typically performed on a worst-case scenario, assuming that tasks execute for a maximum amount of time, known in advance.

Practical experience, however, teaches us that an accurate prediction of the execution behavior of a task is very difficult (if not impossible) to achieve due to a number of reasons. In fact, the execution time of a task is affected by several low-level mechanisms that are typical in modern computing systems, such as caching, prefetching, pipelining, DMA, and interrupts. Such features, although enhancing the average computer performance, introduce a nondeterministic behavior on task execution, making the estimation of the worst-case computation time very unpredictable. Even when running on a predictable hardware architecture, the execution behavior of a task can change very much as a function of the input data for programs having a complex structure. Hence, during system evolution, the overall workload of a real-time application consisting of many concurrent activities can have significant variations that cannot be easily predicted in advance.

In addition, the load can also change because new tasks can be activated in specific circumstances or timing constraints can be changed by the application to react to variations in the environment. For example, the controller of an autonomous vehicle might decide to activate an obstacle avoidance procedure when the proximity sensing system identifies an object close to the robot. Similarly, the acquisition rate of a proximity sensor could be defined as a function of the distance from an object, making the robot more reactive near the obstacles.

From the considerations illustrated above, it is easy to see that the workload of a complex control system can change significantly during system lifetime. If the load increases above the utilization bound of the adopted scheduling algorithm, timing constraints cannot be guaranteed any more and one or more tasks will miss their deadlines, causing an unpredictable performance degradation in the system. On the other hand, if the load is too low, then computational resources are wasted and the system becomes inefficient. In this situation, the available resources could be better exploited to increase system performance or, when dealing with a battery operated system controlled by a voltage variable processor, the supply voltage could be reduced to decrease power consumption and prolong system lifetime.

In real-time systems consisting of periodic activities, a way to react to workload variations is to modify task rates to bring the load to a desired value. Several methods have been proposed in the real-time literature for setting task rates to optimize control performance while ensuring the feasibility of the schedule. They are reviewed in Section 2. However, the consequence of a rate change in terms of control performance is often neglected. In other cases, schedulability analysis has been integrated with control

- *G. Buttazzo is with the Scuola Superiore Sant'Anna, Via Moruzzi, 1, 56100 Pisa, Italy. E-mail: giorgio@sssup.it.*
- *M. Velasco and P. Martí are with the Technical University of Catalonia, Pau Gargallo, 5, 08028 Barcelona, Spain. E-mail: {manel.velasco, pau.marti}@upc.edu.*

Manuscript received 6 Nov. 2005; revised 28 July 2006; accepted 8 Aug. 2006; published online 20 Dec. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0398-1105.

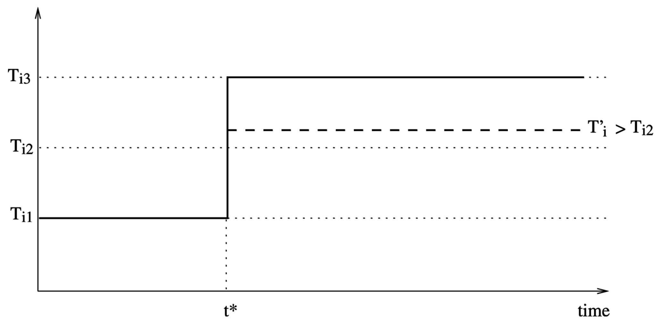

Fig. 1. When the ideal nominal period is not available, the controller must run at the next higher period to preserve system schedulability.

system design to optimize performance, but the methods are based on static parameters and are too complex to be used online.

Indeed, a digital controller is always designed as a function of the sampling period. Hence, if a task period is changed, the control law has to be changed accordingly. If the controller is simple, as is a Proportional-Integral-Derivative (PID) regulator [1], the algorithm can be changed online when the periods are recomputed. However, control algorithms are often designed using numerical methods and specific design tools (e.g., Matlab), hence, updating the period online is too costly in terms of runtime overhead. If computing a new control law as a function of the rate requires too much time to be performed at runtime, a set of controllers has to be designed offline for different rates and the system has to switch to the proper controller in the presence of an overload condition. However, working with a few discrete periods strongly limits the possibility of efficiently reacting to an overload condition, causing a waste of resources. The following example better illustrates the problem and motivates our approach.

1.1 A Motivating Example

Let us consider a system consisting of a set of n digital controllers, $\tau_1, \dots, \tau_n$ (each implemented by a periodic task), where each controller is provided in three different versions, designed at three nominal periods: $\tau_i(T_{i1})$, $\tau_i(T_{i2})$, and $\tau_i(T_{i3})$. In this way, a Quality-of-Control (QoC) manager can react to an overload condition by adapting the controller rates to degrade the system performance in a predictable fashion.

For example, assume that controller τ_i is designed to work with periods $T_{i1} = 100$, $T_{i2} = 200$, and $T_{i3} = 300$ milliseconds and, since the system is not in overload, it is running at its minimum period T_{i1}. Now, suppose that, at time t^*, for some reason (e.g., the activation of a new task), the system experiences an overload condition and the QoC manager decides that the overload can be removed by setting the period of τ_i to a value $T_i' = 220$ ms. Since controller τ_i was not designed for this rate, the period T_i' must be set equal to the next higher period ($T_i' = 300$) in order to keep the system schedulable ($T_i' = T_{ik}$ if $T_{ik-1} < T_i' \leq T_{ik}$). The situation is illustrated in Fig. 1.

However, running the controller τ_i at 300 ms wastes system resources and degrades the performance more than necessary. Two solutions can be applied to overcome this problem:

1. Increase the number of versions for each controller so that the difference between the ideal and the available period can be kept as small as desired.
2. Take a controller $\tau_i(T_{ik})$, designed to work with period T_{ik}, and run it with period T_i', properly selecting T_{ik} in order to reduce degradation.

The first solution is theoretically fine, but it requires keeping the parameters of all the controllers in memory. If the amount of memory required for storing all the controllers is too large for the available resources, this method cannot be applied in practice. In this paper, we exploit the second solution and present a method for reducing the number of controllers to be designed offline, while still guaranteeing a given control performance with a continuous period adaptation.

1.2 Contribution and Summary

In this paper, we propose a novel methodology for managing overload conditions in real-time systems consisting of a set of periodic control tasks running at different rates. The overload is handled by a rate adaptation algorithm, which takes into account both timing constraints, specified according to the elastic task model [5], and performance constraints, specified using a given QoC performance index (see Section 3).

To manage the QoC degradation caused by rate variations, each controller is designed and implemented for different sampling rates (accurately computed offline to minimize memory requirements) so that, at runtime, the QoC manager can switch to the controller instance that guarantees the required control performance. Thus, the proposed method requires two phases to be implemented. In a first phase, each controller τ_i is characterized offline to evaluate how the QoC is degraded when the controller is executed with a period T_i' different from the nominal period T_{ik} which the controller has been designed for. Then, at runtime, the QoC manager reacts to overload conditions by selecting the proper rates that satisfy schedulability and meet the desired performance requirements. Elastic scheduling theory [5] is adopted to promptly react to overload conditions and select a set of feasible periods, whereas controllers are switched online as a function of the selected period. Extensive simulations and practical experiments have been performed on an inverted pendulum to verify the effectiveness of the proposed approach.

The rest of the paper is organized as follows: In Section 3, we address the issue of measuring the quality-of-control and define a performance index that can be used to evaluate the effectiveness of the control algorithm. In Section 4, we evaluate how much the control performance degrades when a controller designed for a specific rate is executed at a different rate. Then, we propose a method for minimizing the number of controllers to be designed offline at different rates for keeping the quality-of-control in a desired range and with a bounded error. In Section 5, we propose an overload management method that adapts task periods using elastic scheduling and switches the controller to meet the performance requirements based on the performance-rate functions computed offline. Section 6 presents some simulation results performed on an inverted pendulum, whereas Section 7 illustrates the experiments carried out on a real inverted pendulum. Finally, Section 8 states our conclusions and future work.

2 Related Work

Several methods have been proposed in the real-time literature for setting task rates to optimize control performance while ensuring the feasibility of the schedule.

For example, Kuo and Mok [15] presented a load scaling technique to gracefully degrade the workload of a system by adjusting the periods of processes. In this work, tasks are assumed to be equally important and the objective is to minimize the number of fundamental frequencies. Other policies to dynamically adjust tasks' rates in overload conditions have been proposed under static [16] and dynamic priority assignments [3]. Buttazzo et al. [5] proposed an elastic approach in which tasks are treated as elastic springs whose utilization can be compressed or expanded by acting on task periods up to a desired value to reach a desired load. Elastic coefficients can be used to encode task importance so that tasks with higher values experience fewer variations.

Elastic scheduling was also extended to work with tasks with unknown and variable computation times for adaptive QoS management [6]. A feedback mechanism was used to monitor task execution, estimate the load, and change the rates accordingly. To compensate for the lower utilization caused by the quantization error, a technique has been proposed to adjust periods to bring the total utilization closer to the desired value. However, the method is not optimal and the utilization error can still be large in the worst case. Elastic scheduling was also employed to work with discrete periods for improving networks scheduling [24] and energy-aware management [22].

The major problem in the techniques cited above is that rates are computed only based on load considerations to meet schedulability constraints, without any concern on the effects that the new periods have on the control performance of the system. Indeed, when a digital controller is implemented as a periodic task, the variation applied on the task period also affects the control law, which needs to be recomputed for the new activation rate [19].

The problem of integrating real-time schedulability analysis with control system design was recently investigated by several authors; however, the consequence of a rate change in terms of control performance was not always taken into account.

For example, Seto et al. [25] proposed a method for determining tasks' periods within a specified range to minimize a control performance index defined over the task set. This approach is effective at a design stage to optimize the performance of a discrete control system, but cannot be used for online load adjustment. Abdelzaher et al. [2] presented a model for QoS negotiation to meet both predictability and graceful degradation requirements during overloads. In this model, the QoS is specified as a set of negotiation options in terms of rewards and rejection penalties. Caccamo et al. [8] introduced a rate optimization framework for real-time control tasks to optimize the control performance and guarantee the schedulability of the task set under worst-case conditions. The approach allows control tasks to execute at optimal frequencies in normal load conditions, while minimum frequencies are guaranteed for overload situations.

Shin and Meissner [26] presented a resource reallocation technique where the effects on control performance of changing periods and moving tasks are evaluated. Martí et al. [20] provided a technique that allows each instance of a control task to choose the current execution period from a set of discrete values to improve control performance and task set schedulability. Cervin et al. [9] presented a scheduling architecture for real-time control tasks in which feedback information from the CPU load is used to adjust the workload of the processor and to optimize the overall control performance by simple rescaling the task periods. The approach works fine if the sampling periods are chosen wisely, that is, for plants sampled reasonably quickly. Similarly, Martí et al. [21] presented an optimal approach to dynamically allocate processor resources as a function of the plant states. Eker and Cervin [12], as well as Palopoli et al. [23], presented specific software tools for evaluating both the control performance and the schedulability of the real-time system.

The previous works have treated different aspects of the codesign of control and real-time systems. However, none of them has focused on determining the minimum number of controllers required to keep a desired quality of control under overload conditions, as we propose here. This work extends a preliminary approach we started investigating but using only simulation experiments [7].

3 Evaluating Control Performance

The primary criterion for evaluating the performance of control systems is to meet stability and response performance specifications, such as transient response and steady-state accuracy. Beyond such requirements, controller design attempts to minimize the system error produced by certain anticipated inputs. The system error is defined as the difference between the desired response of the system and its actual response. The smaller the difference, the better the performance.

Performance criteria (also called performance indexes or cost functions) are mainly based on measures of the system error. Traditional criteria (reported in control textbooks, e.g., [11]), such as IAE (Integral of the Absolute Error), ITAE (Integral of Time-weighted Absolute Error), ISE (Integral of Square Error), or ITSE (Integral of Time-weighted Square Error), provide quantitative measures of a control system response and are used to evaluate (and design) controllers. Some of them weight errors with time, penalizing steady-state errors and discounting the transient response errors.

More sophisticated performance criteria, mainly used in optimal control problems, account for the system error and for the energy that is spent to accomplish the control objective. The higher the energy demanded by the controller, the higher the penalty paid in the performance criterion. In some cases, system error and control energy are multiplied by a weight to balance their relative importance.

For example, in [17] and [10], the performance criterion is only based on the system error, whereas, in [26] and [25], both system error and control energy are considered.

The goal of our approach is to minimize the number of controllers that are required to guarantee a graceful control performance degradation when continuously adapting the period of the control task. The IAE performance criterion will be used to compare the control performance of a task running at different periods. Among all of the available norms, IAE has been selected because it gives the best curves to conceptually illustrate the problem we are addressing (see a further explanation in Section 4.2.3).

The IAE index is defined as follows:

$$IAE = \int_0^\infty |e(t)|\, dt, \tag{1}$$

where $e(t)$ is the system error and $|.|$ denotes an appropriate norm. The integral upper limit of (1) could also be any time t marking the evaluation time interval. If we assume the equilibrium point to be 0 (that is, the system response converges to zero), then the system error is the same as the system output $y(t)$. In particular, since we want to compare the performance of a controller running with different periods, $IAE(T_0, T)$ will denote the IAE value obtained by a controller designed with a nominal period T_0, but running with a period T. Hence, we have:

$$IAE(T_0, T) = \int_0^\infty |y(t)|\, dt, \tag{2}$$

where $y(t)$ is the system output. Notice that, since the integral upper bound of the IAE index is equal to infinity, any closed loop with permanent error will give an infinite value. However, since controllers are designed to remove permanent errors, the value of the index will be finite. In the experimental evaluation, for practical purposes, a time upper bound is used for the integral. For the objective of this work, the mathematical expression of $IAE(T_0, T)$ is required to determine the minimum number of controllers to be designed.

3.1 Mathematical Expression of $IAE(T_0, T)$

In this section, we derive the mathematical expression of the IAE for a controller designed to work at a nominal period T_0 but running with a period T.

Let (3) and (4) be the linear time-invariant system equations of a continuous-time system in state space form [1]:

$$\dot{x} = Ax(t) + Bu(t), \tag{3}$$

$$y(t) = Cx(t), \tag{4}$$

where $x(t)$ and $y(t)$ are the state and the output of the system at time t and $u(t)$ is the control signal applied to the system at time t. The state (3) and output (4) equations defining a given system can be considered an abstract summary of the data obtained by subjecting the system to different inputs (control signals) and observing the corresponding outputs.

Theorem 1. *The mathematical expression of the $IAE(T_0, T)$ of a system specified by (3) and (4), where the excitation input $u(t)$ is given by state feedback with a discrete-time controller designed to work at a nominal period T_0, but running with a period T, is given by*

$$IAE(T_0, T) = \sum_{k=0}^{\infty} \int_0^T |C\Phi_c(t, T_0)x(kT)|\, dt, \tag{5}$$

where $\Phi_c(t, T_0)$ is the discrete-time closed-loop system matrix obtained with a discretization period of t, and $x(kT)$ is the system state at each time kT.

Proof. Consider the discrete-time state space representation of the system (3) and (4), obtained with a discretization period T and output at time $k+1$, given by:

$$x(kT + T) = \Phi(T)x(kT) + \Gamma(T)u(kT), \tag{6}$$

$$y(kT + T) = Cx(kT + T), \tag{7}$$

where

$$\Phi(T) = e^{AT} \qquad \Gamma(T) = \int_0^T e^{As} ds\, B.$$

The output y at any given time, within each sampling period T, is given by:

$$x(kT + t) = \Phi(t)x(kT) + \Gamma(t)u(kT), \tag{8}$$

$$y(kT + t) = Cx(kT + t), \tag{9}$$

with $0 < t < T$. As before, if we consider the equilibrium point to be 0, the system error becomes equal to the system output y. The integral of the absolute error of the system output y during each period T, IAE_T, is given by

$$IAE_T = \int_0^T |y(kT + t)|\, dt, \tag{10}$$

with $0 < t < T$. The $IAE(T_0, T)$ evaluation of the system output is the sum of all the IAE_T values for each period T, hence:

$$IAE(T_0, T) = \sum_{k=0}^{\infty} \int_0^T |y(kT + t)|\, dt. \tag{11}$$

Substituting the output $y(kT + t)$ in (11) by the expressions given by (8) and (9), we obtain:

$$IAE(T_0, T) = \sum_{k=0}^{\infty} \int_0^T |C(\Phi(t)x(kT) + \Gamma(t)u(kT))|\, dt. \tag{12}$$

Considering the controller K designed assuming a nominal period T_0, the state feedback is given by

$$u(kT) = K(T_0)x(kT). \tag{13}$$

Substituting (13) in (12) and reorganizing the resulting expression, we obtain:

$$IAE(T_0, T) = \sum_{k=0}^{\infty} \int_0^T |C((\Phi(t) + \Gamma(t)K(T_0))x(kT))|\, dt. \tag{14}$$

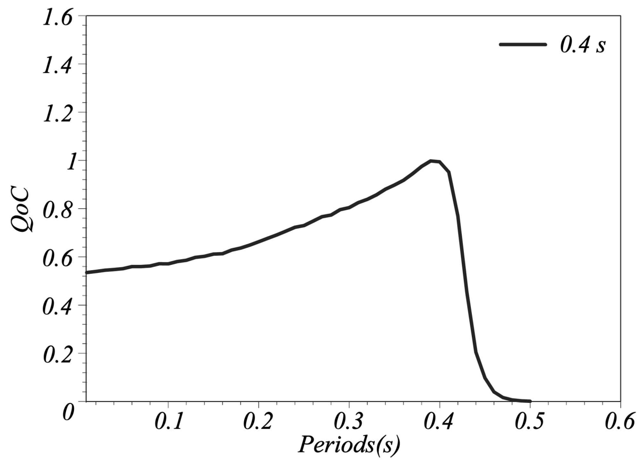

Fig. 2. Performance-rate function of a controller designed to work at a nominal period $T_0 = 0.4s$.

Simplifying (14) by renaming the closed-loop system matrix, $\Phi_c(t, T_0) = \Phi(t) + \Gamma(t)K(T_0)$, the theorem follows. □

3.2 Quality-of-Control Performance Index

As done in [20], instead of using the value given by the $IAE(T_0, T)$ index, we use its inverse as given by (15), thus working with a measure that can be interpreted as a Quality-of-Control (QoC): the smaller the errors, the better the QoC:

$$QoC(T_0, T) = \frac{1}{IAE(T_0, T)}. \tag{15}$$

In the next section, we show how to use the performance index defined above to describe the quality-of-control of a real-time system as a function of the sampling rate.

4 Performance-Rate Functions

Using the performance index defined by (15), our objective is to evaluate how much the control performance degrades when a controller designed for a specific rate is executed at a different rate. Then, by knowing the relation between performance and rate for a specific controller, we can decide the range of periods for which that controller can guarantee a desired level of performance and, thus, decide when to switch to another controller.

4.1 Analysis

To derive the relation between performance and rate, we simulated a control system for an inverted pendulum mounted on a motor driven cart, obtained via discretization of a linear continuous time-invariant state-space representation. The control was derived using simple state feedback (pole placement).

Fig. 2 shows the values of the QoC performance index achieved by the state feedback controller. The controller was designed to work at a nominal period $T_0 = 0.4s$ and tested within a range of periods from $T_{min} = 0.01s$ to $T_{max} = 0.6s$. As we can see from the curve, the quality of control degrades significantly when the sampling period increases with respect to the nominal value, whereas it is less sensitive to periods smaller than T_0.

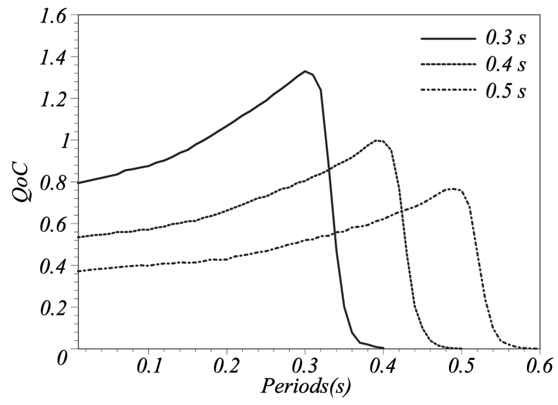

Fig. 3. Performance-rate functions of a controller designed to work at three different nominal periods.

The curve shown in Fig. 2, which relates control performance and controller execution rate, is referred to as the *Performance-Rate Function* (*PRF*) and it is characterized by a nominal period T_0, used to design the controller, and a set of periods $[T_{min}, \ldots, T_{max}]$, used to run the controller.

Each performance-rate function can be formulated in terms of (15) as

$$\begin{aligned} &PRF(T_0, t, T_{min}, T_{max}) = \\ &\{QoC(T_0, t) \mid t \in [T_{min}, \ldots, T_{max}]\}. \end{aligned} \tag{16}$$

Using the notation introduced in (16), the performance-rate function illustrated in Fig. 2 can be expressed as $PRF(0.4s, t, 0.01s, 0.6s)$. The shape of degradation also depends on the nominal period T_0. For example, Fig. 3 shows the performance-rate functions obtained from a controller designed to work with three different sampling periods: $T_1 = 0.3s$, $T_2 = 0.4s$, and $T_3 = 0.5s$, but tested within the same range of periods as before. Note that the degradation is more significant for higher nominal periods. However, for each function, the properties outlined before hold. For periods larger than the nominal period, the QoC of the inverted pendulum response drastically decreased: The system quickly became unstable, making the pendulum fall down. This was an expected behavior because the system was controlled less frequently than it should be.

For rates higher than the nominal one (left side of each curve), the response suffers a graceful and acceptable degradation (smooth slope). In terms of the inverted pendulum response, this means that it takes more time for the pendulum to recover from a perturbation and it can suffer bigger deviations from the desired working point (vertical position). This behavior is less intuitive because we are controlling the system more frequently. However, when a controller is designed for a specific period, the optimal performance is achieved when running exactly with that value. Therefore, even a shorter period implies a performance degradation. A formal justification of this phenomenon would require too much space to fit in this paper, but an intuitive explanation can be given by considering the case of displaying a digital image on a monitor with a given

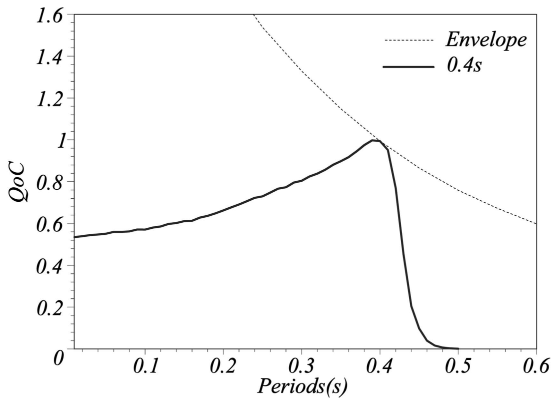

Fig. 4. Performance-rate function of a controller tuned to work with a nominal period $T_0 = 0.4s$ against the ideal controller tuned at any rate.

resolution. If the monitor has the same resolution as the image, the result has the maximum quality. If the monitor has a lower resolution, the image loses quality since pixels are dropped. If the monitor has a higher resolution, the image also loses quality since some pixels are duplicated and, as a consequence, antialiasing algorithms do not work properly, producing an image of degraded quality.

Considering the previous observations, we will allow controllers to execute only with a period less than or equal to the nominal one, trading graceful performance degradation for controller flexibility, as further explained in Section 4.3. In order to prevent instability, a controller is not allowed to execute with periods longer than the nominal one.

A controller running with a period that is different from the nominal one is called a *nontuned controller*. To better evaluate the error produced by a nontuned controller at any running period, it is worth comparing the control performance index with the one achieved by the corresponding tuned controller. Fig. 4 shows the performance function derived from a controller tuned with a period $T_0 = 0.4s$ against the curve achieved by a controller tuned for any rate. Note that this curve is the envelope of the set of performance rate functions.

The curve relative to the ideal controller tuned at any rate is a special case of the performance-rate function expressed by (16), where $t = T_0 \; \forall t \in [T_{min}, \dots, T_{max}]$. Therefore, the performance-rate function of an ideal controller can be formulated as follows:

$$PRF(t, t, T_{min}, T_{max}) = \{QoC(t, t) \mid t \in [T_{min}, \dots, T_{max}]\}. \tag{17}$$

Note that, in (17), if $T_{min} = T_{max} = t$, then the performance-rate function is evaluated in one single period value. And, if $t = T_0$, then $PRF(T_0, T_0, T_0, T_0) = QoC(T_0, T_0)$ corresponds to the maximum of each performance-rate function.

4.2 Discussion

Performance-rate functions will be used for bounding the error during overload conditions. Before presenting our approach, it is worth making some considerations about the amplitude of these curves, their specific shape, and the selection of the IAE index.

4.2.1 Amplitude

It can be easily seen that the amplitude of the performance-rate functions depends on the initial conditions. In fact, given that $x(kT) = \Phi_c^k(T)x(0)$, (5) can be expressed as

$$IAE(T_0, T) = \sum_{k=0}^{\infty} \int_0^T \left| C\Phi_c(t, T_0)\Phi_c^k(T)x(0) \right| dt. \tag{18}$$

From (18), it is clear that the amplitude of each curve depends on the initial condition used to obtain the curve. Therefore, when using several curves, to avoid false comparisons or scaling errors, we normalized them by evaluating $IAE(T_0, T)/|x(0)|$, where $|\cdot|$ is the vector norm associated to the inner integral matrix norm of (18). This will provide curves that do not depend on the initial conditions. For the special case of $x(0) = 0$, the previous expression does not exist. But, for this case, if the initial state is 0, the IAE evaluation will also be 0 as can be deduced from (18).

4.2.2 Shape

The specific shape of the performance-rate functions and their relative position depends on several factors, such as the nature of the plant, the control specifications, the real-time constraints, and the controller design methodology.

Regarding the type of plant, for example, a highly nonlinear plant may always become unstable at the same execution period due to its nonlinearities. Therefore, the relative positions of several curves for different nominal periods will not be as in Fig. 3. They will tend toward 0 at the same time instant.

In a real-time control system, meeting both control specifications and real-time constraints may be somewhat subtle. On one hand, meeting control specifications may require short periods, which may prevent task set schedulability. On the other hand, if the control task period is specified to obtain a feasible task set, the control performance specifications may not be completely achieved. Therefore, a good design must trade control performance versus real-time performance. As a consequence, the shape of the curves must be computed considering these decisions.

It is also important to account for the controller design methodology. For example, performance-rate functions will have different shapes for controllers designed through a discrete-time approximation of a continuous-time design (like discrete-time PID controllers) than for controllers designed through direct discrete-time design methods (e.g., discrete-time pole placement).

In summary, to successfully use performance-rate functions in real-time and control codesign approaches as we do, the designer must be aware of the type of plant, system constraints, and controller design methods to be used. For an initial analysis of these issues, see [27].

4.2.3 The IAE Index

By using the IAE index (or any other standard index) to evaluate the goodness of a plant response, we are hiding the

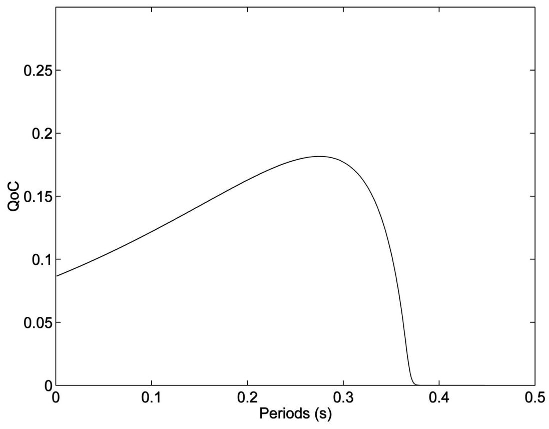

Fig. 5. Performance rate function for a quadratic performance index.

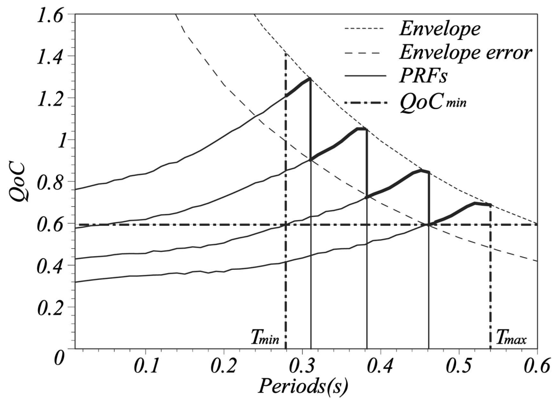

Fig. 6. Sequence of controllers bounding the performance error to a given value.

real dynamics of the plant. That is, two different dynamics, e.g., an oscillating dynamic and a nonoscillating but slow dynamic, can give the same IAE value. This may complicate comprehension of the real behavior of the system.

At the controller design stage, for a nominal period, we specify the exact behavior for our system. However, when the controller is executed with other periods than the nominal, the real behavior cannot be controlled anymore. Therefore, it is not possible to specify a particular family of behaviors.

In Section 3, we argued that the IAE index was selected because it gives the best curves to conceptually illustrate the problem we are addressing.

Among the IAE, ISE, ITAE, and ITSE indexes, ITAE and ITSE are not really useful because they are weighted by time. This shifts the maximum of each curve, thus losing the graphical relation between each nominal period and the corresponding maximum of each performance-rate curve.

Indexes like ISE or any quadratic cost function (such as those used in optimal control problems) will give similar results as IAE. For example, using a standard quadratic cost function such as (19), where Q is a weighting matrix,

$$J(T_0, T) = \int_0^{\infty} y^T(t) Q y(t)\, dt, \tag{19}$$

the type of performance rate functions are equivalent to those obtained by IAE. Fig. 5 shows the equivalent curve to Fig. 2 when J is used.

In summary, performance-rate functions are good tools for codesigning real-time and control systems, but their specific shape depends on several engineering and design choices.

4.3 Bounding the Error during Overloads

To cope with overload conditions, tasks must change rates. However, if we want to always have the best control performance achievable for any given task rate, the execution period of the task must always coincide with the nominal period used for the controller design. As described in [19], this can be achieved either by redesigning the controller at runtime for each new execution period or by accessing it from a table of predesigned controllers. If redesigning the controller is too expensive (in terms of computational overhead) to be done at runtime, a number of controllers must be designed offline and stored into memory: one for each possible rate the task may adapt to cope with overload situations. When the number of possible periods the task is allowed to take is too big, the solution presented above can be unfeasible in terms of memory demand (see [19] for a detailed overhead analysis).

A possible way to overcome this problem is to restrict the task to work only with discrete rate variations. However, working with a small number of discrete periods is not efficient because it may be impossible to reach the desired load after a rate variation. Having a discrete number of rates means having a discrete number of resulting workloads. For example, if, during an overload, the new periods are computed using a typical load compression algorithm (e.g., the elastic compression algorithm [5] or other similar methods [3]), then the resulting periods (which are treated as continuous variables) have to be enlarged to the closest available period for which the controller has been designed.

After resizing all the periods, the system workload may be much lower than the desired one specified in the load compression algorithm. As a consequence, the system would run with low efficiency. To address this issue, some authors [24], [10] proposed an adjustment technique to slightly resize some periods after a discrete load compression to reach a workload closer to the desired one.

However, there is no need to work with discrete periods. Instead of forcing the control tasks to work at predefined rates, one could let them work at a rate resulting from the compression algorithm and switch to the most appropriate controller that bounds the control performance error with respect to the ideal tuned controller, thus posing a trade-off design choice: number of controllers versus QoC.

To bound the error produced by a nontuned controller (with respect to the ideal one tuned at any period), we have to switch controllers as soon as the control performance decreases below a given bound ϵ_{max}. To reduce the number of controllers that have to be designed offline to keep the error below a given value, we can apply the following approach (see Fig. 6 as a reference):

1. The user starts by specifying the minimum admissible QoC level (QoC_{min}) and the maximum error ϵ_{max} (specified in percentage) that can be tolerated

with respect to the optimal curve corresponding to the ideal tuned controller (the *Envelope* curve in Fig. 6).

2. The maximum error, ϵ_{max}, allows the user to derive the minimum performance curve (the *Envelope Error* curve in Fig. 6), which is given by $Envelope - \epsilon_{max} \cdot Envelope$.
3. The range of possible periods is bounded by T_{min} and T_{max}. T_{min} corresponds to the default period of the controller task that is guaranteed by the scheduling algorithm adopted by the system. T_{max} corresponds to the nominal period of the controller whose performance rate function crosses the intersection of the minimum performance curve with QoC_{min}.
4. The first controller can be designed using the nominal period $T_1 = T_{max}$.
5. The next nominal period can be set to the value $T_2 < T_1$ given by the intersection of the performance-rate function tuned at T_1 with the minimum performance curve (i.e., the *Envelope Error* curve) and so on for the other periods while they are not smaller than T_{min}. Note that, since envelopes are monotonically decreasing, the next nominal period (found on the left of T_{max}) will always be an intersection above QoC_{min}.

It is worth noting that QoC_{min} is the worst acceptable quality, meaning that, beyond this limit, the controlled plant would misbehave or crash, whereas ϵ_{max} is the tolerated error of the controlled plant for any given period. Independently of the relative value of the executing period within the allowed range, ϵ_{max} will always imply a quality higher than QoC_{min}.

Fig. 6 illustrates an example showing the sequence of performance-rate functions that bound the error with respect to the tuned controller (the *Envelope* curve) to a value equal to ϵ_{max} (the *Envelope Error* curve). For this case, we need four performance-rate functions (corresponding to nominal periods equal to 0.31s, 0.38s, 0.46s, and 0.54s) to bound the error produced by a nontuned controller executing within $[T_{min}, \ldots, T_{max}] = [0.28s, \ldots, 0.54s]$, where $\epsilon_{max} = 0.3$ and $QoC_{min} = 0.6$. T_{min} is the default task period ($0.28s$) and T_{max} is obtained from QoC_{min}, which is the minimum quality of service level specified by the user. Note that the bold line in Fig. 6 is the expected QoC when controllers are switched in the overload situation. In terms of the inverted pendulum, this translates into a maximum allowed recovery time and pendulum deviation. This specification can be easily mapped into a minimum QoC.

Notice that load variations cannot be predicted at runtime, but the maximum load variation should be estimated at design time. If the overload is too high (beyond the predicted value) to push periods outside the range, the system should generate an exception because this is a design error. The exception could be handled in several ways, e.g., by admission control, but then the consequence of rejecting a task could be even worse than running it at a lower rate. However, this issue is out of the scope of the paper.

Note also that ϵ_{max} relates control performance to system resources in terms of memory requirements (number of controllers to be designed offline and stored for runtime access). If $\epsilon_{max} = 0$, (that is, the *Envelope Error* curve coincides with the *Envelope* curve), the number of controllers to be designed offline would be equal to the number of possible rates the task could run to adapt within the specified range $[T_{min}, \ldots, T_{max}]$, which causes the largest memory demand. The QoC achieved in this case would be the optimal one because, for each execution rate, the system would execute the corresponding tuned controller. As ϵ_{max} increases, fewer controllers need to be designed offline (meaning less storage memory) at the expense of reducing the average achievable QoC.

4.4 Trading Performance with Resources

As outlined in Section 4.3, the value of ϵ_{max} determines the number of controllers to be designed offline and, therefore, the resource requirements. The memory requirements for storing a controller has been discussed by Martí et al. [19]. Here, we are interested in evaluating how the performance degrades as a function of the number of controllers.

It should be pointed out that the specification of ϵ_{max} has to be coherent with QoC_{min} and T_{min}. In fact, it makes no sense to specify an ϵ_{max} that results in a QoC lower than QoC_{min} at T_{min} because it is a contradiction in terms of control performance.

The maximum value allowed for ϵ_{max} can be computed by noting that, at T_{min}, the distance between the *Envelope* curve and the *Envelope error* curve is equal to the distance between the *Envelope* curve and QoC_{min}, that is:

$$\underbrace{QoC(T_{min}, T_{min})}_{Envelope\_curve(T_{min})} - \underbrace{(QoC(T_{min}, T_{min}) - \epsilon_{max} QoC(T_{min}, T_{min}))}_{Envelope\_error\_curve(T_{min})}$$
$$= \underbrace{QoC(T_{min}, T_{min})}_{Envelope\_curve(T_{min})} - QoC_{min}.$$

Hence, the maximum value for ϵ_{max} is given by

$$\epsilon^*_{max} = \frac{QoC(T_{min}, T_{min}) - QoC_{min}}{QoC(T_{min}, T_{min})}. \quad (20)$$

Fig. 7 illustrates a few examples showing the relation between ϵ_{max} and the minimum number of controllers. Each subfigure shows the *Envelope* curve (dotted line), the *Envelope error* curve (dash-dotted line), the QoC_{min} value (horizontal dashed line), the T_{min} value (vertical solid thick line), and the resulting performance rate functions (solid lines) for each experiment (except for Fig. 7d)), that is, the number of controllers to be designed offline. All of the experiments have been carried out for $QoC_{min} = 0.6$ and $T_{min} = 0.3$.

Fig. 7a shows the result achieved by specifying the maximum possible error $\epsilon_{max} = 0.52$, computed using (20). In this case, only one controller is required, with a nominal period of 0.48.

Figs. 7b and 7c show the number of controllers required for $\epsilon_{max} = 0.3$ and $\epsilon_{max} = 0.15$, respectively. It is worth noticing that the relation between ϵ_{max} and the number of controllers is not linear. In fact, three controllers are required with $\epsilon_{max} = 0.3$, whereas seven controllers are needed with $\epsilon_{max} = 0.15$.

Fig. 7d illustrates the case for $\epsilon_{max} = 0$, in which the *Envelope error* curve coincides with the *Envelope* curve. Here, the number of required controllers is not shown since it

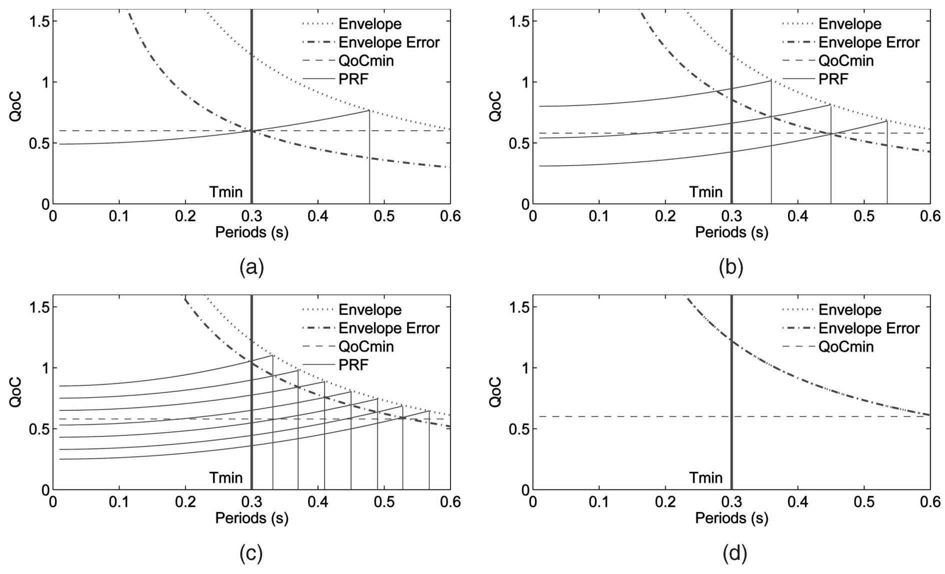

Fig. 7. Trading off performance, ϵ_{max}, and resources, with $QoC_{min} = 0.6$ and $T_{min} = 0.3$. (a) $\epsilon_{max} = 0.52$. (b) $\epsilon_{max} = 0.3$. (c) $\epsilon_{max} = 0.15$. (d) $\epsilon_{max} = 0$.

Algorithm 1: Computing the minimum number of periods (controllers)

```
Nominals ($T_{min}$,$QoC_{min}$, $\epsilon_{max}$)
begin
    $T \leftarrow Solve\,(QoC\,(t,t)\ \text{-}\ \epsilon_{max} \cdot QoC\,(t,t)\ =\ QoC_{min},t)$ [a]
    $T_{max} \leftarrow Solve\,(PRF\,(t,T,T,\infty)\ =\ QoC_{min},t)$ [b]
    $Nominals \leftarrow \{T_{max}\}$
    $t \leftarrow T_{max}$ [c]
    while $t > T_{min}$ do
        $t \leftarrow (t - 1)$
        if $QoC\,(t,t)\ \text{-}\ PRF\,(T_{max},t,T_{min},T_{max})\ =\ \epsilon_{max} \cdot QoC\,(t,t)$ then
            $Nominals \leftarrow Nominals \cup \{T_{min}\}$
            $T_{max} \leftarrow t$

    end
    return $Nominals$
end
```

[a]Find t such that QoC_{min} intersects envelope error curve.

[b]Find a nominal period t-that will be T_{max}- whose performance rate function at time t is equal to QoC_{min}.

[c]Starting with nominal period T_{max}, find the rest of nominal periods, which are given by times t $> T_{min}$ at wich their performance rate functions intersect envelop error curve.

Fig. 8. The Nominals algorithm.

would be theoretically infinite (and practically equal to the number of possible rates, assuming a given granularity).

The results illustrated in Fig. 7, as well as other simulations experiments (not shown here for space limitations), indicate that the number of controllers to be designed offline is inversely proportional to ϵ_{max}:

$$number\ of\ controllers \approx \frac{1}{\epsilon_{max}}. \qquad (21)$$

The exact relationship is not easy to derive since it depends on the exact shape of the performance rate functions, that is, on the matrices that characterize each controlled plant.

4.5 The Algorithm

In this section, we present the detailed algorithm (named *Nominals*) that produces the set of nominal periods that can be used to design the minimal set of controllers that keep the QoC error smaller than ϵ_{max}. The algorithm requires as input arguments the default rate of the controller (T_{min}), the minimum QoC level specified by the user (QoC_{min}), and the maximum tolerated error with respect to the optimal envelope curve (ϵ_{max}). The pseudocode of the algorithm is reported in Fig. 8.

As it can be deduced from the code, the complexity analysis of the algorithm can be split into two levels: the actual search for the set of nominal periods and the evaluation of $QoC(t, t)$ and $PRF(T_{max}, t, T_{min}, T_{max})$. The complexity of evaluating these two functions, which is equivalent to evaluating (15) and (16), basically depends on the order of the model matrices that characterize each controlled plant. Recall, however, that this algorithm is to be executed offline.

After finding T_{max}, the search depends on the processor clock granularity g. In particular, the complexity is given by the number of evaluations of the **if** statement, which is $\frac{T_{max}-T_{min}}{g}$. For the sake of clarity, the algorithm assumes

$g = 1$. If this is not the case, the first statement in the **while** loop, $t \leftarrow (t-1)$, should be replaced with $t \leftarrow (t-g)$.

Taking into account that the number of controllers to be designed offline can be approximated by (21), the computational complexity of the *Nominals* algorithm plus the cost of computing the set of nominal controllers can be expressed as follows:

$$cost = \left\lceil \frac{T_{max} - T_{min}}{g} \right\rceil a + \left\lceil \frac{1}{\epsilon_{max}} \right\rceil b. \qquad (22)$$

The first summand of (22) is the number of evaluations of the **if** statement multiplied by a, which is the cost for evaluating its condition. The second summand is the approximated number of nominal controllers multiplied by b, which is the cost for computing each controller. Altogether, in a standard PC Pentium 4 with 1GB of RAM, the *Nominals* algorithm takes a few hundred milliseconds for a few controllers (large ϵ_{max}) up to a few seconds when the number of controllers is 300 ($\epsilon_{max} = 0$, $g = 1$).

5 Overload Management Policy

The algorithm presented in Section 4.5 can be used to derive the nominal rates of the performance-rate functions that allow keeping the maximum error ϵ_{max} below a given bound. Such nominal rates are then used to design the corresponding controllers that have to be stored into memory for a possible runtime adaptation during transient overload conditions.

Initially, the system starts executing each controller at the nominal rate closest to T_{min}, that is guaranteed by the scheduling algorithm adopted by the system and gives the best QoC. If an overload condition occurs, task periods need to be increased to reduce the load up to a desired value.

In this work, task rate adjustment is performed through the elastic task model [4], [5], according to which task utilizations are treated like springs that can be compressed to a given workload through period variations. The advantage of the elastic model with respect to the other methods proposed in the literature is that a new period configuration can be easily determined online as a function of the elastic coefficients, which can be set to reflect tasks' importance. The greater the elastic coefficient, the more flexible a task is to period variations.

Once elastic coefficients are defined based on some design criterion, the new task utilizations can be quickly computed online depending on the current workload and the final desired load level. Then, the new period configuration can be easily derived from the task computation times and the (compressed) utilizations.

If the period T_i resulting after the compression algorithm falls in the interval $(T_k, T_{k+1}]$ given by two consecutive nominal periods in the performance-rate graph, each control task must select the controller with nominal period equal to T_{k+1}. The way controllers have been designed guarantees that, during overload conditions, as long as periods vary in the range $[T_{min}, T_{max}]$, the QoC degradation will be bounded; that is, the performance error with respect to the ideal tuned controller will be less than ϵ_{max}.

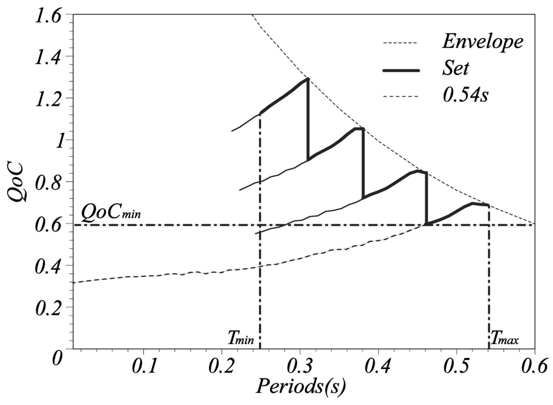

Fig. 9. Performance-rate functions for the Optimal task (*Envelope*), Adaptive task (*Set*), and Static task $(0.54s)$.

Note that the given algorithm might cause fast switching between controllers (chattering) if the workload quickly changes around a switching point. To avoid this problem, hysteresis can easily be added into the algorithm to absorb chattering.

6 Simulation Results

In this section, we evaluate the method we have presented to control the QoC performance under overload conditions. To illustrate the benefits of our approach, we focus on a simple scenario, where the period of a control task can be increased from $0.25s$ to $0.54s$ to cope with an overload condition. The numbers we use here relate to the simulations presented in Section 4 performed on an inverted pendulum. Here, we compare three different controller execution strategies whose performance curves are illustrated in Fig. 9 (which has been extracted from Fig. 6):

1. **Optimal task**. This case considers the execution of a tuned controller for each period computed by the load compression algorithm. Although this solution theoretically provides the best QoC, it may not be practical for the large amount of required memory. In fact, for this particular range of periods, if we assume a system clock granularity of $0.01s$, we need enough memory for storing 30 controllers (corresponding to nominal periods of $0.25s, 0.26s, \ldots, 0.54s$). Note that the number of controllers increases to 300 (or 3,000) with a clock granularity of $1\mu s$ (or $0.1\mu s$). The performance-rate function of this task corresponds to the *Envelope* curve illustrated in Fig. 9.
2. **Adaptive task**. This case considers the set of four controllers, derived with the method presented in Section 4.3 and Section 4.5, that guarantees a maximum error $\epsilon_{max} = 0.3$. According to the overload management policy explained in Section 5, whenever the period of a controller is adapted by the elastic compression algorithm, the system selects the appropriate controller for each period computed by the load compression algorithm. The performance rate function of this task corresponds to the *Set* curve illustrated in Fig. 9.

TABLE 1
Experimental Evaluation Summary

Strategy	Optimal task	Adaptive task	Static task
# Controllers	30	4	1
Avg. QoC	1.15	1.01	0.67
% QoC	100	87.8	58.2
Keep QoC_{min}	OK	OK	fails

3. **Static task**. This case considers the execution of a single controller, regardless of the task execution period in the specified range. The performance-rate function of this task corresponds to the $0.54s$ curve in Fig. 9. For this case, in overload conditions, no single controller is able either to keep the pendulum stable or to guarantee the QoC_{min} specified by the user. Note that if the controller is designed according to any period belonging to the specified range, if the task executes with a longer period, the inverted pendulum may fall down (as discussed in Section 4.1). The only case where the task does not execute with a period longer than the nominal one is when the controller is designed with a nominal period equal to the upper limit of the specified execution range, that is, $0.54s$. However, in this case, as can be seen from Fig. 9, for execution rates ranging form $0.25s$ to $0.46s$, the controller provides a QoC lower than that specified by the user. Also note that the performance-rate function of this task overlaps with the *Set* curve in the time interval from $0.46s$ to $0.54s$.

Table 1 shows the evaluation summary of the experiments performed on the three controller execution strategies: The second row reports the number of required controllers, the third row indicates the average QoC achieved by each method, the fourth row expresses the QoC in average percentage, and the last row indicates whether the method is able to keep the minimum QoC level specified by the user. As we can see from the table, the *optimal task* achieves an average QoC of 1.15 for each execution. However, by drastically reducing the number of controllers from 30 to 4, our *adaptive task* is able to keep an acceptable QoC level, equal to 1.01, corresponding to 87.8 percent of the optimal value, whereas the *static task*, which does not adapt the controller while the task period is increased, only achieves an average QoC of 0.67, corresponding to 58.2 percent of the optimal value.

7 Experimental Results

In this section, we describe the experimental setup that we used to corroborate the theoretical approach, as well as the simulation results. We explain the coding of the algorithms on a real-time kernel and discuss the experimental results.

7.1 Setup

Fig. 10 illustrates the experimental setup. The plant is a pendulum mounted on a cart (from Inteco [14]), where the pole can swing freely only in the vertical plane. The cart is actuated by a DC motor on a straight track, using a belt for transmission. For the sake of balance, two identical joined pendulum rods and loads are attached to the cart (see Fig. 11). Two optical incremental sensors are connected to the system: one for the pendulum angle and one for the cart position. Using these signals, the computer calculates the control outputs and sends them to the DC motor through a Pulse Width Modulator (PWM) module so that the desired control is achieved.

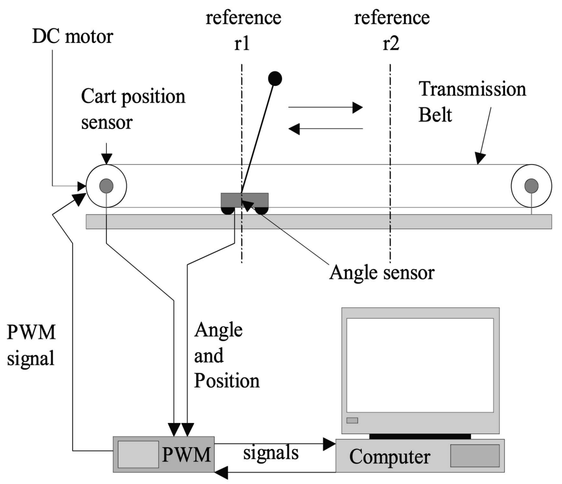

Fig. 10. Experimental setup.

The personal computer runs the SHARK real-time kernel [13], which implements the elastic task model [5].

7.2 The Process Model

To be able to control the pendulum, its dynamics has been derived. In particular, the dynamic relations for the cart position and the pendulum angle are given by (23) and (24),

$$\ddot{x} = \frac{F - ml(\ddot{\theta}\cos\theta + \dot{\theta}^2 \sin\theta) - f_c\dot{x}}{(M+m)}, \tag{23}$$

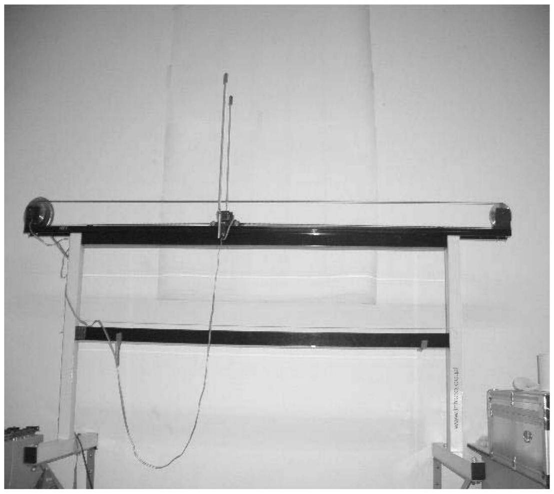
Fig. 11. The inverted pendulum.

TABLE 2
Parameters of the Process

Parameter	Description
x	Cart position
l	Pendulum length
θ	Pendulum angle (from the up-position)
m	Mass of pendulum (load and rod)
M	Cart mass
F	Force applied to cart
I_p	Moment of inertia of the pendulum with respect to the pivot point
f_p	Viscous friction for the pendulum at the pivot point
f_c	Dynamic friction for the cart

$$\ddot{\theta} = \frac{mgl\sin\theta - ml\ddot{x}\cos\theta - f_p\dot{\theta}}{(I_p + ml^2)}, \tag{24}$$

where the parameters are shown in Table 2.

The nonlinear equations (23) and (24) describe the system with four states, x, $\dot{x}$, θ, $\dot{\theta}$, cart position and velocity, and angle and angular velocity, respectively. To be able to use normal stabilization theory, (23) and (24) must be linearized. In the up position, it is natural to choose the equilibrium point ($\theta_0 = 0$, $\dot{\theta}_0 = 0$) and linearize around it. This is the region in which the stabilizing control will be applied. Around the equilibrium point, the linearized versions of (23) and (24) are given by (25) and (26).

$$\ddot{x} = \frac{F - ml\ddot{\theta} - f_c\dot{x}}{(M + m)}, \tag{25}$$

$$\ddot{\theta} = \frac{mgl\theta - ml\ddot{x} - f_p\dot{\theta}}{(I_p + ml^2)}. \tag{26}$$

Taking into account the parameter values reported in Table 3, the A and B matrices of the continuous model of the pendulum in the up position are given by (27) and (28).

$$A = \begin{bmatrix} 0 & 1.00 & 0 & 0 \\ 0 & -1.71 & -0.75 & 0.02 \\ 0 & 0 & 0 & 1.00 \\ 0 & 2.71 & 16.70 & -0.55 \end{bmatrix}, \tag{27}$$

TABLE 3
Parameters Values

Parameter	Value	Unit
l	0.260	m
m	0.120	kg
M	0.572	kg
I_p	0.0116	kg/rad
f_p	0.01	Ns/rad
f_c	1.1	Ns/m

```
Algorithm 2: Controller
  Nominals={$T_{n1}$,$T_{n2}$,.}, ordered in increasing order
  while TRUE do
      $T_i \leftarrow getPeriod()$;
      if $T_i \leq T_{n1}$ then
          $Controller_{n1}$
      else if $T_i \leq T_{n2}$ then
          $Controller_{n2}$
      else if $T_i \leq T_{n3}$ then
          ⋮
      $taskendCycle()$;
  end
```

Fig. 12. Pseudocode of the control task.

$$B = \begin{bmatrix} 0 \\ 1.56 \\ 0 \\ -2.46 \end{bmatrix}. \tag{28}$$

7.3 Controller Design and Implementation

The controller was designed using standard pole placement in the discrete-time domain. Therefore, matrices A and B were discretized using well-known methods [1]. Afterward, several controller gains were obtained using appropriate nominal sampling periods for a given continuous-time pole location, i.e., forcing specific system dynamics.

Fig. 12 shows the pseudocode that each task should execute under SHARK. Each task has the list of nominal periods obtained offline via the *Nominals* algorithm. Each task uses the SHARK system call *getPeriod()* to obtain the period T_i resulting after the compression algorithm. Afterward, it executes the controller whose nominal period T_{ni} is the upper bound of the interval of two consecutive nominal periods that contain T_i.

7.4 Experimental Performance Rate Functions

In order to obtain the performance rate functions for each nominal sampling period, we established the following procedure: We gave two reference positions $r1$ and $r2$ for the pendulum cart, as shown in Fig. 10. For a given nominal period and for an execution period, the cart had to switch 50 times between these two references. Fig. 13 illustrates this procedure over six reference changes. For each reference change, we obtained the corresponding IAE. And, after obtaining the 50 IAEs, we averaged them. By doing this, we filtered false or misleading measurements, which are always prone to occur on experimental setups.

For each nominal period, we repeated this experiment for all execution periods within an appropriate range. Fig. 14 shows the experimental performance rate function we obtained by a nominal period of $30ms$, when the range of tested periods was from near $0ms$ to $50ms$ (with a step of $0.1ms$). As can be seen, the shape is very similar to the performance rate functions obtained by simulation (see Fig. 2 or Fig. 3).

Although the curve shown in Fig. 14 seems continuous, it is not. It is a collection of about 500 points. However, 50 points would be enough to capture the shape of the curve. This would mean testing the range of periods with a step of $1ms$.

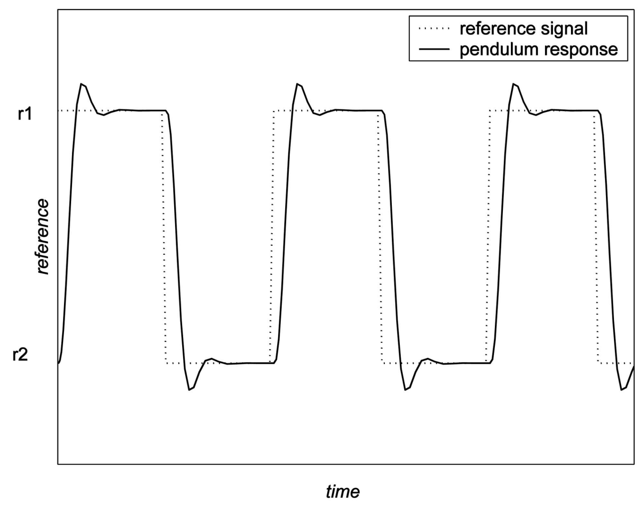

Fig. 13. Reference tracking.

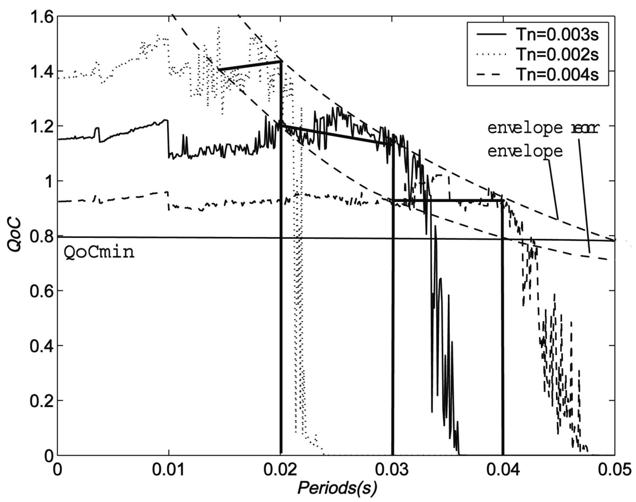

Fig. 15. Experimental sequence of controllers keeping a bounded error.

The similarity between the simulated and experimental performance rate functions allowed us to apply the procedure of finding the minimum number of controllers required to keep a desired QoC in the experimental setup.

7.5 Results

Our algorithm for determining the set of nominal periods was tested for a given value of ϵ in our experimental setup. The sequence of resulting controllers keeping a bounded error is illustrated in Fig. 15. As can be seen, only three nominal periods close to $0.04s$, $0.03s$, and $0.02s$ (approximately) are required to keep the desired QoC within a range of periods given by $[0.015, \ldots, 0.04]$. The thick-downstairs-like line, starting at $0.015s$ and finishing at $0.04s$, indicates the average QoC values that the task will achieve in the overload condition.

Including this set of nominal periods into the task code, we have been able to control the degradation occurring during an overload situation. To do so, we have overloaded the SHARK kernel by a synthetic task set in order to trigger the elastic procedure and cause the control task to change its period.

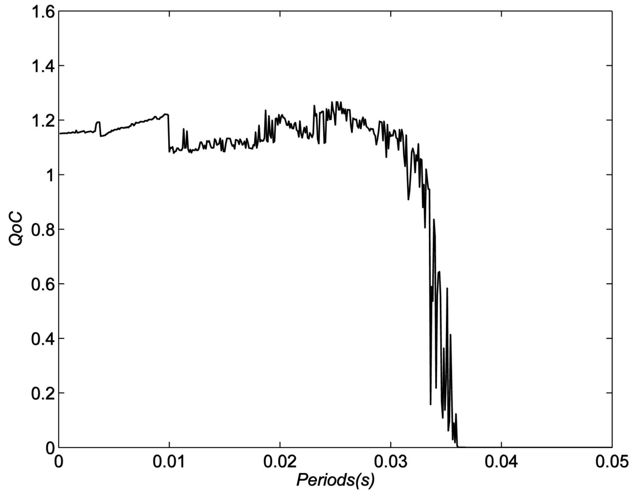

Fig. 14. Experimental performance rate function for $T_0 = 0.03s$.

8 Conclusions

The problem of managing the quality of control (QoC) in real-time control systems subject to overload conditions has been investigated. We assumed that load adjustments were achieved through period variations. To make an efficient use of the available resources while still guaranteeing the feasibility of the schedule, we did not restrict periods to vary on a limited set of predefined values, but allowed them to change continuously, making sure to switch to a proper controller to keep the QoC within a desired range.

By analyzing the performance characteristics of a controller running at a rate different than its nominal one, we proposed an approach that allows the user to design the minimum number of controllers needed to guarantee a desired performance in a set of admissible rates. The method allows the application designer to specify an error with respect to the ideal control performance of a perfectly tuned controller and provides a criterion to balance such an error with control performance and memory requirements.

The effectiveness of the proposed approach has been verified through extensive simulations, as well as real experiments, carried out on an inverted pendulum. The experimental results confirmed the validity of the approach and provided quantitative evidence of the dependency of the specified error from the achieved control performance and the memory requirements.

As future work, we plan to further extend the proposed method so that controllers can be adapted not only to cope with overloads, but also to better conform with the control application dynamics, that is, to provide an integrated QoC management framework for the system and the application, as a whole. This could be achieved by dynamically tuning the elastic coefficients of control tasks according to changes occurring in the controlled plant.

ACKNOWLEDGMENTS

This work has been partially supported by IST-004527 ARTIST2 Network of Excellence on Embedded Systems Design.

REFERENCES

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

[2] T. Abdelzaher, E. Atkins, and K. Shin, "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control," *IEEE Trans. Computers,* vol. 49, no. 11, pp. 1170-1183, Nov. 2000.

[3] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli, "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems," *Proc. 11th IEEE Euromicro Conf. Real-Time Systems,* 1999.

[4] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1998.

[5] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic Scheduling for Flexible Workload Management," *IEEE Trans. Computers,* vol. 51, no. 3, pp. 289-302, Mar. 2002.

[6] G. Buttazzo and L. Abeni, "Adaptive Workload Management through Elastic Scheduling," *Real-Time Systems,* vol. 23, no. 1, pp. 7-24, July 2002.

[7] G. Buttazzo, M. Velasco, and P. Martí, "Managing Quality-of-Control Performance under Overload Conditions," *Proc. 16th Euromicro Conf. Real-Time Systems,* July 2004.

[8] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic Feedback Control," *Proc. 12th IEEE Euromicro Conf. Real-Time Systems,* pp. 121-128, June 2000.

[9] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen, "Feedback-Feedforward Scheduling of Control Tasks," *Real-Time Systems,* vol. 23, nos. 1-2, pp. 25-53, July 2002.

[10] A. Cervin and J. Eker, "The Control Server: A Computational Model for Real-Time Control Tasks," *Proc. IEEE 15th Euromicro Conf. Real-Time Systems,* pp. 113-120, July 2003.

[11] R.C. Dorf and R.H. Bishop, *Modern Control Systems,* seventh ed. Addison-Wesley, 1995.

[12] J. Eker and A. Cervin, "Matlab Toolbox for Realtime and Control Systems Codesign," *Proc. Sixth Int'l Conf. Real-Time Computing Systems and Applications,* Dec. 1999.

[13] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo, "A New Kernel Approach for Modular Real-TIme Systems Development," *Proc. 13th Euromicro Conf. Real-Time Systems,* June 2001.

[14] Inteco Ltd., Intelligent Technology for Control, http://www.inteco.cc.pl/, 2006.

[15] T.-W. Kuo and A.K. Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proc. 12th IEEE Real-Time Systems Symp.,* Dec. 1991.

[16] C. Lee, R. Rajkumar, and C. Mercer, "Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach," *Proc. Multimedia Japan '96,* Apr. 1996.

[17] F. Lian, J. Moyne, and D. Tilbury, "Network Design Consideration for Distributed Control Systems," *IEEE Trans. Control Systems Technology,* vol. 10, no. 2, pp. 297-307, Mar. 2002.

[18] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 40-61, 1973.

[19] P. Martí, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Jitter Compensation for Real-Time Control Systems," *Proc. 22nd IEEE Real-Time System Symp.,* Dec. 2001.

[20] P. Martí, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Improving Quality-of-Control Using Flexible Timing Constraints: Metric and Scheduling Issues," *Proc. 23rd IEEE Real-Time System Symp.,* Dec. 2002.

[21] P. Martí, C. Lin, S. Brandt, M. Velasco, and J.M. Fuertes, "Optimal State Feedback Based Resource Allocation for Resource-Constrained Control Tasks," *Proc. 25th IEEE Real-Time Systems Symp.,* Dec. 2004.

[22] M. Marinoni and G. Buttazzo, "Adaptive DVS Management through Elastic Scheduling," *Proc. 10th IEEE Int'l Conf. Emerging Technologies and Factory Automation (ETFA '05),* Sept. 2005.

[23] L. Palopoli, L. Abeni, and G. Buttazzo, "Real-Time Control System Analysis: An Integrated Approach," *Proc. 21st IEEE Real-Time Systems Symp.,* Dec. 2000.

[24] P. Pedreiras and L. Almeida, "The Flexible Time-Triggered (FTT) Paradigm: an Approach to QoS Management in Distributed Real-Time Systems," *Proc. IEEE Int'l Parallel and Distributed Processing Symp.,* Apr. 2003.

[25] D. Seto, J.P. Lehoczky, L. Sha, and K. Shin, "On Task Schedulability in Real-Time Control Systems," *Proc. 17th IEEE Real-Time Systems Symp.,* pp. 13-21, Dec. 1996.

[26] K. Shin and C. Meissner, "Adaptation of Control System Performance by Task Reallocation and Period Modification," *Proc. 11th IEEE Euromicro Conf. Real-Time Systems,* pp. 29-36, June 1999.

[27] M. Velasco, P. Martí, R. Castañé, R. Villá, and J.M. Fuertes, "Key Aspects for Co-Designing Real-Time and Control Systems," *Proc. Int'l Workshop Real-Time and Control (RTC '05),* July 2005.

Giorgio Buttazzo graduated in electronic engineering from the University of Pisa in 1985 and received the master's degree in computer science from the University of Pennsylvania in 1987 and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna of Pisa in 1991. He is a full professor of computer engineering at the Scuola Superiore Sant'Anna of Pisa. From 1987 to 1988, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory at the University of Pennsylvania, Philadelphia. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He has authored six books on real-time systems and more than 200 papers in the field of real-time systems, robotics, and neural networks. Professor Buttazzo is a senior member of the IEEE.

Manel Velasco graduated in maritime engineering in 1999 and received the PhD degree in automatic control in 2006, both from the Technical University of Catalonia, Barcelona, Spain. Since 2002, he has been an assistant professor in the Department of Automatic Control at the Technical University of Catalonia. He has been involved in research on artificial intelligence from 1999 to 2002 and, since 2000, on the impact of real-time systems on control systems. His research interests include artificial intelligence, real-time control systems, and collaborative control systems, especially on redundant controllers and multiple controllers with self-interacting systems.

Pau Martí received the degree in computer science and the PhD degree in automatic control from the Technical University of Catalonia, Barcelona, Spain, in 1996 and 2002, respectively. Since 1996, he has been an assistant professor in the Department of Automatic Control at the Technical University of Catalonia. From 1999 to 2002, he spent several months as a visiting student at Malardalen University, Vasteras, Sweden. From 2003 to 2004, he held a research fellow appointment in the Computer Science Department at the University of California at Santa Cruz, involved in research on soft real-time systems. His research interests are real-time control systems, with emphasis on the interaction and integration of control systems, real-time systems, and communication systems. He is a member of the IEEE.