Managing Quality-of-Control Performance Under Overload Conditions[*]

Giorgio Buttazzo
University of Pavia
Pavia, Italy
buttazzo@unipv.it

Manel Velasco
Tech. Univ. of Catalonia
Barcelona, Spain
manel.velasco@upc.es

Pau Marti
University of California
Santa Cruz, USA
pmarti@cs.ucsc.edu

Gerhard Fohler
Malardalen University
Vasteras, Sweden
gerhard.fohler@mdh.se

Abstract

A common method for dealing with overload conditions in periodic task systems is to reduce the load by enlarging activation periods. When a periodic task implements 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 off line 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 off line, 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.

1 Introduction

The workload of a complex control system can change significantly during system lifetime for various reasons: new tasks can be activated or terminated in specific circumstances, computation time can vary due to the non deterministic behavior of some low-level mechanisms (e.g., caching, pre-fetching, and interrupts), or timing constraints can be changed by the application to react to variations in the environment. If the load increases above a certain limit, one or more tasks could miss their deadlines, causing an unpredictable performance degradation in the system. 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.

For example, Kuo and Mok [10] presented a load scaling technique to gracefully degrade the workload of a system by adjusting the periods of processes Other policies to dynamically adjust tasks' rates in overload conditions have been proposed under static [11] and dynamic priority assignments [2]. Buttazzo et al. [4] 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 scheduling was also extended to be used with tasks with unknown and variable computation times for adaptive QoS management [5], and, has been used on networks to work with discrete periods [16].

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.

The problem of integrating real-time schedulabilty analysis with control system design was recently investigated by several authors [17, 1, 6, 14, 7, 15], however the consequence of a rate change in terms of control performance was not always taken into account.

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 control law is simple (e.g., a PID regulator), the algorithm can be changed on line when the periods are recomputed. 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 off line 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 peri-

[*]This work has been partially supported by Departament d'Universitats, Recerca i Societat de la Informaci de la Generalitat de Catalunya and by Spanish Ministerio de Ciencia y Tecnologia Project ref. DPI2002-01621.

ods strongly limits the possibility of efficiently reacting to an overload condition, causing a waste of resources.

In this paper, we present a method for reducing the number of controllers to be designed off line, while still guaranteeing a given control performance with a continuous period adaptation. The proposed approach can be easily integrated with the elastic scheduling theory to promptly react to overload conditions. Extensive simulations have been performed on an inverted pendulum to verify the effectiveness of the proposed approach.

2 Evaluating control performance

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. Performance criteria (also called performance indexes or cost functions) are mainly based on measures of the system error. Traditional criteria (reported in control text-books, e.g. [9]), 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 case, system error and control energy are multiplied by a weight to balance their relative importance. For example, in [12] and [8] the performance criterion is only based on the system error, whereas in [18] and [17] 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 the available norms, IAE has been selected because it gives the best curves to conceptually illustrate the problem we are addressing.

It is defined as follows:

$$IAE = \int_0^\infty |e(t)|\, dt \qquad (1)$$

where $e(t)$ is the system error and $|.|$ denotes an appropriate norm. The integral upper limit of equation (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:

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

where $y(t)$ is the system output. 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.

2.1 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 system equations of a continuous-time system.

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

$$y(t) = Cx(t). \qquad (4)$$

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*

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

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 system (3) and (4), obtained with a discretization period period T and output at time $k + 1$ given by:

$$x(kT + T) = \Phi(T)x(kT) + \Gamma(T)u(kT) \qquad (6)$$

$$y(kT + T) = Cx(kT + T). \qquad (7)$$

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 equation (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 equation (12) and reorganizing the resulting expression, we obtain:

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

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

2.2 Quality-of-Control performance index

As done in [14], instead of using the value given by the $IAE(T_0, T)$ index, we use its inverse, as given by equation (15), thus working with a measures 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.

3 Performance-rate functions

Using the performance index defined by equation (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.

3.1 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).

Figure 1 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.

The curve shown in Figure 1, which relates control performance and controller execution rate, is referred to as *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.

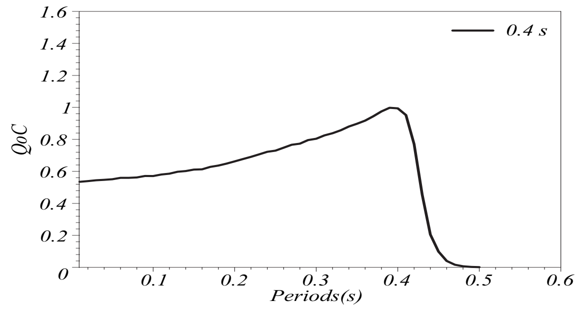

Figure 1. Performance-rate function of a controller designed to work at a nominal period $T_0 = 0.4s$.

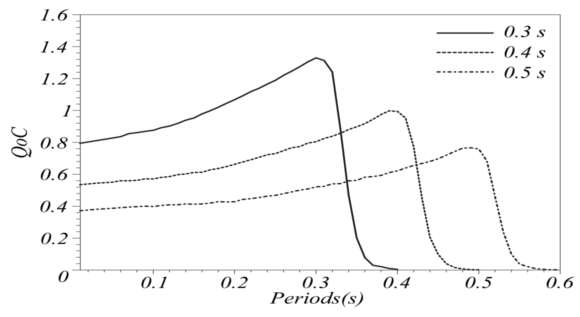

Figure 2. Performance-rate functions of a controller designed to work at three different nominal periods.

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

$$PRF(T_0, t, T_{min}, T_{max}) = \{QoC(T_0, t) \,|\, t \in [T_{min}, \ldots, T_{max}]\}. \quad (16)$$

Using equation (16), the performance-rate function illustrated in Figure 1 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, Figure 2 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 to fall down. This was an expected behavior, because the system was controlled less frequently that 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. In this case, the control does not fail but looses accuracy. The best control is achieved when the execution period is the same as the nominal one (that is, when $t = T_0$ in equation (16)). Considering the previous observations, we will allow controllers to execute with periods shorter than the nominal one, trading graceful performance degradation with controller flexibility, as further explained in Section 3.2. But, in order to prevent instability, we will not allow controllers to execute with periods longer than the nominal one.

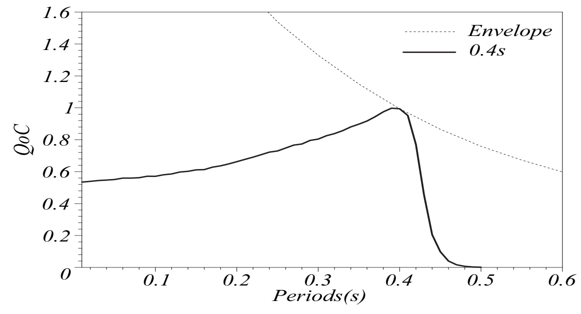

Figure 3. Performance-rate function of a controller tuned to work with a nominal period $T_0 = 0.4s$ against the ideal controller tuned at any rate.

A controller running with a period that is different than the nominal one is called a *non-tuned controller*. To better evaluate the error produced by a non-tuned controller at any running period, it is worth to compare the control performance index with the one achieved by the corresponding tuned controller. Figure 3 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 envelop 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 equation (16), where $t = T_0 \; \forall t \in [T_{min}, \ldots, T_{max}]$). Therefore, the performance-rate function of an ideal controller can be formulated as follows:

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

Note that in equation (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.

3.2 Bounding the error during overloads

To cope with overload conditions, tasks must change rates. However, if we want to have always 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 [13], this can be achieved either by re-designing the controller at runtime for each new execution period, or by accessing it from a table of pre-designed controllers. If re-designing the controller is too expensive (in terms of computational overhead) to be done at runtime, a number of controllers must be designed off line 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 [13] 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 [4] or other similar methods [2]), 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 [16, 8] proposed an adjustment technique to slightly resize some periods after a discrete load compression, to reach a workload closer to the desired one.

Actually, there is no need to work with discrete periods. Instead of forcing the control tasks to work at a predefined rates, one could let them work at a rate resulting from the compression algorithm, but 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 vs. QoC.

To bound the error produced by a non-tuned controller (with respect to the ideal one tuned at any period), we have to switch controller as soon as the control performance decreases below a given bound ϵ_{max}. To reduce the number of controllers that have to be designed off line to keep the error below a given value, we can apply the following approach (see Figure 4 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 *Envelop* curve in Figure 4).

2. The maximum error ϵ_{max} allows the user to derive the minimum performance curve (the *Envelop Error* curve in Figure 4), which is given by *Envelop*$-\epsilon_{max}\cdot$*Envelop*.

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 *Envelop Error* curve); and so on for the other periods, while they are not smaller than T_{min}.

Figure 4 illustrates an example showing the sequence of performance-rate functions that bound the error with respect to the tuned controller (the *Envelop* curve) to a value equal to ϵ_{max} (the *Envelop 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 non-tuned controller executing within $[T_{min}, \ldots, T_{max}] = [0.25s, \ldots, 0.54s]$, where $\epsilon_{max} = 0.3$ and $QoC_{min} = 0.6$. T_{min} is the default task period ($0.25s$) and T_{max} is obtained from QoC_{min}, which is the minimum quality of service level specified by the user. 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.

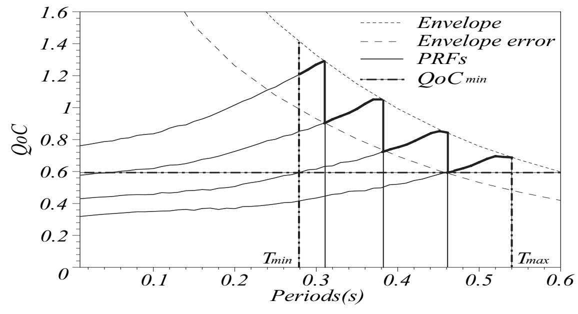

Figure 4. Sequence of controllers bounding the performance error to a given value.

Note also that ϵ_{max} relates control performance to system resources in terms of memory requirements (number of controllers to be designed off line and stored for runtime access). If $\epsilon_{max} = 0$, (that is, the *Envelop Error* curve coincides with the *Envelop* curve), the number of controllers to be designed off line 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, less controllers need to be designed off line (meaning less storage memory), at the expenses of reducing the average achievable QoC.

3.3 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 envelop curve (ϵ_{max}).

```
Nominals ($T_{min}$, $QoC_{min}$, $\epsilon_{max}$)
{
  $T := t \leftarrow (QoC(t,t) - \epsilon_{max} \cdot QoC(t,t) = QoC_{min})$;
  / Find $t$ such that $QoC_{min}$ intersects envelope error curve /
  $T_{max} := t \leftarrow (PRF(t,T,T,\infty) = QoC_{min})$;
  / Find a nominal period $t$ - that will be $T_{max}$ - whose
  performance rate function at time $t$ is equal to $QoC_{min}$ /
  $Nominals := \{T_{max}\}$;
  $t := T_{max}$;
  / Starting with nominal period $T_{max}$, find the rest of nominal
  periods, which are given by times $t > T_{min}$ at which their
  performance rate functions intersect envelop error curve /
  While $(t > T_{min})$
        $t--$;
        If $QoC(t,t) - PRF(T_{max}, t, T_{min}, T_{max}) =$
        $\epsilon_{max} \cdot QoC(t,t)$
              $Nominals := Nominals \cup \{t\}$;
              $T_{max} := t$;
        End
  End
  Return ($Nominals$);
}
```

4 Overload management policy

The algorithm presented in Section 3.3 can be used to derive the nominal rates of the performance-rate functions that allow to keep 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 in memory for a possible runtime adaptation during transient overload conditions.

Initially, the system starts executing each controller at a rate (T_{min}) that can be guaranteed by the scheduling algorithm adopted by the system. 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 [3, 4], 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 easily be determined on line 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 to period variations.

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

If the period T_i resulting after the compression algorithm falls in the interval given by two consecutive nominal periods $[T_k, T_{k+1}]$ in the performance-rate graph, the system 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 $[T_{min}, \ldots, T_{max}]$, the QoC degradation will be bounded; that is, the performance error with respect to the ideal tuned controller will be less than ϵ_{max}.

5 Experimental evaluation

In this section we evaluate the method we have presented to control the QoS 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 3 performed on an inverted pendulum. Here, we compare three different controller execution strategies, whose performance curves are illustrated in Figure 5 (which has been extracted from Figure 4):

a) **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 3000) with a clock granularity of $1\mu s$ (or $0.1\mu s$). The performance-rate function of this task corresponds to the *Envelop* curve illustrated in Figure 5.

b) **Adaptive task**. This case considers the set of four controllers, derived with the method presented in Section 3.2 and Section 3.3, that guarantees a maximum error $\epsilon_{max} = 0.3$. According to the overload management policy explained in Section 4, 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 Figure 5.

c) **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 Figure 5. 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 3.1). The only case 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 it can be seen from Figure 5, 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 $0.54s$.

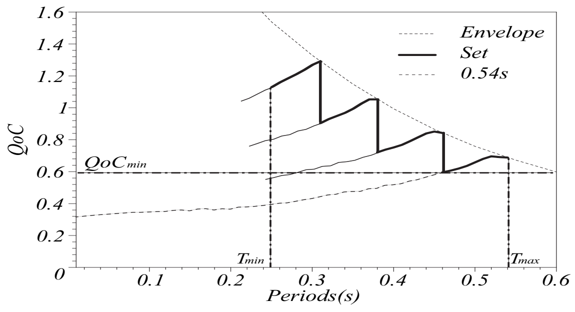

Figure 5. Performance-rate functions for the Optimal task (*Envelop*), Adaptive task (*Set*) and Static task (*0.54s*).

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

Table 1. Experimental evaluation summary.

Table 5 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% 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% of the optimal value.

6 Conclusions

In this paper we addressed the problem of controlling the quality of control (QoC) in a dynamic real-time system subject to overload conditions, where load adjustment is performed 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 experiments carried out on an inverted pendulum. The experimental results confirmed the validity of the approach and provided a quantitative evidence of the dependency of the specified error from the achieved control performance and the memory requirements.

As a 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.

References

[1] T. Abdelzaher, E. Atkins, and K. Shin, "QoS Negotiation in Real-Time Systems and Its Applications to Automated Flight Control," *Proc. of the IEEE Real-Time Technology and Applications Symposium*, June 1997.

[2] G. Beccari, S. Caselli, M. Reggiani, F. Zanichelli, "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems," *IEEE Proc. of the 11th Euromicro Conference on Real-Time Systems*, York, UK, 1999.

[3] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control", *Proc. of the IEEE Real-Time Systems Symposium*, December 1998.

[4] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic Scheduling for Flexible Workload Management," *IEEE Transactions on Computers*, Vol. 51, No. 3, pp. 289-302, March 2002.

[5] G. Buttazzo and L. Abeni, "Adaptive Workload Management through Elastic Scheduling," Real-Time Systems, Vol. 23, No. 1, pp. 7-24, July 2002.

[6] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic Feedback Control," IEEE Proceedings of the 12th Euromicro Conference on Real-Time Systems, pp. 121-128, June 2000.

[7] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. rzn, "Feedback-Feedforward Scheduling of Control Tasks," Real-Time Systems, 23:1, 2002.

[8] A. Cervin, and J. Eker, "The Control Server: A Computational Model for Real-Time Control Tasks," IEEE Proceedings of the 15th Euromicro Conference on Real-Time Systems, pp. 113-120, Porto, Portugal, July 2003.

[9] R.C. Dorf, and R.H. Bishop, *Modern Control Systems. Seventh Edition*, Addison-Wesley, 1995.

[10] T.-W. Kuo and A. K, Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proceedings of the 12th IEEE Real-Time Systems Symposium*, December 1991.

[11] C. Lee, R. Rajkumar, and C. Mercer, "Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach," *Proceedings of Multimedia Japan 96*, April 1996.

[12] F. Lian, J. Moyne, and D. Tilbury, " Network Design Consideration for Distributed Control Systems," IEEE Transactions on Control Systems Technology, Vol.10, No.2, March 2002

[13] P. Marti, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Jitter Compensation for Real-time Control Systems," Proceedings of the 22rd IEEE Real-Time System Symposium, London, UK, December 2001.

[14] P. Marti, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Improving Quality-of-Control using Flexible Timing Constraints: Metric and Scheduling Issues," Proc. of the 23rd IEEE Real-Time System Symposium, Austin, TX, USA, December 2002.

[15] L. Palopoli, L. Abeni, and G. Buttazzo, "Real-Time control system analysis: an integrated approach," Proc. of the 21st IEEE Real-Time Systems Symposium, Orlando, Florida, December 2000.

[16] P. Pedreiras, and L. Almeida "The Flexible Time-Triggered (FTT) Paradigm: an Approach to QoS Management in Distributed Real-Time Systems,"International Parallel and Distributed Processing Symposium, April, 2003.

[17] D. Seto, J. P. Lehoczky, L. Sha, and K. Shin, "On task schedulability in real-time control systems," Proc. of the 17th IEEE Real-Time Systems Symposium, Washington, DC, pp. 13-21, December 1996.

[18] K. Shin and C. Meissner, "Adaptation of control system performance by task reallocation and period modification," IEEE Proc. of the 11th Euromicro Conference on Real-Time Systems, pp. 29-36, June 1999.