# Optimization of Control Systems Performance via Soft Real-Time Quality-of-Service Management Techniques

## Technical Report (UCSC-CRL-04-05)

Pau Marti, Caixue Lin, Scott A. Brandt, Manel Velasco and Josep M. Fuertes

***Abstract*— Quality-of-Service (QoS) techniques improve system performance by allocating resources to the applications that will provide the most benefit. In this paper we describe how to employ QoS techniques to improve the performance of concurrently executing control applications. We present results from a real implementation that show that these techniques can reduce control system error by more than 20% compared to a non-adaptive control system on the same sequence of randomly generated control system perturbations.**

**Keywords:** real-time systems, control theory, optimization, quality of service

## I. Introduction

Control systems performance optimization is a discipline that belongs *by default* to optimal control theory. Optimal control deals with the design of admissible control laws, satisfying constraints on the states and inputs, that optimize a criterion (in the form of a performance index), for a given process. As we show in this paper, if we *zoom out* from the process level to the system level, optimization of control systems performance becomes a discipline that also belongs to real-time systems theory, specifically to Quality-of-Service (QoS).

The system we consider is a multitasking real-time system where a set of controllers share a CPU. Due to resource limitations, the controllers cannot all execute at the same time with the highest rate, i.e., providing the best possible control performance equivalent to what they would provide if they were running alone on the CPU. In order to optimize the performance delivered by this set of controllers we apply soft and adaptive real-time QoS techniques to dynamically reallocate resources to the controllers that need them the most, i.e., those with the greatest error. If we consider system outputs to be the performance provided by the set of controllers, the main goal of the QoS framework becomes the optimization of control systems performance.

Just as the high computational requirements posed by optimal control problems sometimes limit their applicability, QoS optimization can sometimes be prohibitively expensive or even computationally infeasible. However, in this case we find that QoS optimization can feasibly be accomplished in real-time with significant control system performance improvement.

We have implemented our QoS management system in the Rate-Based Earliest Deadline (RBED) integrated real-time system [1]. The QoS system bases its resource allocation decisions on *feedback* about the application performance. In this case, the feedback is a direct specification of control system performance. Apart from monitoring resources (as QoS managers usually do), it monitors whether each controlled system is affected by perturbations in order to reassign resources accordingly, adapting resources to the dynamically determined application *real* needs.

Our control applications are capable of running at different rates with time-varying resource allocations (as suggested in [2] and [3]), providing greater benefit when given more resources. The performance delivered by these classically designed controllers is simply improved by the run-time resource allocator because the controllers get more resources exactly when they need it the most, i.e. when they are experiencing the greatest error.

We discuss our implementation of both the QoS system and the controllers and present experiments demonstrating both the feasibility and the benefits of this approach. We compare several different resource allocation policies including an optimal policy and characterize their performance in terms of performance of the control systems. Finally, we discuss *pros* and *cons* of the presented framework.

## II. Related Work

Optimization of control systems performance subject to resource constraints has been examined before. Seto et al. [4] optimized task frequencies at the design stage in order to minimize a control performance index defined over the task set. Rehbinder and Sanfridson [5] proposed an off-line scheduling method based on optimal control theory. None of the previous work has examined run-time adaptive QoS management for optimal control system performance, as we do.

Different approaches to runtime control performance and resource allocation optimization have also been examined before. Shin and Meissner [6] presented a resource allocation technique for multiprocessor systems where tasks are reallocated and periods are changed while taking into account control performance. Caccamo et al. [7] allowed tasks' computation times to range from average to worst case computation times and adjusted periods at runtime to

This work was supported in part by Intel Incorporation.

Pau Marti, Caixue Lin and Scott A. Brandt are with the Computer Science Department, University of California, Santa Cruz, CA 95064, USA. pmarti,lcx,scott@cs.ucsc.edu

Manel Velasco and Josep M. Fuertes are with the Automatic Control Department, Universitat Politècnica de Catalunya, Barcelona, Spain. manel.velasco,josep.m.fuertes@upc.es

optimize control performance and enhance schedulability using server approaches. Cervin et al. [8], proposed a scheduling architecture for real-time control tasks where the scheduler uses feedback from execution time measurements and feedforward from workload changes to adjust the sampling periods of the control tasks so that the combined performance of the controllers is optimized. However, none of the previous work uses feedback from the control application in order to reassign resources as we do.

The resource allocation policies for control tasks that we present in this paper, including the optimal, consider feedback in terms of changes in the controlled plants as suggested by Yepez et al. [9] and which we initially explored in the context of discrete QoS adaptation [10]. The approach we present resembles the model we presented previously [11], where resources and plant dynamics are jointly considered. However, no formal proof on the optimality of the model was given in that work, nor was that work implemented in a real system.

## III. QoS Management in a Soft Real-Time System

Quality-of-service management frameworks trade off application performance and computing resources. In unpredictable environments, run-time adaptation techniques are highly effective at optimizing system quality.

Two main approaches can be identified in adaptive QoS management techniques. In the first approach, the adaptation is performed by each application itself. Each application adapts its computational demands based on the available resources and/or the required service level [12]. An example of this technique for control applications is [13].

In the second approach, the adaptation is (discretely [12] or continuously [14]) performed by a system (or middleware) resource manager that monitors and controls the utilization of resources by each application. The amount of resources assigned to each application is usually a function of the current workload and the relative benefit provided by each application [12], [1]. It is generally true that the higher the output quality, the larger the resource needs. An example of this technique for control applications is [15]. In this paper we extend these adaptive techniques by enabling the resource manager to assign resources as a function of the state of each control loop, as we explain in section IV.

Our system is implemented in the RBED integrated real-time system [1], in which allocation and dispatching are managed separately. RBED allocates resources to processes as a percentage of the CPU such that the total allocated is less than or equal to a desired utilization factor. It then schedules all processes with the Earliest Deadline First (EDF) algorithm [16]. RBED dynamically changes allocated resources and application periods without violating EDF constraints, guaranteeing that tasks never misses their assigned deadlines. A previous implementation of RBED provided support for discrete QoS levels, allowing applications to switch among multiple discrete algorithms depending on their needs and the system load to meet the application goals. We have extended this model to include continuous QoS support by allowing each application to specify a function that relates its resource usage to the benefit it provides. In previous implementations, the benefit specification of the applications has been static. In our current implementation, our resource allocator re-scales the application benefits according to the application dynamics and analyzes how to optimize the global benefit to determine its allocation strategy.

We can formally define the objective of our QoS framework as finding the solution of the following constrained optimization problem

$$\begin{aligned} &\text{maximize} && gb = g(s_i(t)\, f_i(u_i)) \\ &\text{subject to} && \sum_{i=1}^{n} u_i \leq U_d \end{aligned} \tag{1}$$

where $u_i$ is the amount of resources to be assigned to each application, $f_i$ is the continuous or discrete function that relates quality (or benefit) for a given amount of resources for each application (or task), $s_i(t)$ is the run-time re-scaling factor depending on the current time $t$, $g$ is the function that links all application benefits, $gb$ stands for *global benefit*, and $U_d$ is the desired global resource utilization, which constraints the resource allocation problem.

For any time $t$, the resource allocation problem is reduced to find the absolute maximum $\vec{u} = [u_1, u_2, \ldots, u_n]$ of the function $g$ restricted to the utilization feasibility constraint. The absolute maximum $\vec{u}$ may lie either in the interior, on the boundary or at the extreme points of the feasibility set. A generic algorithm that would find the solution can be summarized into four steps (see further details in [17]).

*Step 1:* Search for local relative maximums in the interior of the feasibility set by solving the following equation set

$$\frac{\partial g}{\partial u_1} = 0,\ \frac{\partial g}{\partial u_2} = 0,\ \ldots,\ \frac{\partial g}{\partial u_n} = 0$$

and keep those $\vec{u}$ that conforming with the restriction $u_1 + u_2 + \ldots + u_n \leq U_d$ maximize $gb$.

*Step 2:* Search for local relative maximums in the boundary of the feasibility set by solving the following equations

$$\frac{\partial g(U_d - u_2 - u_3 - \ldots - u_n, u_2, u_3, \ldots, u_n)}{\partial u_i} = 0,\ i \leq n, i \neq 1$$

$$\frac{\partial g(u_1, U_d - u_1 - u_3 - \ldots - u_n, u_3, \ldots, u_n)}{\partial u_i} = 0,\ i \leq n, i \neq 2$$

$$\vdots$$

$$\frac{\partial g(u_1, u_2, \ldots, U_d - u_1 - u_2 - \ldots - u_{n-1})}{\partial u_i} = 0,\ i \leq n, i \neq n$$

and keep those $\vec{u}$ that maximize $gb$.

*Step 3*: Search for local relative maximums in the feasibility set extremes. That is to evaluate

$$g(u_1, 0, \ldots, 0),\ g(0, u_2, \ldots, 0),\ \ldots,\ g(0, 0, \ldots, u_n)$$

and keep those $\vec{u}$ that maximize $gb$.

*Step 4:* Choose a $\vec{u}$ among the ones obtained in *Step 1*, *2* and *3* that maximizes $gb$.

The discrete QoS optimization problem is in general NP-complete [18]. It is not generally possible to calculate an optimal resource allocation that maximizes global benefit. QoS levels managed with heuristic resource allocation algorithms that come close to the optimal solution are thus often the best alternative. For continuous QoS support, depending on functions $f_i$ and $g$, and depending on the re-scaling policy $s_i$, solving the optimization problem may not be feasible for an on-line real-time resource manager. If it is too expensive, again, heuristic solutions may be considered.

## IV. Exploiting QoS Techniques for Control Systems

In order to apply the RBED soft QoS management to control applications, we need to specify a) the controlled processes, b) the type of controllers we consider (which must be capable of running at several rates), c) the static benefits, which relate resources and control performance, and d) the run-time information used as feedback measure from control applications that re-scales the static benefits. In the following subsections we discuss each of these.

### A. Controlled processes

In our implementation we simulate a collection of inverted pendulums. The linear time-invariant state space model we used for each inverted pendulum mounted on a cart is given by

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)\cdot g}{M\cdot l} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-m\cdot g}{M} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-1}{M\cdot l} \\ 0 \\ \frac{1}{M} \end{bmatrix} u(t)$$

where $\theta$ is the pendulum angle, $\omega$ is the angular velocity, $x$ is the cart position and $v$ its velocity. For the simulation, we customized all pendulums as follows: mass of the cart $M = 2kg$, mass of the pendulum $m = 0.1kg$, length of the pendulum stick $l = 0.5m$ and gravity $g = 9.81m/s^2$.

We have defined all of the controlled processes to be the same because it simplifies the performance analysis. However, different processes would not materially affect the results.

### B. Adaptive Controllers

In order to switch (continuously or discretely) among QoS levels, control applications have to implement *adaptive* control algorithms according to different resources requirements. To do so, we design controllers using classical design procedures but allow them to execute the same control law at different rates.

Given the state space representation of each discrete-time system with period $h$ (2),

$$x(kh+h) = \Phi(h)x(kh) + \Gamma(h)u(kh) \tag{2}$$

and taking into account the closed-loop requirements, we close each loop using state feedback, where the gain matrix L (given by (3)) can be obtained by classic design methods such as pole placement or optimization approach (see for example [19].

$$u(kh) = -L(h)x(kh) \tag{3}$$

At the end, we have for each process a control algorithm that depends on a sampling period $h$. We specify a range of periods for which the closed loop requirements are met and allow the controller to execute with a run time period that belongs to the specified ones (for full details, see [3]), adapting the gain accordingly. System stability is analyzed using [20].

For our inverted pendulum, the type of responses that each controller (designed using simply pole placement) obtains for some of the allowed rates (specified from 0.06s to 0.1s) can be seen in Figure 1.

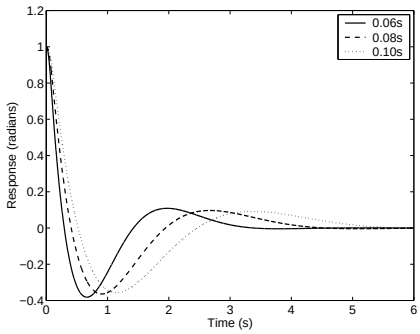
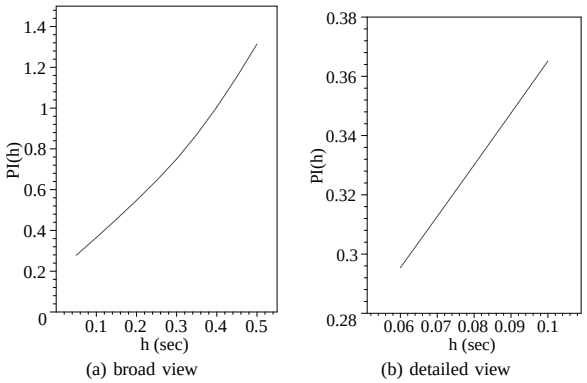



Fig. 1. Responses of Controllers

### C. Control performance vs. resources

In the QoS framework we need to define the static benefits, that is, the set of functions $f_i$ (in (1)) that relate control performance and resources. Looking at control systems, linear or quadratic performance indexes are traditional tools for the analysis and design. Therefore, we are interested in the relation between these indexes and resources.

For example, if for the inverted pendulum example we pick as a performance index $P_I$ the integral of the absolute value of angle (treated as a error because we would like it to be zero), and plot its value for a broad range of sampling rates, $P_I(h)$ (from $h = 0.05 \ldots 0.5$ sec), we obtain the *almost* linear relation shown in Figure 2(a). In fact, as outlined in [8], the relation between control performance (when measured using standard quadratic or linear performance index) and a range of allowed sampling periods can be approximated by a linear relationship. Note that this approximation is more or less accurate depending on the range of allowed sampling periods (within the broad range shown in Figure 2(a)). Figure 2(b) shows the relation between the allowed sampling periods for each controller in our setup (from $h = 0.06 \ldots 0.1$ sec) versus control performance. The relationship is linear.



(a) broad view (b) detailed view

Fig. 2. Sampling Period vs. Error (in terms of angle)

Therefore, in our QoS framework, for either the discrete or continuous approach, we will assume this linear relation. Note that this relation also establishes the basic optimizing operation of the QoS framework: *the shorter the sampling period (i.e., the higher the utilization), the better the control performance (i.e., the lower the error).*

### D. Measuring Instantaneous Control Performance

Our QoS framework differs from previous works because the resource allocator makes decisions based both on system load and on the dynamics of the applications. Recall that we are considering a real-time system where resources are limited and shared among applications in such a way that all of the controllers cannot simultaneously execute with the highest rate.

Looking at control systems, we can assume that if a controlled process is in equilibrium, having the corresponding controller executing at its highest rate would be a *waste* of resources. Therefore, given a set of allowed sampling periods, it is desirable to execute at a lower rate (with guaranteed minimum performance) if the controlled process is in equilibrium, thus saving resources that may be needed for other controllers. And if the process is perturbed and brought out of the equilibrium point, it is desirable to increase the rate of the controller whenever possible.

To do so, the run-time information that the resource allocator should consider at any given time in order to reassign resources is whether processes are affected by perturbations or not. And this can be detected by simply looking at each controlled system state, as we enable in our QoS framework. In fact, if we consider the equilibrium point to be zero for each closed-loop, the norm of each system state vector is a distance measure that evaluates how far each system is from its desired state, capturing all system dynamics. Therefore, more or less resources may be assigned at run-time to each controller taking into account the norm of each state vector (sometimes also called just *error*): the higher the norm (the higher the error), the more urgent a controller requires more resources.

## V. Performance Optimization of Control Systems

The generic optimization problem for QoS frameworks was formulated in (1). However, for control applications, and taking into account all of the discussions of section IV, the optimization problem can be highly simplified.

Assuming that each controller is independent in the sense of being in charge of controlling a plant, the function $g(.)$ that links all of the benefits can be considered as the sum (that may be weighted) of all individual benefits obtained by each controller (which are given by $f_i$). In addition, the dynamic relation between resources and application dynamics, given by $s_i(t)$, that re-scales the static benefits is given by the norm of each plant state vector, $s_i(t) = |x_i(t)|$.

Therefore, given a set of $n$ controllers, we can rewrite the optimization problem as in (4)

$$\begin{aligned} \text{maximize} \quad & gb = \sum_{i=1}^{n} |x_i(t)|\, f_i(u_i) \\ \text{subject to} \quad & \sum_{i=1}^{n} u_i \leq U_d \end{aligned} \tag{4}$$

The complexity of solution of the problem stated in (4) depends on each function $f_i(u_i)$, due to the fact that *Step 1* and *2* in the algorithm presented in section III have been simplified to the following set of equations (because $g$ has turned into a sum), where $b_i = |x_i(t)|\, f_i(u_i)$:

$$\frac{\partial b_1}{\partial u_1} = 0,\ \frac{\partial b_2}{\partial u_2} = 0,\ \ldots,\ \frac{\partial b_n}{\partial u_n} = 0$$

$$\begin{aligned} \frac{\partial b_1(U_d - u_2 - u_3 - \ldots - u_n)}{\partial u_i} &= 0,\ i \leq n, i \neq 1 \\ \frac{\partial b_2(U_d - u_1 - u_3 - \ldots - u_n)}{\partial u_i} &= 0,\ i \leq n, i \neq 2 \\ &\vdots \\ \frac{\partial b_n(U_d - u_1 - u_2 - \ldots - u_{n-1})}{\partial u_i} &= 0,\ i \leq n, i \neq n \end{aligned}$$

If $b_i$ are polynomial functions of grade less than five, an analytical solution can be found, turning the solution into a feasible (in terms of computational complexity) algorithm for a run-time resource manager. Otherwise, numerical methods may be applied, resulting in too expensive algorithms.

If $b_i$ are linear (as we assume in our system, see section IV-C), the optimization problem becomes linear, and we will find the optimal solution in one of the extreme points of the hypersurface defined by $[u_1, u_2, \ldots, u_n, b_1 + b_2 + \ldots + b_n]$ and delimited by the projection given by the hyperplane $u_1 + u_2 + \ldots + u_n \leq U_d$. In terms of utilization factors, that means to assign all the available CPU (that is, $U_d$) to the control task with maximum $|x_i(t)|\, f_i(u_i)$. And if all the functions $f_i$ are the same, it is to assign all the available resources to the control task facing a biggest error.

Therefore, the solution can be found by performing a simple search (i.e., performing *Step 3*) because all equations in *Step 1* and *2* will not have solution.

Figure 3 illustrates the case for two control tasks, where $U_d = 0.8$, both with the same benefit function ($f_1(u_1) = u_1$ and $f_2(u_2) = u_2$) but $controller_1$ at time $t$ facing a bigger error ($|x_1(t)| = 4$) than $controller_2$ ($|x_2(t)| = 1$). As it can be seen, the maximum benefit considering the schedulability constraints is found at $\vec{u} = [0.8, 0.0]$ (extreme point).

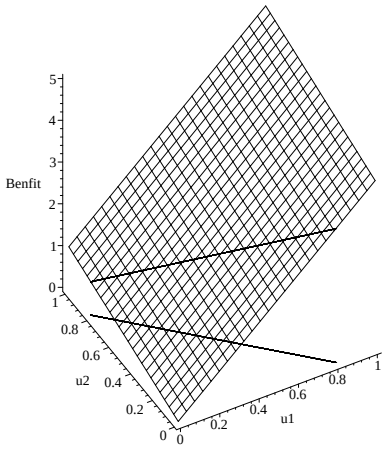



Fig. 3. Optimal solution.

## VI. Results

In order to demonstrate the benefits of using adaptive QoS techniques for control applications at the system level, we compare the performance of four different QoS managers: Static, Discrete-QoS, Proportional Continuous-QoS, Optimal Continuous-QoS. For each technique we performed experiments in which we ran three control tasks implementing the same control law. Perturbations were randomly generated and and the control performance was recorded. Each controller in charge of a simulated inverted pendulum can run at any sampling period ($h_i$) within 0.06s and 0.10s (if the QoS manager allows it) and has a constant worst-case execution time ($w_i$) of 0.027s.

### A. Implementation of the QoS allocation policies

As mentioned above, we implemented four different QoS managers. The first, Static, does no QoS optimization at all; all controllers always share the available resources equally and no dynamic resource allocation policy is used. RBED always reserves a minimum of 3% of the CPU capacity to the set of best-effort (general purpose) processes. Therefore, the available CPU for the three controllers is $U_d = 97\%$ and each controller is given $\frac{97}{3}\%$ of the CPU for the duration of the experiments. This policy implements traditional static control and is used as a baseline for examining the performance of the QoS-enabled control systems.

*1) Discrete-QoS:* We implemented a discrete level-based QoS for control tasks because it is the default QoS policy of RBED and because it is extremely simple. We use the QoS configuration defined in Table I for the three control tasks, where *%CPU* and *Benefit* are given by (5)

$$benefit = f_i(u_i) = \alpha_i u_i = \alpha_i \cdot \frac{w_i}{h_i} \tag{5}$$

where $\alpha_i$ (approximately 1.78 for all controllers) is the slope of the curve in in Figure 2(b), which corresponds to $f_i(u_i)$ in (4). That is, we defined three QoS levels (corresponding to three different sampling period $h_i$) for each controller.

TABLE I
QoS configuration for control tasks

| Number of QoS Levels: 3 | | | |
|---|---|---|---|
| Level | Benefit | % CPU | Period |
| 1 | 0.80 | 45% | 0.06 s |
| 2 | 0.60 | 33.7% | 0.08 s |
| 3 | 0.48 | 27% | 0.10 s |

With this configuration, if there are three control tasks in the system, then none of them will execute at their highest level (45%) since $(45\% + 2 * 27\%) = 99\% > 97\% = U_d$ (that is, the required CPU load would exceed what is available). A heuristic algorithm [10] chooses the level for each controller such as global benefit is maximized.

In order to determine for each error which QoS level to choose (and thus which period to assign to each controller), we assigned to each QoS level consecutive ranges of possible system error values. In this way, application sampling rate are changed only when system error moves from one range to another.

*2) Proportional Continuous-QoS:* For control applications characterized as we have explained, it is possible to apply continuous QoS support because we can specify a continuous function that relates control performance and resources. In fact, this function was already specified in (5) but now for any $h_i \in [0.06s \ldots 0.1s]$.

We first implemented a *fair* resource policy that we called Proportional Continuous-QoS. This policy assigns resources to each controller as a proportion of the controlled system error (in terms of the norm of its state vector) over the sum of all controlled system errors. In our implementation, the resource allocation algorithm is given by

$$u_i = \frac{|x_i(t)|}{\sum_{i=1}^{n} |x_i(t)|} \cdot U_d \tag{6}$$

Fairness comes from the fact that any controller whose controlled process is subject to a perturbation increases its utilization according to the its relative degree of error.

If (6) gives a utilization that results in a smaller sampling period than the lower bound of the allowed sampling period range, then, the controller will run at that lower bound. This mechanism allows us to guarantee a minimum sampling rate for all the controllers (which is $0.1s$ in our implementation).

*3) Optimal Continuous-QoS:* Finally we implemented an optimal allocation policy. As we discussed in section V, the optimal solution is to assign all the available resources to the controller with process facing the largest error (if all the closed-loops have the same $\alpha_i$ factor, which is our case), taking into account that the system must also guarantee a minimum rate for the remaining controllers. If we would have different $\alpha_i$ (meaning different type of controllers and/or different controlled processes), the resources would be assigned to the controller having the highest $|x_i(t)|\alpha_i$. This is summarized in (7)

$$u_i = \begin{cases} U_d - (n-1)\cdot u_{min}, & \textit{if } |x_i(t)|\alpha_i \text{ is maximum} \\ u_{min}, & \textit{otherwise} \end{cases} \quad (7)$$

where $n$ is the number of controllers and $u_{min}$ is the minimum utilization that is guaranteed for the remaining controllers.

### B. Workload Generation

For each of the QoS management policies, we show the results of running three controllers with same control law for 10 minutes and randomly generated perturbations for each inverted pendulum. A total of 176 perturbations of the same magnitude affected the pendulums (57, 57 and 62 respectively for each process). For the performance analysis and a better understanding of the results, we defined different scenarios in terms of whether perturbations overlapped or not between controllers. At any given time, it is important to know how many pendulums are simultaneously affected by perturbations. For a given perturbation, we defined the following three scenarios:

1) Non-Overlapping: At the time a perturbation affects a process, no other processes are affected by perturbations and none of them will be affected during at least the following 3.5s. That is, there is at least 3.5s between consecutive perturbations to any process. This 3.5s time interval relates to the settling time of the controlled process (see Figure 1). This scenario presents the most opportunity for performance improvement as only one process can benefit from running with a higher sampling frequency.
2) Partially Overlapping: At the time a perturbation affects a process, one or more other processes have been affected by perturbations in the last 3.5s but not within the last 0.1s. This scenario presents some opportunity for improvement, but less than the non-overlapping case because multiple applications are simultaneously in need of additional resources.
3) Fully Overlapping: At the time a perturbation affects a process, one or more other processes have been affected by perturbations within the last 0.1s. This scenario presents the least opportunity for performance improved performance as multiple applications have essentially equivalent need for additional resources at the same time.

If we look at Figure 4(a) (where that lower part of the y-axis represents the angle dynamics of each inverted pendulum affected by randomly generated perturbations), a non-overlapping perturbation affecting the pendulum controlled by task3 happens for example at about time 12s. Three partially overlapping perturbations happen between 0s and 10s for the three pendulums. And three fully overlapping perturbations occur between 40s and 50s for the three pendulums. Among the total of 127 perturbations of each experiment, 29 are not overlapped, 113 are partially overlapped and 34 are totally overlapped. Thus the randomly generated perturbations cover all of the three scenarios in a general way.

### C. Performance Results

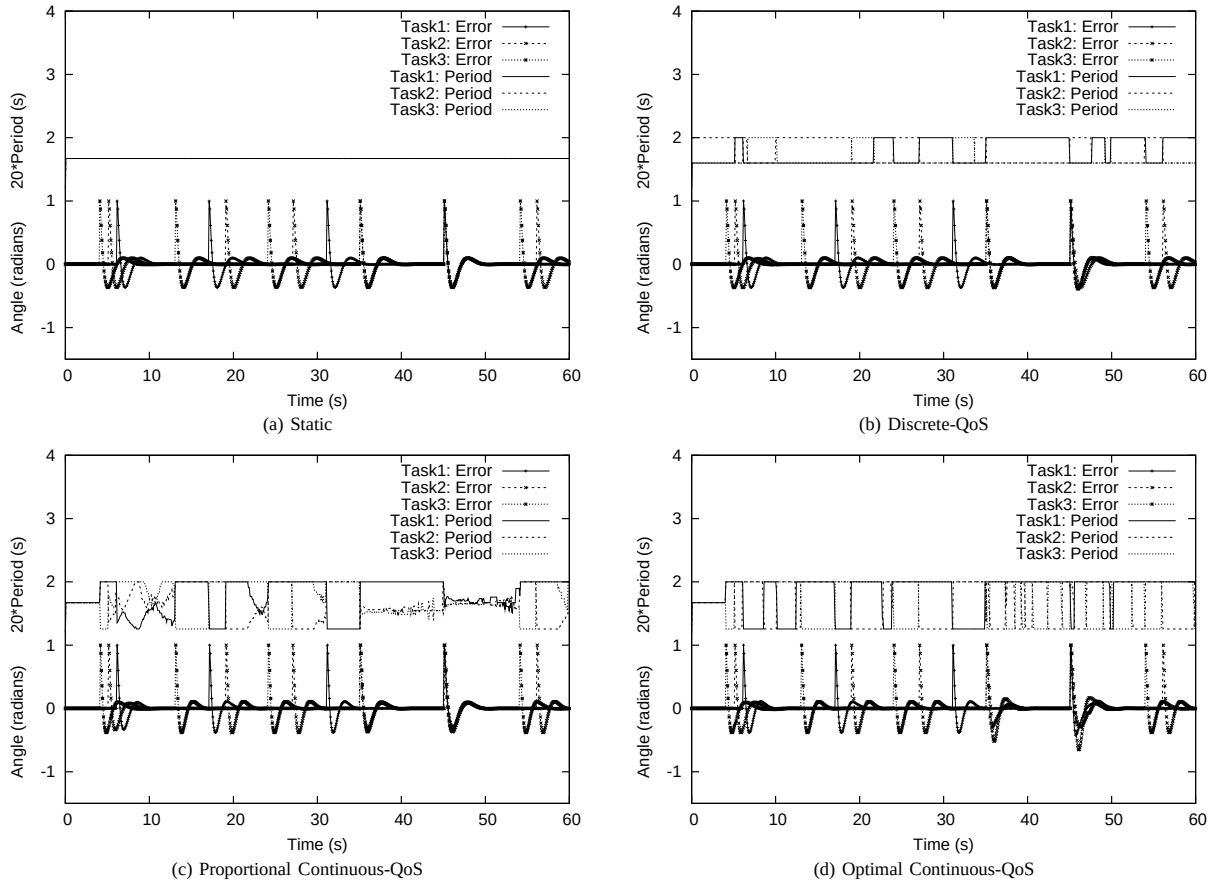
In this section we first detail which are the effects on the execution rate of each controller when using the different QoS management policies we have presented. Afterwards, we look at the control performance that can be achieved by these techniques.

*1) Execution rate patterns:* Figure 4(a)-4(d) shows the resulting performance of using the four different policies: Static, Discrete-QoS, Proportional Continuous-QoS and Optimal Continuous-QoS respectively. In order to look into the detailed progress of the three control tasks, we only show the first 60s of the long run, i.e. 600s. In each figure, the x-axis represents time and the y-axis represents both the angle (error) of each inverted pendulum and the corresponding controller sampling period (multiplied by 20 to normalize it to the same range). Note that the error ranges from $-1$ to 1 and the $20*$period ranges from 1.2 to 2 because the allowed period ranges from 0.06s to 0.1s. Because the variations of the angle dynamics are hard to appreciate in these figures, we will use Figure 5 for the control performance evaluation.

With the Static QoS policy (figure 4(a), every control task has the same sampling period ($h_i \approx 0.084$s, which comes from $h_i = w_i/u_i = \frac{0.027}{U_d/3}$) and does not change regardless what perturbation scenario occurs. With both Discrete-QoS and Continuous-QoS (Figure 4(b)-(d)), perturbations force the controllers to dynamically adapt their rates. The specific periods for each controller are determined by each QoS policy and according to the perturbation arrival pattern.

With the Discrete-QoS policy, not all the three tasks can run at their second QoS level (corresponding to $h_i = 0.08$) at the same time because of the limited available resources. Thus, the only possible case that can maximize the global benefit is the following: two of the control tasks with larger error running at their second QoS level and the third control task with smaller error running at its lowest QoS level (Figure 4(b)).

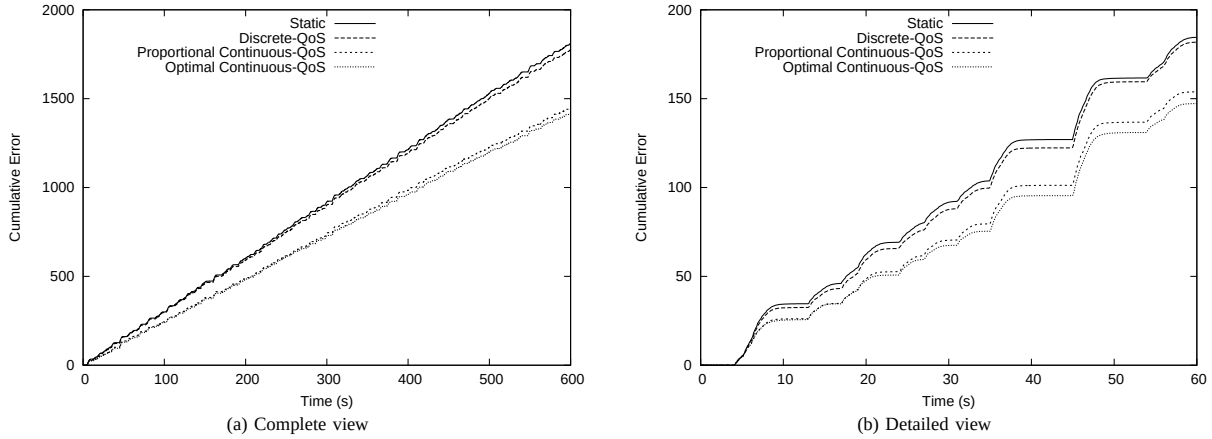
However, with the Proportional and Optimal Continuous-QoS policies, the period can be any value between the predefined range, depending on the dynamics of the three pendulums. It is important to note that the dynamic period change with the Discrete-QoS policy is less frequent than with both Continuous-QoS algorithms (Figures 4(c) and




(a) Static (b) Discrete-QoS

(c) Proportional Continuous-QoS (d) Optimal Continuous-QoS

Fig. 4. Sampling Period Vs Error (in terms of angle)




(a) Complete view (b) Detailed view

Fig. 5. Performance Evaluation in terms of Cumulative Error

(d)). This is due to the fact that with the Discrete-QoS policy, changes in the system dynamics may not result in changes in the QoS level, as explained in section VI-A.1.

Looking at the Continuous-QoS policies, the sampling rate adaptation also has different frequencies. With Optimal Continuous-QoS adaptation, the order of magnitude of the errors determines the period assignment: the one with largest error always runs at a highest rate ($h_i \approx 0.063$) provided the other two at least can run at their lowest rate ($h_i = 0.1$s). With Proportional Continuous-QoS adaptation, any variation in the errors causes a period adjustment among the three controllers, resulting in more variation than the Optimal algorithm.

The type of perturbations also affects the resource allocation. For non-overlapped perturbations (e.g. the perturbation at about 13s), the Proportional Continuous-QoS allocation

is similar to that of Optimal Continuous-QoS, while for for fully overlapped perturbations (e.g. the perturbations between 40s and 50s), it distributes the resources to the three controllers almost fairly, like the Static policy.

*2) Control performance:* Figure 5(a)-5(b) shows the performance evaluation in terms of the summation of the cumulative errors of the three controllers using the different algorithms, expressed as the integral over the simulation time of the norm of each pendulum state vector.

From Figure 5(a), we can clearly see large gaps between the cumulative error of the Static/Discrete-QoS policies and the Proportional/Optimal Continuous-QoS policies. Overall, using continuous QoS management we improved the performance of the system by about 21.6% in terms of the reduction of the accumulated error. However, there is almost no gap either between the Static and the Discrete-QoS or between the Proportional and the Optimal Continuous-QoS. The reason for the former is the lack of room for adaptation due to the limited number of QoS levels. The reason for the latter is the fact that when perturbations are not overlapped, the Optimal and the Proportional policies behave almost the same. Nevertheless, the Optimal policy performs somewhat better.

Figure 5(b) is a close-up view for the first 60s of the long run. Compare Figure 5(b) with any simulated scenario in Figure 4: clearly, each perturbation incurs an obvious error increase. However perturbations of different overlap scenarios result in different amounts of error increment. Nevertheless, from a general view, the gaps are getting larger and larger as time progress, which proves that adaptive QoS techniques can be used to improve control systems performance and that the optimal QoS policy (in the long run) is the one that optimizes control performance.

### D. Discussion

As we pointed out, different perturbation overlap scenarios yield differing opportunities for performance improvement. As expected, the Optimal Continuous-QoS generally performs better than the Proportional Continuous-QoS. However, we found in practice that the control performance of the Optimal policy may sometimes be slightly worse then the Proportional policy when faced with fully overlapping perturbations, contradicting our theoretical results from section V. In particular, we have observed that the drastic sampling period changes that the optimal policy implies, depending on the specific controller setting and allowed sampling period variability, may slightly decrease rather than increase control performance. We are still analyzing this puzzling result and hope to address it in the future. Nevertheless, in a long run the Optimal Continuous-QoS slightly outperforms the Proportional Continuous-QoS with respect to the global benefit achieved and QoS techniques have been proven to improve control systems performance.

## VII. Conclusions

Traditional control systems employ static controllers designed to operate at a single sampling frequency. Traditional real-time systems are designed to support these applications with static resource allocations and hard performance guarantees. Recent work in both areas has examined softening these constraints to allow for time-varying adaptation. We have merged these two areas and shown that significant performance benefits can be obtained by employing Quality of Service techniques to control systems. We have presented a real implementation and shown that adaptive QoS techniques can improve control system performance by over 20% in our experiments. Further, we have presented a framework for doing adaptive Quality of Service control and developed highly effective policies for doing resource allocation within this framework.

## References


[1] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes," in *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, Dec. 2003, pp. 396–407.

[2] P. Marti, R. Villa, J. M. Fuertes, and G. Fohler, "On real-time control tasks schedulability," in *Proceedings of the European Control Conference*, Sept. 2001.

[3] P. Marti, G. Fohler, K. Ramamritham, and J. M. Fuertes, "Improving quality-of-control using flexible time constraints: Metric and scheduling issues," in *Proceedings of the 23nd IEEE Real-Time Systems Symposium (RTSS 2002)*, Dec. 2002.

[4] D. Seto, J. Lehoczky, L. Sha, and K. Shin, "On task schedulability in real-time control systems," in *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS 1996)*, Dec. 1996.

[5] H. Rehbinder and M. Sanfridson, "Integration of off-line scheduling and optimal control," in *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, June 2000, pp. 137–143.

[6] K. G. Shin and C. Meissner, "Adaptation and graceful degradation of control system performance by task reallocation and period adjustment," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, 1999, pp. 29–37.

[7] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic feedback control," in *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, June 2000, pp. 121–128.

[8] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, pp. 25–53, 2002.

[9] J. Yepez, J. Fuertes, and P. Marti, "The large error first (LEF) scheduling policy for real-time control systems," in *Work in Progress Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS WIP 2003)*, Dec. 2003, pp. 63–66.

[10] C. Lin, P. Marti, S. A. Brandt, S. Banachowski, M. Velasco, and J. M. Fuertes, "Improving control performance using adaptive quality of service in a real-time system," in *submitted*, Dec. 2003.

[11] M. Velasco, J. Fuertes, and P. Marti, "The self triggered task model for real-time control systems," in *Work in Progress Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS WIP 2003)*, Dec. 2003, pp. 67–70.

[12] S. Brandt and G. Nutt, "Flexible soft real-time processing in middleware," *Real-Time Systems*, vol. 22, pp. 77–118, 2002.

[13] D. Henriksson, A. Cervin, J. Akesson, and K.-E. Arzen, "On dynamic real-time scheduling of model predictive controllers," in *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NV, Dec. 2002.

[14] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "Practical solutions for QoS-based resource allocations," in *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, Dec. 1998.

[15] G. C. Buttazzo, M. Velasco, and P. Marti, "Managing quality-of-control performance under overload conditions," in *submitted*, Dec. 2003.

[16] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[17] E. K. Chong and S. H. Zak, *An Introduction to Optimization*. John Villey and Sons, Inc, 1996.
[18] M. R. Garey and D. S. Johnson, *Computers and Intractability*. W.H. Freeman, 1979.
[19] K. J. Astrom and B. Wittenmark, *Computer-Controlled Systems. Third Edition.* Prentice–Hall, 1997.
[20] M. Dogruel and U. zgner, "Stability of a set of matrices: A control theoretic approach," in *Proceedings of the 34th Conference of Decision and Control*, Sept. 1995.