Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems *

G. Beccari S. Caselli M. Reggiani F. Zanichelli
Dipartimento di Ingegneria dell'Informazione
Università di Parma - 43100 Parma - Italy
e-mail: {beccari,caselli,reggiani,mczane}@ce.unipr.it

Abstract

Due to the high number of sensors managed and need to perform complex reasoning activities, real-time control systems of autonomous robots exhibit a high potential for overload, i.e., real-time tasks missing their deadlines. In these systems overload should be regarded as a likely occurrence and hence managed accordingly. In this paper we illustrate a novel scheduling technique for adaptation of soft real-time load to available computational capacity in the context of autonomous robot control architectures. The technique is based on rate modulation of a set of periodic tasks in a range of admissible rates. The technique is shown to be easily computable and several variations in implementation are reviewed within the paper.

1 Introduction

Robot control systems have always relied upon real-time operation for their safe, deterministic behavior and smooth trajectory control. However, in the majority of *traditional* robot control systems real-time constraints have been enforced by means of well-engineered implementations largely based on resource underutilization and a limited set of guaranteed computations (often a single real-time loop).

This approach has become inadequate in the design of control architectures for *autonomous* robots, operating in dynamic and possibly unstructured environments. Autonomous robots are required to adapt their behavior to novel operating conditions and possibly negotiate an effective course of actions taking into account multiple goals. They achieve adaptability and autonomous operation by means of richer perceptual, motor, and reasoning capabilities. However, guaranteeing compliance to real-time constraints in autonomous robots is much more challenging than in traditional robot systems, because of the complex logical control architecture and of the dynamic environments they must cope with [4, 18, 20].

In an autonomous robot architecture a number of events may lead to sudden variations in real-time load and eventually to an overload. In such a situation, the soft real-time nature of certain computations of the control architecture can be exploited to bring the system fully under control or implement a graceful degradation scheme. In this paper, after a discussion of the main real-time issues arising in the design of autonomous robot systems, we describe a scheduling approach adapting soft real-time tasks to available capacity during overloads or occasional high-priority computations.

2 Issues in autonomous robot architectures

A distinctive feature of autonomous robots is the large number of sensors they must be equipped with (most notably *exteroceptive* sensors). These sensors are needed to perceive environmental changes and overcome the many sources of uncertainty. The control architecture must process the rich sensory information, perceive change, react quickly, and plan ahead an appropriate execution strategy for the current task.

A number of structuring approaches are being pursued in the design of control architectures for autonomous robots, from purely reactive architectures to deliberative systems. Among them, *hybrid control architectures* attempt to incorporate both reactive and deliberative functionalities in a multi-layered structure [10, 18, 20, 22]. The underlying architectural approaches emphasize modularity, concurrency, and extensibility. Hybrid architectures are typically structured in terms of behaviors, often encompassing critical reactive functionalities, and reasoning modules [4, 17]. Behaviors may be activated and de-activated based upon the current application context.

Further reasons for sudden variations in computational load are the following:

- *Intermittent sensing* - Certain sensors are permanently in use for low level proprioception and alerting purposes, whereas other sensors are exploited in a context dependent manner.
- *Highly variable execution times* - Sensor interpretation and motion planning algorithms are characterized by high variability in their computation time. Even fundamental robot control algorithms, such as inverse kinematics and inverse dynamics

*Partial support for this research has been provided by MURST, Italy (Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications") and by C.N.R., Italy (Special Project "Grasp and Exploration Using a Dextrous Robotic Hand", Grant N. 97.04583.CT07).

[9, 19], may exhibit significant variability in computation times depending upon input values and specific branches taken.

A prevalent fraction of real-time computational load for robot control consists of periodic activities. These include low-level control loops, trajectory interpolation schemes, kinematic coordination loops, proprioceptive sensor acquisition, and exteroceptive sensing for purposeful behaviors. Periodic activities should be performed in a synchronous (per task) manner, as actuation jitter could degrade accuracy in motion execution.

During system development the rate of most periodic activities can be chosen in a range of admissible values, as long as pertinent control-level goals (e.g., stability) or task-level goals (e.g., perception of a moving target) are attained. For certain activities, their period directly affects the function or the algorithm being computed. This implies that changing their rate should be accompanied by a change of the algorithm (i.e., of the *task*, in real-time terminology). For all these activities (including control loops and trajectory interpolation) the rate should be set off-line or only modified in well-defined situations, such as with the robot idle and parked in a safe position. On the other hand, for other periodic activities, such as exteroceptive sensing and periodic monitoring or reasoning tasks, the rate could conceivably be modified dynamically, provided that this is accounted for at the robot mission control level.

We regard robot control architectures as being mostly *soft real-time*. In fact, in most robot control systems, hardly a single real-time deadline miss (typically from a periodic thread) will determine a global system malfunctioning. Rather, occasional deadline misses will simply affect overall quality in robot task execution (e.g., trajectory smoothness). Of course, systematic misses or starving threads will eventually lead to a global malfunctioning and make the task fail. This remark also emphasizes the need for timing fault management to prompt recovery and adaptation of the computation [24]. Furthermore, the soft real-time nature of many robot systems, along with the need to integrate non–real-time reasoning engines, may suggest the use of general-purpose operating systems providing basic soft real-time features [23].

3 Graceful degradation policy

Let $\mathcal{T} = \{ \tau_1, \tau_2, \ldots, \tau_N \}$ be a set of N real-time periodic tasks. Each task τ_i has a *worst case execution time (WCET)* C_i, a relative deadline D_i (relative to its *ready/request* time), and a period T_i, i.e., τ_i must be executed once every T_i time units. We assume that $\mathcal{T}$ is partitioned in two subsets $\mathcal{T}_h$ and $\mathcal{T}_s$, so that $\mathcal{T} = \mathcal{T}_h \bigcup \mathcal{T}_s$ and $\mathcal{T}_h \bigcap \mathcal{T}_s = \emptyset$. $\mathcal{T}_h$ is the set of hard real-time tasks and $\mathcal{T}_s$ is the set of soft real-time tasks. Tasks in $\mathcal{T}$ are sorted in such a way that τ_i has a period T_i lower than the period T_{i+1} of τ_{i+1}:

$$T_i < T_{i+1}, \;\; i = 1 \ldots N-1. \tag{1}$$

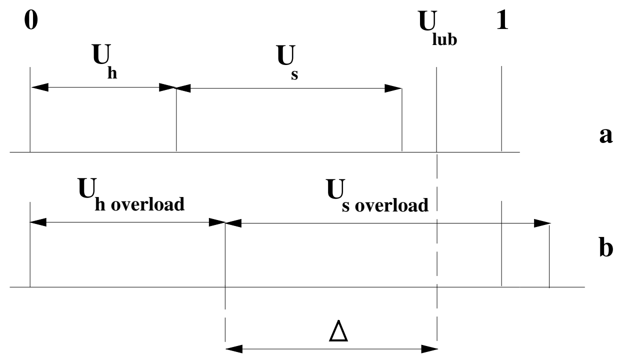

Figure 1: Requested CPU utilization before and after overload.

Tasks in $\mathcal{T}_s$ satisfy the following hypothesis of non-null *rate modulation range*[1]:

$$\begin{aligned} T_{i,min} &\leq T_i \leq T_{i,max} \\ T_{i,min} &\neq T_{i,max} \end{aligned} \tag{2}$$

i.e., under proper task conditions, the rate of soft real-time tasks can be adjusted within an allowable range.

Tasks are scheduled by *Rate Monotonic Scheduling* (RMS) [16] and priorities are fixed *a-priori* accordingly: namely, if P_i is the priority of τ_i and P_{i+1} is the priority of τ_{i+1}, then $P_i > P_{i+1}$. The processor *Utilization Factor* of each task τ_i is C_i/T_i. The least upper bound for a task set to be feasibly scheduled by the RMS algorithm is $L(n) = n(2^{\frac{1}{n}} - 1)$, i.e. if $\mathcal{U}(\mathcal{T}) = \sum_{i=1}^{n} C_i/T_i \leq L(n)$ then the task set is guaranteed to be schedulable by the RMS algorithm [16]. Other results [1, 15, 5, 12] can also be applied to get an improved least upper bound for $\mathcal{U}(\mathcal{T})$ under RMS; nevertheless, since this work is not directly concerned with this bound, in the following we assume $\mathcal{U}_{lub}$ to be one of them.

The utilization factor can be written as:

$$\mathcal{U}(\mathcal{T}) = \mathcal{U}(\mathcal{T}_h) + \mathcal{U}(\mathcal{T}_s) \tag{3}$$

where $\mathcal{U}(\mathcal{T}_h)$ is the utilization factor of the hard real-time subset and $\mathcal{U}(\mathcal{T}_s)$ is the utilization factor of the soft real-time subset (Figure 1 (a)).

The proposed graceful degradation policy, hereafter termed *rate modulation policy*, is based on a controlled *relaxation* of the $\mathcal{U}(\mathcal{T}_s)$ utilization factor as a means to manage overload conditions. These include hard or soft real-time tasks taking longer than expected, hard real-time sporadic or aperiodic tasks unaccounted for, and modification of scheduling parameters induced by the deliberative level of the hybrid robot control architecture. In general, this *relaxation* is achieved by means of a reduction of the rate $f_i = 1/T_i$ for some or all the tasks $\tau_i \in \mathcal{T}_s$. In the following, we analyze the behavior

[1] Without loss of generality, in the paper we assume that all soft real-time tasks satisfy this hypothesis. The technique requires that the hypothesis be satisfied for at least one task.

of the rate modulation policy in the overload case. Similar techniques can be applied when rate modulation is mandated by higher levels of the control system.

The following inequality:

$$\Delta = \mathcal{U}_{lub} - \mathcal{U}(\mathcal{T}_h) \geq 0 \tag{4}$$

holds even during overload, or otherwise some hard real-time deadline could be missed and the system would fail. Δ is the processor utilization effectively available to soft real-time tasks.

When an overload occurs $\mathcal{U}(\mathcal{T})$ becomes greater than $\mathcal{U}_{lub}$, and some tasks miss their deadlines. Let $\Delta_{overload}$ be the available processor utilization for soft real-time tasks in an overload situation. Whatever the source of overload, since equation (4) must hold, we have:

$$\mathcal{U}(\mathcal{T}_s)_{overload} > \Delta_{overload} \tag{5}$$

Note that *both* members in (5) might have changed from their values prior to overload (Figure 1 (b)). The system could be recovered if the adopted policy can reduce $\mathcal{U}(\mathcal{T}_s)$ below its present $\mathcal{U}(\mathcal{T}_s)_{overload}$ value while still executing soft real-time tasks at acceptable rates. The proposed degradation policy exploits the degree of freedom offered by the $\mathcal{T}_s$ subset, which allows choosing in a range of possible rates for each task in order to re-establish

$$\mathcal{U}(\mathcal{T}_s) \leq \Delta_{overload} \tag{6}$$

This policy determines a controlled exit from the overload situation as long as a feasible one (compatible with constraints (2)) exists. However, the application level control system must be notified of the new set of rates, as the high-level control policy may need to be changed accordingly.

The system can be successfully adapted to the new operating conditions by rate modulation when:

$$\sum_{j=1}^{S} \frac{C_j}{T_{j,max}} = \mathcal{U}(\mathcal{T}_s)_{min} \leq \Delta_{overload} \tag{7}$$

where $S = |\mathcal{T}_s|$. However, even when (7) does not hold, the rate modulation policy helps in taking the system to a possibly safer state, i.e., in general less tasks will be missing their deadlines after adaptation.

4 Rate modulation algorithms

In order to simplify the presentation of algorithms implementing the graceful degradation policy, we initially assume that tasks in $\mathcal{T}_s$ are sorted in such a way that τ_j has a period T_j lower than the period T_{j+1} of τ_{j+1}, with $j = 1 \ldots S-1$. More precisely, because of the rate modulation range hypothesis in equation (2), we assume that:

$$\begin{array}{rcll} T_{j,max} & < & T_{j+1} & \text{or} \\ T_{j,max} & < & T_{j+1,min} & \text{or} \\ T_j & < & T_{j+1,min} & \end{array} \tag{8}$$

Furthermore, we assume that $\forall \tau_j \in \mathcal{T}_s, T_j < T_{j,max}$ in nominal operative conditions.

4.1 Simple rescaling

Let us define the *Rescaling Factor* $\mathcal{F}$ as:

$$\mathcal{F} = \frac{\Delta_{overload}}{\mathcal{U}(\mathcal{T}_s)_{overload}} \tag{9}$$

and its reciprocal *Relaxation Factor* η as:

$$\eta = \frac{1}{\mathcal{F}} \tag{10}$$

Algorithm 1 (Simple rescaling): If $\forall \tau_j \in \mathcal{T}_s, T_j * \eta \leq T_{j,max}$ then the *rate modulation policy* is obtained by replacing T_j with $\eta * T_j$.

This straightforward replacement will re-establish the real-time guarantees for the system, as required by equation (6). Note however that if:

$$\exists \tau_j \in \mathcal{T}_s \;\; : \;\; T_j * \eta > T_{j,max} \tag{11}$$

a simple rescaling cannot be applied. In general, finding out a feasible modulation that guarantees the task set of the robot control architecture cannot be achieved by simple rescaling due to the constraints about rate modulation ranges. We next describe alternative algorithms dealing with these constraints.

4.2 General formulation

The goal is to determine a set of periods T_j for tasks in $\mathcal{T}_s$ so that the constraint (6) holds. Of course, there is an inverse relation between task periods and utilization factors, $\mathcal{U}_j = C_j/T_j$, where the C_j can be assumed as constants.

We have investigated several approaches that search a guaranteed state along different directions in the space of the *Utilization Factors*:

$$\begin{array}{rcl} \mathcal{U} & = & \{\mathcal{U}_j : \mathcal{U}_{j,min} \leq \mathcal{U}_j \leq \mathcal{U}_{s,max}\} \\ \text{with} & & \mathcal{U}_{j,min} = \dfrac{C_j}{T_{j,max}} \\ \text{and} & & \mathcal{U}_{j,max} = \dfrac{C_j}{T_{j,min}} \end{array} \tag{12}$$

Let Δ^* be a known value set to less than or equal to the maximum available utilization factor for soft real-time tasks, e.g., $\Delta^* = \Delta_{overload}$ in (6). Let $x_j = \mathcal{U}_j$ $j = 1 \ldots S$. Then the x_j are real variables, Δ^* is a real constant, and $0 \leq x_j, \Delta^* \leq 1$.

We translate the rate modulation constraints (2) and the capacity constraint (6) in terms of the new variables x_j, leading to the following *linear programming* (LP) formulation:

Find

$$\max \sum_{j=1}^{S} x_j \tag{13}$$

subject to

$$\left\{ \begin{array}{lcl} x_{j,min} & \leq & x_j \leq x_{j,max}, \;\; j = 1 \ldots S \\ \sum_{j=1}^{S} x_j & \leq & \Delta^* \end{array} \right.$$

where $x_{j,min}$ and $x_{j,max}$, $j = 1 \dots S$, are known constants, $x_{j,min} = \mathcal{U}_{j,min}$, and $x_{j,max} = \mathcal{U}_{j,max}$.

Algorithm 2 (LP): Any standard LP solver.

As well known, linear programming problems can be solved with polynomial algorithms, such as the ellipsoid method [11], or with the standard and usually efficient simplex algorithm. Due to its simple structure, the particular LP instance (13) can be solved in linear time with a *greedy* approach.

Algorithm 3 (Greedy): Let $\mathcal{X}_{min} = \sum_{j=1}^{S} x_{j,min}$ and $\mathcal{X}_{max} = \sum_{j=1}^{S} x_{j,max}$. If $\mathcal{X}_{max} \leq \Delta^*$ we set $x_j = x_{j,max}$ for each τ_j, i.e., each soft real-time task can be set to its maximum rate or to any desired rate compatible with the rate modulation constraints (2). If $\mathcal{X}_{min} > \Delta^*$ than no feasible solution exists, and overload must be managed by a second order graceful degradation mechanism. Nontheless, all periods could be set to their maximum values to simplify overload management and reduce the number of tasks experiencing uncontrolled deadline misses.

The greedy algorithm re-establishes equation (6) by maximizing the utilization factor of a subset of tasks in $\mathcal{T}_s$ and relaxing the residual subset of tasks so that if after L steps we have:

$$\sum_{j=1}^{L} x_{j,max} + \sum_{j=L+1}^{S} x_{j,min} < \Delta \tag{14}$$

and after step $L+1$:

$$\sum_{j=1}^{L+1} x_{j,max} + \sum_{j=L+2}^{S} x_{j,min} > \Delta \tag{15}$$

then a feasible solution is given by the following choice:

$$\begin{aligned} x_j &= x_{j,max} \quad j = 1 \dots L \\ x_j &= x_{j,min} \quad j = L+2 \dots S \\ x_{L+1} &= \Delta - (\sum_{j=1}^{L} x_{j,max} + \sum_{j=L+2}^{S} x_{j,min}) \end{aligned} \tag{16}$$

While simple, the greedy approach does not yield a solution ideally suited for robot control architectures, as it is strongly biased toward maximizing x_j for higher priority (lower indexes) tasks at the expense of lower priority ones. In this respect, the greedy algorithm may be considered as the opposite of the rescaling algorithm in a fairness spectrum. Furthermore, in robot control architectures usually the reference period for soft real-time threads does not coincide with the minimum one.

4.3 Saturation approaches

Several algorithms could be conceived to overcome the limitations of the simple rescaling algorithm in (11). We describe two simple algorithms differing in the way they choose the tasks that will be set to their maximum allowed period and hence only partially rescaled (*saturated* tasks). Both algorithms have quadratic complexity in the number of soft real-time tasks ($\mathcal{O}(S^2)$), but in the general case they obtain a more balanced allocation of processor utilization than the greedy algorithm.

Algorithm 4 (Iterative saturation): Let $\mathcal{T}_{s,sat} \subset \mathcal{T}_s$ be the set of tasks $\tau_j \in \mathcal{T}_s \ : \ T_j * \eta > T_{j,max}$ in (11). These tasks are saturated, i.e.:

$$\forall \tau_k \in \mathcal{T}_{s,sat} \ \ T_k = T_{k,max} \tag{17}$$

$\mathcal{T}_s$ is then partitioned in such a way that:

$$\mathcal{T}_s = \mathcal{T}_s^* \bigcup \mathcal{T}_{s,sat} \tag{18}$$

On the current set of unsaturated tasks $\mathcal{T}_s^*$ we attempt again the rescaling $\eta * T$ mechanism, with:

$$\Delta = \mathcal{U}_{lub} - \mathcal{U}(\mathcal{T}_h) - \mathcal{U}(\mathcal{T}_{s,sat}) \tag{19}$$

and $\mathcal{F}, \eta$ modified accordingly. Since additional tasks may need saturation, the algorithm continues until equation (6) is re-established.

Algorithm 5 (Prioritized saturation): This algorithm obtains a set of acceptable rate modulation values by saturating tasks in order of increasing priority. Thus, at each iteration K:

$$T_j = T_{j,max} \ \ j = S \dots K \tag{20}$$

$\mathcal{T}_s$ is partitioned so that:

$$\begin{aligned} \mathcal{T}_s &= \mathcal{T}_s^k \bigcup \mathcal{T}_{s,sat} && \text{where} \\ \mathcal{T}_s^k &= \{\tau_1, \dots, \tau_{K-1}\} && \text{and} \\ \mathcal{T}_{s,sat} &= \{\tau_S, \dots, \tau_K\} \end{aligned}$$

Tasks in $\mathcal{T}_s^k$ are modulated, if possible, by replacing T_j with $\eta * T_j$, where:

$$\Delta = \mathcal{U}_{lub} - \mathcal{U}(\mathcal{T}_h) - \mathcal{U}(\mathcal{T}_{s,sat}) \tag{21}$$

and $\mathcal{F}, \eta$ are modified accordingly. Iterations start with $K = S$ and continue with $K = S-1, S-2, \dots$ until equation (6) is re-established.

4.4 Partially ordered rate modulation ranges

All rate modulation algorithms described in the previous section do not affect task priorities. In fact, equation (8) assures that all tasks maintain their original RMS ordering after rescaling. In many realistic robot control architectures the disjoint rate modulation ranges constraints in (8) do not hold. In general, any two period ranges could be partially overlapping or properly included. Because of rate modulation, the priority of a subset of tasks, expressed by the RMS ordering, could be modified from the original assignment.

The saturation approaches described can be applied also when rate modulation ranges are only partially ordered. However, a task re-ordering operation must be executed at each iteration when RMS consistency is not preserved owing to modulation.

The linear programming formulation (13) and the greedy algorithm remain also valid with general rate

Tasks	C	T_{min}	$T_{nominal}$	T_{max}	U_{max}	U	U_{min}
τ_1	3	20	30	40	0.15	0.1	0.075
τ_2	4	40	60	80	0.1	0.0667	0.05
τ_3	20	80	120	160	0.25	0.1667	0.1111
τ_4	44	180	270	300	0.2444	0.1630	0.1467
τ_5	60	320	550	600	0.1875	0.1091	0.1
τ_6	150	600	920	1200	0.25	0.163	0.1250
U_{tot}					1.1819	0.7685	0.6078

Table 1: Timing parameters for a set of soft real-time tasks (all times in ms).

	Greedy		Iterative Sat.		Priorit. Sat.	
Tasks	T	policy	T	policy	T	policy
τ_1	20	*hard*	20	*hard*	20	*hard*
τ_2	40	*min*	70	*adapt*	66	*adapt*
τ_3	176	*adapt*	140	*adapt*	132	*adapt*
τ_4	300	*max*	300	*max*	297	*adapt*
τ_5	600	*max*	600	*max*	600	*max*
τ_6	1200	*max*	1078	*adapt*	1200	*max*

Table 2: Adaptation policies chosen by rate modulation algorithms (periods in ms).

modulation ranges. Found solutions, though, might modify the original priority assignments to the task set due the to non-uniform rate modulation.

Algorithm 6 (LP with priority preservation): In order to maintain the original priority assignment, if a feasible one exists, we may simply include an additional set of constraints (expressing the rate ordering preserving constraint (1)) in the LP formulation in (13):

$$a_j\ x_j \geq a_{j+1}\ x_{j+1},\quad j = 1 \dots S-1 \qquad (22)$$

where the a_j are known constants, $a_j = 1/C_j,\ j = 1 \dots S$.

Indeed, a major value of the LP formulation is that it provides a framework where several constraints can be coherently formulated based on the application needs. Unfortunately, some constraints move the problem out of the LP domain. An example of such constraints we are currently considering is the *harmonic* constraint, to specify that the new rate values be harmonic:

$$k_j\ a_j\ x_j = a_{j+1}\ x_{j+1},\quad j = 1 \dots S-1 \qquad (23)$$

where k_j are integer variables.

5 Simulation results

Several simulated experiments have been performed to assess how the algorithms described in the paper modify task periods in order to re-establish a guaranteed scheduling. Table 1 presents the parameters for one of these simulated experiments. All tasks in $\mathcal{T}_s = \{\tau_1 \dots \tau_6\}$ are soft real-time tasks, with periods variable in allowable ranges. Ranges are neither interleaved nor nested, and hence the total ordering assumption holds. Tasks are scheduled by Rate Monotonic Scheduling, thus task indexes in Table 1 reflect their priority assignment.

At time t_o task τ_1 makes a request to be computed with a shorter period, $T_1 = 20\ ms$. This event increases the overall requested utilization factor to $\mathcal{U}_{tot} = \sum_{i=1}^{6} \mathcal{U}_i = 0.8185$, resulting in the possibility of a timing fault. The real-time analysis shows that tasks in $\mathcal{T}' = \{\tau_1 \dots \tau_5\}$ are still guaranteed, as $\sum_{i=1}^{5} \mathcal{U}_i = 0.6555 \leq L(5) = 0.7435 \leq M(3) = 0.7798$, whereas $\mathcal{T}' \bigcup \{\tau_6\}$ is no longer guaranteed by RMS and τ_6 could miss its deadlines ($\sum_{i=1}^{6} \mathcal{U}_i = 0.8185 \geq M(3)$) [2].

Table 2 shows how the rate modulation algorithms described in the paper modify task rate parameters to re-establish scheduling guarantees. Being the highest priority task, τ_1 can be regarded as a hard real-time task, and hence is labeled as *hard* in Table 2. In the solution obtained by the *greedy* algorithm the task set is partitioned in such a way that task τ_2 is executed at its maximum rate, i.e. with its minimum period (*min*), task τ_3 is executed at an intermediate rate (*adapt*), whereas all other tasks are maximally relaxed, i.e. their periods are set to the maximum allowable values (*max*). The *iterative saturation* algorithm results into a more balanced solution, adapting periods as much as possible while saturating a reduced set of them to their maximum period when need. The *prioritized saturation* algorithm clearly favors high priority tasks while saturating low priority ones. The *simple rescaling* algorithm is not applicable under this particular operative condition. Finally, given the underconstrained nature of this problem the results provided by the general *linear programming* formulation are not particularily useful and are omitted from Table 2.

The *iterative saturation* appears more suitable for generic robot control architectures, owing to its ability to maintain task rates closer to their nominal values. The *prioritized saturation* algorithm may prove more effective in specific applications, where rate assignements closely reflect the relative importance even of soft real-time tasks.

[2] M(k) is Mok's upper bound [14] for a k-base harmonic task set.

Timing params.	τ_1	τ_2	τ_3	τ_4
C	8	8	8	8
T	20	40	80	160
T_{min}	15	35	75	-
T_{max}	35	65	125	-
U	0.4	0.2	0.1	0.05
$Priority$	60	59	58	57
$C_{overload}$	8	16	16	16
$U_{overlod}$	0.4	0.4	0.2	0.1
$T_{modulated}$	30.45	60.9	121.8	160

Table 3: Timing parameters in [ms] for the set of threads. Priority level refers to Solaris 2.6 priority.

6 Experimental results

Preliminary experiments on rate modulation have been performed on a VME-based Force 3CE computer, with MicroSPARC CPU and Solaris 2.6 operating system. This system currently operates as the main controller of a Puma robot manipulator at the University of Parma [3]. Experiments leverage upon Solaris multithreading capabilities and real-time oriented features, including reduced latency to interrupts, frequent kernel preemption points, three separate scheduling classes (*Real-Time, Sys, User*) for thread execution, process memory locking to avoid paging, and high resolution real-time timers.

In order to simplify the design of real-time robot control architectures we have developed a library on top of Solaris POSIX facilities. This library, termed `rt-lib` [3], implements mechanisms for periodic computations, handling of timing faults, monitoring of CPU utilization, and procedures that allow on-line modification of thread scheduling parameters (period, deadline and priority). Times in `rt-lib` are measured using `gethrtime/gethrvtime` calls (1 μs resolution), whereas periodic computations exploit the `nanosleep` system call (1 ms resolution).

We have developed a thread-based test application to validate the *simple rescaling* algorithm described in the paper, whereas evaluation of the other rate adaptation algorithms is in progress. The set of threads $\tau_1 \ldots \tau_4$ of the application is characterized in Table 3. Note that $\mathcal{T}_s = \{\tau_1, \tau_2, \tau_3\}$ and $\mathcal{T}_h = \{\tau_4\}$.

In the application all threads are initially activated with harmonic rates. Since:

$$\mathcal{U} = \mathcal{U}(\mathcal{T}_h) + \mathcal{U}(\mathcal{T}_s) \quad = \quad 0.75 \leq L(4) = 0.7568 \leq 1$$

the system is guaranteed. Nominally, in a 80 hyperperiod test interval, τ_1 should be executed 640 times, τ_2 320 times, τ_3 160 times, and τ_4 80 times. After 10 hyperperiods (about 1600 ms) the execution times of τ_2, τ_3, and τ_4 are artificially doubled so that the requested utilization becomes $\mathcal{U} = 1.1$ and overload occurs.

Table 4 shows some statistics about the number of overruns and cycle skips undergone by τ_3 and τ_4 when

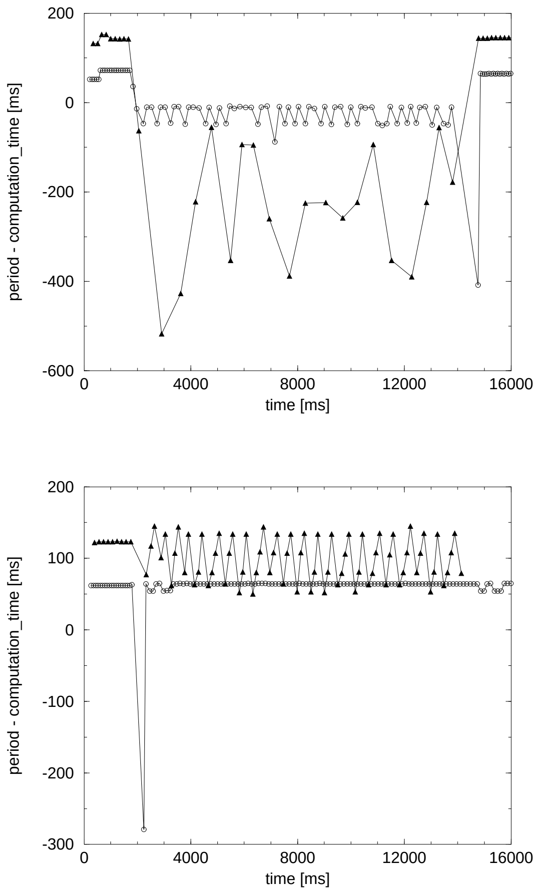

Figure 2: An overload situation (top) and its management by means of rate modulation (bottom). White circle: τ_3. Black triangle: τ_4

rate modulation is not applied. During overload, since

$$\mathcal{U}(\tau_1) + \mathcal{U}(\tau_2) = 0.8 \leq L(2) = 0.82$$

τ_1 and τ_2 are still guaranteed. On the other hand, τ_3 and τ_4 miss several deadlines and their computation times become greater than their periods for the 1600-15000 interval of Figure 2 (top). As automatic skips are performed by the `rt-lib` library, low priority thread execution is delayed while τ_1 and τ_2 complete their given number of cycles earlier (nearly at 15000 ms). After that, no overrun is detected, being τ_3 and τ_4 amply guaranteed. The spike exhibited by τ_3 (lasting 408 ms and occurring nearly at 15000 ms) is caused by the extra-load due to `pthread_exit()` calls.

Table 5 shows statistics about the number of overruns and skips occurring to τ_3 when rate modulation is enabled. Modulation is performed at the first overrun detection (Figure 2 (bottom)) by the *timing fault handler* procedure hooked to τ_3. As the rate ranges are wide enough, a simple rescaling ($\eta * T$, Algorithm 1)

	τ_3	τ_4
Activation requests	64	20
Overruns	64	20
Skips	134	61
Max consecutive Skips/cycle	7	5
Avg. consecutive Skips/cycle	2.09	3.05
Max compTime-Period [ms]	408	516
Avg. compTime-Period [ms]	32	235

Table 4: Events and statistics during an unmanaged overload situation.

	τ_3	τ_4
Activation requests	1	0
Overruns	1	0
Skips	5	0
Max consecutive Skips/cycle	5	0
Avg. consecutive Skips/cycle	5	0
Max compTime-Period [ms]	279	0
Avg. compTime-Period [ms]	279	0

Table 5: Events occurring during an overload situation managed by means of rate modulation.

can be applied. The following parameters are used to compute this rate modulation:

$$\begin{aligned}\Delta_{overload} = \mathcal{U}_{lub} - \mathcal{U}(\mathcal{T}_h) &= 0.7568 - 0.1 = 0.6568 \\ \mathcal{F} = \frac{\Delta_{overload}}{\mathcal{U}(\mathcal{T}_s)_{overload}} &= \frac{0.6568}{0.4 + 0.4 + 0.2} \\ \eta = \frac{1}{\mathcal{F}} &= 1.52\end{aligned}$$

The adopted rescaling fully guarantees the computation of τ_4, which was required to be hard real-time but also retains the lower priority in the system.

7 Relevant literature

The use of predefined rate schedules for different operating modes has been around for a long time as an empirical technique to manage variability in computational load. However, in the real-time literature relatively few approaches have addressed the overload problem. Load adjustable algorithms [14] and value based policies [7, 13] are the main techniques proposed for graceful recovery from overload. A load adjustment mechanism is proposed in [14] in order to handle periodic processes with varying temporal parameters. The aim of this work is to determine feasible time parameter configurations (execution time C and period T) and thus modify the real-time computation for collections of tasks. The configuration selection problem is solved by a novel harmonics approach achieving the maximum exploitation of the computational resources under any time parameter configuration. While appealing, this approach does not lend itself to the real-time subsystems of autonomous robots, where execution times, in spite of their variability, cannot be set or chosen by the designer.

Value based policies, e.g., RED [7] and D_{over} [13], adopt reject procedures that suspend low value tasks according to the application privileges they own. This approach is particularly suited for hard real-time applications. In control architectures for autonomous robots, while it is acceptable that certain soft real-time threads are computed at a lower rate, typically few or no threads can be rejected for an arbitrary amount of time without a serious impairment of robot skills.

Other research deals with real-time policies compliant with soft time constraints. For instance, the jitter analysis in [2] makes the assumption that inaccuracy characterizes actual systems, hence some jitter, e.g., the uncertainty of the arrival times of individual frames in a communication system, should be taken into account during the design of real systems. The skip approach in [8] accepts more flexible timing constraints than those allowed by usual real-time approaches. It assumes that certain tasks could abort some instance during a periodic execution, especially for data transmission applications. In our view such a soft real-time approach is well-suited also for robot control architectures.

The recent contributions in [6] and [21] can be directly compared to the research described in this paper. In [6] periodic computations are modeled as springs with given elastic coefficients and minimum lengths. Requested variations in task execution rates or overload conditions are managed by changing the rates based on the spring elastic coefficients. In [21] task rates are allowed to vary within given ranges. Each task is assumed to be characterized by a performance index such that the index is a monotonically decreasing and convex function of the task rate. Task rates are determined by means of a combination of search and Lagrangian multipliers. While [6] and [21] address a problem similar to the one discussed in the present paper, we believe that our approach provides a more general formulation and straightforward solution.

8 Conclusions

The paper has motivated the demand for overload management and graceful degradation mechanisms in autonomous robot control architectures. The main contribution of the paper is a novel scheduling technique based on rate modulation of periodic soft real-time tasks characterized by a range of admissible rates. Several variations of the modulation technique have been presented for totally ordered rate ranges, being all amenable to easy computation. Furthermore, these algorithms have been shown to be extendible to the case of partially ordered rate modulation ranges, although in general with the modification of the original task priorities.

All algorithms described in this paper follow a rate-monotonic formulation, i.e. slower processes (low priority) are considered the less important ones. Unfortunately, robotic applications sometimes regard a low rate activity (hence lower priority in RMS) to be more valuable than higher frequency processes (higher RMS priority). In our current work we are investigating value-based policies taking into account task-related thread priority in the modulation of the soft real-time threads of a robot control system.

References

[1] N.C. Adusley, A. Burns, M.F. Richardson, K. Tindell, A.J. Weillings, "Hard real-time scheduling: the deadline monotonic approach", IEEE Workshop on Real-Time Operating Systems and Software, May 1991.

[2] S.K. Baruah, D. Chen, A.K. Mok, "Jitter Concerns in Periodic Task Systems", IEEE Real-Time Systems Symposium, RTSS'97, San Francisco CA, December 1997.

[3] G. Beccari, S. Caselli, M. Reggiani, F. Zanichelli, "A Real-Time Library for the Design of Hybrid Robot Control Architectures", IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'98, Victoria, BC, Canada, October 1998.

[4] R.P. Bonasso, R.J. Firby, E. Gat, D. Kortenkamp, D. Miller, M. Slack, "Experiences with an Architecture for Intelligent, Reactive Agents", J. of Experimental and Theoretical Artificial Intelligence, Vol. 9, No. 1, 1997.

[5] A. Burchard, J. Liebeherr, Y. Oh, S.H. Son, "New Strategies for assigning real-time tasks to multiprocessor systems", IEEE Trans. on Computers, 44(12):1429-1442, December 1995.

[6] G.C. Buttazzo, G. Lipari, L. Abeni, "Elastic Task Model for Adaptive Rate Control", IEEE Real-Time Systems Symposium, RTSS'98, Madrid, Spain, December 1998.

[7] G. Buttazzo, J. Stankovic, "RED: A Robust Earliest Deadline Scheduling Algorithm", Proc. of 3rd International Workshop on Responsive Computing Systems, Austin, TX, 1993.

[8] M. Caccamo, G. Buttazzo, "Exploiting Skips in Periodic Tasks for Enhancing Aperiodic Responsiveness", IEEE Real-Time Systems Symposium, RTSS'97, San Francisco, CA, December 1997.

[9] J.J. Craig, *Introduction to Robotics – Mechanics and Control*, 2nd edition, Addison-Wesley, Reading, MA, 1989.

[10] R.J. Firby, "An Investigation into Reactive Planning in Complex Domains", Nat. Conf. on Artificial Intelligence, AAAI'87, 1987.

[11] M.R. Garey, D.S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., New York, 1979.

[12] C.-C. Han, H.-y. Tyan, "A Better Polynomial-Time Scheduling Test for Real-Time Fixed Priority Scheduling Algorithms", IEEE Real-Time Systems Symposium, RTSS'97, San Francisco, CA, December 2-5, 1997.

[13] G. Koren, D. Shasha, "D-over: An Optimal on-line Scheduling algorithm for overloaded real-time systems", IEEE Real-Time Systems Symposium, 1992.

[14] T.-W. Kuo, A.K. Mok, "Incremental Reconfiguration and Load Adjustment in Adaptive Real-Time Systems", IEEE Transactionon Computer, Vol. 46, No. 12, December 1997, pp. 1313-1324.

[15] J. Lehoczky, L. Sha, Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior", IEEE Real-Time Systems Symposium, RTSS'89, Santa Monica, CA, December 1989.

[16] C.L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", J. of the ACM, Vol. 20, No. 1, January 1973, pp. 46-61.

[17] D.M. Lyons and A.J. Hendriks, "Plannig for Reactive Robot Behavior", IEEE Int. Conf. on Robotics and Automation, ICRA'92, Nice, France, May 1992.

[18] D.J. Musliner, E.H. Durfee, K.G. Shin, "CIRCA: A Cooperative Intelligent Real-Time Control Architecture", IEEE Trans. on Systems, Man, and Cybernetics, Vol. 23, No. 6, November/December 1993.

[19] R.P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, MIT Press, Cambridge, MA, 1981.

[20] M. Schoppers, "A Software Architecture for Hard Real-Time Execution of Automatically Synthesized Plans or Control Laws", Conf. on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS'94), AIAA, March 1994.

[21] D. Seto, J.P. Lehoczky, L. Sha, K.G. Shin, "On Task Schedulability in Real-Time Control Systems", IEEE Real-Time Systems Symposium, RTSS'96, Washington, DC, December, 1996.

[22] R.G. Simmons, "Structured Control for Autonomous Robots", IEEE Trans. on Robotics and Automation, Vol. 10, No. 1, February 1994.

[23] J.A. Stankovic, et al., "Strategic Directions in Real-Time and Embedded Systems", ACM Computing Surveys, Vol. 28, No. 4, December 1996

[24] D.B. Stewart, P.K. Khosla, "Mechanisms for Detecting and Handling Timing Errors", Communications of the ACM, Vol. 40, No. 1, January 1997.