

Novel Scheduling Policies in Real-Time Multithread Control System Design

AUTHORS:

Luigi Palopoli Luca Abeni Fabio Conticelli
Gabriele Bolognini Benedetto Allotta

CORRESPONDING AUTHOR:

Prof. Benedetto Allotta
Sezione di Meccanica Applicata
Dipartimento di Energetica "Sergio Stecco"
Università di Firenze
Via Santa Marta, 3 - 50139 Firenze, Italy
Tel. +39-055-4796529 Fax +39-055-4796342

Submitted as a *Paper* to the
CONTROL ENGINEERING PRACTICE

Contents

1	Introduction	2
2	Real-Time Scheduling Revisited	5
3	A New View-Point on Multithread Control Software	7
3.1	The CBS Scheduling Approach	8
3.2	RTSIM Simulator	12
4	The case studies	16
4.1	Case study 1: inverted pendulum stabilization	18
4.2	Case study 2: a mobile robot	21
5	Conclusions and Future Work	26

List of Figures

.1	The CBS algorithm.	34
.2	Matrix for computing the probabilistic deadline PDF.	35
.3	Serving some jobs divided in chunks.	35
.4	Timing of an ideal digital controller: the execution time is neglected.	35
.5	Timing of a real digital controller: the execution time and scheduling jitter determine stochastic delays in the output release.	36
.6	Structural view on the RTSIM software package.	36
.7	An example of a thread schedule reconstructed using the RTSIM simulator.	37
.8	Architectural design of the inverted pendulum controller.	37
.9	Statistical distribution of execution times for the tracking thread.	38
.10	Dynamics resulting from experiments on the real system (dotted line) and from simulations (solid line).	38
.11	Inverted Pendulum Cost function vs T_1 (T_2 fixed) using EDF: simulations (left) and experiments (right).	39
.12	Cost Function vs. bandwidth using CBS: $T_1 = 10.0ms$, $T_2 = 2.0ms$.	39
.13	Schematic representation of the nonholonomic vehicle.	40
.14	Implementation scheme for the discrete-time multithread controller.	40
.15	Mobile Robot Cost function vs T_1 (T_2 fixed) using EDF.	41
.16	Mobile Robot Cost function vs B_1 bandwidth using CBS: $T_1 = 200ms$ and $T_2 = 6.5ms$.	41
.17	Mobile Robot: dynamics of the displacement $\tilde{\mathbf{x}}_r$ and for the command variables in a simulation run. The initial relative displacement error is $\tilde{\mathbf{x}}_r(0) = [2.8, 0.5]^T$. a) \tilde{x}_r b) \tilde{y}_r , c) τ_r d) τ_l .	42

Novel Scheduling Policies in Real-Time Multithread Control System Design

Luigi Palopoli palopoli@sssup.it, Luca Abeni luca@sssup.it,

Gabriele Bolognini bolognini@gandalf.sssup.it,

Scuola Superiore S. Anna, Via Carducci 40 56127 Pisa, Italy

Benedetto Allotta¹ ben@sssup.it,

Sezione di Meccanica Applicata, Dipartimento di Energetica "Sergio Stecco",

Università di Firenze, Italy

Fabio Conticelli fabio.conticelli@it.abb.com,

ABB Corporate Research, Italy

Abstract

In this paper soft real-time scheduling approaches (Resource Reservations) are applied to multithread digital control design and implementation. The advantage of this choice is the possibility of raising the sampling frequencies beyond the hard real-time boundaries, while retaining control on the transient overloads. An important component of the proposed framework is a simulation tool, which allows for a fine tuning of the scheduling parameters. Experimental and simulation results on two case studies show a significant performance improvement with respect to hard real-time scheduling, especially when the feedback controller operates on dataflows requiring statistically widespread computation times.

¹ Corresponding author: Tel. +39-055-4796529 Fx: +39-055-4793342

1 Introduction

Computer based control architectures allow to realize very complex and flexible real-time controllers by using low cost *off the shelf* components. An important application is represented by control system using multiple communicating loops activated at different frequencies. Each of these loops is usually implemented as a concurrent computation activity, called thread, which is periodically activated. The term multithread controllers is used for this kind of controllers. A first important feature of multithread controllers is that they are multirate digital systems. When dealing with linear systems, the analysis and synthesis of multirate controller is a relatively well known issue (K.J. Åström and B. Wittenmak, 1997; T. Chen and B. Francis, 1995; P. Colaneri et al., 1992; S. Bittanti et al., 1984; J. Tornero et al., 1999). As far as nonlinear multirate systems are concerned, analytical tools and techniques are beginning to appear in the recent literature (S. Monaco and D. Normand-Cyrot, 1996; Bo Hu and A.N. Michel, 1999; A.N. Michel and Bo Hu, 1999), but most of the work still lies ahead. Moreover, when a multirate controller is implemented on a computer architecture, many problems due to the limitation of computation and communication resources arise. In fact, many unmodeled phenomena – including data dependent loops and conditional branches in the programs or random delays in the network – introduce stochastic variations in the execution time of threads. A further source of uncertain delays is the scheduling of multiple concurrent activities on the same computer. To cope with these problems, an appropriate mixture of analytical and simulation techniques, arising from a synergistic effort of control and computer engineers, is required. A novel thread of research activities has been concentrating its attention in this direction.

Email addresses: 1 (Luigi Palopoli), 2 (Luca Abeni), 3 (Gabriele Bolognini), 4 (Benedetto Allotta), 5 (Fabio Conticelli).

In (J. Nilsson et al., 1998) stochastic delays introduced by networks are explicitly considered in the control synthesis. In (Albertos et al., 2000), the authors consider the impact of multithread scheduling on the controller performance. The investigation presented in this paper offers a possible solution to this specific problem.

The focus of the paper is restricted to the case of controllers realized by locally connecting a computer to a plant (without any network in between). Both the sensor data and the controller outputs, applied to the actuators, are supposed to be sampled with fixed periods. The communicating control loops are realized by periodic *threads*. Each thread executes a *job* upon the arrival of new sensor data. A typical approach for developing this kind of real-time controllers is a downfall process comprising two steps. The first step is the synthesis of an ideal controller, designed as if it should operate in the continuous time domain. The second one consists of realizing a digital approximation for the control law by decomposing it into a set of cooperating threads. The flow of information between the two steps is often weak. The controller design phase includes the determination of upper bounds on the sampling periods for preserving the system stability and its performance; most often the choice can only be done on empirical basis. As far as the computational aspect is concerned, a set of techniques, provided by the real-time community, guarantee that if certain upper bounds on the activation frequencies are respected, the set of threads is *schedulable*, i.e no job will ever execute beyond a time instant called *deadline* (usually equal to the next periodic activation). However, concepts like schedulability and deadline cannot easily be related to the quality of the dynamics of the controlled plant. In particular, the strict respect of each deadline (hard real-time guarantee) potentially brings to a strong underutilization of the system computational power; on the other hand it is known that a limited number of failures in updating the actuators values can be tolerated (K.G. Shin and Xianzhong Cui, 1995).

The main contribution of this paper is the application of novel scheduling policies, called *Resource Reservation* (RR), to the multithread real-time control of dynamical systems. These approaches permit to push the threads activation rates beyond the boundaries of hard schedulability, introducing a controlled ratio of deadline misses; in this way the

control system performance can be remarkably improved. In order to assess the influence of the scheduling choices on the control quality, a *codesign* tool, called RTSIM (Real-Time Controller Simulator) (L. Palopoli et al., 2001), has been realized by interfacing the RTLIB scheduling simulator to the OCTAVE mathematical package (see <http://www.octave.org>). The RTSIM tool is able, at one time, to reconstruct the functional and timing behaviour of an embedded controller and to simulate the dynamical evolution of the controlled plant. Moreover, it is possible to define performance metrics and collect statistics over stochastic parameters. The application of the tool provides useful guidelines for the choice of such parameters as the sensors' sampling rates and, more generally, permits to evaluate the impact of architectural choices on the system performance. A similar approach was proposed by Eker and Cervin (J. Eker and A. Cervin, 1999). In their work a real-time kernel is simulated in a Simulink block; the advantage is the good integration with a well known design environment. RTSIM, on the other end, offers a wide choice of pre-defined objects to simulate state-of-the-art scheduling algorithms; moreover its clear object oriented design allows for an easy extension of the library to simulate novel scheduling solutions.

The effectiveness of soft real-time approaches in the multithread implementation of control systems using data from heterogenous sensor sources is shown in two meaningful case-studies. The first case-study is the classical stabilization of an inverted pendulum where the position of the cart is acquired through a digital camera, while the angle of the pendulum is acquired through a potentiometer. The second case study is the tracking of a moving target by a mobile robot. In both cases, the control scheme consists of two hierarchical control loops fed with data having different timing properties. Both applications were modeled and simulated by RTSIM. Furthermore, experimental data were collected for the first case study. Simulations and experiments reveal that releasing the “zero deadline misses” requirement leads to actual performance improvements, which are more evident if the RR scheduling is used.

The paper is organized as follows. In Sec. 2, some concepts of real-time scheduling are reviewed. Sec. 3 presents the proposed reservation mechanism for scheduling control threads

and the performance evaluation tool. The two case studies are described in Section 4, along with the results of simulations and experiments. Finally, Sec. 5 summarizes the most important contribution of this work and outlines future investigations.

2 Real-Time Scheduling Revisited

The real-time multithread controller model assumed in this paper consists of a set of periodic real-time threads \mathcal{T}_i , $i = 1, \dots, N$. A real-time thread \mathcal{T}_i is a stream of *instances*, or *jobs* $J_{i,j}$, $j \in \mathbb{N}$, where \mathbb{N} denotes the set of integer nonnegative numbers. Each job $J_{i,j}$ is characterized by

- an *arrival time* $r_{i,j}$, i.e. the instant when job $J_{i,j}$ is required to start;
- an *absolute deadline* $d_{i,j}$;
- a *job execution time* $c_{i,j}$;
- a *finishing time* $f_{i,j}$.

In general, thread \mathcal{T}_i is characterized by three parameters (C_i, T_i, D_i) , where $C_i = \max_j \{c_{i,j}\}$ is the thread's Worst Case Execution Time (WCET), $T_i = \min_j \{r_{i,j+1} - r_{i,j}\}$ is the minimum interarrival time, and D_i is the relative deadline such that $d_{i,j} = r_{i,j} + D_i$. A real-time thread is said to be periodic if $r_{i,j+1} = r_{i,j} + T_i$; in this work, the controller is implemented using periodic threads with the relative deadline equal to the period ($D_i = T_i$). Moreover, c_i is used to indicate the best case execution time of thread \mathcal{T}_i ; hence, each job $J_{i,j}$ will have to execute for a time varying in the $[c_i, C_i]$ range; finally, \bar{c}_i indicates the thread mean execution time.

In classical real-time scheduling theory, the goal is to guarantee that every job in the system terminates before its absolute deadline (hard guarantee): $\forall(i, j), f_{i,j} \leq d_{i,j}$. To respect each job's deadline, suitable scheduling algorithms based on fixed (Rate Monotonic, RM) or dynamic (Earliest Deadline First, EDF) priorities can be used. Basic results on these algorithms can be found in a seminal work by Liu and Layland (C.L. Liu and J. Layland, 1973). Two prongs of a consistent research activity were originated from these results and a collection of well settled results has been produced and it is now systematically

summarized in different text books (M.H. Klein et al., 1993; J. Stankovic et al., 1998).

The most important result of (C.L. Liu and J. Layland, 1973) is that EDF and RM guarantee that *every* job meets its deadline if the following admission test is verified:

$$\sum_{i=1}^N \frac{C_i}{T_i} \leq U_l \quad , \quad (1)$$

where $U_l \leq 1$ is a constant depending on the scheduling algorithm.

The problem with this approach is that the admission test considers the worst case situation, so the system can be underutilized if $C_i \gg \bar{c}_i$. This overly conservative design may lead to large activation periods and, potentially, to a poor performance. As a matter of fact, the effectiveness of a controller is generally enhanced by the quantity of information it collects from the plant dynamics and hence by choosing shorter sampling periods. If more optimistic execution time estimates are used in the admission test, it is possible to decrease the activation periods, hopefully achieving improvements on the system performance. A first possibility is simply using RM or EDF algorithms with a modified admission test. The new admission test can use more optimistic choices than C_i for the execution time estimate. When the job execution time exceeds the optimistic estimates, the system is said to be *overloaded*. In (L. Palopoli et al., 2000b), well known results from the queuing theory (L. Kleinrock, 1975), are used to prove that, if

$$\sum_{i=1}^N \frac{\bar{c}_i}{T_i} \leq U_l \quad ,$$

then the overload condition is not permanent. The typical situation using the RM algorithm in overload conditions is that the threads having longest period miss their deadlines, while the ones with shortest period, being assigned the highest priorities, tend to respect their temporal constraints. Using the EDF algorithm, the situation is less formally predictable since the priorities are dynamically assigned. However, from a practical point of view, the privileged threads result to be the ones that are activated more frequently. Therefore, in both cases the only way for attaching importance to a thread is decreasing its period. However, in some cases the threads having the biggest influence on the system performance are necessarily activated with the longest periods (for instance the threads

using data coming from a camera sensor).

This problem can be solved by the scheduling techniques known as CPU reservations (L. Abeni and G. Buttazzo, 1998a; R. Rajkumar et al., 1998). Using a CPU reservation technique, each thread \mathcal{T}_i is assigned a couple (Q_i, T_i) , where T_i is the thread period, and Q_i is the amount of CPU time reserved to \mathcal{T}_i in a period. If $\sum_j \frac{Q_j}{T_j} \leq U_l$, \mathcal{T}_i is guaranteed to execute for Q_i time units every T_i , independently from the behaviour of the remaining threads. It is remarkable that such a guarantee can be given individually on every single thread whereas, in the case of classical *hard guarantee*, the schedulability is a property of the entire set of periodic threads. By using this approach, the most important threads can be made to respect more deadlines than the less important ones, independently from the periods. Proportional share scheduling (I.Stoica et al., 1996) introduces similar concepts, but gives less control over CPU allocation. In this work, a particular reservation technique, called Constant Bandwidth Server (CBS), is used and it will be described in section 3.1.

3 A New View-Point on Multithread Control Software

The most frequent assumption made during the control synthesis is that the time interval between the acquisition of the sensor data and the release of the controller output is negligible or fixed. Considering the case of a single-input/single-output system, this ideal assumption is depicted in Figure .4: the controller reacts in a negligible time to the arrival of a new sensor data emitting its output. In this case the only degradation of the system performance is due to the delays introduced by the sampling operation itself. A more realistic situation is depicted in Figure .5: the feedback law computation is associated to a periodic thread which has a non-null computation time and can be preempted by higher priority threads. As a result, there is a stochastically varying time lag between the data acquisition and the output emission. While the theory of digital controllers provides, at least for linear systems, some tools for catering with sampling inherent delays, wide stochastic parametric variations like the ones shown in Fig. .5 are much more difficult to treat. This problem entails a good deal of efforts aimed at ensuring some form of

deterministic behaviour for the controller implementation.

Traditional real-time scheduling theory concentrated on a particular notion of determinism based on respecting the jobs deadline. Referring once again to the example in Fig. 5, the idea is that if the control thread is known to terminate within its deadline, at least an upper bound on the maximum delay between the data acquisition and the output emission can be established. However, considering more complex topologies, where multiple threads activated at different rates communicate with each other, a real correlation between the threads deadlines and the controlled system properties (stability, performance, robustness) is much harder to find. As said earlier, this form of determinism can be very expensive in terms of performance loss, since the worst case conservative assumptions impose slow thread activation rates and sampling frequencies.

The alternative proposed in this paper is based on a looser form of determinism: the probabilistic guarantee. As shown in the previous section, the reservation based approaches decouple the importance of a thread from its activation rate and to perform individual guarantee on every thread. In this section, a brief description of a particular RR algorithm, named CBS is provided. The algorithm allows to perform a probabilistic guarantee on each thread: i.e. given the distribution of the threads execution time, the scheduling parameters can be chosen so that the probability of a deadline miss for a given thread be lower than a required value. In other words, the activation frequencies of the threads can be raised while a further adjustment is possible by concentrating the deadline misses on the less important threads. A concrete utilization of this approach is possible only if the scheduling parameters can be tuned so as to optimize the system performance. For complex systems this goal is very difficult to be accomplished analytically; hence an extensive use of simulation techniques is needed. To this end a simulation tool has been realized, which will be described in the last part of this section.

3.1 The CBS Scheduling Approach

The Constant Bandwidth Server (CBS) is a reservation technique based on dynamic priorities, first applied in the field of multimedia applications (L. Abeni and G. Buttazzo,

1998a). As previously stated, the goal of the scheduler is to reserve to thread \mathcal{T}_i an amount Q_i of time for each period T_i .

The CBS uses a global EDF scheduler which gives the highest priority to the threads having the shortest absolute *scheduling deadlines* (in practice, the ready threads queue is ordered by absolute deadline).

Hence, in order to be scheduled by EDF, each thread has to be assigned a proper scheduling deadline: this is done by a dedicated mechanism, called *server*. Since the CBS is the entity that enforces the reservation (by assigning proper deadlines), each thread \mathcal{T}_i is served by a dedicated CBS.

The server description and some remarkable properties are reported below.

Algorithm 1 *The CBS is defined as follows:*

- *A CBS is characterized by a current budget c_s and by a ordered pair (Q_s, T_s) , where Q_s is the maximum budget, T_s is the server period, and $c_s \in [0, Q_s]$. The ratio $B_s = Q_s/T_s$ is denoted as the server bandwidth. At each instant, a fixed deadline $d_{s,k}$ is associated with the server. At the beginning $d_{s,0} = 0$.*
- *Each served job $J_{i,j}$ is assigned a dynamic deadline $d_{i,j}$ equal to the current server deadline $d_{s,k}$.*
- *Whenever a served job executes, the budget c_s is decreased by the same amount.*
- *When $c_s = 0$, the server budget is recharged to the maximum value Q_s and a new server deadline is generated as $d_{s,k+1} = d_{s,k} + T_s$. Notice that there are no finite intervals of time in which the budget is equal to zero.*
- *A CBS is said to be active at time t if there are pending jobs (remember that the budget c_s is always greater than 0); that is, if there exists a served job $J_{i,j}$ such that $r_{i,j} \leq t < f_{i,j}$. A CBS is said to be idle at time t if it is not active.*
- *When a job $J_{i,j}$ arrives and the server is active the request is enqueued in a queue of pending jobs according to a given (arbitrary) non-preemptive discipline (e.g., FIFO).*
- *When a job $J_{i,j}$ arrives and the server is idle, the server checks if the last assigned deadline can be used. In particular, if $c_s \geq (d_{s,k} - r_{i,j})B_s$ the server generates a new deadline $d_{s,k+1} = r_{i,j} + T_s$ and c_s is recharged to the maximum value Q_s , otherwise the*

job is served with the last server deadline $d_{s,k}$ using the current budget.

- When a job finishes, the next pending job, if any, is served using the current budget and deadline. If there are no pending jobs, the server becomes idle.
- At any instant, a job is assigned the last deadline generated by the server.

As shown in (L. Abeni and G. Buttazzo, 1998a), the CBS algorithm presents some important properties: the first one is the *temporal isolation* property, stating that

Consider a set of periodic threads \mathcal{T}_i and assume that each of them is served by a dedicated CBS with budget Q_i and, hence, has an assigned bandwidth $B_i = \frac{Q_i}{T_i}$. If $\sum_i B_i \leq 1$ then each \mathcal{T}_i executes for Q_i time units every T_i , whatever the behaviour of the other threads.

In order to prove it, it is to be noted that, according to the algorithm definition, if no scheduling deadline is missed, than each thread will receive the reserved amount of time. On the other hand, since the EDF basic scheduling algorithm is used, no deadline is missed if and only if the total CPU utilization is $U \leq 1$. Hence, the following theorem (introduced in (L. Abeni and G. Buttazzo, 1998a)), is equivalent to the temporal isolation property as stated above.

Theorem 1 *A CBS with parameters (Q_s, T_s) contributes to the total CPU utilization U for an amount $U_s = \frac{Q_s}{T_s}$*

Proof:

See Appendix. □

A second important property is the *hard schedulability* property, stated by the following lemma:

Lemma 1 *If $B_i \geq \frac{C_i}{T_i}$ (that is to say, $Q_i \geq C_i$), the \mathcal{T}_i thread will not miss any deadline;*

Proof:

For any job, it holds $r_{i,j+1} - r_{i,j} = T_i$ and $c_{i,j} \leq Q_i$. Hence, by definition of the CBS, each job is assigned a scheduling deadline $d_{i,j} = r_{i,j} + T_i$ and it is scheduled with a budget $Q_i \geq C_i$. Moreover, since $c_{i,j} \leq Q_i$, each job finishes no later than the budget is exhausted,

hence the deadline assigned to a job does not change and is exactly the same as the one used by EDF. \square

Finally, if $B_i < C_i$ but a stochastic distribution for the threads execution times is known, the probability of a deadline miss can be bounded, as shown in (L. Abeni and G. Buttazzo, 1998b). In that paper, a CBS is modeled as a queue, and queuing theory is used to compute a bound for the deadline miss probability (see the cited paper for theorems and proofs). More formally, if $U_i(c) = P\{c_{i,j} = c\}$ is the Probability Distribution Function (PDF) of the execution times, it is possible to find the probability $X_{(\delta)} = P\{f_{i,j} - r_{i,j} \leq \delta\}$ that a job will finish within a relative *probabilistic deadline* δ .

Definition 1 *A thread τ_i is said to respect a probabilistic constraint (δ, P) if $X_i(\delta) \geq P$. In order to compute the PDF of the probabilistic deadline $X_i(\delta)$, a state variable $v_{i,j}$ has to be introduced, representing the amount of work to be served by the i^{th} CBS immediately after the j^{th} arrival:*

Definition 2 *The state $v_{i,j}$ of a CBS S_i is defined as*

$$v_{i,1} = c_{i,1}$$

$$v_{i,j} = \max\{0, v_{i,j-1} - Q_i\} + c_{i,j}$$

and indicates the length of the queue modeling the CBS at time jT_i , that is the units of times that are still to be served when the job $J_{i,j}$ arrives.

In (L. Abeni and G. Buttazzo, 1998b), the following lemma, relating δ and $v_{i,j}$ was proven:

Lemma 2 *The probability that a job finds a CBS state equal to v when it arrives (that is to say, the probability of finding a CBS queue of length v) is a lower bound for the probability that the job finishes within a relative deadline*

$$\delta = \left\lceil \frac{v}{Q_i} \right\rceil T_i.$$

Moreover, a way for computing the PDF of v is provided (index i has been omitted in order to simplify notation) by the following:

Theorem 2 *If $\pi_k^{(j)} = P\{v_j = k\}$ is the state probability of process v_j , $U(h) = P\{c_j = h\}$*

is the probability that an arriving job requires h units of computation time (since c_j is time invariant, $U(h)$ does not depend on j), and $Q > \bar{c}$, then $\pi_k^{(i)}$ can be computed by solving the equation

$$\Pi^{(j)} = M\Pi^{(j-1)} \quad , \quad (2)$$

where

$$\Pi^{(j)} = \left[\pi_0^{(j)} \quad \pi_1^{(j)} \quad \pi_2^{(j)} \quad \pi_3^{(j)} \quad \dots \right]^T$$

and M is shown in Figure .2.

There are two observations to be made. First, for the scheduling system to remain stable (i.e. for the overload to be transient), Q_i has to be chosen in the $[\bar{c}_i \ C_i]$ range. Second, reserving a bandwidth $B_i \leq \frac{C_i}{T_i}$ corresponds to the optimistic assumption that since $Q_i > \bar{c}_i$, if a job requires more than Q_i time units, then one of the following jobs could require less than Q_i , so if a thread misses a deadline, it will be probably soon be able to respect the following ones.

Summarizing, the use of a CBS scheduling approach for the control threads adds a degree of freedom in the implementation design by allowing to reserve a fraction of the total CPU bandwidth to each of them. In Section 4.2.2 it is shown that this approach permits to achieve remarkable improvements on the control performance.

3.2 RTSIM Simulator

In order to evaluate the performance of a given Hardware/Software architecture when it is employed to control a continuous time plant, it is needed a tool able to co-simulate the continuous time system along with the scheduling of the controller threads. From a practical viewpoint, a scheduling simulator has to reconstruct the controller schedule, computing the time instants when each control thread starts and finishes, while an ODE package is needed to integrate the equations describing the plant between such instants. For this purpose, the Real-Time controller SIMulator (**RTSIM**) has been developed.

RTSIM is designed to achieve a high level of computational efficiency; this requirement is motivated by the need for treating very complex systems (either in the plant equations

or in the controller structure) and for executing a big number of simulation runs to collect statistics over stochastically varying parameters (e.g. the threads execution times). Another important feature of the package is the modeling flexibility: it is possible to describe arbitrary topological arrangements for the controller computational activities and to test a wide variety of scheduling choices and mutual exclusion protocols. From the modeling viewpoint, a plant is described by a set of ordinary differential equations (ODE) depending on the physical parameters and on the external inputs. The real-time controller is composed of the following elements:

- a set of samplers, which periodically read the output of the plant;
- a set of computing threads, described by statistical timing parameters; they perform the computation and share the CPU according to the assigned scheduling policy;
- a set of memory buffers used to store intermediate results and to implement the communication between the threads; mutually exclusive accesses can be enforced by different protocols;
- a set of ZOHs (Zero Order Hold), which convert the numerical outputs into piecewise constant signals to be used as input variables for the plant.

RTSIM is entirely written in C++ and is based upon the functionalities of two existing packages: OCTAVE and RTLIB. OCTAVE provides a numerical library endowed with an intuitive support for linear algebra and ODE integration. RTLIB is a real-time systems simulator able to model a wide variety of scheduling algorithms and protocols. A high level structural view of RTSIM is reported in Figure .6. As it is possible to see from the figure, there are essentially three software components corresponding to logically different operations. Roughly speaking, two modules are mainly used to simulate the controller (RTLIB, and the data processing module) and the third one (numerical module) is used to simulate the plant dynamics.

More specifically, the numerical module is used to perform the ODE integration for the plant and provides all the support for the linear algebra and other numerical computation. It is essentially based on the OCTAVE C++ library. However, a set of interface classes encapsulate all the library technicalities providing an abstract interface; in this way the

base library can be easily changed (for example using MATLAB, GSL or others) without affecting the rest of the RTSIM structure. Different plants can be described by simply providing the implementation of a limited set of pure virtual methods; the most important of these methods is the differential equation which can be written in a very natural way using the mathematical abstractions provided by the numerical module.

RTLIB is a modular library devised to simulate distributed real-time systems. Some of its basic features are reported here, referring the reader to the cited paper for further details. A RTLIB simulation consists of a set of entities: entities react to events changing their internal status and possibly generating other events. All the computations in the simulator are performed at certain particular instants corresponding to events triggering. Thus, the status of an entity at any time t_2 depends on its status at any previous time t_1 and by the sequence of events between t_1 and t_2 . RTLIB structure is layered. The lowest level, called METASIM, offers a set of basic classes for creating entities and events, along with a set of important utilities including execution tracers, statistics collectors and random variables generators. The upper levels of RTLIB offer a wide set of predefined entities for simulating real-time systems. Real-time threads can be constructed as aggregations of pseudoinstructions having fixed or random duration; they can be handled by kernels endowed with different scheduling policies (Cyclic Executive, RM and EDF, Proportional share), or by aperiodic servers (Polling server (J.P. Lehoczky et al., 1987), Sporadic Server (B. Sprunt et al., 1989), Constant Bandwidth Server (L. Abeni and G. Buttazzo, 1998a), etc.). It is also possible to simulate mutually exclusive accesses to resources handled by different protocols (FIFO, Priority Inheritance/Priority Ceiling (L. Sha et al., 1990), Stack Resource Policy (T.P. Baker, 1990)). Another important feature under development is the support for network distributed systems, but it is not in the scope of this paper. Summarizing, once a structure of threads has been specified along with their scheduling policy, RTLIB permits to construct a timed sequence of events associated with the jobs activation and termination instants, with the beginning and the end of a pseudoinstruction and so forth. Fig. .7 shows an execution trace reconstructed using the RTLIB simulator. The trace visualization tool is part of RTLIB suite and is written entirely in JAVA.

In the original RTLIB design, threads and their pseudoinstructions cannot perform “actual” computations. A third module (the data processing module) was introduced to fill in this gap: its goal is essentially to simulate the computational part of a digital controller, RTLIB being focused on reconstructing its timing behaviour. The data module offers two types of components: computing units and storage units; both of these components are framed within a hierarchy of classes. In order to specify a computing unit, the programmer has to derive it from an abstract class and has to provide an implementation for three pure virtual methods. The first method acquires data from storage units and save them into the objects internal state. The second method is used to compute the output values based on the current state. The third one writes the data onto storage units. The module provides three type of storage units: input buffers, memory buffers, output buffers. Input buffers are used to contain the data sampled from the plant; computing units are only allowed to read from them. Output buffers are models of a Zero Order Hold behaviour for the data applied to the plants simulated actuators; computing units are allowed to write into this type of storage units. Finally Read/Write buffers model memory location used by the computing units to communicate intermediate results one another; so they can be accessed by the computational units in both directions. Computational and storage unit are associated to a symbolic meaningful name; in this way their composition is less error-prone and the simulation programs are easy to read. To conclude this *bird-eye* watch on RTLIB, some simplified remarks on the interactions between the modules are given. The information flow between the numerical module and the data processing module is bidirectional (see Figure .6). A plant, when integrating its differential equations, needs to read the data held in the output buffers. On the other hand, input buffers, whenever requested, have to read the data from the plant.

The relation between RTLIB and the data processing module can be explained considering that the computation units operations and the plant sampling have to be triggered at specific instants. Such instants are associated with the events produced by the RTLIB module. For example, suppose a computation unit be associated with a thread; more specifically, some of the pseudoinstructions of the thread can be associated with methods

calls on the computation unit. In this way the event raised when the first pseudoinstruction is scheduled may cause the computation unit to read the input data and store them into its internal state. After an arbitrary time, determined by pseudoinstruction temporal duration, a second pseudoinstruction could make the computation unit compute the output, and so on. Similarly the plant sampling is executed by the input buffer object whenever a periodic event is generated by an entity belonging to the RTLIB.

Finally the interaction between RTLIB and the numerical module is determined by the plant ODE integration. Such an integration is performed by a method call requiring as input a time interval. A useful property of this model is that the integration between two RTLIB generated events can be performed assuming constant command signals. So the RTLIB generated events can be used to drive the integration: every time a new event is handled, the plant's integrating method can be called using as parameter the time interval separating the latest produced event and the one being handled.

In order to build a simulation, the user is required to program a C++ file where he/she defines and connects the components of the simulation (plant, computing units, threads etc.) as instances of the library classes. Moreover, objects for collecting statistics and/or tracing the thread execution on files can easily be defined. Program templates are available and alleviate the effort for constructing the program. In the near future, a graphical GUI will be realized to further simplify this task.

Further details on the RTSIM package can be found in (L. Palopoli et al., 2001). All the software components necessary to perform the simulations are covered by GPL license and they can freely be downloaded from the RTSIM web site: <http://rtsim.sssup.it>. Interested readers can take a look at the examples provided with the tool.

4 The case studies

In order to show how the scheduling algorithm can influence the controller performance, a couple of case studies have been considered. In the first case (an inverted pendulum) the controller performance has been evaluated both by simulation and by implementing the

controller on real hardware. The results showed a good accordance between simulation and experiment validating the use of the RTSIM tool. For the second system, consisting of a mobile robot, only simulation results have been produced.

The considered applications cover an important class of control systems, using dataflows from heterogeneous sources with different sampling rates (multirate systems); for these systems an implementation based on a set of cooperating threads turns out to be the most natural choice. Moreover, a very important point of contact between the two applications is the presence of multimedia threads (e.g. image processing) in the control loop. This situation is more and more common in recent control applications and it appears as particularly interesting, since a design based on worst case execution times, entails slow sampling rates thus leading to a poor control performance. The assessment of the influence of the scheduling choices on the system performance has been based on well-defined metrics. Following the directions in (K.G. Shin et al., 1985; D. Seto et al., 1996), the following quadratic cost function has been introduced:

$$\mathcal{J} = E\left[\int_0^{+\infty} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u}) dt \right], \quad (3)$$

where \mathbf{x} and \vec{u} represent the appropriate state-space vector and actuator command, Q, R are two positive definite weighting matrices, and $E[\]$ denotes the statistical expectation value. Actually, since the execution times are stochastic variables, an average of the cost function over several runs is needed. Moreover, in both cases the performance assessment is aimed at finding a good (possibly the best) digital implementation of a continuous-time (ideal) controller. Thus, an unbiased performance estimation is obtained subtracting from the cost function (3) the value \mathcal{J}_c which could be attained by the continuous-time controller:

$$\Delta \mathcal{J} = E\left[\int_0^{+\infty} (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u}) dt \right] - \mathcal{J}_c. \quad (4)$$

4.1 Case study 1: inverted pendulum stabilization

4.1.1 Description of the system.

The case study consists in a state-space stabilization of a classical inverted pendulum plant. The state vector is composed of $[x, \dot{x}, \theta, \dot{\theta}]^T$, where x is the linear position of the cart, \dot{x} is its velocity, θ is the pendulum angle and $\dot{\theta}$ is the angular velocity of the pendulum. The system is controlled by a dynamic feedback controller: the state vector reconstruction is accomplished by an observer from the measured system outputs (cart position and pendulum angle) and then the reconstructed state is fed back by means of a constant matrix gain. The angular position of the pendulum is acquired through a potentiometer at a high sampling rate, while the position of the cart is obtained through the tracking of a digital camera image, and hence at a much slower rate. This leads to a multirate control scheme. Asymptotic stability and convergence properties of the proposed control scheme, as well as an effective design for the observer, can be derived from the application of standard techniques for multirate design (P. Colaneri et al., 1992; S. Bittanti et al., 1984). However, these techniques do not cater with stochastic effects due to execution delays and scheduling jitter. In the following, effective scheduling solutions to minimize the performance degradation due to these effects will be shown on a multithread implementation of the system. The implementation is based on two control threads (except for those involved in data logging and man/machine interface): the first one (`Thread1`) is dedicated to the extraction of the cart position, while the second one (`Thread2`) manages the acquisition of the angular position from the sensor, as well as the state reconstruction and the control actuation through asynchronous communication channels. The architectural design of the feedback controller, comprised of all threads, is shown in Figure .8.

4.1.2 Simulation and experimental results

The goal of the studied control application was to locally stabilize the origin $[x, \dot{x}, \theta, \dot{\theta}]^T$ of the system state-space given the cart position, pendulum angle and their velocities. In the simulations reported in this section the pendulum initial state was set to $[-0.1, 0, 0, 0]$. Trajectories arising from different initial states were also examined both with the RTSIM

simulator and with the experimental setup; in all cases, the obtained results are similar to those reported below for the chosen configuration.

The control algorithm for the experimental device was implemented using the SHARK real-time operating system (P. Gai et al., 2001), expressly designed to permit the easy use of different scheduling policies (WEB site <http://shark.sssup.it> for details). The simulations were carried out by RTSIM using the dynamical parameters of the experimental device, and also considering the effects of the computation delays. To include the latter effects, the probability distribution function (PDF) of the execution times were first profiled from several experimental executions (for example, Fig. .9 plots the PDF for Thread1), and then imported into the simulator through a suitable class for random variables' generation. The good accordance between the dynamics resulting from the experiments and from the simulations can be evaluated looking at Figure .10.

Every value of the cost function was computed as an average of more than 20 executions, ensuring a confidence interval lower than the 3% of the mean value.

As it can be seen from Fig. .9, the execution times of the image processing thread are spread over a wide range of values. In particular, it is evident that, in this case, the mean calculation time \bar{c}_i is much smaller than the WCET C_i . The system performance was examined using different scheduling algorithms. In each case the parameters used to schedule the control threads have been varied, whereas the parameters of the remaining threads have been left unchanged.

In a first experiment, the EDF priority assignment has been used. In this case, as shown earlier, the real-time scheduling theory guarantees that no deadline is missed as long as test (1) is enforced in choosing the threads' activation periods: $\sum \frac{C_i}{T_i} \leq 1$. In order to show the performance degradation due to this conservative choice, also scheduling alternatives based on a "soft" EDF policy were used. In particular periods were chosen so that the worst case load is greater than 1, but the mean system load is less than 1, thus ensuring the absence of permanent overloads.

The system performance was measured varying the period T_1 of the tracking thread and keeping fixed the period T_2 of Thread2. Figure .11 plots the cost function, obtained both

from simulations and experimental runs. In all the experiments, the WCETs for **Thread1** and **Thread2** are $C_1 = 12.1ms$, and $C_2 = 0.5ms$ respectively; the CPU utilization assigned to the remaining threads is 0.23, hence the guarantee test (1) imposes $C_1/T_1 + C_2/T_2 = 0.77$ ($U_i = 1$ for EDF). In the reported experiments the following periods are used: $T_1 = 23.0ms, T_2 = 2.0ms$, $T_1 = 27.9ms, T_2 = 1.5ms$, and $T_1 = 44.4ms, T_2 = 1.0ms$ for the hard real-time assignments; in the soft EDF case the same values for T_2 were used, while T_1 was allowed to vary.

As a first consideration, the behaviour of the cost function evaluated using the RTSIM simulator is essentially the same as in experiments. The value of the cost function $\Delta\mathcal{J}$ is higher in the experimental plots, mainly because of unmodeled dynamics and sensor noise. From the figure it is clear that, for increasing activation rates, the controller's performance improves until a limit condition is reached.

When the mean system load approaches 1, overload conditions become longer and longer making the schedule completely unpredictable; as a result, the controller performance degrades. Furthermore, in Figure .11, a cross on each curve marks the lower bound for the choices of T_1 (T_2 is fixed) making the worst case load less than 1 (hard schedulability). Considering the experimental plot corresponding to $T_2 = 1.0ms$ the best value attained in this situation is 116.6, whereas the minimum value is 60.8. This is a clear evidence of the price to be paid for the strict respect of every deadline.

In the second experiment, the system performance has been analyzed using the CBS scheduling policy, in order to compare the results with those previously obtained from soft EDF scheduling. The activation periods were kept fixed, while varying the bandwidth of the two control threads (the total bandwidth of the remaining threads was again set to 0.23). Figure .12 plots the cost function $\Delta\mathcal{J}$ with respect to the bandwidth assigned to **Thread2**, for $T_1 = 11.0ms$ and $T_2 = 2.0ms$, obtained both from simulations and experiments.

The horizontal lines represent the value of cost function obtained using soft EDF scheduling and the same periods (see Fig. .11). It is clear from the figure that the choice of the bandwidth assigned to the control threads can greatly modify the system performance; it

can be seen that a fine bandwidth tuning of the CBS bandwidth brings to the same results as the EDF soft (actually slightly better). However, it is important to point out that the properties of the CBS algorithm bring to more reliable results. Namely, it is possible to decide, during the system design, the probability of a deadline miss for a thread starting from the distribution of its execution time; this result is guaranteed independently of the behaviour of other threads in the system. On the contrary, the lack of a precise knowledge/control of the timing behaviour of tasks in overload conditions under EDF makes the results more sensitive to the particular tasks configuration. A more effective control on the overload conditions becomes a prominent performance factor when there is the need for protecting as much as possible some important threads from deadline misses.

4.2 Case study 2: a mobile robot

4.2.1 Modeling and control

Consider a mobile robot modeled as a two wheels nonholonomic vehicle, endowed with an external sensor (e.g. a camera sensor (F. Conticelli et al., 1999; F. Conticelli, B. Allotta and P. K. Khosla, 1999)) able to determine the Cartesian relative position of a moving target, whose coordinates with respect to an inertial frame $\langle w \rangle$ are $\mathbf{x}_g = [x_g \ y_g]^T$. The analytical dynamics of the vehicle is determined using the principle of virtual displacements and the d'Alembert-Lagrange equations (J.I. Neimark and N.A. Fufaev, 1972). In particular, the state of the mechanical system is given by five generalized coordinates (see Figure .13): the point of intersection $[x, \ y]^T$ (expressed in the inertial frame $\langle w \rangle$) of the symmetry axis of the carriage with the wheel axle, the orientation angle θ between the symmetry axis of the carriage and a fixed axis of the frame $\langle w \rangle$ and the angles of rotation of the left-hand wheel ϕ_l and of the right-hand one ϕ_r .

The *control problem* is formulated as follows: the mobile robot has to track a moving target until the relative displacement, expressed in a frame attached to the robot $\langle c \rangle$, becomes a desired value $\mathbf{x}_r^{(d)} = [x_r^{(d)} \ y_r^{(d)}]^T$. The relative displacement is given by $\mathbf{x}_r = [x_r \ y_r]^T = R [x_g - x \ y_g - y]^T$, being R the rotation matrix between the frame attached to

the robot $\langle c \rangle$ and the inertial frame $e \langle w \rangle$

$$R = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}. \quad (5)$$

An analytical formulation for the control law is: $\lim_{t \rightarrow \infty} \tilde{\mathbf{x}} = [\tilde{x}_r \ \tilde{y}_r]^T = \mathbf{x}_r - \mathbf{x}_r^{(d)} = 0$.

The tracking control problem is solved using the *backstepping approach* (M. Krstić et al., 1995) which, in a natural manner, leads to a nonlinear feedback controller organized in multiple levels. Details on the derivation of the nonlinear controller for this system can be found in (L. Palopoli et al., 2000a). For the purposes of the present work, it is sufficient to discuss the structure of the hierarchical controller and of its digital multirate implementation. Introduce the vectors $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_2^*$ where $\mathbf{X}_1 = [x_r, y_r, \tilde{x}_r, \tilde{y}_r, \dot{x}_g, \dot{y}_g]^T$, $\mathbf{X}_2 = [\dot{\phi}_r, \dot{\phi}_l]^T$ and $\mathbf{X}_2^* = [\dot{\phi}^*]$, where $\dot{\phi}^*$ denotes reference values for the angular velocities of the wheels. The control law is organized in two levels. The outermost level receives as its input the quantities contained in the \mathbf{X}_1 vector and produces a reference angular velocity according to a continuous map \mathcal{C}_1 . The innermost level uses a continuous map \mathcal{C}_2 to compute the value of the torques to be applied to the motors. Inputs to \mathcal{C}_2 are the reference value for the wheels' angular velocities \mathcal{C}_1 , the value of the actually measured velocities \mathbf{X}_2 and inputs applied to the outermost subcontroller.

The overall structure of the control law can be written as

$$\begin{aligned} \mathbf{X}_2^* &= \mathcal{C}_1(\mathbf{X}_1) \\ \boldsymbol{\tau} &= \mathcal{C}_2(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_2^*) \end{aligned} \quad (6)$$

here $\boldsymbol{\tau}$ is the vector of the motor torques and the maps $\mathcal{C}_1, \mathcal{C}_2$ can be derived as shown in (L. Palopoli et al., 2000a). This notation enables us to expose the general structure of controllers developed using the backstepping approach. As a matter of fact, this triangular cascade of maps could be generalized for as many levels as required by different applications. Moreover, observing that the \mathcal{C}_i maps are static, the controller results to be stateless and its digital multirate implementation is straightforward.

Assume that \mathbf{X}_1 and \mathbf{X}_2 be sampled with the periods T_1 and T_2 respectively where $T_1 \geq T_2$. Therefore, the subcontrollers will be fed with the piecewise constant signals

$\mathbf{X}_1^d, \mathbf{X}_2^d$ where:

$$\mathbf{X}_i^d(t) = \mathbf{X}_i(kT_i), \forall t \in [kT_i, (k+1)T_i), i = 1, 2 \quad (7)$$

In general, each subcontroller

$$\mathcal{C}_i(\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_{i-1}, \mathbf{X}_2^*, \dots, \mathbf{X}_{i-1}^*), \quad i = 1, \dots, N$$

is implemented by a periodic thread \mathcal{T}_i , $i = 1, \dots, N$, which, upon every activation, reads its input data, computes the output value and writes it out onto memory buffers or output ports. The produced value is therefore held until the next activation of the thread. Each thread's activation rate is assumed to be equal to the fastest rate of its sensor data. Therefore, in the considered case study, the outermost thread is activated with a period equal to T_1 whereas the innermost one is activated with a period equal to T_2 (see Figure .14).

As can be seen the sensor data undergo a double sampling. For example the vector \mathbf{X}_1 is sampled with periodicity T_1 yielding the signal \mathbf{X}_1^d ; afterwards the threads sample \mathbf{X}_1^d each at its own period (i.e. \mathcal{T}_1 with period T_1 and \mathcal{T}_2 with period T_2). The same remark applies to the reference values \mathbf{X}_2^* . Notice that the periodic threads are completely asynchronous one another; in other words there is not any precedence constraint for their execution. The communication of the reference values between the different levels can be achieved by any kind of lock-free protocol (e.g. see (D. Clark, 1989)).

4.2.2 Simulation Results

In order to evaluate the influence of the scheduling choices on the control performance, the closed loop dynamics of the system has been simulated by using the RTSIM simulator. The robot controller uses data having different timing characteristics. In particular the outer loop uses data coming from an external sensor (e.g. a camera) while the inner loop uses data sampled from a tachometer sensor. The weighting matrices used in the cost function (4) were chosen as follows: $Q = I$, $R = 10^{-3}I$

A preliminary study of this system (L. Palopoli et al., 2000a), performed assuming zero

computation delays, revealed that the activation rate of thread \mathcal{T}_1 is the one that influences most the performance of the system.

The execution times of \mathcal{T}_1 were assumed to be uniformly distributed between the values $c_1 = 40ms$ and $C_1 = 100ms$; the bounds chosen for \mathcal{T}_2 were $c_2 = 1ms$, $C_2 = 2ms$. In order to model computation activities different from the control threads - for example data logging - a periodic thread \mathcal{T}_3 has been used having a fixed computation time $C_3 = c_3 = 4ms$ and an activation period $T_3 = 20ms$. This activity “steals” a processor utilization factor $u_3 = \frac{c_3}{T_3} = 0.2$.

The vehicle is initially located at the origin of the inertial frame and is required to track a target located at $\mathbf{x}_g = [3 \ 1.5]^T$. The final displacement in the robot frame $\langle c \rangle$ is required to be $\mathbf{x}_r^{(d)} = [0.2 \ 1]^T$. Every value for the cost function was computed averaging through twenty runs and yielding a 90% confidence interval lower than 9% of the obtained average. Simulation plots for the cost function are reported in Figure .15. Each plot is obtained computing the cost function with respect to the choice of T_1 , while T_2 is kept constant. As in the first case study a cross on each plot is used to mark the lower bound for the choice of T_1 respecting the EDF schedulability test (1). From the figure, it is evident that conservative choices for the activation periods lead to poor performance. As an example, consider the curve obtained by choosing $T_2 = 4ms$: pushing the activation period beyond the hard schedulability limit pays off a 50% performance improvement. On the contrary, if T_1 does not even respect the relaxed test (obtained using the mean execution times of tasks) the schedule becomes completely unpredictable and the performance worsens (the curves have been interrupted when this situation occurs).

As explained above, the RR approaches allow to obtain a more reliable management of the overload situations, permitting to decide the importance of threads independently from their activation periods. To show the effectiveness of this approach, a set of simulations using the CBS scheduler have been performed. Figure .16 shows the results when the periods of the control tasks are kept fixed and their assigned bandwidth is made to vary. In particular, the chosen periods were $T_1 = 200ms$ and $T_2 = 6.5ms$. The thread \mathcal{T}_3 was assigned a fixed bandwidth of 20 %. Consequently the bandwidths assigned to \mathcal{T}_1 and \mathcal{T}_2

respect the condition $B_1 + B_2 = 0.8$. The chart shows that the performance improves as more importance is attached to \mathcal{T}_1 (by raising B_1), until a local minimum is reached. If B_1 is increased beyond the local minimum the performance begins to degrade since the bandwidth B_2 reserved to the \mathcal{T}_2 thread becomes too little. Notice that the minimum is reached at $B_1 \approx 0.65$, $B_2 \approx 0.15$, that is $B_1 > B_2$. This effect can be observed also with other frequency choices and on other trajectory and it is a remarkable evidence that the \mathcal{T}_1 thread has a greater importance than \mathcal{T}_2 . Another interesting effect is that the cost function does not vary lowering B_1 below 0.55. To understand this point, recall that in the considered situation there are only three threads in the system: the \mathcal{T}_1 , \mathcal{T}_2 control threads, and a dummy thread \mathcal{T}_3 . A property of the CBS scheduler is that it is possible to guarantee all the jobs of an individual thread if it is provided with a bandwidth greater or equal than to the ratio between the maximum execution time and the activation period. For the \mathcal{T}_3 thread, it holds $B_3 = 0.2 \geq \frac{C_3}{T_3}$; thus all of its jobs are guaranteed to terminate within the deadline. The jobs of \mathcal{T}_2 enjoy the same property if $B_2 \geq \frac{C_2}{T_2}$; in this case a conservative choice is $B_2 \geq 0.31$, that is $B_1 \leq 0.49$. Lowering B_1 below 0.49 do not give any further advantage to \mathcal{T}_2 . On the other hand, decreasing B_1 exposes \mathcal{T}_1 to being slowed down by other threads. However in this case there is no other thread to take advantage from this effect. Hence, \mathcal{T}_1 executes in background with respect to \mathcal{T}_2 , \mathcal{T}_3 independently from its assigned bandwidth and the cost function remains constant.

A visual assessment of performance using different scheduling policies can be made by looking at the dynamics for the $\tilde{\mathbf{x}}_r$ variables and for the torques, for some of the simulations in Figure .17. As it is possible to see, the evolution obtained with the CBS scheduler is the closest to the ideal (continuous time) curve. The advantages are even more evident if attention is restricted to the output dynamics. If different weights were chosen for the cost function penalizing the command dynamics, the quantitative improvement using CBS would have been higher.

5 Conclusions and Future Work

In this paper a novel scheduling policy for realizing software architectures targeted to multithread real-time controllers has been presented. Two meaningful cases have been reported to show that the performance of classical real-time scheduling approaches, based on a strict respect of every deadline, can be poor, if compared to soft real-time approaches able to tolerate transient overload situations. Practical evidence of control performance improvements by using the real-time controller simulator RTSIM has been given.

Future work will be aimed at achieving a deeper analytical insight into this problem. In particular, open analytical issues are the derivation of stability conditions in the case of real-time overload situations based on optimistic EDF scheduling and RR techniques, and the analysis of control robustness with respect stochastic perturbations (delays, jitter) introduced by the real-time implementation. The final goal is to devise a nonlinear optimal control design procedure and a software tool able to produce either the control law and the scheduling parameters to be assigned to each control thread.

References

- L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real Time Systems Symposium*, Madrid, Spain, December 1998.
- L. Abeni and G. Buttazzo. Qos guarantee using probabilistic deadlines. In *Proceedings of the IEEE Euromicro Conference on Real-Time*, York, England, June 1998.
- K. J. Åström and B. Wittenmark, *Computer Controlled Systems (3rd edition)*, Prentice Hall, Englewood Cliffs, N.J., 1997.
- P. Albertos, A. Crespo et al., RT control scheduling to reduce control performance degrading. *Proceedings of the 39th IEEE Conference on Decision and Control* Sidney, December 2000.
- T.P. Baker, A Stack-Based Policy for Real-time Processes, In *Proc. of IEEE Real-Time Systems Symposium*, Dec. 1990.

- S. Bittanti, P. Colaneri and G. Guardabassi, Periodic solutions of periodic Riccati equations, *IEEE Trans. Aut. Contr.*, 29(7), 1984.
- T. Chen and B. Francis, *Optimal Sampled-Data Control Systems*, Springer, 1995.
- D. Clark, HIC: an Operating System for Hierarchies of Servo Loops, *Proc. of IEEE International Conference on Robotics and Automation*, 1989.
- P. Colaneri, R. Scattolini and N. Schiavoni, The LQG problem for multirate sampled-data systems, *IEEE Trans. Aut. Contr.*, 37(5), 1992.
- F. Conticelli, B. Allotta, C. Colombo, Hybrid visual servoing: a combination of nonlinear control and linear vision, *Robotics and Autonomous Systems*, 29 (1999), pp. 243-256.
- F. Conticelli, B. Allotta, P. K. Khosla, Image-based visual servoing of nonholonomic mobile robots, *IEEE Conference on Decision and Control*, Phoenix, Arizona, USA, december 1999.
- J. Eker and A. Cervin, A Matlab toolbox for Real-time and Control Systems Co-Design *Proc. of The Real-Time Computing Systems and Applications*, Hong Kong, December 1999.
- P. Gai, L. Abeni, M. Giorgi and G. Buttazzo, New Kernel Approach for Modular Real-Time systems Development, *Proceedings of the 13th IEEE Euromicro Conference on Real-Time Systems*, June 2001.
- Bo Hu and A.N. Michel, Some Qualitative Properties of Multirate Digital Control Systems, *IEEE Trans. Automat. Contr.*, vol. 44, no. 4, pp. 765-770, April 1999.
- L. Kleinrock, *Queuing Systems*, Wiley-Interscience, 1975
- Klein, M.H., M. Gonzalez-Harbour et al. "A Practitioners' Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems." Kluwer Academic Publishers, Boston 1993
- M. Krstić and I. Kanellakopoulos and P. Kokotović, *Nonlinear and Adaptive Control Design*, John Wiley & Sons, Inc., 1995.
- J.P. Lehoczky, L. Sha and J.K. Strosnider, Enhanced periodic responsiveness in hard real-time environment. In *Proc. of the IEEE Real-Time Systems Symposium*, Dec. 1987.

- C.L. Liu and J. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.
- A.N. Michel and Bo Hu, Towards a Stability Theory of General Hybrid Dynamical Systems, *Automatica*, 35, pp. 371-384, 1999.
- S. Monaco and D. Normand-Cyrot, *Nonlinear Systems*, vol. 3, chap. 5, Edit. A. J. Fossard and D. Normand-Cyrot, Chapman & Hall, 1996.
- J.I. Neimark and N.A. Fufaev, *Dynamics of Nonholonomic Systems*, American Math. Society, Providence, Rhode Island, 1972.
- Nilsson, J., B. Bernhardsson, and B. Wittenmark, Stochastic Analysis and Control of Real-Time Systems with Random Time Delays, *Automatica*, Vol 34:1, 57-64, 1998.
- L. Palopoli, F. Conticelli, B. Allotta, Multi-level stabilizing control of nonholonomic vehicles and its multirate digital implementation, *IEEE Conference on Robotics and Automation*, Stanford, USA, April 2000.
- L. Palopoli, L. Abeni, F. Conticelli, M. Di Natale, and G. Buttazzo Real-Time control system analysis: an integrated approach, In *Proceedings of the IEEE Real Time Systems Symposium*, Orlando, Florida, December 2000.
- L. Palopoli, G. Lipari, Luca Abeni and others, A tool for simulation and fast prototyping of embedded control systems *Proceedings of the ACM SIGPLAN workshop on language, compilers and tools for embedded systems (LCTES01)*, Snowbird, U.S.A., June 2001.
- R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.
- D. Seto, J.P. Lehoczky, L. Sha and K.G. Shin, On Task Schedulability in Real-Time Control Systems In Proc. of IEEE Real Time System Symposium, December 1996
- L. Sha, R. Rajukumar and J.P. Lehoczky, Priority Inheritance: An Approach to Real-Time Synchronization, *IEEE Trans. on Computers*, 39, 9, Sept. 1990.
- K.G. Shin, C.M. Krishna and Y.H. Lee, A unified method for evaluating real-time computer controllers and its application *IEEE Transactions on Control System Technology*, 3, 2, June 1995.

- K. G. Shin and Xianzhong Cui, Computing time delay and its effects on Real-Time Control Systems *IEEE Transactions on Control System Technology*, 3, 2, June 1995.
- B. Sprunt, J.P. Lehoczky and L. Sha, Aperiodic task scheduling for hard real-time systems. *Journal of Real-Time Systems*, 1, July 1989.
- J. Stankovic, K. Ramamritham, M. Spuri, and G. Buttazzo, "Deadline Scheduling for Real-Time Systems", Kluwer Academic Publishers, Boston, 1998.
- I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, and C.G. Plaxton, A proportional share resource allocation algorithm for real-time, time-shared systems, In *Proceedings of the IEEE Real Time Systems Symposium*, December 1996.
- J. Tornero, J. Salt and P. Albertos LQ optimal control for multirate sampled data systems IFAC World Congress, Beijing 1999.

Appendix

Proof of Theorem 1

To prove the theorem, it is shown that a CBS with parameters (Q_s, T_s) cannot demand a CPU bandwidth greater than $U_s = Q_s/T_s$. That is, if $D_s(t_1, t_2)$ is the processor demand of the CBS in the interval $[t_1, t_2]$, it holds:

$$\forall t_1, t_2 \in N : t_2 > t_1, \quad D_s(t_1, t_2) \leq \frac{Q_s}{T_s}(t_2 - t_1).$$

Recall that under a CBS a job J_j is assigned an absolute time-varying deadline d_j which can be postponed if the thread requires more than the reserved bandwidth. Thus, each job J_j can be thought as consisting of a number of chunks $H_{j,k}$, each characterized by a release time $a_{j,k}$ and a fixed deadline $d_{j,k}$. An example of chunks produced by a CBS is shown in Figure .3. To simplify the notation, all the chunks generated by a server are indicated with an increasing index k (in the example of Figure .3, $H_{1,1} = H_1$, $H_{1,2} = H_2$, $H_{2,1} = H_3$, and so on).

The release time and the deadline of the k^{th} chunk generated by the server will be denoted by a_k and d_k , c will indicate the actual budget and n the number of requests in server

queue. These variables are initialized in the following manner:

$$d_0 = 0$$

$$c = 0$$

$$n = 0$$

$$k = 0$$

Using these notations, the server behavior can be expressed as in Figure .1.

Indicating with e_k the server time demanded in the interval $[a_k, d_k]$ (that is, the execution time of chunk H_k), it holds:

$$\forall t_1, t_2, \quad \exists k_1, k_2 : \quad D_s(t_1, t_2) = \sum_{k: a_k \geq t_1 \wedge d_k \leq t_2} e_k = \sum_{k=k_1}^{k_2} e_k.$$

If $c_s(t)$ is the server budget at time t and f_k is the time at which chunk H_k ends to execute, we can see that $c_s(f_k) = c_s(a_k) - e_k$, while $c_s(a_{k+1})$ is calculated from $c_s(f_k)$ in the following manner:

$$c_s(a_{k+1}) = \begin{cases} c_s(f_k) & \text{if } d_{k+1} \text{ was generated by Rule 2} \\ Q_s & \text{if } d_{k+1} \text{ was generated by Rule 1 or 3.} \end{cases}$$

Using these observations, the theorem can be proved by showing that:

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s}.$$

The demonstration proceeds by induction on $k_2 - k_1$, using the algorithmic definition of CBS shown in Figure .1.

Inductive base. If in $[t_1, t_2]$ there is only one active chunk ($k_1 = k_2 = k$), two cases have to be considered.

Case a: $d_k < a_k + T_s$.

If $d_k < a_k + T_s$, then d_k is generated by Rule 2, so $a_k + \frac{c_s(f_{k-1})}{Q_s} T_s < d_k$ and $a_k = f_{k-1}$,

that is

$$a_k + \frac{c_s(a_k)}{Q_s} T_s < d_k.$$

Being $c_s(f_k) = c_s(a_k) - e_k = c_s(a_k) - D_s(a_k, d_k)$, it holds

$$a_k + \frac{D_s(a_k, d_k) + c_s(f_k)}{Q_s} T_s < d_k$$

hence

$$D_s(a_k, d_k) + c_s(f_k) < (d_k - a_k) \frac{Q_s}{T_s}.$$

Case b: $d_k = a_k + T_s$.

If $d_k = a_k + T_s$, then $D_s(a_k, d_k) + c_s(f_k) = e_k + c_s(f_k) = Q_s$. Hence, in both cases, it holds:

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) = D_s(a_k, d_k) + c_s(f_k) \leq (d_k - a_k) \frac{Q_s}{T_s} = (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s}.$$

Inductive step. The inductive hypothesis

$$D_s(a_{k_1}, d_{k_2-1}) + c_s(f_{k_2-1}) \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s}$$

is used to prove that

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s}.$$

Given the possible relations between d_k and d_{k-1} , three cases have to be considered:

- $d_k \geq d_{k-1} + T_s$. That is, d_k is generated by Rule 3 or Rule 1 when $r_j \geq d_{j-1}$.
- $d_k = d_{k-1}$. That is, d_k is generated by Rule 2.
- $d_{k-1} < d_k < d_{k-1} + T_s$. That is, d_k is generated by Rule 1 when $r_j < d_{j-1}$.

Case a: $d_{k_2} = d_{k_2-1} + T_s$.

In this case d_{k_2} can be generated only by Rule 1 or 3. Adding e_{k_2} to both sides of the inductive hypothesis, it follows:

$$\sum_{k=k_1}^{k_2-1} e_k + e_{k_2} \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2-1}) + e_{k_2}$$

and, since $c_s(f_k) = c_s(a_k) - e_k$, it holds

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2-1}) + c_s(a_{k_2}) - c_s(f_{k_2}).$$

Since d_{k_2} is generated by Rule 1 or 3, it must be $c_s(a_{k_2}) = Q_s$, therefore:

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2-1}) + Q_s - c_s(f_{k_2})$$

$$\sum_{k=k_1}^{k_2} e_k + c_s(f_{k_2}) \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2-1}) + Q_s \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} + Q_s$$

and finally

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} + Q_s = (d_{k_2-1} + T_s - a_{k_1}) \frac{Q_s}{T_s}$$

$$D_s(a_{k_1}, d_{k_2}) + c_s(f_{k_2}) \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s}.$$

Case b: $d_{k_2} = d_{k_2-1}$.

If $d_{k_2} = d_{k_2-1}$, then d_{k_2} is generated by Rule 2. In this case,

$$\sum_{k=k_1}^{k_2-1} e_k + e_{k_2} \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2-1}) + e_{k_2}$$

but, being $d_{k_2} = d_{k_2-1}$, $c_s(f_{k_2}) + e_{k_2} = c_s(a_{k_2})$ and $c_s(a_{k_2}) = c_s(f_{k_2-1})$ (by Rule 2), it holds:

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s} - c_s(a_{k_2}) + e_{k_2} = (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2})$$

hence

$$D_s(k_1, k_2) + c_s(f_{k_2}) = \sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s}.$$

Case c: $d_{k_2-1} < d_{k_2} < d_{k_2-1} + T_s$.

If $d_{k_2} < d_{k_2-1} + T_s$, d_{k_2} is generated by Rule 1, so $a_{k_2} + \frac{c_s(f_{k_2-1})}{Q_s} T_s \geq d_{k_2-1}$, hence $c(f_{k_2-1}) \geq$

$(d_{k_2-1} - a_{k_2}) \frac{Q_s}{T_s}$. Applying the inductive hypothesis, it follows

$$\sum_{k=k_1}^{k_2-1} e_k + e_{k_2} \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2-1}) + e_{k_2}$$

from which

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2-1} - a_{k_1}) \frac{Q_s}{T_s} - (d_{k_2-1} - a_{k_2}) \frac{Q_s}{T_s} + e_{k_2}$$

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2-1} - d_{k_2-1} - a_{k_1} + a_{k_2}) \frac{Q_s}{T_s} + e_{k_2}.$$

Now, being $e_{k_2} = Q_s - c_s(f_{k_2})$, it holds:

$$\sum_{k=k_1}^{k_2} e_k \leq (-a_{k_1} + a_{k_2}) \frac{Q_s}{T_s} + Q_s - c_s(f_{k_2}) = (a_{k_2} + T - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2})$$

but, from Rule 1 and 3, it follows $d_k = a_k + T$, hence

$$\sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s} - c_s(f_{k_2})$$

and

$$D_s(k_1, k_2) + c_s(f_{k_2}) = \sum_{k=k_1}^{k_2} e_k \leq (d_{k_2} - a_{k_1}) \frac{Q_s}{T_s}.$$

```

When job  $J_j$  arrives at time  $r_j$ 
    enqueue the request in the server pending request queue;
    n = n + 1;
    if (n == 1) /* (the server is idle) */
        if ( $r_j + (c / Q_s) * T_s \geq d_k$ )
            /*-----Rule 1-----*/
            k = k + 1;
             $a_k = r_j$ ;
             $d_k = a_k + T_s$ ;
            c =  $Q_s$ ;
        else
            /*-----Rule 2-----*/
            k = k + 1;
             $a_k = r_j$ ;
             $d_k = d_{k-1}$ ;
            /* c remains unchanged */
When job  $J_j$  terminates
    dequeue  $J_j$  from the server queues;
    n = n - 1;
    if (n != 0) begin to serve the next job in queue with deadline  $d_k$ ;
When job  $J_j$  served by  $S_s$  executes for a time unit
    c = c - 1;
When (c == 0)
    /*-----Rule 3-----*/
    k = k + 1;
     $a_k = \text{actual\_time}()$ ;
     $d_k = d_{k-1} + T$ ;
    c =  $Q_s$ ;

```

Figure .1. The CBS algorithm.

$$M = \begin{pmatrix} \overbrace{U(0) U(0) \dots U(0)}^{Q+1} & 0 & 0 & \dots & \dots \\ U(1) U(1) \dots U(1) & U(0) & 0 & \dots & \dots \\ U(2) U(2) \dots U(2) & U(1) & U(0) & 0 & \dots \\ U(3) U(3) & U(3) & U(2) & U(1) & U(0) & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

Figure .2. Matrix for computing the probabilistic deadline PDF.

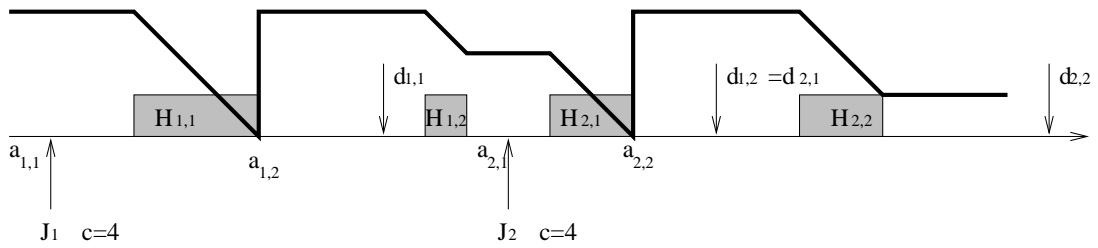


Figure .3. Serving some jobs divided in chunks.

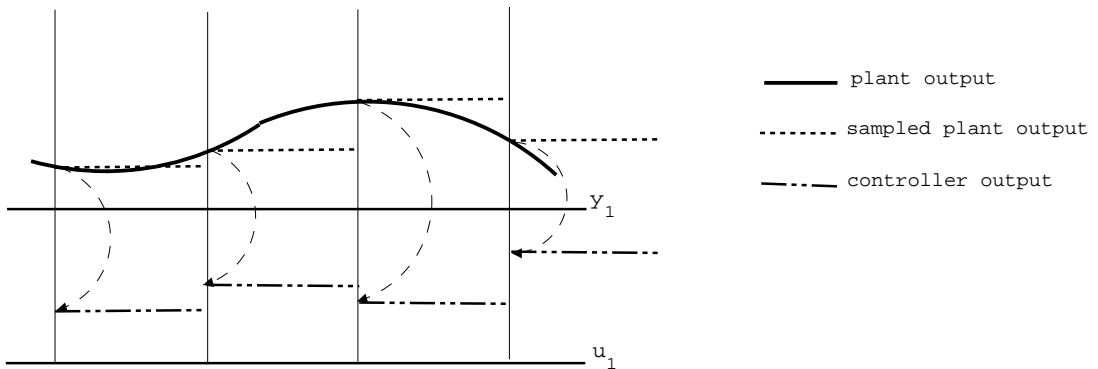


Figure .4. Timing of an ideal digital controller: the execution time is neglected.

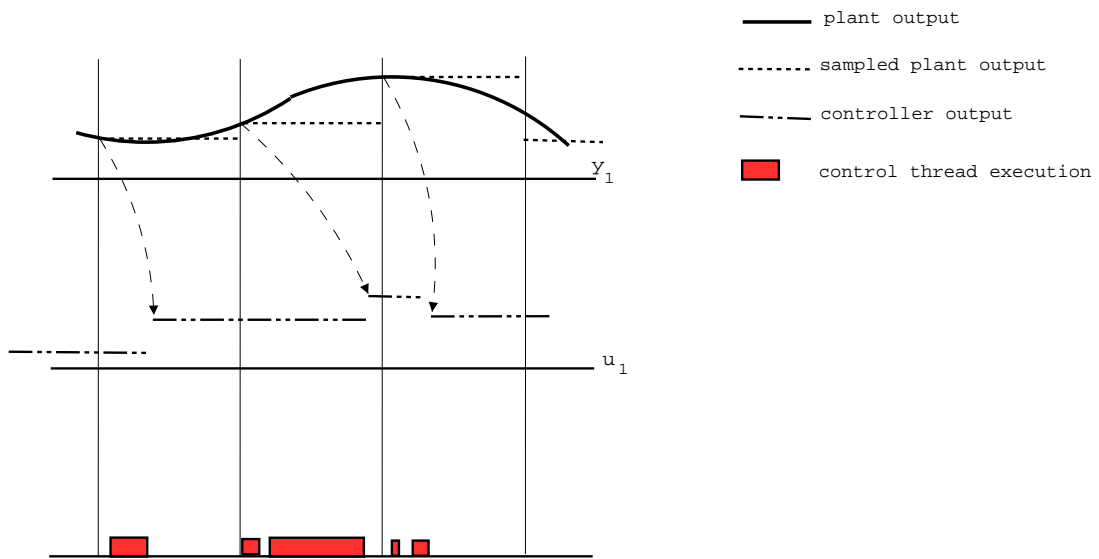


Figure .5. Timing of a real digital controller: the execution time and scheduling jitter determine stochastic delays in the output release.

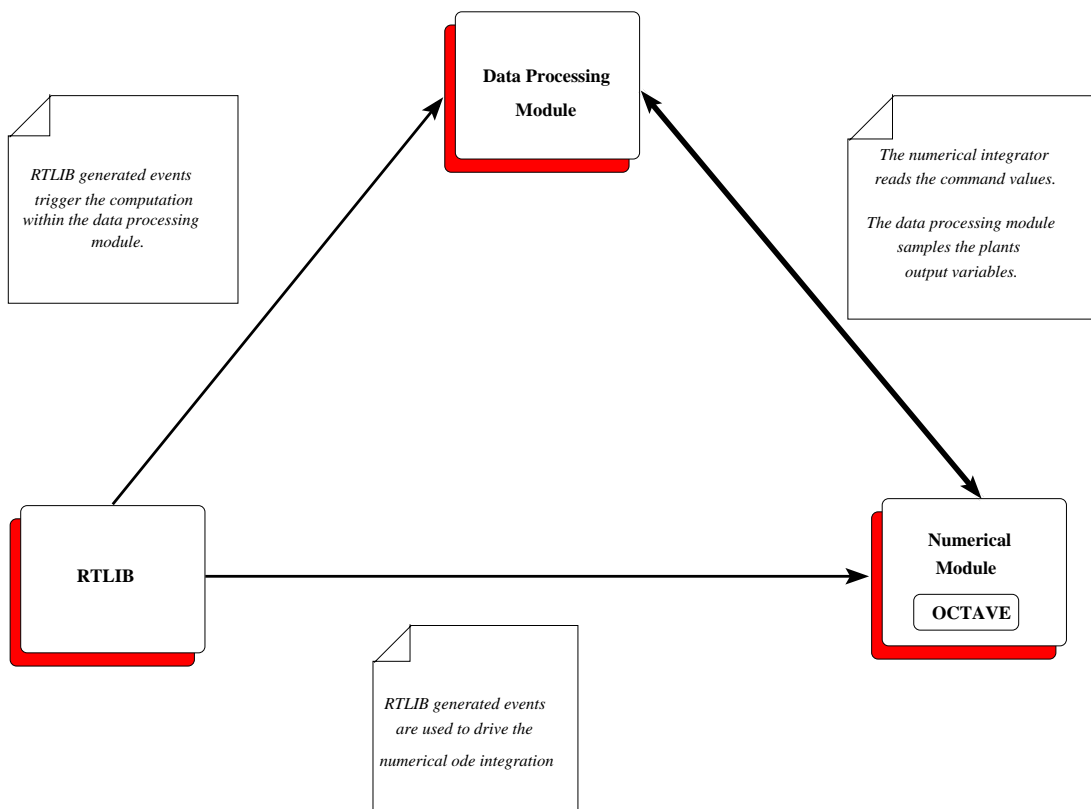


Figure .6. Structural view on the RTSIM software package.

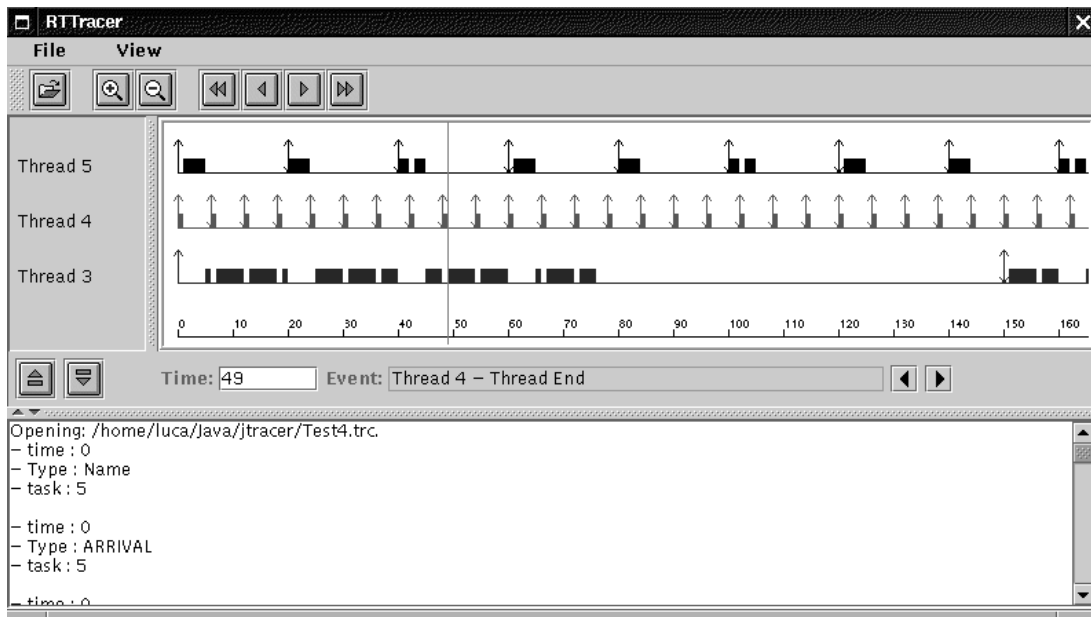


Figure .7. An example of a thread schedule reconstructed using the RTSIM simulator.

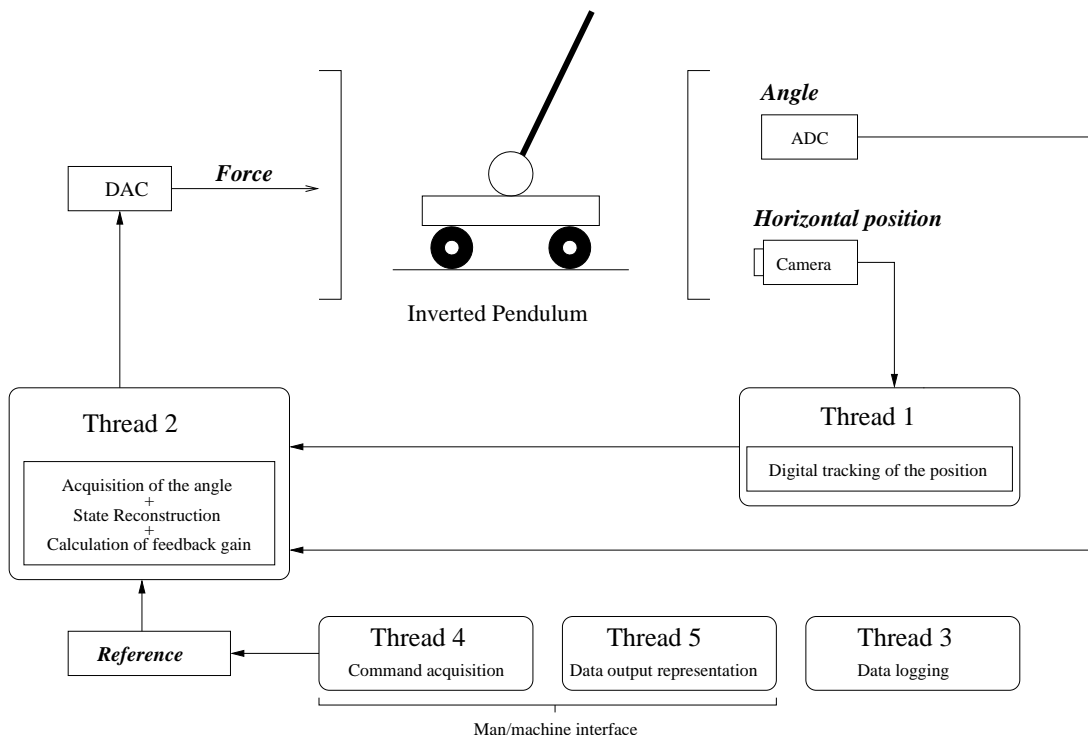


Figure .8. Architectural design of the inverted pendulum controller.

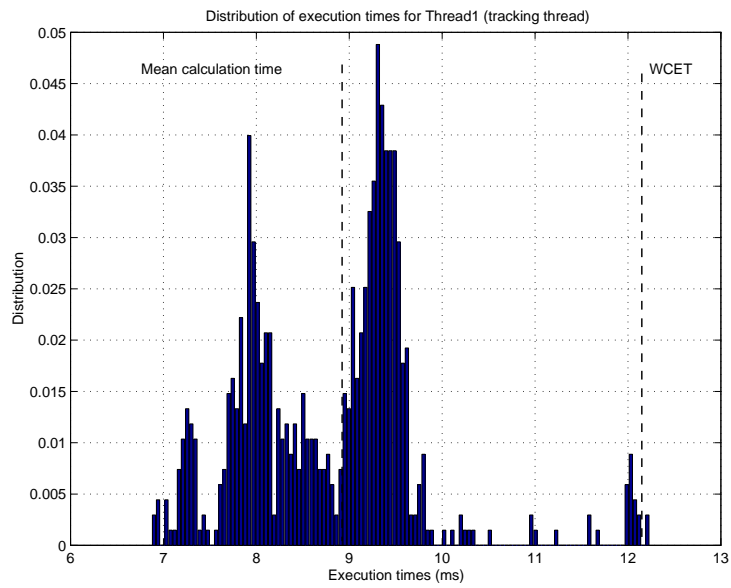


Figure .9. Statistical distribution of execution times for the tracking thread.

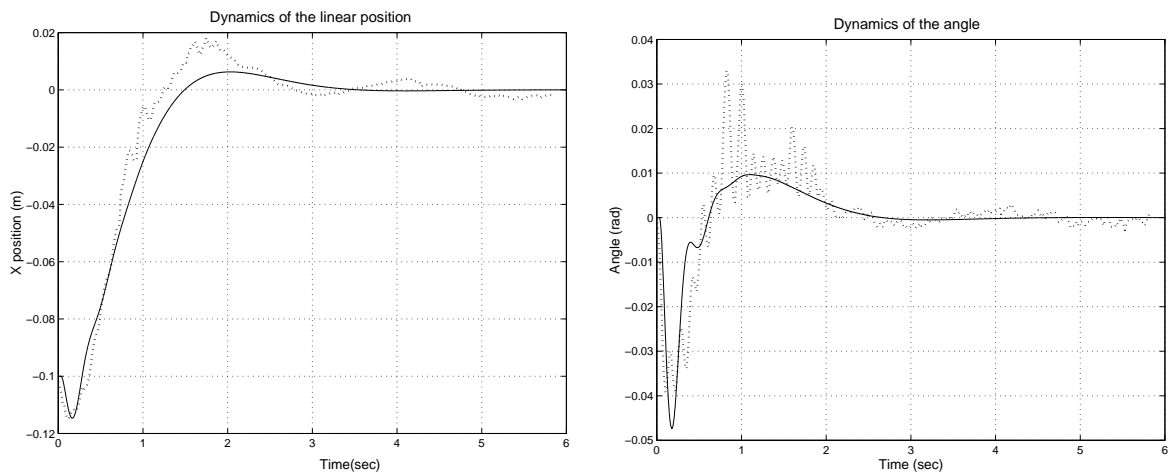


Figure .10. Dynamics resulting from experiments on the real system (dotted line) and from simulations (solid line).

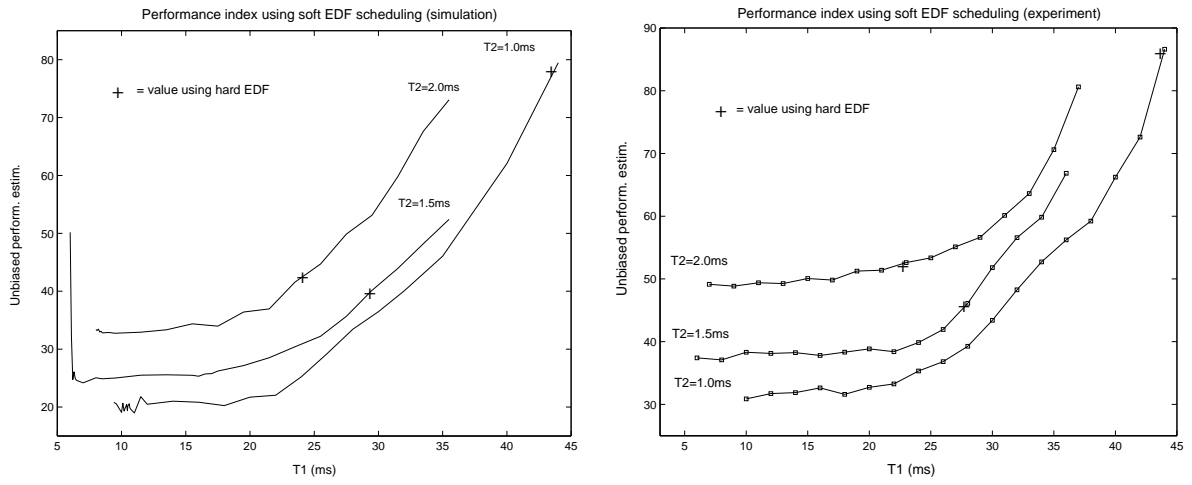


Figure .11. Inverted Pendulum Cost function vs T_1 (T_2 fixed) using EDF: simulations (left) and experiments (right).

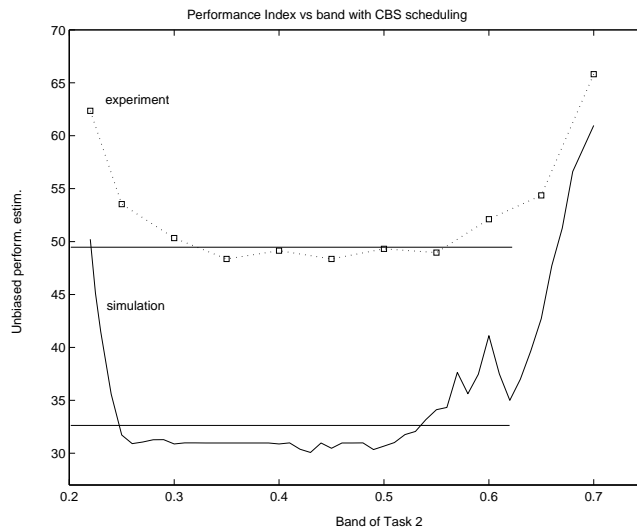


Figure .12. Cost Function vs. bandwidth using CBS: $T_1 = 10.0ms$, $T_2 = 2.0ms$.

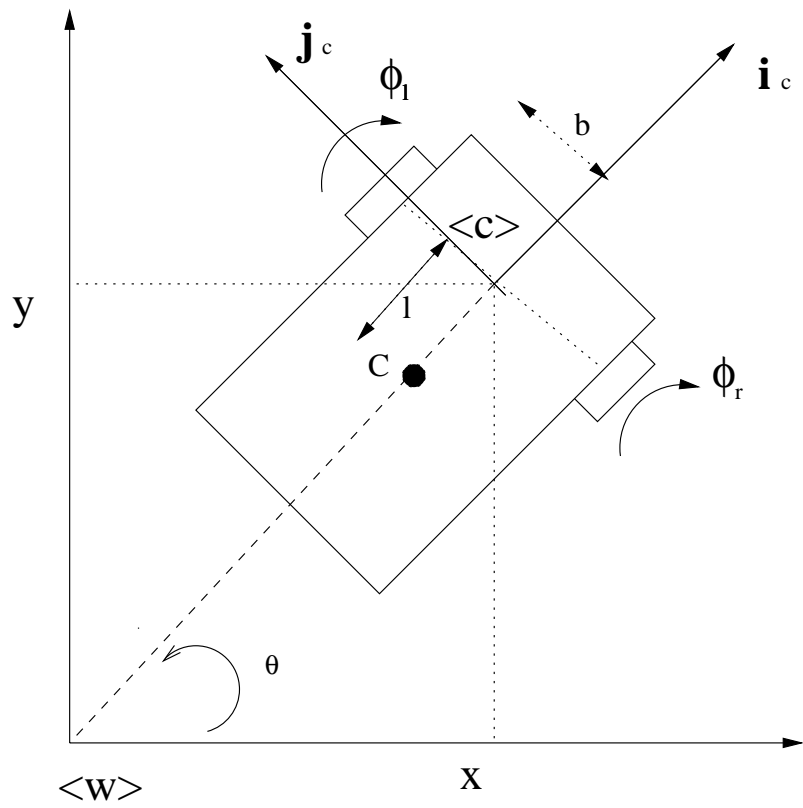


Figure .13. Schematic representation of the nonholonomic vehicle.

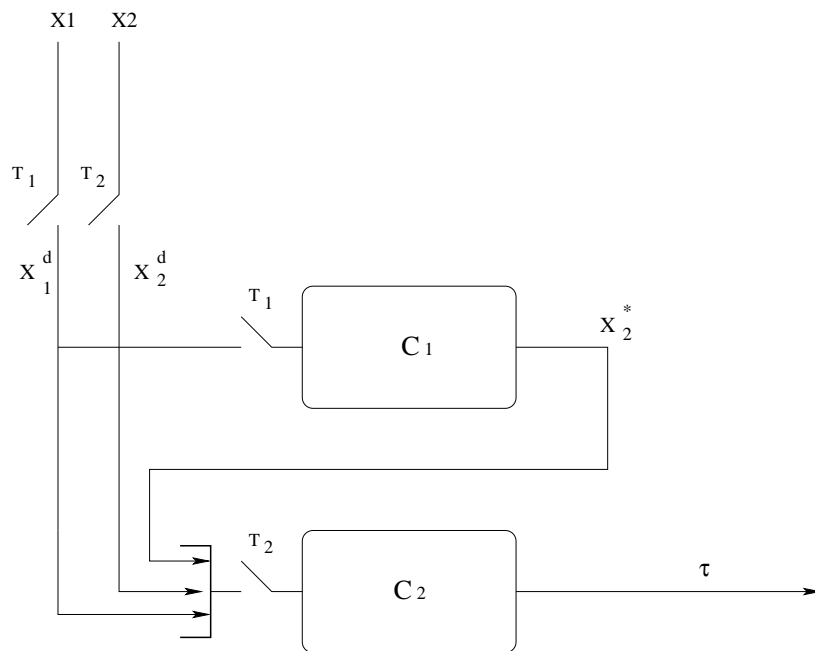


Figure .14. Implementation scheme for the discrete-time multithread controller.

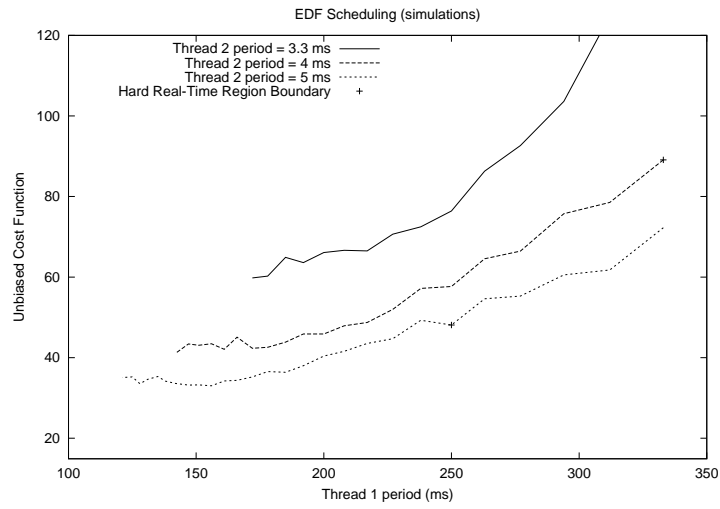


Figure .15. Mobile Robot Cost function vs T_1 (T_2 fixed) using EDF.

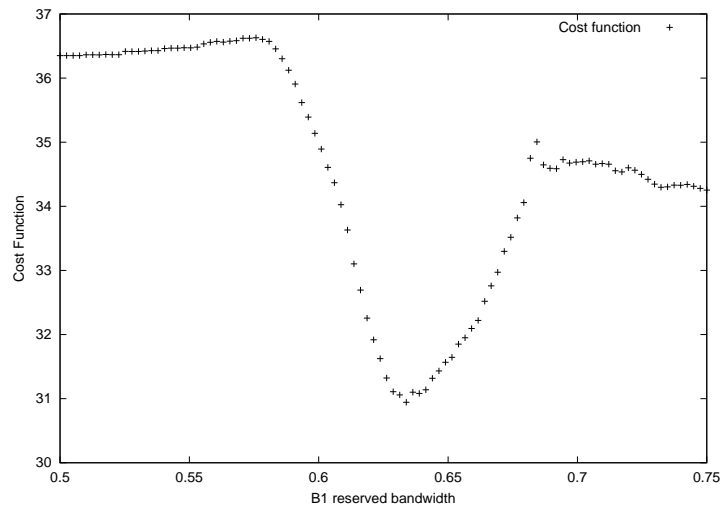
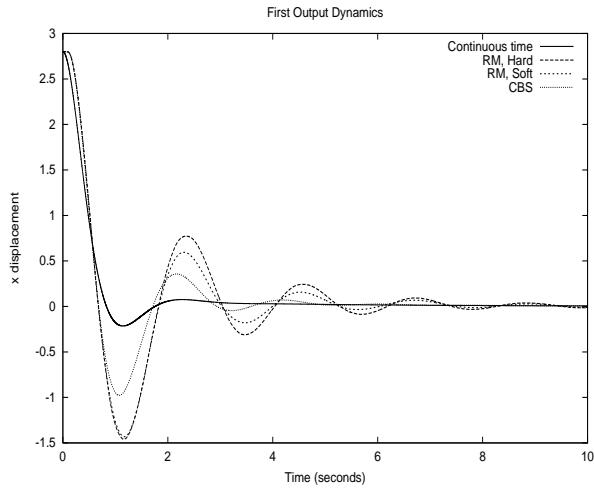
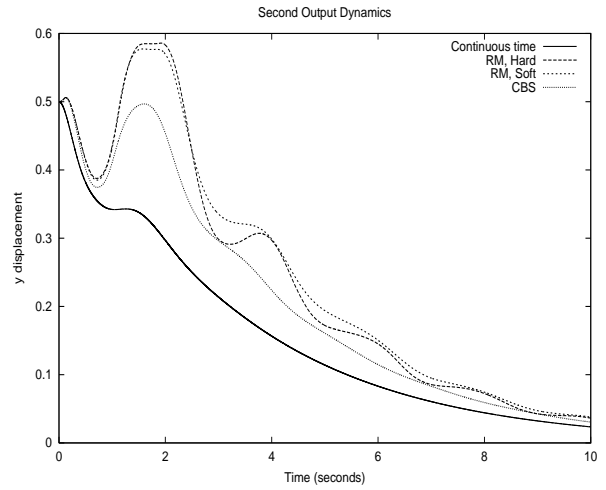


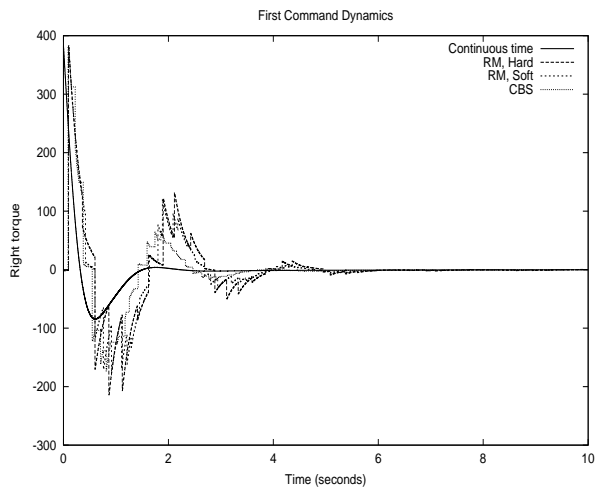
Figure .16. Mobile Robot Cost function vs B_1 bandwidth using CBS: $T_1 = 200ms$ and $T_2 = 6.5ms$.



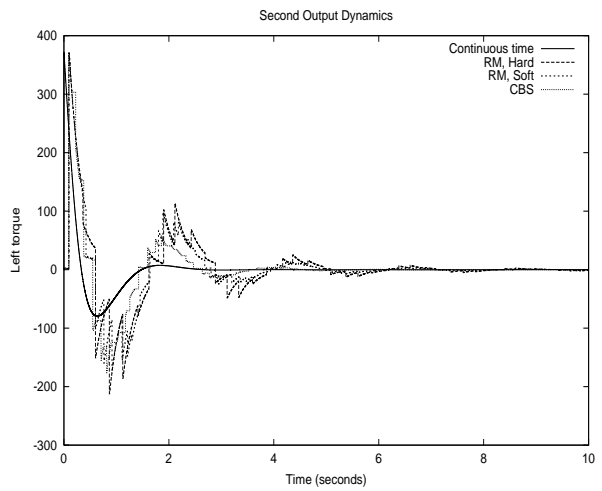
a)



b)



c)



d)

Figure .17. Mobile Robot: dynamics of the displacement $\tilde{\mathbf{x}}_r$ and for the command variables in a simulation run. The initial relative displacement error is $\tilde{\mathbf{x}}_r(0) = [2.8, 0.5]^T$. a) \tilde{x}_r b) \tilde{y}_r , c) τ_r d) τ_l .