Designing Real-Time Embedded Controllers using the Anytime Computing Paradigm

Andrea Quagli*, Daniele Fontanelli†, Luca Greco**, Luigi Palopoli† and Antonio Bicchi*

***Abstract*— In this paper we present a methodology for designing embedded controller with a variable accuracy. The adopted paradigm is the so called any-time control, which derives from the computing paradigm known as "imprecise computation". The most relevant contributions of the paper are a procedure for designing an incremental control law, whose different pieces cater for increasingly aggressive control requirements, and a modeling technique for the execution platform that allows us to design provably correct switching policies for the controllers. The methodology is validated by both simulations and experimental results.**

I. INTRODUCTION

In recent years, the presence of embedded controllers has become one of the most important drivers of innovation for a large class of industrial applications. As an example, in the automotive industry the electronic component is known to take the lion's share in the development efforts for new products, and it "sneaks" even in low-price product lines. In this scenario, system engineers are confronted with a task of challenging complexity: how to make this pervasive introduction of controllers cost effective, preserving the levels of safety and reliability required by increasingly tough regulations.

One of the most popular ways for achieving this goal is by an aggressive sharing of hardware resources amongst different control functions. The paradigm one application-one system is dead and buried. Each computing unit (ECU) is required to sustain several applications (either independent or interacting), and this workload can increase throughout the different re-design iterations to introduce bug fixes or to meet new demands collected from the customers. The price to pay for resource sharing is a reduced predictability of the timing behaviour. An application can receive a different availability of resources and suffer time-varying delays depending on the "interference" suffered from other applications.

This new scenario is not the typical one addressed by classical digital control design methodologies. Indeed, typical digital control approaches consider null or fixed delays inside the control loop. The subsequent development flows are based on a kind of interface between the control applications and the hardware/software platform. Control algorithms are designed assuming fixed delays. Such delays are regarded as real-time "deadlines" for the tasks implementing the control loop and the real-time operating systems parameters are tweaked in such a way as to guarantee that all deadlines will be met. Although perfectly reasonable, this approach bears a considerable drawback: designing the system for the worst case prevents the control designer from capitalizing on the additional availability of resources when the system workload is low. As an example, suppose, in an automotive application, that an ECU is both used for spark ignition control and for other control applications. The spark ignition task is activated by an "angular" event (i.e., the piston reaching the deadend). Therefore, it generates a workload increasing with the rotation per minute (RPM) of the engine. Designing the remaining applications for the worst case, is in this case tantamount to assuming that the engine always rotates at the maximum speed, which is evidently an infrequent occurrence. On the other hand, empirical solutions can be found and tuned through extensive simulation and prototyping activities but the outcome hardly meets the robustness and portability requirements posed by the modern industry. A more appropriate way can be found working in two convergent directions: revising the interface between platform and applications and adopting a different control design methodology that leverages the time-varying "extra" availability of resources.

This is the direction taken by this paper. The fundamental conceptual tool that we rely on is the notion of "anytime" control. The computation of the control law can be split into a sequence of segments. Each segment refines the result of the previous one catering for increasingly aggressive control goals. The first segment has always to be executed, while the remaining ones are executed only when there is a sufficient availability of computation. This idea has been introduced in our previous work [11], [9], where we also discussed such important aspect as the stability of the resulting controller. In this work we deal with the issue of how to make it applicable in an effective development methodology. To this end, we first provide a systematic way to design a set of controllers with increasing closed loop performance, which are executed in subsequent code segments. Each controller satisfies a well established control goal, provided as a share of the design. How to efficiently combine the given platform and the controllers is the second main issue addressed in the paper, which is related to the model derived for the run-time mechanism. The main contribution of this paper

This work was supported by the EC under contract "CHAT - Control of Heterogeneous Automation Systems"

*Authors are with the Interdept. Research Center "Enrico Piaggio", University of Pisa, via Diotisalvi, 2, 56100 Pisa, Italy. Phone: +39050553639. Fax:+39050550650. andrea.quagli@gmail.com, bicchi@ing.unipi.it

†Authors are with the Department of Engineering and Information Science, University of Trento, Via Sommarive 14, 38050 Trento (TN), Italy. Phone: +390461883967. Fax:+390461882093. fontanelli@disi.unitn.it, palopoli@dit.unitn.it

**L. Greco is with the DIIMA, University of Salerno, Italy, greco@dsea.unipi.it

is then a design methodology based on anytime control. The most critical phases (i.e., the production of the anytime control law) of the methodology are supported by a prototype software tool.

II. Related Work

The idea of anytime algorithms is borrowed from the field of *imprecise computation* ([19], [20]). The characteristic of anytime algorithms is to always return an answer on demand; however, the longer they are allowed to compute, the better (e.g. more precise) an answer they will return. The periodic task is split in a *mandatory* part and one or more *optional* parts. The criticality of hard RT tasks is preserved ensuring only that the mandatory parts satisfy the time constraints ([19]). In this paper, we propose a particular application of this paradigm to the case of control application. The theoretical foundations of controllers have been described in [11], where the design of a probabilistic switching policy ensuring the "Almost Sure" stability of the closed loop system is proposed ([8], [2]). The results in [11] and the controllers' design constraints in [9] build upon the assumption of a probabilistic description of the preemptive scheme, derived in this paper for a realistic framework. As pointed out in the introduction, in this paper we display a methodology of practical interest based on these theoretical achievements.

The problem of control/scheduling co-design has raised a remarkable interest in the past few years. A remarkable thread of research [6], [21] has focused on optimal parameter selection (w.r.t. a control theoretical performance metric) for set of periodic tasks implementing digital control loops. In all of these pieces of work, the classical model of digital control (i.e., a discrete–time controller) activated with a fixed frequency is assumed. This work presents a significant departure from this assumption opening to the possibility of multiple incremental implementation for the feedback controller.

Remarkable proposals to overcome the limitation of the classical, although on a different conceptual line than the one presented in this paper, have been presented in [4], where the authors remodulate the periods in response to an overload condition, and in [25], where the authors present event-triggered task models as opposed to the classical time-triggered alternative.

The thread of work closest to ours is probably that related to Firm Real Time Systems (FRTSs) [22], [1]. Since in FRTSs occasional deadline misses are allowed, the task instances that miss their deadlines are considered valueless and they are dropped. In a series of papers [14], [15], [17], [18], Lemmon and co-workers consider performance of Networked Control Systems (NCSs) in a FRTS framework, introduce a Markov Chain model to describe the task dropout process, and provide a general QoS constraint. Our model differs from the one used in the FRTSs literature as we define our probabilities on the space of execution times rather than on deadline misses. More substantial differences, are that we regard the scheduler characteristics to be a given in our problem, rather than a design objective, while the design methodology is related to control law implementation to be implemented in the embedded system.

III. Problem Description

A. The real-time task model

We consider a set $\mathscr{S} = \{s_1, \ldots, s_m\}$ of real-time tasks. Each task s_i is a stream of jobs $J_{i,k}$ with $k \in \mathbb{N}$. Job $J_{i,k}$ is activated at time $a_{i,k}$ and is associated with a duration $c_{i,k}$ and a deadline $d_{i,k}$. Tasks use a shared CPU that is managed according to a *scheduling policy*, i.e., an algorithm whereby a task is selected amongst the ones eligible for execution. The job remains active until it receives a CPU assignment equal to $c_{i,k}$. This instant is called finishing time and will be denoted by $f_{i,k}$. Job $J_{i,k}$ is said to *meet* the deadline if $f_{i,k} \leq d_{i,k}$ and to *miss* it otherwise.

A task is periodic if the activation time $a_{i,k}$ is given by $a_{i,k} = \phi_i + kT_i$, where T_i is said activation period (or simply period), and ϕ_i is said initial offset.

In this work we will consider *preemptive* scheduling algorithms, i.e., algorithms that can suspend (resume) the execution of a job when a higher priority job become active (terminates). We denote by $\Omega_{i,k}$ the time that in the interval $\left[a_{i,k}, d_{i,k}\right]$ is reserved to tasks receiving a higher scheduling priority than s_i. Clearly, the job $J_{i,k}$ meets its deadline if $\Omega_{i,k} + c_{i,k} \leq d_{i,k} - a_{i,k}$.

B. The control problem

In the set $\mathscr{S}$, the element s_j, with $j \in \{1, 2, \ldots, n\}$, is used to control a plant whose "nominal" transfer function is $G(s)$. The plant is affected by an external disturbance term $d(t)$ and by internal additive uncertainties $\Delta(s)$. In plain terms, if $u(t)$ is the input function ($U(s)$ being its Laplace transform) the output of the plant is given by $Y(s) = (G(s) + \Delta(s))U(s) + D(S)$.

The controller is activated with period T_j. At time $a_{j,k} = \phi_j + kT_j$, the hardware takes a sample of the plant's output and computes the control value. The deadline is chosen as $d_{i,k} = a_{i,k} + D_i$, with the relative deadline $D_i \leq T_i$. Between two subsequent releases of the control output, a hardware device holds the last computed value (ZoH).

It is then possible to compute a discrete time equivalent of the plant $G(z)$ using the standard conceptual tools of digital control [10]. Likewise we will use the discrete time counterparts for the the disturbance ($D(z)$) and for the plant uncertainties ($\Delta(z)$). The Z-transform of the sampled output of the system is given by:

$$Y(z) = (G(z) + \Delta(z))U(z) + D(z). \tag{1}$$

The control output is released (with a high accuracy) at time $d_{i,k}$ and is held until time $d_{i,k+1}$. This computation model (commonly called *time-triggered* model of computation [24]) requires the use of a specialized hardware (e.g., an output compare block) and allows us to deal with fixed computation delays, i.e., the effect of output jitter is null. For the purposes of control design, this delay can be treated as additional memory elements (one memory element if the

delay is shorter than a period) inside $G(z)$. The closed loop evolution of the system is given by:

$$Y(z) = \frac{C(z)\tilde{G}(z)}{1+C(z)\tilde{G}(z)}R(z) + \frac{D(z)}{1+C(z)\tilde{G}(z)}, \qquad (2)$$

where $\tilde{G}(z) = G(z) + \Delta(z)$.

The control problem is now described with a list of control goals ordered with respect to control relevance and, hence, suitable for the anytime control approach:

1) Closed loop asymptotic stability of the nominal system,
2) Rejection of the disturbance term $D(z)$,
3) Minimization of an appropriate $\mathscr{H}_\infty$ norm.

It is worthwhile to note that each of the control goals here proposed is customary in the control literature ([10], [7]) and represents only a possible choice among all the existing techniques. Nevertheless, it allows to define a systematic way to design each compounding controller of the sequence.

The rationale behind this choice is readily explained. Asymptotic stability (Goal 1), instrumental to any other goal in control engineering, means that the closed loop system, starting from an equilibrium, is able to react to possible perturbations remaining in a neighbourhood of the equilibrium point and eventually restoring the equilibrium condition. The rejection of the disturbance term $D(z)$ (Goal 2) means that we can attenuate the effect of the term $\frac{D(z)}{1+C(z)\tilde{G}(z)}$ in Equation (2). The minimization of $\mathscr{H}_\infty$ norm (Goal 3) allows us to consider all the previous requirements at once and to tolerate the parametric variations (modelled in the additive uncertainties $\Delta(z)$).

The price to pay for increasingly aggressive goals is computing power. Indeed, as detailed below, the computation power required to achieve Goal 2 is greater than the one required to achieve Goal 1, and the computation power required to achieve Goal 3 is (far) greater than the one required to achieve Goal 1 and Goal 2.

C. Problem Formulation

In the setting described above, the time available for the computation of the control value is, for the $k-th$ job, given by $D_j - \Omega_{j,k}$. The classical (worst case) way of designing the controller is to choose such a control law $C(z)$ as can be accommodated in the time interval $D_j - \max_k \Omega_{j,k}$. This way, the number of deadline misses is limited ([16]). Nevertheless, in the case of control tasks on fixed computational power budget platforms, the algorithms are drastically simplified to be computable within the allotted time.

In this paper, we take a different approach. Assuming that, for all tasks, $c_{i,k}$ is a stochastic process with known distribution, $\Omega_{j,k}$ and the available time $D_j - \Omega_{j,k}$ are stochastic processes in their turn. Assuming the presence of kernel mechanism that notifies the task when its available time has been exhausted, we can build a mechanism according to the following lines: 1) the execution of a controller ensuring the attainment of Goal 1 has to be guaranteed in the worst case, 2) in case the available time is sufficient we can execute additional pieces of computation that attain Goal 2 and Goal 3.

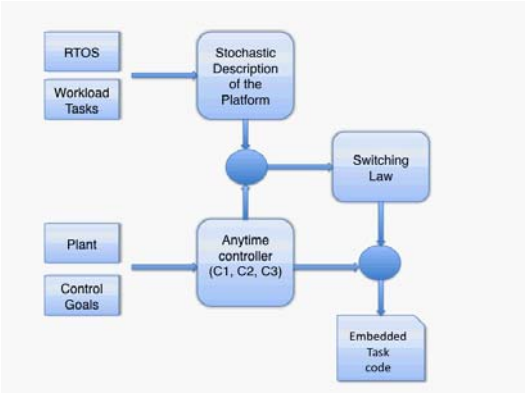


Fig. 1. Our methodology to produce the real-time code for anytime controllers.

In order to make this approach viable we propose a development process organized as shown in Figure 1. The process is organized in the following steps:

- Provide a characterization for the execution environment of s_j, i.e., a mathematical description of the process $D_j - \Omega_{j,k}$,
- Design a controller such that the three goals introduced above are incrementally obtained by executing different "subroutines" in the control algorithm,
- Choose a switching policy for the three controllers such that, given the process $D_j - \Omega_{j,k}$, some systemwide properties are achieved for the closed loop system (first and foremost stability).

The last point deserves some attention. Since we switch between different controllers, the resulting closed loop system is a linear switching system ([12]). It is well known that arbitrary switching between linear asymptotically stable can very well produce unstable dynamics (see [13]). Therefore, we need a systematic way to avoid the switching sequences that could determine this anomaly. In the rest of the paper, we will separately show how each of these steps is performed.

IV. A Markov Description of Stochastic Execution Time

In this section, we show how to come up with a stochastic model for the behaviour of the execution platform (see Figure 1).

Throughout the section, we will assume that the task executes on a CPU managed by a RTOS implementing a fixed priority scheduler [16], [3]. Tasks are ordered according to their scheduling priority ($i < h$ means that s_i has a higher priority than s_h).

a) Preliminaries: As we said above, we focus on a control task s_j coded as a sequence of n different subroutines. The execution of the sequence can be interrupted, if necessary, at "any time" by all tasks s_i having a priority greater than s_j. Each of the n subroutines is the software implementation of a discrete–time dynamic controller, whose state space representation is generically referred to as Γ_p for $p = 1,\ldots,n$. The objective of this section is to model the time $\bar{c}_{j,k} = D_j - \Omega_{j,k}$ available for the k-th job of s_j in an interval of length D_j. Since the relative deadline D_j is

constant, the time dependence of $\bar{c}_{j,k}$ is only related to $\Omega_{j,k}$, i.e. the workload generated by tasks s_i with $i < j$.

Let $\bar{c}_{j,k} \in [t_{min}, t_{max}]$. By definition, we have $\tau_{max} \leq D_j$. Moreover, if $WCET_p$ is the worst case execution time of the first p subroutines of the task s_j, we *require* $WCET_1 \leq t_{min}$ and $WCET_n \leq t_{max}$.

Define an event set $L_\tau \triangleq \{\tau_1, \cdots, \tau_n\}$, and a map

$$\begin{array}{rcccc} \mathscr{T} & : & [\tau_{min}, \tau_{max}] & \rightarrow & L_\tau \\ & & \bar{c}_{j,k} & \mapsto & \tau(k) \end{array}$$

where

$$\tau(k) = \begin{cases} \tau_1, & \text{if} \quad \bar{c}_{j,k} \in [\tau_{min}, WCET_2) \\ \tau_2, & \text{if} \quad \bar{c}_{j,k} \in [WCET_2, WCET_3) \\ \vdots & \text{if} \quad \vdots \\ \tau_n, & \text{if} \quad \bar{c}_{j,k} \in [WCET_n, \tau_{max}] \end{cases}$$

Intuitively, each symbol τ_h is associated with a range of available computation times and it expresses the subroutines that can be invoked at the considered job (for instance the τ_2 symbol means that the controller can execute the first two subroutines).

Therefore, the stochastic description of the platform is in fact the description of a discrete valued stochastic process $\tau(k)$ taking values in L.

The distribution of this process is directly inherited from the probability distribution of $\bar{c}_{j,k}$ The simplest situation is one where $\bar{c}_{j,k}$ is independent and identically distributed (i.i.d.). In this case, the $\tau(k)$ is i.i.d. in its turn.

b) Markov based model for workload generation: In this paper, instead, a slightly more complex, but more general, model allowing for non-stationary probability distributions is provided. The considered situation is one where the higher priority tasks s_i, interfering on the execution of s_j, are periodic and can have different execution modes, i.e. the duration $c_{i,k}$ and the associated period $T_{i,k}$ will vary between different jobs taking a discrete set of values. More precisely, let the set of possible modes for s_i be described by the set of feasible choices $\mathscr{W}_i = \{(c_i^p, T_i^p)\}$ with $m_{s_i} = \#\mathscr{W}_i$ finite and $p = 1, \ldots, m_{s_i}$. For example, if the process associated to the task s_i may be not active, it follows that $(0,0) \in \mathscr{W}_i$. The mode changes are due to asynchronous events but, if they take place during a job of s_i, their execution is deferred to the end of the job, i.e., $c_{i,k+1} \neq c_{i,k}$ and $T_{i,k+1} \neq T_{i,k}$ will be the new time duration and the new execution period after the deadline $d_{i,k}$ has expired.

The mode switches are triggered by a discrete–time Markov process, associated to a an irreducible and aperiodic Markov chain, whose states represent the different modes. For the sake of simplicity, in our context, we assume that mode switches are *synchronized at the end of the current job of* s_j. This behaviour is very simple to implement (e.g., by using a semaphore), but it clearly corresponds to a delay in executing the mode switch. Simplifications like this are not infrequent in the literature of real-time systems with multiple modes. They do not significantly affect the behaviour of the system as far as the sampling periods of the task are very close to each other and the time scale of the triggering event is much longer than that the sampling periods (e.g., this is not unrealistic if the triggering events are determined by the choices of a human operator). Hence, for the p–th mode the number of activations of s_i that cause interference are given by: $\hat{l}_{i,p} = \left\lceil \frac{D_j}{T_i} \right\rceil$. Since the number of possible different duration for s_i is exactly m_{s_i}, we can construct a Markov chain χ_i expressing the interference of s_i on task s_j which is exactly coincident with the Markov chain governing the mode switches. The p–th state of χ_i is tagged with a total duration of $\hat{l}_{ip} c_i$.

If we have multiple tasks with higher priority than s_j, we can generalize this construction approach by computing the composition. The stochastic description of the overall chain χ is given by the Kronecker products of the $j-1$ transition probability matrices of χ_i, and hence χ preserves the property and the steady state probabilities of the chains χ_i ([11]). Again, each state is tagged with the sum of the durations of each of the $j-1$ compounding states. Therefore, a finite state discrete-time homogeneous irreducible aperiodic Markov chain for the available time $\bar{c}_{j,k}$ is thus derived, referred in the rest of the paper as the *scheduler chain*. In this case, more flexible and general than the model that can be obtained using i.i.d. processes, $\Pr\{\tau(k) = \tau_i\} = \pi_{\tau_i}(k)$ is time-dependent, while transition probabilities $p_{ij} \triangleq \Pr\{\tau(k+1) = \tau_j \mid \tau(k) = \tau_i\}$ are time-independent.

c) A motivational example: The choice described above has pretty strong industrial motivations. Consider the following example taken from the automotive domain. In this case, a higher priority task s_i can be, for instance, related to the spark ignition controller that changes $(c_{i,k}, T_{i,k})$ with respect to the drive shaft angular event. Therefore, it is directly dependent from the number of drive shaft revolutions per minute (RPM). Therefore, different driving conditions for the vehicle generate different workloads. Other high priority automotive tasks may be the active suspensions controller, the Electronic Stability Program (EPS) or the Traction Control System (TCS), all related to the driving style. Since it is customary that the driver behavioral model related to the vehicle accelerations and decelerations is obtained using hidden Markov model estimators, the task modes are in their turn modelled as finite states of a Markov chain. In this paper, we assume that all the descriptive chains are finite state discrete-time homogeneous irreducible aperiodic Markov chains. Less important than the previously presented safety systems, is the gaseous emission controller. To gain the best as possible pollution reduction, such a control task cannot be completely removed but, instead, it can be implemented using a lower priority anytime task s_j.

V. Anytime Controllers' Design

In this section we present the relevant implementation aspects of the methodology. In particular we focus on the automatic generation of control laws suitable for the Anytime paradigm and then we briefly summarize the control theoretical issues, like stability and performance, of the resulting switching scheme.

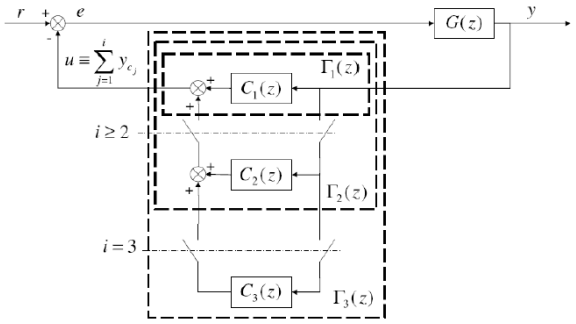


Fig. 2. Anytime control closed loop scheme

A. Bottom-up design technique

The first step towards the implementation of Anytime control on a true RTOS is the development of an automatic procedure to design controllers suitable to Anytime approach.

The main conceptual problem in designing controllers for the anytime paradigm is that the controllers interact in feedback with dynamic systems, which fact entails issues such as:

- *Hierarchical Design:* controllers must be ordered in a hierarchy of increasing performance;
- *Practicality:* implementation of both control and scheduling algorithms must be simple (limited resources);
- *Composability:* computation of higher controllers should exploit computations of lower controllers (recommended);
- *Switched System Performance:* stability and performance of the switched system must be addressed.

We propose here an automatic tool implementing a simple bottom-up design technique based on classical cascade design [9]. This method addresses the issues of hierarchical design, practicality, and composability.

The closed loop structure adopted for implementing this procedure is described in Figure 2 for a three-levels hierarchy.

Looking at the scheme in Figure 2 it is apparent that the composability requirement is addressed since any controller Γ_i inherits the whole structure of the lower controllers.

We define $\Sigma_i = G(z) + \Gamma_i,\ i \in I \triangleq \{1,\ldots,n\}$, the closed loop between the system and (at least asymptotically stabilizing) controller Γ_i. The procedure starts designing Γ_2 according to its performance index, then try to produce C_1 as a proper (or strictly proper) transfer function (t.f.) extracted from Γ_2. In others words, the whole set of stabilizing sub-controllers obtained taking poles and zeros from the t.f. of Γ_2 is produced and a suitable C_1 is chosen according to one or more of the following criteria: rising time, settling time, steady state error, minimum complexity (hence minimum WCET), etc.

C_1 is designed in this way in order to reduce the complexity of the first controller (practicality requirement). Once C_1 is chosen, C_2 is simply given by the difference $\Gamma_2 - C_1$.

If no stabilizing sub-controllers can be produced, the problem of designing C_1 is directly addressed by an optimization problem that has the numerator and denominator coefficients of the controller's t.f. as optimization variables. A possible objective function to be minimized is given by the spectral radius of the closed loop $\max(\mathrm{abs}(\mathrm{eig}(\Sigma_1)))$.

Since keeping low the controller complexity is a main concern (practicality requirement), the procedure starts with a proportional controller and increases its complexity by adding poles and zeros until a stabilizing controller is found. Any candidate controller is described by a proper or strictly proper transfer function. It is worth noting that stability is only one of the possible objective function this optimization problem can use to find a controller.

After C_1 is found, the design of C_2 is performed on the closed loop system Σ_1 according to Σ_2 desired specifications and C_3 on Σ_2 according to Σ_3 desired specifications.

We implemented a software tool that, given the discrete transfer function of the system $G(z)$, is able to provide three controllers ensuring that the related closed loop systems Σ_i have the following performance:

- Σ_1 is asymptotically stable (Goal 1);
- Σ_2 minimizes a quadratic index cost (LQG) (Goal 2);
- Σ_3 minimizes the $\mathscr{H}_\infty$ norm of the transfer function between reference and error, i.e. minimizes $\|T_{re}(z)\|_\infty$ ($\hat{e} = y - r$) (Goal 3).

B. Stability and Performance Under Stochastic Switchings

The stochastic switching policy in Figure 1 is now defined with respect to important control aspects, such as stability and performance. Let the closed-loop system Σ_i be described by a linear discrete–time dynamic $x_{k+1} = \widehat{A}_i x_k$, where the process governing the switchings among different controllers is the Markov chain modelled in Section IV. The stochastic closed loop system is a Markov Jump Linear System (MJLS). Since switching among asymptotically stable controllers may result in an overall unstable behaviour ([12]), in order to actually execute an anytime controller it is necessary to determine a *switching policy* that prevent harmful switches between the controllers. Such a policy selects which controller (among the available ones) will be executed in the next period in the task s_j. Clearly, by no means is it guaranteed that the execution will be permitted since, due to the stochastic interference of higher priority tasks, the available time in the job could not be sufficient. Therefore, the switching policy should ensure stability in a stochastic meaning, named "Almost Sure Stability" in [8]. In [11] a design procedure that based on a the solution of a linear program produces a Markov chain conditioning the evolution of the scheduler in the desired way has been proposed. The switching policy is then a Markov chain $\sigma(k)$, with the same number of states as $\tau(k)$ and with the same properties. Picking a realization of the Markov chain, it is then possible to stochastically select the controller to be executed in the next period.

VI. Simulation Examples

In this section, we present some simulation results that show the application of the design methodology. The simulation has been carried out using the true-time tool, which allowed us to model both the anytime controller and the

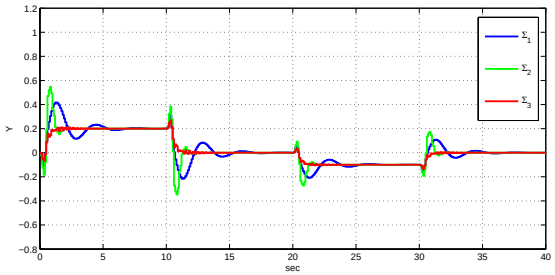


Fig. 3. Output of $\Sigma_1, \Sigma_2, \Sigma_3$. The reference signal is a square wave with variable amplitude.

execution environment [5]. To the purpose of evaluating performance we used an extension of the bounded real lemma [23] to switching systems that allows us to compute the $\mathscr{H}_\infty$ norm and use it as a performance index.

A. Controllers design

In order to test the effectiveness of the design technique presented in the previous section we used a set of randomly generated discrete-time unstable t.f. $G(z)$ of different orders. The tool succeeded in finding the three controllers for each system.

For the sake of completeness, we report one of these systems and the related controllers. The open loop system is described by the following t.f.:

$$G(z) = \frac{-1.84(z+3.58)(z+1.24)(z-1.51)}{(z^2+1.26z+0.64)(z^2-2.4z+1.87)}, \qquad (3)$$

with sample time of 0.1 sec. The algorithm firstly finds an LQG controller (Γ_2), from which four stabilizing sub–controller C_1 are derived by decomposition in this case. If no particular criterion is specified, the algorithm selects by default the lower order controller. In this example the lower order C_1 component controller is $C_1(z) = -0.11\frac{(z-0.91)}{(z-0.88)}$; and C_2, given by $\Gamma_2 - C_1$, is $C_2(z) = 0.11\frac{(z+0.53)(z^2-0.44z+0.38)}{(z+0.92)(z^2+1.16z+1.6)}$. For the third controller we use a classical $\mathscr{H}_\infty$ control synthesis. A stabilizing $\mathscr{H}_\infty$ optimal controller is designed minimizing the $\mathscr{H}_\infty$ norm of the t.f. $T_{re}(z)$ between the reference r and the error e of Σ_2. This control synthesis provides the following controller:

$$C_3(z) = -0.044\frac{(z-0.63)(z+1.09)(z-0.14)}{(z-0.88)(z+1.01)(z+0.92)} \times \frac{(z^2+0.87z+0.28)(z^2+1.40z+0.71)}{(z^2+1.91z+1.02)(z^2+0.87z+1.16)}.$$

The closed loop output to a square wave reference with variable amplitude is reported in Figure 3.

As the controllers are available, we consider the performance of the switching system generated by the Anytime control paradigm.

B. Bounded Real Lemma for Performance Index

To use the Bounded Real Lemma index in [23] for the Anytime control systems we have to pay attention to some particular aspects, related to the switching nature of our system. To this end, the index is modified using weighting terms that are strictly related to the entries of the transition probability matrix of the conditioned Markov chain, i.e., the Markov chain of the scheduler τ conditioned by the Markov chain of the switching policy σ.

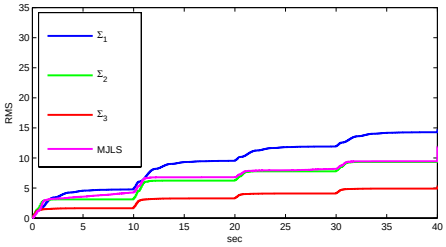


Fig. 4. RMSs of $\Sigma_1, \Sigma_2, \Sigma_3$ and Anytime control

Assuming in the present example that $\pi_\tau = [0.05 \quad 0.25 \quad 0.7]$ is the invariant probability distribution of the scheduler, a solution to the switching policy problem for the system and controllers presented in section VI-A is given by $\pi_\sigma = [0.0462 \quad 0.9014 \quad 0.0524]$, the invariant probability distribution of the switching policy.

The resulting value of the $\mathscr{H}_\infty$ of the t.f. between "r" and "e" for the three systems ($\|T_{re}(z)\|_\infty$) was 8.64 for Σ_1, 8.22 for Σ_2 and 3.92 for Σ_3 and 8.23 for the anytime MJLS. As expected the three controllers provide increasing performance, while the anytime controller (as expected from the probability π_σ) performs very closely to Σ_2.

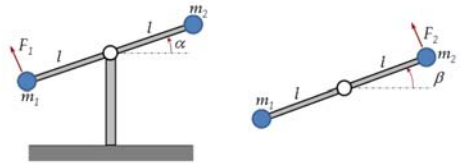
C. TrueTime

Using Matlab/Simulink TrueTime toolbox, we simulated a task set composed of two tasks: the *Anytime* control task and a task, given a higher priority, that generates scheduling interference. The latter task executes randomly in three different modes, named *Load1*, *Load2* and *Load3*, with fixed execution times. The computational time of *Load1* is such that the time available in the period after its execution is sufficient to compute only C_1. The time available after the execution of *Load2*, instead, allows for the computation of controllers up to the second one, etc. We recall that the probability of execution of every controller reflects the one assigned by the π_τ of the scheduler, while the actual execution of the *Anytime* controllers is conditioned by the chain σ used to define the switching policy.

In Figure 4 we compare of the Root Mean Square (RMS) of the MJLS induced by the Anytime control with the RMSs of Σ_1, Σ_2 and Σ_3. The reference signal is the same we used in section VI-A.
The performance of the MJLS is very similar to the one Σ_2. Hence, the Markov policy behaves almost like computing always the second controller, which is not a feasible policy (from a hard real-time perspective) since when the interfering tasks executes *Load1*, Σ_2 is not schedulable.

VII. Experimental Results

In this section we show the application of our methodology to a real-system. (See Figure 5). The system is a 2 DOF helicopter model, consisting of a rotating base linked to a rod having length $2l$. Two fan actuators, producing forces F_1 and F_2, are installed at the two ends of the rod. We name



(A)

(B)

Fig. 5. 2 DOF helicopter model. A) schematic diagram , B) picture of the system

Name	Description	Value
m_1	mass	0.110 kg
m_2	mass	0.090 kg
l	rod half length	0.2 m
c_α	friction coefficient on α	0.01 kg s^{-1}
c_β	friction coefficient on β	0.01 kg s^{-1}
J_1	complete moment of inertia on x	0.00463 kg m^2
J_y	moment of inertia on y	0.00023 kg m^2
J_z	moment of inertia on z	0.00364 kg m^2
I_b	base axial moment of inertia	0.00023 kg m^2

TABLE I

NUMERIC VALUES OF THE HELICOPTER CONSTANTS

the pitch angle α and the yaw angle β.
The system can be described by the following equations:

$$\ddot{\alpha} = -\frac{J_2}{J_1}\dot{\beta}^2 \sin\alpha\cos\alpha - \frac{c_\alpha}{J_1}\dot{\alpha} - \frac{l}{J_1}F_1 + \frac{(m_2-m_1)lg}{J_1}\cos\alpha$$

$$\ddot{\beta} = -\frac{2J_2}{J_3(\alpha)}\dot{\alpha}\dot{\beta}\sin\alpha\cos\alpha - \frac{c_\beta}{J_3(\alpha)}\dot{\beta} + \frac{l\cos\alpha}{J_3(\alpha)}F_2,$$

where $J_1 = J_x + (m_1+m_2)l^2$, $J_2 = J_y - J_z - (m_1+m_2)l^2$, $J_3(\alpha) = J_y\sin^2\alpha + \left(J_z + (m_1+m_2)l^2\right)\cos^2\alpha + I_\beta$.

The description and the numeric value of the helicopter constants are reported in Table I. Linearizing the system around the operating point $\bar{\alpha} = \bar{\beta} = 0$ with $\bar{F}_1 = (m_2 - m_1)g$ and discretizing the continuous-time model with a sample time of 1 millisecond, we obtain a diagonal discrete-time transfer function matrix $G(z)$. Therefore, unlike the non linear model, the dynamics of α and β are decoupled in the linearised system, hence the Multiple Input Multiple Output (MIMO) system can be regarded as two independent Single Input Single output (SISO) systems. $G(z)(1,1) = \frac{-2.16\times10^{-5}(z+0.999)}{(z-1)(z-0.998)}$ represents the t.f. between F_1 and α. In this case, the developed tool gives $C_1(z) = -0.499$, $C_2(z) = -3196.71\frac{(z-0.56)}{(z^2+0.28z+0.62)}$ and $C_3(z) = -19352.47\frac{(z-0.81)(z+0.38)(z^2+0.15z+0.63)}{(z^2+0.6z+0.143)(z^2+0.29z+0.61)}$. $G(z)(2,2)$ represents the t.f. between F_2 and β. For space limits, the $G(z)(2,2)$ and the associated controller are not exlicitly reported.

Closed Loop systems	$\|T_{re11}(z)\|_\infty$	$\|T_{re22}(z)\|_\infty$
Σ_1	4.8741	9.631
Σ_2	1.0914	1.045
Σ_3	1.0003	1
Anytime MJLS	1.1327	1.7847

TABLE II

$\mathscr{H}_\infty$ NORM OF T.F. BETWEEN "R" AND "E" FOR EACH Σ_i AND FOR MJLS.

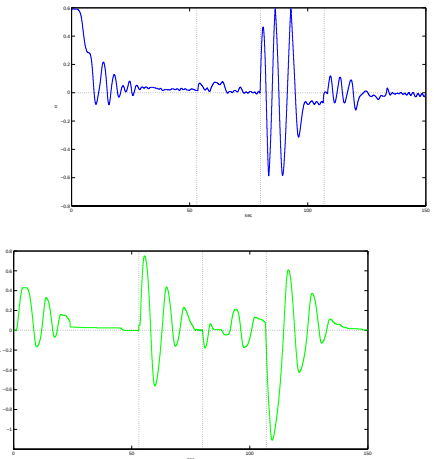


Fig. 6. Pitch and yaw angle of the Anytime controlled helicopter.

Assuming $\pi_\tau = [0.05 \quad 0.25 \quad 0.7]$ is the invariant probability distribution of the scheduler, a solution to the linear programming problem for the system is given by $\pi_\sigma = [0.1042 \quad 0.8103 \quad 0.0855]$.
In Table II the $\mathscr{H}_\infty$ norm of the closed loop systems Σ_i and of the Anytime MJLS for each SISO system is reported. The norm value of the Anytime controlled system is very close to that of Σ_2, as expected by the values of π_σ.

For the platform, we used one of large industrial employment, having an on-board RTOS and providing an intuitive visual programming language: the PXI (*PCI eXtensions for Instrumentation*) modular instrumentation platform from *National Instruments* and its on-board proprietary RTOS. To monitor and to program the PXI platform, we used LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench). In particular we used LabVIEW to implement our three level Anytime control scheme.

We applied the resulting architecture to some test regulation tasks. Let us illustrate one of them. The helicopter starts from the initial position $\alpha = 33.84^\circ$ and $\beta = 0^\circ$ and reaches the horizontal position $\alpha = \beta = 0^\circ$, i.e. the operating point. The system is perturbed by external forces in order to test its robustness.

Figure 6 depicts the measured outputs (α and β). Starting from the initial position, the helicopter reaches the horizontal position in about 40 seconds. At $T = 53$ sec an external impulsive force is applied to the system in order to destabilize the yaw posture. The dynamics of α turns out to be perturbed as well. The system returns to the equilibrium point at $T = 80$ and after a few seconds another impulse is applied, in order to destabilize the pitch posture. Another impulse perturbing mainly β is applied at $T = 107$. The plots in Figure 6 show how the Anytime control stabilises the

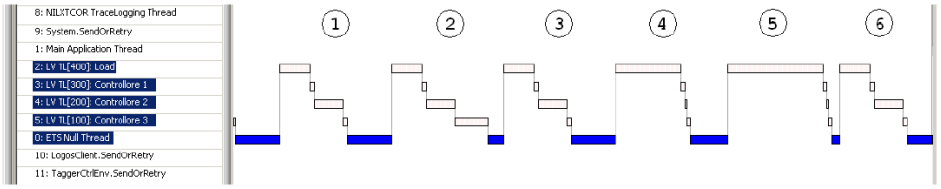


Fig. 7. Excerpt of the task set scheduling

system even in presence of strong external disturbances and the anytime regulation task fulfil the control objective in the presence of noise in the input and output channels and of model uncertainties (e.g., resulting from the linearisation).

Using a LabVIEW toolkit (*Real-Time execution Trace toolkit*), the task scheduling of the RTOS has been traced and then partially depicted in figure Figure 7 for the helicopter experiment shown in this section. In the few sample periods reported, all the typical behaviours of the anytime control are depicted.

For the implementation of the paradigm, we split the Anytime task into three different tasks, each implementing one of the three subroutines, while another dummy task generates the workload with stochastic Markovian execution time. For completeness, the *ETS Null Thread* is added as the *dummy* process of the RTOS. For each of the previously introduced tasks, we report in Figure 7, left, the priority value (higher value, lower priority, except for the *ETS Null Thread*) in square brackets. The computation time of the tasks were as follows: load task $200 \div 900\mu s$, C_1 $10 + 23\mu s$ (23 μs is the input acquisition), C_2 $185\mu s$ and C_3 $190 + 26\mu s$ (26 μs writing activities, included in the mandatory part).

VIII. Conclusions

The main contributions of this paper are in the direction of providing a methodology for an automatic control design complying with the Anytime control paradigm, and in formalizing a stochastic model for a particular, practically motivated, scheduler for the interaction with the control design.

The effectiveness of the proposed approach has been proved by both simulating the designed control tasks and the stochastic scheduling policy in TrueTime, and by a real experimental setup. The good adherence with the theoretical expectations, as well as the evident improvement with respect to the hard real-time setting, is suggesting of interesting future development in terms of achieving the best exploitation of computational resources for embedded systems.

References

[1] G. Bernat, A. Burns, and A. Llamosi. Weakly hard real-time systems. *IEEE Trans. on Computers*, 50(4):308–321, April 2001.

[2] P. Bolzern, P. Colaneri, and G. De Nicolao. On almost sure stability of discrete-time Markov jump linear systems. In *Proc. 43rd IEEE Conf. On Decision and Control*, volume 3, pages 3204–3208, 2004.

[3] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, 1997.

[4] Giorgio C. Buttazzo, Manel Velasco, and Pau Martí. Quality-of-control management in overloaded real-time systems. *IEEE Trans. Computers*, 56(2):253–266, 2007.

[5] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

[6] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 2002.

[7] B. M. Chen. *Robust and H-infinity Control*. Springer, first edition, 30th March 2000.

[8] Y. Fang, K. A. Loparo, and X. Feng. Almost sure and δ-moment stability of jump linear systems. *Int. J. Control*, 59(5):1281–1307, 1994.

[9] D. Fontanelli, L. Greco, and A. Bicchi. Anytime control algorithms for embedded real-time systems. In *Proc. IEEE Int. Conf. on Hybrid Systems: Computation and Control*, pages 158–171, St. Louis, MO, April 2008.

[10] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, third edition, 1998.

[11] L. Greco, D. Fontanelli, and A. Bicchi. Almost sure stability of anytime controllers via stochastic scheduling. In *Proc. IEEE Int. Conf. on Decision and Control*, pages 5640–5645, New Orleans, LO, 12–14 December 2007.

[12] D. Liberzon. *Switching in Systems and Control*. Birkhauser, January 2003.

[13] D. Liberzon and A. S. Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19(5):59–70, October 1999.

[14] Q. Ling and M.D. Lemmon. Robust performance of soft real-time networked control systems with data dropouts. In *Proc. IEEE Int. Conf. on Decision and Control*, volume 2, pages 1225–1230, 10–13 December 2002.

[15] Q. Ling and M.D. Lemmon. Soft real-time scheduling of networked control systems with dropouts governed by a Markov chain. In *Proc. American Control Conf.*, volume 6, pages 4845–4850, 4–6 June 2003.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 1973.

[17] D. Liu, X.S. Hu, M. Lemmon, and Q. Ling. Scheduling tasks with Markov-chain based constraints. In *Proc. 17th Euromicro Conf. on Real–Time Systems*, pages 157–166, 6–8 July 2005.

[18] D. Liu, X.S. Hu, M. Lemmon, and Q. Ling. Firm real-time system scheduling based on a novel QoS constraint. *IEEE Trans. on Computers*, 55(3):320–333, March 2006.

[19] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung. Imprecise computation. *Proceedings of the IEEE*, 82(1):83–93, January 1994.

[20] Jane W. S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, J.-Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *Computer*, 24(5):58–68, 1991.

[21] L. Palopoli, C. Pinello, A. Bicchi, and A. Sangiovanni-Vincentelli. Maximizing the stability radius of a set of systems under real-time scheduling constraints. *Automatic Control, IEEE Transactions on*, 50(11):1790–1795, Nov. 2005.

[22] P. Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Trans. on Parallel and Distrib. Syst.*, 10(6):549–559, 1999.

[23] P. Seiler and R. Sengupta. A bounded real lemma for jump systems. *IEEE Transactions on Automatic Control*, 48(9):1651 – 1654, Sept. 2003.

[24] C.M. Kirsch T. Henzinger, B. Horowitzm. Embedded control systems development with giotto. In *Proc. of ACM SIGPLAN 2001 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'2001)*, June 2001.

[25] P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *Automatic Control, IEEE Transactions on*, 52(9):1680–1685, Sept. 2007.