Design of Embedded Controllers Based on Anytime Computing

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

***Abstract*—In this paper we present a methodology for designing embedded controllers based on the so–called any-time control paradigm. A control law is split into a sequence of subroutine calls, each one fulfilling a control goal and refining the result produced by the previous one. We propose a design methodology to define a feedback controller structured in accordance with this paradigm and show how a switching policy of selecting the controller subroutines can be designed that provides stability guarantees for the closed loop system. The cornerstone of this construction is a stochastic model describing the probability of executing, in each activation of the controller, the different subroutines. We show how this model can be constructed for realistic real-time task sets and provide an experimental validation of the approach.**

***Index Terms*—Embedded systems, Real–time control, Anytime control.**

I. Introduction

A trend that is widely recognised in modern embedded control development is toward a very high utilisation of hardware resources. Indeed, the increasing demand of control functionalities even in low price product lines cannot be satisfied by a proliferation of dedicated electronic devices, both for cost reasons and for the unacceptable engineering complexity of the resulting system. The price to pay for resource sharing is a reduced predictability of the timing behaviour: an application receives a different availability of resources and suffer time-varying delays depending on the "interference" suffered from other applications.

In this new context, classical digital control design methodologies suffer important limitations. Indeed, they require that the computation of the control task always terminate within a deadline. With this approach, it is imperative for the designer to consider the worst case situation for the task computation times. Hence, the designer is forced to conservative choices in terms of computation complexity. The evident drawback is that the control designer cannot capitalise on the additional availability of resources when the system workload is low. In modern control systems strong fluctuations in the workload are not infrequent. As an example, suppose, in an automotive application, that a Electronic Computing Unit (ECU) is used both 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 dead-end). 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 clearly an infrequent occurrence. Even more evident are the workload fluctuations generated by control tasks using multimedia data (e.g., visual servoing). In this case, depending on the complexity of the scene and on the presence of obstructions, the computation time required to track a feature of interest in the image plane can change significantly.

While empirical approaches can be used to fine-tune the application operating with the prototypical implementations, the result rarely meets the robustness and portability requirements posed by modern industry. Indeed, the empirical fine-tuning is made only with a specific architecture and it is not guaranteed that the specific workload conditions considered in the prototype tests will be the ones actually encountered in the production system. In contrast, we advocate a theoretically sound solution to this problem based on the notion of *anytime* control. By using this paradigm, the computation of the control law is split into a sequence of segments. Each segment refines the result of the previous one catering for increasingly aggressive control goals. The execution of the first segment is mandatory, while the subsequent ones are executed only when there is a sufficient availability of computation time. The theoretical foundation of this approach has been laid in our previous work [1], [2]. In these papers, we have shown how to synthesise a switching policy that guarantees closed loop stability (more precisely "almost sure stability") of the controlled system *given* the anytime controller and the probability distributions of the availability of CPU time for each of its executions (jobs).

The purpose of this paper is to show the practical applicability of this idea in the context of realistic control applications. To achieve this result, we offer two key contributions. The *first one* is a design procedure that enables one to synthesise an anytime control algorithm. This critical design step is carried out starting from a list of increasingly aggressive goals on the desired performance and simultaneously generating a sequence of functional blocks that attain each of them when executed iteratively. In essence, a block designed for a goal in the hierarchy builds on top of the result obtained by the previous

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7 under grant agreement n° IST-2008-224428 "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@centropiaggio.unipi.it, 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@disi.unitn.it

**L. Greco is with the LSS - Supélec, 3, rue Joliot-Curie, 91192 Gif sur Yvette, France. lgreco@ieee.org

blocks (rather than starting the computation from scratch). The *second contribution* is the construction of a stochastic model representing the availability of computing time. This model is based on a discrete–time Markov Chain that accounts for the interference that a control task can suffer from stochastically changing the execution time of higher priority tasks. Similar models have been proposed in the past for soft real-time systems. In our context, this model is used to estimate the availability of time for each execution of the control task. In some sense, it can be thought of as a "contract" between the control application and the platform, much more descriptive of its dynamic behavior than the worst case response time. Notably, we show how a similar model can indeed be used to describe the workload generated by *realistic task sets*.

The outcome of the anytime control design and the model of the execution platform are perfectly compliant with the requirements of the theoretical framework constructed in the previous work. Thereby, we are now in condition to propose a *full-fledged design procedure* based on this novel paradigm, whose most critical steps are supported by a prototypical CAD tool suite. More precisely, the software tool takes as an input the task set, along with the stochastic description of the computation times, and the plant model, producing as an output the switching policy and the embedded task code of each controller.

The methodology has been applied to the design of the feedback controller of a real system (a 2 DoF helicopter). The experimental results that we report on the paper are in good accordance with our theoretical expectations and display a remarkable performance gain with respect to the application of standard and conservative hard real-time approaches. These results are consistently present in a large batch of simulations, suggesting an interesting potential for the future application of this technique.

II. Related Work

The problem of control/scheduling co-design has raised a strong interest in the past few years. A remarkable line of research [3], [4] 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 departure from this assumption opening to the possibility of incremental implementations for the controllers.

Important proposals to overcome the limitation of the classical model, although on a different conceptual line than the one presented in this paper, have been presented in [5], where the authors remodulate the periods in response to an overload condition, and in [6], where the authors present event-triggered task models as opposed to the classical time-triggered alternative. But, the thread of work closest to ours is probably that related to Firm Real Time Systems (FRTSs) [7], [8]. 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 [9], [10], [11], [12], 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 because we define our probabilities on the space of execution times rather than on the space of 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 focused on the control law implementation.

Param.	Description
w_i	i-th task in $\mathscr{W}$
T_i	period of w_i
ϕ_i	initial offset of w_i
D_i	relative deadline of w_i ($D_i \leq T_i$)
$J_{i,k}$	k-th job of w_i ($k \in \mathbb{N}$)
$a_{i,k}$	activation time of $J_{i,k}$ ($a_{i,k} = \phi_i + kT_i$)
$c_{i,k}$	execution time of $J_{i,k}$
$d_{i,k}$	absolute deadline of $J_{i,k}$ ($d_{i,k} = a_{i,k} + D_i$)
$f_{i,k}$	finishing time of $J_{i,k}$
$\Omega_{i,k}$	interference on w_i in the interval $[a_{i,k}, d_{i,k}]$ from higher priority tasks

Table I
Notation used for the task set $\mathscr{W}$.

The resulting idea of anytime algorithms is closely related to the notion of *imprecise computation* [13], [14]. The characteristic of anytime algorithms is to always return a result on demand; however, the longer they are allowed to compute, the better (e.g. more precise) the result they will return. The periodic task is split in a *mandatory* part and one or more *optional* parts. The mandatory part of the task is the only one subject to hard execution constraints [13]. In this paper, we apply this paradigm to control applications. The theoretical foundations of controllers have been described in [1], where the design of a probabilistic switching policy ensuring the "Almost Sure" stability of the closed loop system is proposed [15], [16]. The results in [1] and the controllers' design constraints in [2] build upon the assumption of a probabilistic description of the preemptive scheme. In this paper, we flesh up this scheme showing how it can be used to model realistic systems of real-time tasks. As pointed out in the introduction, in this paper we display a methodology of practical interest based on these theoretical achievements. This paper extends and subsumes the preliminary results on the topic presented in [17]. In particular, the tools needed to derive the stochastic description of the scheduler are formally provided in Section IV. An interesting inspiration for this part of our work was offered us by the work of Kim and co-workers [18]. The first part of the discussion in Section IV-B on the stochastic model for the platform was partially covered in a previous work [19]. This paper extends and subsumes the previous results that we presented in the ETFA Conference [17].

III. Problem Description and Solution Overview

A. Real-time task model

We consider a set $\mathscr{W} = \{w_1, \ldots, w_m\}$ of real-time periodic tasks, whose relevant parameters are given in Table I. We

assume a RTOS implementing a fixed priority scheduler [20] allowing the adoption of *preemptive algorithms*. Hence, tasks are scheduled according to a decreasing priority order and the execution of a job can be suspended (resumed) when a job with higher priority becomes active (finishes).

Each task generates a stream of jobs. Job $J_{i,k}$ is started at time $a_{i,k} = \phi_i + kT_i$ and finishes at time $f_{i,k}$; it is said to *meet* the deadline if $f_{i,k} \leq d_{i,k}$ and to *miss* it otherwise. Clearly, $J_{i,k}$ meets its deadline if $\Omega_{i,k} + c_{i,k} \leq d_{i,k} - a_{i,k}$, $\Omega_{i,k}$ being the interference suffered by job $J_{i,k}$ from higher priority tasks.

B. The control problem

In this setting, *one* of the tasks, in the set $\mathscr{W}$, Task w_m, is used to control a plant described by the "nominal" transfer function $G(s)$. The plant is affected by an external disturbance $d(t)$ (Laplace transform $D(s)$) and by internal additive uncertainties $\Delta(s)$. In plain terms, if $u(t)$ is the input function ($U(s)$ its Laplace transform) the output of the plant is given by $Y(s) = (G(s) + \Delta(s))U(s) + D(s)$. At each $a_{m,k}$ $k \in \mathbb{N}$, the hardware takes a sample of the plant's output and then w_m computes the control value. Using a time-triggered computation model [21], [22], the control output is released (with high accuracy) within a relative deadline from the arrival of the sample(e.g., equal to the arrival of the next sample $d_{m,k}$) and is held constant until time $d_{m,k+1}$ by means of a Zero order Hold (ZoH). This way, the output jitter is nullified.

It is then possible to compute a discrete time equivalent $G(z)$ of the plant using the standard conceptual tools of digital control [23]. Likewise we will use the discrete time counterparts for the disturbance ($D(z)$) and for the plant uncertainties ($\Delta(z)$). The Z-transform of the sampled output of the system is given by:

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

For the purposes of control design, the fixed delay introduced by the time-triggered model can be accounted for inside $G(z)$. The closed loop evolution of the system is given by:

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

where $\tilde{G}(z) = G(z) + \Delta(z)$ and $C(z)$ represents the Z-transform of the control algorithm.

In the control literature [23], [24] it is customary to translate the practical control problem in a set of goals or objectives to be achieved. Each of these goals catches a specific desired performance for the controlled system. The first essential requirement, instrumental to any other goal in control engineering, is the stability. Dealing with linear systems, asymptotic stability is commonly called for, as it ensures that the closed loop system reacts to a perturbation of the equilibrium by remaining in a neighborhood of the equilibrium point and by eventually restoring it. Performance specifications are given by control engineers both in time and frequency domain. Examples of time domain specifications are those related to the transient and steady state response of the system to a step input: rise time, settling time, overshoot, undershoot, steady state error, etc. They are particularly useful whenever the typical input signal is constant or slowly varying for long periods. More complex control objectives can be specified in terms of frequency shaping of meaningful transfer functions (such as for instance the sensitivity and complementary sensitivity transfer functions) or in terms of gain and phase margins. Requirements typically specified in the frequency domain are disturbance rejection and robust stability. With reference to the transfer function (2), disturbance rejection aims at attenuating the effect of the term $\frac{D(z)}{1+C(z)\tilde{G}(z)}$, while robust stability aims at guaranteeing asymptotic stability against parametric variations and model uncertainties (modelled by the term $\Delta(z)$). Another important requirement is represented by the simplicity of the controller. Indeed, it is often desirable to keep the controller's order as much low as possible.

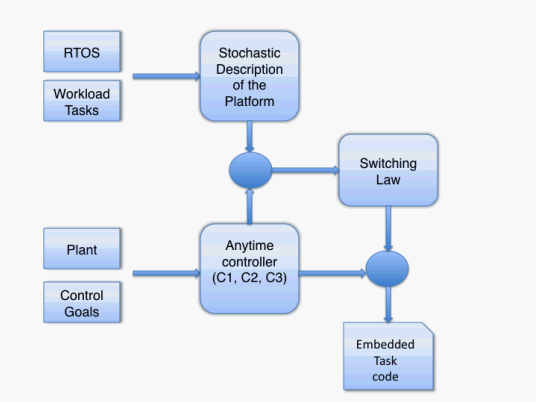

Figure 1. Our methodology to produce the real-time code for anytime controllers.

A classical control design try to build an overall controller to match the entire set of goals, whilst in the anytime approach we pursue an incremental design. A set of controllers with increasing complexity has to be designed to ensure progressively the desired performance. A typical set of control goals for the system (2) is as follows:

1) Closed loop asymptotic stability of the nominal system,
2) Rejection of the disturbance term $D(z)$,
3) Robust stability w.r.t. model uncertainties $\Delta(z)$.

The computing time and the complexity of the controllers achieving the previous goals increase with the goal (Goal 3 being by large the most demanding).

C. Problem Formulation

In the setting described above, the time available for the computation of the control value, for the k-th job execution, is given by $D_m - \Omega_{m,k}$. The classical (worst case) way of designing the controller is to choose a control law $C(z)$ such as it can be accommodated in the time interval $D_m - \max_k \Omega_{m,k}$. This way, the number of deadline misses is null ([25]). To meet this constraint, control algorithms have to be simplified to be computable within the allotted time.

In this paper, we take a different approach. Assuming that, for all tasks w_j, $\{c_{j,k}\}_{k\in\mathbb{N}}$ is a stochastic process with known stochastic characterisation, the available time $D_m - \{\Omega_{m,k}\}_{k\in\mathbb{N}}$

for the task w_m $m > j$, is a known stochastic process in its turn. As long as the RTOS notifies the task when its available time has been exhausted, we can build an anytime controller as follows: 1) the execution of a controller ensuring the attainment of the first goal (stability) has to be guaranteed in the worst case before the relative deadline chosen to release the output, 2) in case the available time is sufficient we can execute additional pieces of computation that attain progressively the other performance goals (e.g. disturbance rejection and robust stability).

To make this approach viable we propose a development process organised in the following steps (see Figure 1): a) Provide a characterisation for the execution environment of w_m, tightly related to the stochastic process $D_m - \left\{\Omega_{m,k}\right\}_{k\in\mathbb{N}}$, b) Design a controller such that a set of increasing control goals are incrementally obtained by executing different "subroutines" c) Choose a switching policy for the subroutines such that, given the process $D_m - \left\{\Omega_{m,k}\right\}_{k\in\mathbb{N}}$, global stability of the resulting system is guaranteed in a stochastic sense.

The last step deserves some attention. Since we switch between different controllers, the resulting closed loop system is a (linear) switching system ([26]). It is well known that arbitrary switching between linear asymptotically stable systems can likely produce unstable dynamics (see [27]). 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. Stochastic description of the Platform

The anytime control task w_m is coded as a sequence of n different subroutines. Each of the n subroutines is the software implementation of a discrete–time dynamic controller, whose state space representation is denoted as Γ_p for $p = 1,\ldots,n$.

According to the theoretical results presented in our previous work [1], a stabilising switching policy, choosing among the various Γ_p, can be synthesised if, 1) for every job we know the distribution for the probability of completing the different subroutines, 2) this distribution eventually reaches a steady state.

In the rest of the section we will show how to construct this information. Our procedure is in two steps and it assumes that the anytime control task w_m share the CPU with a number of higher priority periodic tasks. The first step takes as input the probability mass functions (PMFs) of the higher priority tasks. Since these tasks can preempt the execution of the w_m, they generate an interference $\Omega_{m,k}$ on its execution. The goal of this step is to find a stochastic model for the interference. More specifically, we are interested in the steady state distribution of the available computation time $\bar{c}_{m,\widetilde{k}}$.

The second step starts from this model and produces a different discrete valued stochastic model describing the probability of executing each subroutine, the information required by the anytime control theory. Essential is the ability to show that this stochastic process reaches an invariant distribution.

A. Step 1: A Stochastic Model for the Interference

In this section, we will provide a time–varying description of the stochastic process $\left\{\Omega_{m,k}\right\}_{k\in\mathbb{N}}$ and show that it eventually reaches a steady state.

a) The probability distributions of the execution time: The starting point of our procedure is a model of the stochastic process $\{c_{j,k}\}$ describing the execution time of any task w_j with a higher priority than w_m. This process takes values in the discrete set $\mathscr{C}_j \triangleq \{0,1,\ldots,l_j\}$. We have adopted a normalised notation whereby $c_{j,k} = h$, $h \in \mathbb{N}$ means that the execution time of w_j in the k^{th} period is $h\Delta T$, where the ΔT constant is the time *granularity* of the CPU (related to the clock period). Since in a hard real-time setting relative deadlines have to be greater than the worst case execution time, the execution time is upper bounded by l_j.

The simplest situation is one for which the process $\{c_{j,k}\}$ is independent and identically distributed (i.i.d). In this case the PMF of the process is a time–invariant function $\overline{g}_j$ defined as $\overline{g}_j(\cdot) : \mathscr{C}_j \to [0,1]$, where $\overline{g}_j(h)$ represents the probability $\Pr\{c_{j,k} = h\}$ that the execution time in the k^{th} job of task w_j is exactly h.

In this paper, we consider a more general model: each task w_j can have r_j different execution modes, and each mode is associated with a discrete random variable c_j^q modelling the execution time of w_j in mode $q \in \mathscr{I}_j \triangleq \{1,\ldots,r_j\}$. We denote with $g_{j,q}$ the PMF related to c_j^q. For example, if the task w_j is allowed to skip some executions, there exists a $q \in \mathscr{I}_j$ s.t. $g_{j,q}(0) = \Pr\{c_j^q = 0\} = 1$.

The mode switches are due to asynchronous events but, if they take place during a job, their execution is deferred to the end of the job. We assume that such events are triggered by a random process described by a Finite-State discrete-time Homogeneous Irreducible Aperiodic Markov Chain (FSHIA–MC) $\eta_j(k)$, taking values in $\mathscr{C}_j$, with transition matrix $S_j = (s_{pq}^j)_{l_j\times l_j}$, $s_{pq}^j \triangleq \Pr\{\eta_j(k+1) = q \mid \eta_j(k) = p\}$, and with initial probability measure $\pi_{\eta_j}(0) = [\pi_1^j(0),\ldots,\pi_{r_j}^j(k)]$.

In such a model, the PMF associated with the random variable $c_{j,k}$ (execution time of w_j in the k-th period) is represented by the time dependent function $g_j(\cdot,\cdot) : \mathbb{N}\times\mathscr{C}_j \to [0,1]$, where $g_j(k,a)$ represents the probability $\Pr\{c_{j,k} = a\}$. Using the Bayes Theorem we can write $\forall a \in \mathscr{C}_j$

$$g_j(k,a) = \sum_{q=1}^{r_j} \pi_q^j(k) g_{j,q}(a).$$

Due to the fact that $g_j(\cdot,\cdot)$ assumes finitely many values, we can compactly write it as a vector $g_j(k) = [g_j(k,0),\ldots,g_j(k,l_j)]$. Collecting all $g_{j,q}(a)$ $\forall q \in \mathscr{I}_j$ and $\forall a \in \mathscr{C}_j$ in the stochastic matrix

$$G_j = \begin{bmatrix} g_{j,1}(0) & \cdots & g_{j,1}(l_j) \\ \vdots & & \vdots \\ g_{j,r_j}(0) & \cdots & g_{j,r_j}(l_j) \end{bmatrix} \tag{3}$$

allows us to write

$$g_j(k) = \pi_{\eta_j}(k) G_j. \tag{4}$$

Hence, $g_j(k)$ is the probability distribution associated with the stochastic discrete-time process $\left\{c_{j,k}\right\}_{k\in\mathbb{N}}$.

Motivational example The Markov model proposed above has very strong industrial motivations. For instance, in the

automotive domain, the spark ignition controller task changes its computation time with respect to the drive shaft angular event. This task, as well as the active suspensions controller, the Electronic Stability Program (EPS) or the Traction Control System (TCS), are all related to the driving style. In the literature, it is customary to model the behaviour of a driver by using a hidden Markov Model, where each state represents a driving mode (e.g., constant speed, accelerations, neutral). The observations of the behaviour of each driver can in this case be fed into a hidden Markov Model estimator. The DTMC describing the mode changes for the control tasks is then easily obtained.

b) Interference generated by a single task: Let us now consider the problem of computing the interference experienced by the anytime control task w_m due to a higher priority task w_j, with $j < m$. For the sake of simplicity, we make two assumptions: 1) the two tasks do not have any activation offset (i.e., the first job starts at 0 for both), 2) the period of w_m is an integer multiple of the period of w_j: $T_m = h_j T_j$.

Each job of task w_m contains h_j activations of w_j. The amount of interference suffered by the job then depends on the sequence of operating modes for each job of w_j. A probability distribution for the interference can then be found merging two different pieces of information: the distribution that we get for a specific sequence of modes and the probability associated with the different sequences of modes.

Interference for a specific sequence of modes. Consider the $\widetilde{k}^{th}$ job of the task w_m. Let $\widetilde{p} = (p_1, p_2, \ldots, p_{h_j})$ represents the sequence of modes associated to w_j during the $\widetilde{k}^{th}$ job. The interference of w_j is then given by the random variable $\Omega_{m,\widetilde{k}} = \sum_{k=1}^{h_j} c^{p_k}_{j,(\widetilde{k}-1)h+k-1}$. Due to the independence of each $c^q_{j,k}\ \forall q \in \mathscr{I}_j$ for fixed $j \in \{1,\ldots,m-1\}$ and $k \in \mathbb{N}$, we have that the PMF associated to the sequence of modes $\widetilde{p}$ is given by the convolution of all the PMFs of the modes in the sequence:

$$g_{j,\widetilde{p}} = g_{j,p_1} * g_{j,p_2} * \cdots * g_{j,p_{h_j}}, \tag{5}$$

defined in the set $\widetilde{\mathscr{C}_j} \triangleq \mathscr{C}_j^{h_j}$, i.e., the set of all the sequences of h_j computation times.

Probability distributions for mode sequences. Each sequence of modes $\widetilde{p}$ is an element of the set $\widetilde{\mathscr{I}_j} = \mathscr{I}_j \times \ldots \times \mathscr{I}_j$ and can be associated with a possible state of a *lifted* MC obtained from the MC that describes the mode transitions. Strictly speaking, each state of the lifted MC is obtained by collecting h_j samples of the states of $\eta_j(k)$.

A simple example can clarify this idea. Suppose that task w_j has period 1, that task w_m has period 2 and that the operating modes of w_j are 1 and 2. Based on the notation introduced above, the transition from mode p to mode q of the FSHIA–MC $\eta_j(k)$ is associated with a probability $s^j_{p,q}$. The state space of the lifted Markov Chain is, in this example, given by $\{(1,1),(1,2),(2,1),(2,2)\}$. Suppose we want to compute the transition probability between the sequence $(1,2)$ and the sequence $(1,1)$. By applying the Markov property, this probability is simply the one associated with the two transitions $2 \to 1$ and $1 \to 2$; therefore, it is given by $s^j_{2,1} s^j_{1,1}$.

In more general terms, for the lifted MC the state at step $\widetilde{k}$ is given by $\widetilde{\eta}_j(\widetilde{k}) = \Big[\eta_j(\widetilde{k}h_j), \cdots, \eta_j(\widetilde{k}h_j + h_j - 1)\Big]$. The cardinality of the state space is obviously given by $(r_j)^{h_j}$. The transition matrix can be found as follows. Let $\widetilde{p} = (p_1, p_2, \ldots, p_{h_j}) \in \widetilde{\mathscr{I}_j}$ and $\widetilde{q} = (q_1, q_2, \ldots, q_{h_j}) \in \widetilde{\mathscr{I}_j}$, the transition matrix $\widetilde{S}_j$ has elements $\widetilde{s}^j_{\widetilde{p}\widetilde{q}} \triangleq \Pr\{\widetilde{\eta}_j(\widetilde{k}+1) = \widetilde{q} \mid \widetilde{\eta}_j(\widetilde{k}) = \widetilde{p}\}$ given by $\widetilde{s}^j_{\widetilde{p}\widetilde{q}} = s^j_{p_{h_j} q_1} \prod_{l=1}^{h_j - 1} s^j_{q_l q_{l+1}}$.

An important observation to make is that since the original MC (the one describing the mode changes) is a FSHIA–MC it reaches a steady state; hence each mode has a invariant probability distribution (i.p.d). This property is inherited by the lifted MC (which in fact is a different description of the same process). Therefore the lifted MC has an i.p.d. whose $\widetilde{p}$-th element is given by $\widetilde{\overline{\pi}}^j_{\widetilde{p}} = \prod_{l=1}^{h_j - 1} s^j_{p_l p_{l+1}} \overline{\pi}^j_{p_1}$.

Computation of the PMF of the interference. The two results displayed above can be merged exploiting the theorem of total probability. In plain words, the probability that the interference is equal to a specific value is given by the sum of the probabilities that this event occurs for a specific sequence of modes for task w_j weighted by probability that this sequence takes place:

$$\Pr\{\Omega_{m,k} = a\} = \sum_{\widetilde{p} \in \mathscr{I}_j} \Pr\{\Omega_{m,k} = a | \widetilde{p}\} \widetilde{\overline{\pi}}^j_{\widetilde{p}}.$$

Recalling the vector notation (4), we can write the complete PMF for $c_{j,\widetilde{k}}$ as follows

$$\widetilde{g}_j(\widetilde{k}) = \widetilde{\pi}_{\eta_j}(\widetilde{k}) \widetilde{G}_j, \tag{6}$$

where $\widetilde{\pi}_{\eta_j}(\widetilde{k})$ is the probability distribution of the lifted FSHIA–MC $\widetilde{\eta}_j(\widetilde{k})$ and $\widetilde{G}_j = G_j \otimes \cdots \otimes G_j$ (h_j times), where $\otimes$ represents the Kronecker product (see [28]). Due to the use of the Kronecker product, the vector $\widetilde{g}_j(\widetilde{k})$ in (6) has $(l_j)^{h_j}$ elements. However, since we are assuming that the mandatory part of the anytime control task is schedulable, the interference can be no larger than T_m. Therefore, only the first $l_{\max} \triangleq T_m/\Delta T$ elements can be not null. To get ride of the null elements, we can truncate the vector to its first $l_{\max}$ elements by right multiplying $\widetilde{g}_j(\widetilde{k})$ for the matrix $\widetilde{N} = \left[I_{l_{\max} \times l_{\max}},\ 0_{l_{\max} \times \left((l_j)^{h_j} - l_{\max}\right)}\right]^T$.

The probability distribution of the available computation time $\bar{c}_{m,\widetilde{k}}$ of the control task w_m in the $\widetilde{k}$-th period, is simply obtained by $\widetilde{g}_j(\widetilde{k})$ recalling that $\bar{c}_{m,\widetilde{k}} = D_m - \Omega_{m,\widetilde{k}}$ and that, in the case of only one higher priority task, $\Omega_{m,\widetilde{k}} = c_{j,\widetilde{k}}$. Indeed, it will amount only to flipping the truncated vector by using a permutation matrix $\widetilde{M}$, which is a $l_{\max} \times l_{\max}$ matrix with null entries except for the antidiagonal of 1's. Finally, in the case of a unique higher priority task w_j, we can write

$$\pi_{\bar{c}_m}(\widetilde{k}) = \widetilde{\pi}_{\eta_j}(\widetilde{k}) \widetilde{G}_j \widetilde{N} \widetilde{M}.$$

c) Multiple higher priority tasks: A straightforward generalization of this result is derived for more than one higher priority task. We want to write the probability distribution $\pi_{\Omega_m}(k)$ of the total interference $\Omega_{m,k}$ on the control task w_m exerted in the k-th period by tasks w_j having higher priorities. Due to the independence of each FSHIA–MC $\widetilde{\eta}_j(\widetilde{k})$ and each

$c_{j,k}^{\widetilde{p}}$ $\forall j \in \{1,\dots,m-1\}$, $\forall \widetilde{p} \in \widetilde{\mathscr{I}}_j$ for $\widetilde{k}_j \in \mathbb{N}$, it can be written as follows

$$\pi_{\Omega_m}(k) = \bigotimes_{\alpha=1}^{m-1} \widetilde{g}_\alpha(\widetilde{k}_\alpha) = \left(\bigotimes_{\alpha=1}^{m-1} \widetilde{\pi}_{\eta_\alpha}(\widetilde{k}_\alpha)\right)\left(\bigotimes_{\alpha=1}^{m-1} \widetilde{G}_\alpha\right), \quad (7)$$

where we used (6) and the mixed product rule for Kronecker product. A deeper insights in the previous relation is achieved if one recognizes that $\bigotimes_{\alpha=1}^{m-1} \widetilde{\pi}_{\eta_\alpha}(\widetilde{k}_\alpha)$ is the probability distribution of the FSHIA–MC $\rho(k)$ describing all the mode changes occurring in the period T_m to the processes with higher priority than the process w_m. Again due to the independence of each FSHIA–MC $\widetilde{\eta}_j(\widetilde{k}_\alpha)$, the transition matrix R of $\rho(k)$ is simply given by $R = \bigotimes_{\alpha=1}^{m-1} \widetilde{S}_\alpha$. The use of the Kronecker product makes the vector $\pi_{\Omega_m}(k)$ in (7) have $\prod_{\alpha=1}^{m-1}(l_\alpha)^{h_\alpha}$ elements. Since the schedulability of the mandatory part requires that the first $l_{\max}$ be not null, we can define (as in the case of a single task) a truncation matrix N and flipping matrix M of suitable dimension to write

$$\pi_{\bar{c}_m}(k) = \pi_{\Omega_m}(k)NM = \pi_\rho(k)\left(\bigotimes_{\alpha=1}^{i-1} \widetilde{G}_\alpha\right)NM. \quad (8)$$

It can be easily verified that, being the time varying distribution $\pi_{\bar{c}_m}(k)$ linearly related to $\pi_{\Omega_m}(k)$, it converges to a unique invariant distribution since $\rho(k)$ is a FSHIA–MC.

B. Step 2: Probability of execution for the subroutines

As we discussed above, in our model, the *available computation time* $\bar{c}_{m,k}$ is generated by a stochastic process with time-varying distribution. Since this vector takes finitely many values, we can compactly write it as the vector $\pi_{\bar{c}_m}(k) = [\pi_{\bar{c}_m^0}(k), \cdots, \pi_{\bar{c}_m^{l_m}}(k)]$.

The value of $\bar{c}_{m,k}$ for the k^{th} job of the control task is clearly unknown before its execution: the subroutines are sequentially executed until the deadline expires. Hence, the computation of Γ_i cannot start until the computation of Γ_{i-1} has terminated. Let us define $c_{m,k,p}$ as the "computation time required by the k-th job of the control task w_m to execute the first p subroutines". The sequence $\{c_{m,k,p}\}_{k\in\mathbb{N}}$ is assumed an independent and identically distributed (i.i.d.) stochastic process taking values in $\mathscr{C}_m$. Its (time invariant) PMF is given, in vector notation, by $\overline{\pi}_{c_p} = [\overline{\pi}_{c_p^1}, \cdots, \overline{\pi}_{c_p^{l_m}}]$, where $\overline{\pi}_{c_p^q} \triangleq \Pr\{c_{m,k,p} = q\}$, $0 \le \overline{\pi}_{c_p^q} \le 1$ and $\sum_{q=0}^{l_m} \overline{\pi}_{c_p^q} = 1$. Notice that for each subroutine of task w_m, a vector of probabilities $\overline{\pi}_{c_p}$ is needed.

We can now define the *scheduler stochastic process* $\{\tau_k\}_{k\in\mathbb{N}}$, taking values in the discrete set $\mathscr{L}_\tau \triangleq \{1,\cdots,n\}$. In this setting, $\tau_k = p$ means that in the k^{th} job the subroutines up to Γ_p can be executed but Γ_{p+1} can not. Since $\tau_k \ge 1$ the execution of the mandatory part is always guaranteed.

As shown in our previous work [19], defining the cumulative distribution of $c_{m,k,p}$ as $\kappa_{c_{m,p}} = \left[\overline{\pi}_{c_p^1}, \overline{\pi}_{c_p^1} + \overline{\pi}_{c_p^2}, \cdots, \sum_{q=1}^{l_m} \overline{\pi}_{c_p^q}\right]$, we can write the relation between the distributions of the stochastic processes $\bar{c}_{m,k}$ and τ as follows:

$$\pi_\tau(k) = \pi_{\bar{c}_m}(k)L, \quad (9)$$

where

$$L = \begin{bmatrix} \kappa_{c_{m,1}} - \kappa_{c_{m,2}} \\ \vdots \\ \kappa_{c_{m,n-1}} - \kappa_{c_{m,n}} \\ \kappa_{c_{m,n}} \end{bmatrix}^T$$

is a stochastic matrix. Relation (9) expresses the probability distribution of completion of each different subroutine as a linear function of the probability distribution of the available time for the execution of the control task. Therefore, if $\pi_{\bar{c}_m}(k)$ eventually converges to a steady state (invariant) distribution regardless of the initial condition $\pi_{\bar{c}_m}(0)$ (i.e. $\overline{\pi}_{\bar{c}_m} \triangleq \lim_{k\to\infty} \pi_{\bar{c}_m}(k)$), then $\pi_\tau(k)$ converges as well. If we assume, without loss of generality, that $l_m = l_{\max}$, then the distribution in (8) is exactly the one needed in (9) for building $\pi_\tau(k)$. Hence, the machinery of anytime control can soundly be applied.

C. Example

Consider an embedded platform with a real-time operating system (RTOS) that uses the Rate Monotonic scheduling. Let w_1, w_2 and w_3 be the tasks with higher priority than the control task w_4. In order to have the best exploitation of the platform computational resources, we assume a task set with harmonic periods. More in depth, $T_1 = 125\Delta T$, $T_2 = 250\Delta T$, $T_3 = 500\Delta T$ and $T_H = T_4 = 1000\Delta T$, where $\Delta T = 1$ μs.

Assume that w_1 and w_2 have a number of execution modes $r_1 = r_2 = 2$, governed by two FSHIA-MCs, whose transition matrices are respectively

$$S_1 = \begin{bmatrix} 0.25 & 0.75 \\ 0.3 & 0.7 \end{bmatrix} \text{ and } S_2 = \begin{bmatrix} 0.4 & 0.6 \\ 0.35 & 0.65 \end{bmatrix}.$$

The task w_3 has, instead, only one mode of execution.

Since the tasks are always active in the considered time window, $\mathscr{C}_i \triangleq \{1,\cdots,l_i\}$, $\forall i = 1,\cdots,4$. The PMFs $g_{i,j}$ of the workload tasks differ from zero only in the subset $\overline{\mathscr{C}} \subset \mathscr{C}_i$, $\forall i = 1,\cdots,3$, with $\overline{\mathscr{C}} \triangleq \{10,\cdots,70\}$. The PMFs values for the three tasks are summarized in Table II.

On the other hand, let the control task w_4 be implemented by three subroutines. The execution time PMFs are considered to be Kronecker delta centered respectively at $90\Delta T$ for Γ_1, $280\Delta T$ for Γ_1 and Γ_2 and $480\Delta T$ up to Γ_3. The assumption is not infrequent for a task implementing a linear controller, since very often there are no code branches and the computation only accounts for matrices multiplications and vector sums.

Notice that the task set $\mathscr{W}$ of this example is schedulable under Rate Monotonic considering only the first subroutine Γ_1 of w_4. In plain words, the platform will always execute Γ_1, hence a control input to the plant is always produced. By Equation (8), we have that the stationary distribution of the scheduler is given by $\overline{\pi}_\tau = [0.0143, 0.7471, 0.2385]$.

V. Anytime Controller Design

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

PMF	c_j^q												
	10	15	20	25	30	35	40	45	50	55	60	65	70
$g_{1,1}$	0	0	0.3	0	0.31	0	0	0.17	0.22	0	0	0	0
$g_{1,2}$	0	0	0	0	0	0	0.61	0	0	0.03	0.36	0	0
$g_{2,1}$	0	0	0	0	0	0	0.46	0	0.23	0	0.31	0	0
$g_{2,2}$	0.57	0	0.01	0	0	0	0	0	0	0	0	0	0.42
$g_{3,1}$	0.56	0	0.01	0	0	0	0	0	0.01	0	0.42	0	0

Table II
PMFS FOR THE TASK SET $\mathscr{W}$ IN SECTION IV-C.

The design problem for anytime controllers is to obtain a family of controllers which, when executed under the constraints dictated by the schedulability conditions, exhibit an overall performance which exceeds that of the conservative worst-case (non-switching) controller.

It should be observed that designing anytime controllers is a much more complex problem than designing other anytime algorithms, such as e.g. anytime filters, because controllers interact in feedback with a dynamic system, thus giving raise to stability problems.

The main guidelines of our approach to the design of anytime controllers can be summarized as follows.

- *Switched System Performance:* stability and performance of the switched system must be addressed.
- *Hierarchical Design:* controllers must be ordered in a hierarchy of increasing performance;
- *Practicality:* implementation of both the control and the scheduling algorithms must be efficient;
- *Modularity:* computation of more complex controllers should exploit computations of the simpler ones (recommended);

Let the closed-loop system Σ_i obtained by connecting the i–th controller $\Gamma_i(z)$ with the plant $G(z)$ be described by the state-space dynamics $x_{k+1} = \widehat{A}_i x_k$. It can be safely assumed that each of the n ensuing closed loop systems, individually considered, is asymptotically stable (i.e., $\widehat{A}_i$ is a Schur matrix). However, if the execution of different controllers at different times is imposed (e.g. by schedulability constraints as described in Section IV) the resulting Jump Linear System (JLS) may well result in an overall unstable behavior ([26]).

In order to prevent harmful switches between controllers, one can design a *switching policy* that selects which controller should be executed in the next period of the control task.

More precisely, a switching policy is defined as a map $s : \mathbb{N} \to \mathscr{L}_\tau$, $k \mapsto s(k)$, and determines an *upper bound* to the index i of the controller to be executed at time k. In other terms, for the k^{th} period T_m, the system starts computing the controller algorithm until it can provide the output of $\Gamma_{s(k)}$, unless the computational resources are not sufficient. In such a case, a preemption event occurs forcing the control task w_m to provide only the output of $\Gamma_{\tau(k)}$. By applying a switching policy s to the set of controllers Γ_i, $i \in \mathscr{L}_\tau$, a Markov JLS is generated. The set of all the possible switching policies ranges from the most conservative one $s(k) \equiv 1$, i.e. forcing always the execution of the simplest controller Γ_1 of the mandatory part, to the most aggressive one $s(k) \equiv n$, which leads to providing Γ_τ for all k. While the conservative policy always generates a stable system (with poor performance), the aggressive may generate an unstable system.

In our present approach, the switching policy aims at ensuring stability in the stochastic meaning of "Almost Sure" (AS) Stability [15]. In [19] a linear program is presented that, given a set of controllers and a stochastic description of the available computation time, produces a stochastic switching policy σ which guarantees AS stability.

With respect to that work, in this section we consider a more general, and very practically relevant problem of jointly designing the controllers *and* the switching policy.

The general problem of anytime controller design, in our framework, can be put in these terms:

Problem 1:

Given

- A1 a plant $G(z)$;
- A2 an invariant probability distribution $\overline{\pi}_{\overline{c}_m}$ describing the available computation time $\overline{c}_m$ (cf. sec. IV),
- A3 a set of performance requirements P_i, $i = 1,2,\ldots,n$, prioritized such that $P_i \Rightarrow P_{i-1}$, $\forall i$;

find

- B1 a set of controllers Γ_i, $i = 1,2,\ldots,n$;
- B2 an invariant probability distribution $\overline{\pi}_\sigma$ describing the switching policy,

such that

- C1 performance is maximized,

under the constraint that

- D1 the switched system is stable.

Points (A1) and (A2) above have been discussed previously. Typical performance requirements (A3) for Problem 1 are e.g. closed-loop asymptotic stability, steady-state error, rise and settling time, overshoot and undershoot. Alternatively, performance can be specified using quadratic index costs (LQG), and/or in the frequency domain by requiring certain stability margins, or a given $\mathscr{H}_\infty$ norm of the complementary sensitivity function.

The unknown controllers in point (B1) can be parameterized in several different ways. The practicality guideline on the controller implementation however excludes useful but computationally expensive techniques such as Youla parameterization of stabilizing controllers. We therefore recur to a minimal representation of controllers in terms of their transfer function coefficients. As a consequence of the hierarchical design guideline, the order of the Γ_i controller is greater than the order of Γ_{i-1}. The modularity guideline can be implemented by choosing a parallel structure for the controllers, i.e. Γ_i: $\Gamma_1 \triangleq C_1$ and $\Gamma_i \triangleq \sum_{j=1}^{i} C_j$.

Based on the hypothesis that controllers are hierarchically ordered by their closed-loop performance (i.e., use of controller Γ_i provides better results than Γ_j, $j < i$) a natural "Quality of Control" metric for point (C1) can be simply given as

$$J(\overline{\pi}_d) = \sum_{i=1}^{n} d_i^2 \overline{\pi}_{d_i}, \tag{10}$$

i.e. the second moment of a random variable $d \in \{d_1, \ldots, d_n\}$, where d_i is the performance index associated with Γ_i, $0 < d_i < d_j$, $i < j$, associated with the i.p.d. $\overline{\pi}_d$ describing the probability of controller Γ_i to be executed under the switching policy above.

Finally, the crucial condition (D1) can be verified through the sufficient condition for AS-stability (cf. [15]) that there exist a matrix norm $\|\cdot\|_p$, such that

$$\prod_i \|\widehat{A}_i\|_p^{\overline{\pi}_{d_i}} < 1. \tag{11}$$

Notice that this inequality does not specify any particular p–norm, hence $0 < p \leq \infty$ should be considered as a further variable of the optimization problem. p–norm can be efficiently computed by means of the algorithm presented in [29].

Problem 1 is a complex nonlinear constrained optimization problem, whose numerical solution is greatly simplified by the Lemma below.

Lemma 1: For a plant $G(z)$ and a set of controllers Γ_i, $i = 1, 2, \ldots, n$, an AS-stabilizing switching policy exists for the scheduler i.p.d. $\overline{\pi}_\tau = \overline{\pi}_{\overline{c}_m} L$ if the following condition is satisfied

$$\min\{\overline{\pi}_\tau M_c\} < 0 \tag{12}$$

where $(M_c)_{ij} = \ln\left(\left\|\widehat{A}_{\min(i,j)}\right\|\right)$.

As a consequence, the design problem can be subdivided in two steps: I) solve the optimization problem obtained from Problem 1 by removing the variables (B2) and replacing (11) with (12) for (D1); II) once a set of controllers Γ_i and a value of the norm index p is obtained, apply the linear program in [19] to obtain the switching function.

VI. A REAL SYSTEM: SIMULATION AND EXPERIMENT

In this section we report the application of our methodology to a real system. After describing the model of the system, we will show how the control design is carried out. Then, we will provide evidence of the effectiveness of the approach both in simulations and in the experiments. The simulation of the JLS resulting from the closed loop connection has been carried out using TrueTime [30], a tool for co-simulation of a system and real-time embedded controllers. For the implementation, we used a prototyping platform endowed with a real-time operating system (RTOS) that features real-time scheduling based on fixed priorities.

A. The system

The system is a 2 DOF helicopter model (see Figure 2), 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 the pitch angle α and the yaw angle β.

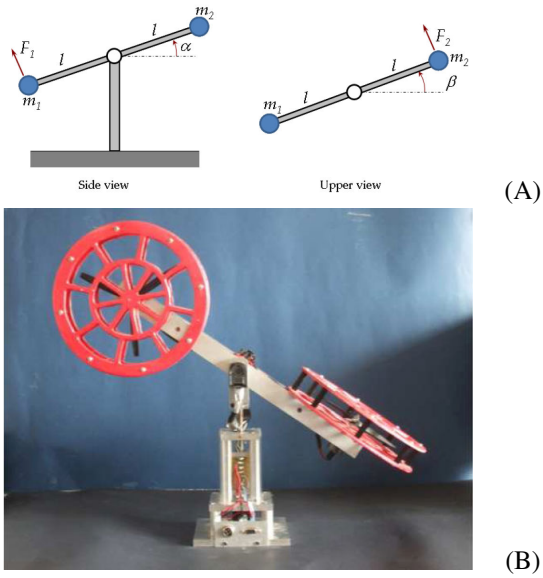

Figure 2. 2 DOF helicopter model. A) schematic diagram , B) picture of the system

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 III
NUMERIC VALUES OF THE HELICOPTER CONSTANTS

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

Numeric values for the helicopter constants are reported in Table III. Linearizing the system around the point $\bar{\alpha} = \bar{\beta} = 0$ rad with $\bar{F}_1 = (m_2 - m_1)g$ and discretizing the continuous-time model with a sample time of 1 ms, we obtained a diagonal discrete-time transfer function matrix $G(z)$. Hence, the dynamics of α and β in the linearized system can be regarded as two independent Single Input Single Output (SISO) systems: $\frac{\alpha(z)}{F_1(z)} = G(z)(1,1) = \frac{-2.16\ 10^{-5}(z+0.999)}{(z-1)(z-0.998)}$ and $\frac{\beta(z)}{F_2(z)} = G(z)(2,2) = \frac{8.42\ 10^{-6}(z+1)}{(z-1)(z-0.999)}$.

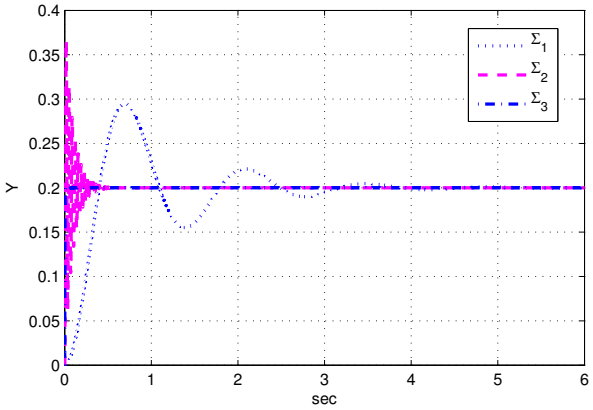

Figure 3. Output of $\Sigma_1, \Sigma_2, \Sigma_3$. The reference signal is a step

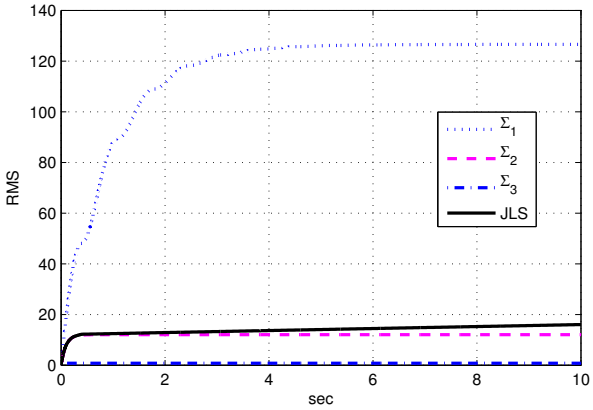

Figure 4. RMSs of $\Sigma_1, \Sigma_2, \Sigma_3$ and Anytime control

B. Controllers design

Due to space constraints, we present the results given by the methodology proposed in Section V only for $G(z)(1,1)$. In addition to stability, the performance are specified by a quadratic cost index and by constraints in the frequency domain. More in depth, the controllers produced by the methodology are: $\Gamma_1(z) = -0.499$ for stability, $C_2(z) \triangleq \Gamma_2 - C_1 = -3196.71\frac{(z-0.56)}{(z^2+0.28z+0.62)}$, which is based on an LQG design, and

$$C_3(z) = -19352.47\frac{(z-0.81)(z+0.38)(z^2+0.15z+0.63)}{(z^2+0.6z+0.143)(z^2+0.29z+0.61)},$$

which minimizes the $\mathscr{H}_\infty$ norm of the t.f. $T_{re}(z)$, i.e., the t.f. between the reference r and the tracking error e. The closed loop output to a step reference is reported in Figure 3.

Assuming the computation times of the three subroutines implementing the controllers and, hence, the invariant probability distribution $\overline{\pi}_\tau$ are given in the example in Section IV-C. A solution to the steady state invariant distribution for the switching policy is then given by $\overline{\pi}_\sigma = [0.0342 \;\; 0.9356 \;\; 0.0302]$. As we can see, the switching policy most likely chooses Σ_2.

C. Simulation Results

Using Matlab/Simulink TrueTime toolbox, we simulated the execution environment with the three load tasks *Load1*, *Load2* and *Load3* having the stochastic characterization given in the example in Section IV-C.

In Figure 4 we compare the Root Mean Square (RMS) of the JLS 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-B.

The performance of the JLS 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.

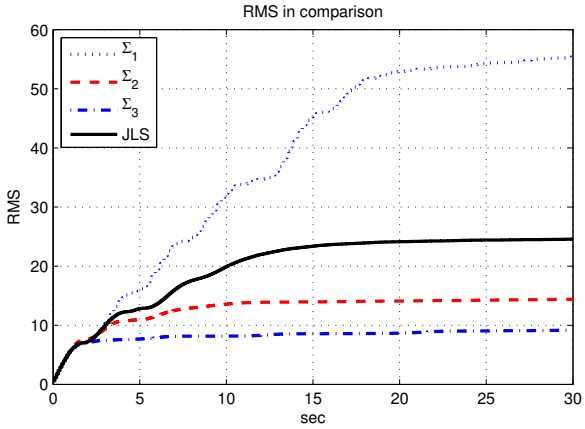

Figure 5. RMSs of $\Sigma_1, \Sigma_2, \Sigma_3$ and Anytime control in the PXI RTOS

D. Experimental Results

The choice of a real hardware-software Real Time platform to implement the Anytime control has been done considering different criteria. We decided for a widespread industrial platform, executing its proprietary RTOS and providing an intuitive visual programming language, the PXI (*PCI eXtensions for Instrumentation*) modular instrumentation platform from *National Instruments*. To monitor and to implement the anytime controller on the PXI platform, we used LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench).

For the implementation of the paradigm, we use the task set $\mathscr{W}$ introduced in Section IV-C, in which we design the three sub-routines of the control task w_4 with the real execution time of the Γ_i controllers that we have found in VI-B.

We applied the resulting architecture to some test regulation tasks. Let us illustrate one of them. The helicopter starts from the initial position $\alpha = 0.59$ rad and $\beta = 0$ rad and reaches the horizontal position $\alpha = \beta = 0$ rad, i.e. the operating point. In Figure 5 we compare the Root Mean Square (RMS) of the JLS with the RMSs of Σ_1, Σ_2 and Σ_3.

Figure 6 depicts the measured outputs (α and β) when the system is perturbed by external forces in order to test its robustness. Starting from the initial position, the heli-

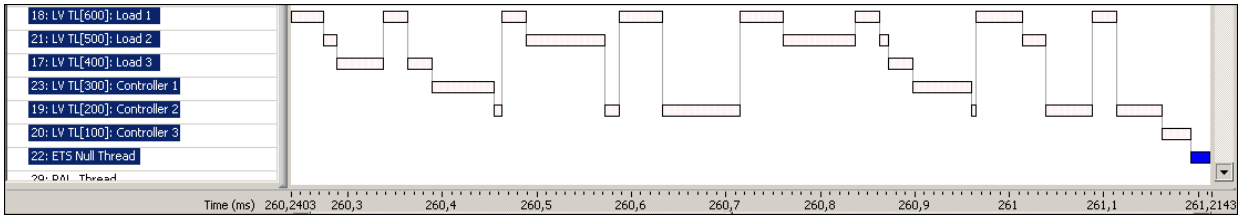

Figure 7. Excerpt of the task set scheduling

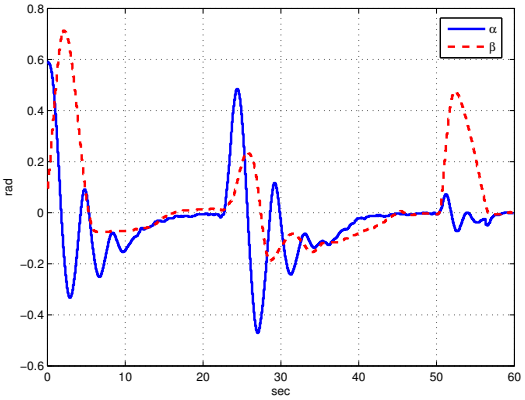

Figure 6. Pitch (α, solid) and yaw (β, dashed) angles of the Anytime controlled helicopter.

copter reaches the horizontal position in about 20 seconds. At $T = 23$ s an external impulsive force is applied to the system in order to perturb the pitch posture. The dynamics of β turns out to be perturbed as well. The system returns to the equilibrium point at $T = 46$ s and after a few seconds another impulse is applied, in order to perturb the yaw posture. The plots in Figure 6 show how the Anytime control stabilizes the system even in presence of strong external disturbances and the anytime regulation task fulfils the control objective in the presence of noise in the input and output channels and of model uncertainties (e.g., resulting from the linearization).

Using a LabVIEW toolkit (*Real-Time execution Trace toolkit*), the task scheduling of the RTOS for the experiment shown in this section could be traced. In Figure 7 we report one hyperperiod as an example. We highlighted the three *Load* task (w_1, w_2 and w_3), the *Controller* task, that are the subroutines of w_4, and, for completeness, the *ETS Null Thread*, 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. It is worth noting that in the computational time of the first controller task there is some additional time (23 μs) accounting for the input acquisition from the helicopter sensors as well as in the third controller task there are 26 μs that are spent in writing the outputs to the motor drivers (hence, a little portion of the third controller is always executed and can be considered as a deferred fragment of the mandatory part).

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, discloses interesting perspectives for the application of this technique to resource constrained embedded controllers.

REFERENCES

[1] 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*, New Orleans, LO, 12–14 December 2007, pp. 5640–5645.

[2] 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*, St. Louis, MO, April 2008, pp. 158–171.

[3] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-feedforward scheduling of control tasks," *Real-Time Systems*, 2002.

[4] 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*, vol. 50, no. 11, pp. 1790–1795, Nov. 2005.

[5] G. C. Buttazzo, M. Velasco, and P. Martí, "Quality-of-control management in overloaded real-time systems," *IEEE Trans. Computers*, vol. 56, no. 2, pp. 253–266, 2007.

[6] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *Automatic Control, IEEE Transactions on*, vol. 52, no. 9, pp. 1680–1685, Sept. 2007.

[7] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE Trans. on Parallel and Distrib. Syst.*, vol. 10, no. 6, pp. 549–559, 1999.

[8] G. Bernat, A. Burns, and A. Llamosi, "Weakly hard real-time systems," *IEEE Trans. on Computers*, vol. 50, no. 4, pp. 308–321, April 2001.

[9] Q. Ling and M. Lemmon, "Robust performance of soft real-time networked control systems with data dropouts," in *Proc. IEEE Int. Conf. on Decision and Control*, vol. 2, 10–13 December 2002, pp. 1225–1230.

[10] ——, "Soft real-time scheduling of networked control systems with dropouts governed by a Markov chain," in *Proc. American Control Conf.*, vol. 6, 4–6 June 2003, pp. 4845–4850.

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

[12] ——, "Firm real-time system scheduling based on a novel QoS constraint," *IEEE Trans. on Computers*, vol. 55, no. 3, pp. 320–333, March 2006.

[13] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computation," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–93, January 1994.

[14] J. 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*, vol. 24, no. 5, pp. 58–68, 1991.

[15] Y. Fang, K. A. Loparo, and X. Feng, "Almost sure and δ-moment stability of jump linear systems," *Int. J. Control*, vol. 59, no. 5, pp. 1281–1307, 1994.

[16] P. Bolzern, P. Colaneri, and G. D. Nicolao, "On almost sure stability of discrete-time Markov jump linear systems," in *Proc. 43rd IEEE Conf. On Decision and Control*, vol. 3, 2004, pp. 3204–3208.

[17] A. Quagli, D. Fontanelli, L. Greco, L. Palopoli, and A. Bicchi, "Designing Real-Time Embedded Controllers using the Anytime Computing Paradigm," in *Proc. IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, Mallorca, Spain, 22-26 September 2009, *Fumio Harashima* Best Paper Award in Emerging Technologies.

[18] K. Kim, J. Diaz, L. Bello, J. Lopez, C.-G. Lee, and S. L. Min, "An exact stochastic analysis of priority-driven periodic real-time systems and its approximations," *Computers, IEEE Transactions on*, vol. 54, no. 11, pp. 1460 – 1466, nov. 2005.

[19] L. Greco, D. Fontanelli, and A. Bicchi, "Design and Stability Analysis for Anytime Control via Stochastic Scheduling," *IEEE Trans. on Automatic Control*, 2010, conditionally Accepted.

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

[21] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[22] C. K. 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.

[23] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems*, 3rd ed. Addison-Wesley, 1998.

[24] B. M. Chen, *Robust and H-infinity Control*, 1st ed. Springer, 30th March 2000.

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

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

[27] D. Liberzon and A. S. Morse, "Basic problems in stability and design of switched systems," *IEEE Control Systems Magazine*, vol. 19, no. 5, pp. 59–70, October 1999.

[28] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[29] N. J. Higham, "Estimating the matrix p-norm," *Numerische Mathematik*, vol. 62, no. 1, pp. 539–555, December 1992.

[30] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén, "How does control timing affect performance?" *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, Jun. 2003.

Andrea Quagli was born in Empoli, Italy, on March, 1981. He received bachelors degree in Computer Engineering in 2004 and the masters degree in Automation Engineering from the University of Pisa in 2008. He is currently with the Robotics Group at the Interdepartmental Research Center "E. Piaggio", University of Pisa. His main research interests are in Real-Time Embedded Control and Networked Control Systems.

Daniele Fotanelli received the M.S. degree in Information Engineering in 2001, and the Ph.D. degree in Automation, Robotics and Bioengineering in 2006, both from the University of Pisa, Pisa, Italy. He was a Visiting Scientist with the Vision Lab of the University of California at Los Angeles, Los Angeles, US, from 2006 to 2007. From 2007 to 2008, he has been an Associate Researcher with the Interdepartment Research Center "E. Piaggio", University of Pisa. From 2008 he joined as an Associate Researcher the Department of Information Engineering and Computer Science, University of Trento, Trento, Italy. His research interests include robotics and visual servoing, embedded system control, wireless sensor networks, networked and distributed control.

Luca Greco received the M.S. degree in Information Engineering in 2001, and the Ph.D. degree in Automation, Robotics and Bioengineering in 2005, both from the University of Pisa, Pisa, Italy. From 2005 to 2007, he has been Associate Researcher at the Interdepartment Research Center "E. Piaggio", University of Pisa. From 2007 to 2009, he has been Associate Researcher at the DIIMA, University of Salerno, Salerno, Italy. He is presently Associate Researcher at LSS - Supélec, Gif-sur-Yvette, France. His main research interests are in hybrid and switching systems, embedded system control, networked control systems.

Luigi Palopoli graduated with a degree in computer engineering from the University of Pisa, Pisa, Italy, in 1992, and received the Ph.D. degree in computer engineering from Scuola Superiore Sant'Anna, Pisa, in 2002. He is an Assistant Professor of Computer Engineering at the University of Trento. His main research activities are in embedded system design with a particular focus on resource-aware control design and adaptive mechanisms for QoS management. He has served in the program committee of different conferences in the area of real-time and control systems.

Antonio Bicchi received the "Laurea" degree in Mechanical Engineering from the University of Pisa in 1984, and the Doctoral degree from the University of Bologna in 1989. After a post-doctoralf fellowship at the Artificial Intelligence lab , Massachusetts Institute of Technology, he joined the Faculty of Engineering in the University of Pisa in 1990. He is Professor of Systems Theory and Robotics in the Department of Electrical Systems and Automation (DSEA) of the University of Pisa and the Director of the Interdepartmental Research Center "E. Piaggio" of the University of Pisa , where he has been leading the Automation and Robotics group since 1990.
His main research interests are in Theory and control of nonlinear systems, in particular hybrid (logic/dynamic, symbol/signal) systems; Dynamics, kinematics and control of complex mechanical systems, including robots, autonomous vehicles, and automotive systems; Haptics, and Dexterous manipulation. He has published more than 250 papers on international journals, books, and refereed conferences. Antonio Bicchi is a Fellow of IEEE.