On the Benefits of Relaxing the Periodicity Assumption for Control Tasks

Adolfo Anta and Paulo Tabuada
Dept. of Electrical Engineering
University of California, Los Angeles
E-mail: {adolfo,tabuada}@ee.ucla.edu

***Abstract*— Feedback control laws have been traditionally treated as periodic tasks when implemented on digital platforms. Although this approach facilitates the scheduling of control tasks, it also leads to inefficient implementations. In this paper we seek to demystify the periodicity assumption in favour of aperiodic self-triggered implementations of control tasks. We show that by adopting aperiodic models for control tasks we can considerably reduce processor utilization while ensuring stability and desired levels of control performance. Based on previous work by the authors, a modification of Cervin and Eker's control server is proposed to fully exploit the benefits afforded by aperiodic self-triggered control tasks. We illustrate the proposed techniques on the control of two jet engine compressors.**

I. Introduction

Historically, control applications have been developed by adopting a separation of concerns between control engineering and real-time scheduling: control engineers design feedback control laws under the assumption that implementation effects are negligible (zero delays and zero computation times) while software engineers schedule control tasks by minimizing jitter and input-output latency in the control loop. This approach leads to overly conservative designs since the same period is used for the control task independently of the processor load and the behavior of the system being controlled. Moreover, the period is designed in order to provide performance guarantees under worst case conditions even if these only rarely occur.

Recently many authors have proposed an integrated study of control design and real time scheduling. Seto et al [SLSS96] approach the problem as an optimization problem, by defining a performance index as a function of both the sampling frequency and the dynamical response of the control system. In [AS90], an online modification of the controller parameters is used to compensate for the implementation effects. Another solution proposed in [CE00] uses feedback from the current state of the tasks to improve the scheduling. Most of this research has been done at the scheduling stage, assuming periodicity of control tasks and therefore unnecessarily overconstraining the design. That is, the starting point for many codesign problems is already based on far-from-optimal design choices. In contrast, a first attempt to study self-triggered models for control tasks was developed in [VFM03], by discretizing the plant; in [LCHZ07], where the computation of the transition matrix is required, making the approach inefficient; and in [AT08], where the scheduling problem was not addressed. We claim that the periodicity assumption is not needed, as it leads to overly conservative designs. This claim is substantiated by previous work by the authors on the real-time requirements of control tasks, reviewed in Section III, and by a modification of the control server, proposed in this paper, that exploits the benefits offered by aperiodic self-triggered models for control tasks.

This research was partially supported by the National Science Foundation EHS award 0712502 and Mutua Madrileña Automovilista.

In addition to advocate the use of aperiodic self-triggered models for control tasks, our contribution is twofold: a modification of the control server to utilise the advantages of the self-triggered model for control tasks; and a particular choice of interface between the control task and the real-time scheduler that facilitates the codesign. This interface allows an online modification of the relative deadlines under overload conditions while preserving stability and performance. We finally illustrate the proposed techniques on the control of two jet engine compressors.

II. Problem statement

The starting point is a control system:

$$\dot{x} = f(x,u), \qquad x \in \mathbb{R}^n,\ u \in \mathbb{R}^m \tag{II.1}$$

for which a feedback controller:

$$u = k(x) \tag{II.2}$$

has been designed, rendering the closed loop system $\dot{x} = f(x, k(x))$ stable. The feedback control law (II.2) is typically implemented in a digital platform by measuring the state x at time instant t_i, computing $u(t_i) = k(x(t_i))$, and updating the actuator values at time instant $t_i + \Delta_i$, where $\Delta_i \geq 0$ represents the time elapsed between the sensor measurement of the state to the update of the actuators.

The problems we are trying to solve can now be posed as follows.

- *How can we adjust deadlines, for control tasks, online so as to guarantee performance and reduce processor usage?*
- *Once deadlines for the control tasks are set, how can we schedule these tasks in a real-time environment?*
- *How can we define a simple interface between control tasks and schedulers that facilitates system codesign?*

To tackle these issues, we will explore the real-time requirements of control tasks discussed in [Tab07] and reviewed in the next section.

III. Event-triggered stabilization of linear systems

Although the results of this paper apply to nonlinear systems, we shall review the event-triggered stabilization in a linear context for simplicity of presentation. In the linear case, the control system defined in (II.1) becomes:

$$\dot{x} = Ax + Bu \tag{III.1}$$

and is asymptotically stabilized by a linear feedback:

$$u = Kx \tag{III.2}$$

The dynamics of the closed loop system under the controller $u = Kx(t_i)$ is given by:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + BKx(t_i) \\ &= (A + BK)x(t) + BKe(t) \end{aligned} \tag{III.3}$$

where the measurement error e is defined by:

$$t \in [t_i + \Delta_i, t_{i+1} + \Delta_{i+1}[\quad \Longrightarrow \quad e(t) = x(t_i) - x(t)$$

Since (III.2) is a stabilizing controller, it is well known from control theory that there exists a Lyapunov function V satisfying:

$$\dot{V} \leq -a|x|^2 + b|x||e| \qquad a, b > 0 \tag{III.4}$$

where $|\cdot|$ denotes the Euclidean norm. If we restrict the error to satisfy:

$$b|e| \leq \sigma\, a|x| \tag{III.5}$$

the dynamics of V is bounded by:

$$\dot{V} \leq (\sigma - 1)a|x|^2$$

thus guaranteeing that V decreases provided that $\sigma < 1$. In the context of nonlinear systems, equation (III.4) becomes:

$$\dot{V} \leq -\alpha(|x|) + \gamma(|e|) \tag{III.6}$$

where α and γ are strictly increasing continuous functions with $\alpha(0) = \gamma(0) = 0$; and (III.5) is replaced by:

$$\gamma(|e|) \leq \sigma\alpha(|x|) \tag{III.7}$$

to preserve stability of the control loop. Inequality (III.5) can be enforced by executing the control task whenever:

$$|e| = \sigma\frac{a}{b}|x| \tag{III.8}$$

Every time the control task is executed, the current state is measured, making $x(t_i) = x(t)$ which implies $e(t) = x(t_i) - x(t) = 0$ and thus enforcing (III.5). Equality (III.8) generates a sequence of deadlines at which the control task has to be executed in order to guarantee stability. This strategy leads to a lower number of executions than the conservative periodic task model, since the controller is only updated when it is indeed required. The parameter σ represents the rate of convergence of the dynamical system and at the same time it determines how frequently the controller will be updated. Thus this parameter σ represents a simple abstraction of the control performance that will facilitate the codesign.

IV. Self-triggered stabilization of nonlinear systems

An event-triggered implementation based on equality (III.8) would require testing (III.8) frequently. Unless this testing process is implemented in hardware, one might run the risk of consuming the processor time freed-up by using an event-triggered implementation to test (III.8). A better solution that we propose here is to use the current measurement of the state to set the next deadline for the task, that is, a self-triggered control task.

To find the sequence of deadlines $\{d_i\}$ described by equality (III.8) we need to analyze the dynamics of the control system, that determines the evolution of the ratio $\frac{|e|}{|x|}$. The procedure is described in detail in our previous work [AT08]. Due to space limitations, we briefly summarize the idea here:

- To derive a self-triggered condition, the relative deadlines of a control task should be expressed in terms of the measured state. At a particular state $x(t_j)$, the relative deadline $d(x(t_j))$ is related to the deadline for another state $d(x(t_i))$ according to the formula:

  $$d(x(t_j)) = \chi(x(t_j)) \cdot d(x(t_i)) \tag{IV.1}$$

  where $\chi(\cdot)$ is a function that is determined by the dynamics of the closed loop system. This equation allows us to obtain a sequence of relative deadlines once the initial deadline is known.
- In order to apply equation (IV.1) online, it is necessary to find a deadline preserving stability for the initial condition. It was shown in ([Tab07]) that this deadline can be obtained from the following equation:

  $$\tau^* = \alpha_1 + \alpha_2 \arctan(\alpha_3 + \sigma \cdot \alpha_4) \tag{IV.2}$$

  where each α_i is a function of the dynamics of the system (II.1) and the controller (II.2).
- In equation (IV.1), if we let $d(x(t_j))$ be the next relative deadline d_j and $d(x(t_i))$ be the initial deadline τ^*, we obtain the following self-triggered condition to be used online:

  $$d_j = \chi(x(t_j)) \cdot \tau^* \tag{IV.3}$$

  Hence the deadlines depend on the current state and on τ^*, which is in turn a function of σ, the control performance. The scheduler could modify online the value of τ^* to adjust for the processor load or to optimize global performance.

V. Scheduling self-triggered control tasks

Most of the current scheduling techniques for control tasks assume periodicity, and tend to reduce latency and jitter. When designing controllers, it is difficult to deal with unknown delays but feasible to account for a constant input-output latency. One way to achieve this fixed delay (and to keep it as low as possible) is through the control server, introduced in [CE03]. Although the control server was developed for periodic control tasks, here it will be extended for sporadic tasks: hence we will work with densities rather

than dealing with utilization factors. We assume at the outset preemptive EDF scheduling in a uniprocessor system.

A. Schedulability

Two categories of tasks are considered:

- Control tasks C_i, that appear herein as sporadic tasks. As it was mentioned before in equation (IV.3), the deadlines are functions of the control performance, and any value of σ less than 1 guarantees stability. Hence we can talk about a range $[\hat{d}_i^k, \widetilde{d}_i^k]$ of possible deadlines associated with a range $[\hat{\sigma}_i, \widetilde{\sigma}_i]$ of allowed performance. Here $\hat{\sigma}_i$ represents the desired performance and $\widetilde{\sigma}_i$ is the lowest performance allowed (i.e., maximum value of σ).
- Other hard tasks O_i, either periodic or aperiodic.

Each hard task is comprised of a string of jobs $\{J_i^k\}_{k\in K} = \{J_i^1, J_i^2, \ldots\}$ with execution times c_i^k, relative deadlines d_i^k, density $\beta_i^k = c_i^k/d_i^k$ and instantaneous utilization $\max_k \beta_i^k$. For the control tasks, instead of a fixed deadline d_i^k we have the range $[\hat{d}_i^k, \widetilde{d}_i^k]$ and the corresponding density range $[\widetilde{\beta}_k^i, \hat{\beta}_k^i]$. It is well known that this set of tasks is schedulable if the total sum of the instantaneous utilizations is less than 1:

$$\sum_{C_i} \max_k \widetilde{\beta}_i^k + \sum_{O_i} \max_k \beta_i^k \leq 1 \qquad \text{(V.1)}$$

It is straightforward to check schedulability under this setup since an upper bound for the density $\widetilde{\beta}_i^k$ is known. To achieve a fixed latency we resort to the control server, that is briefly reviewed in the next section.

B. The control server

To reduce the latency, the job of a control task J_i^k may be split into several segments $\{S_{ij}^k\}_{j\in J} = \{S_{i1}^k, S_{i2}^k, \ldots\}$. Each segment is assigned a relative deadline d_{ij}^k (or length) according to:

$$d_{ij}^k = \frac{c_{ij}^k}{\beta_i^k}$$

where c_{ij}^k is the computation time of segment j. This assignment of deadlines preserves the density of the job of the control task while achieving a shorter latency (that is in fact the length of the corresponding segment).

We extend this concept to reduce even further the latency of the control tasks: if there is some available time in the CPU, density β_i^k can be increased in order to decrease d_{ij}^k, as shown in Figure 1. This procedure creates an artificial segment (S_{i3}^k in the diagram) with an assigned density β_i^k (since density has to be the same for all segments of a task), and this spare segment can be alloted to low priority tasks. More precisely, let the total density of the task set be $\Gamma^k = \sum_i \beta_i^k$. Hence the spare density becomes $\Delta\Gamma^k = 1 - \Gamma^k$ and it could be split between the n control tasks to increase their density (and thus reducing the latency). For instance, if we consider different weights ω_i for each of the n control tasks, the new densities will be given by:

$$\beta_{i_{new}}^k = \beta_i^k + \frac{\omega_i}{\sum_i \omega_i} \Delta\Gamma^k$$

Thus the new latencies become $d_{ij_{new}}^k = c_{ij}^k/\beta_{i_{new}}^k$. This strategy preserves schedulability while reducing delays in the control loops. At the same time, the scheduler could modify online the value of σ in order to allocate more resources for high priority tasks or to accept new incoming tasks.

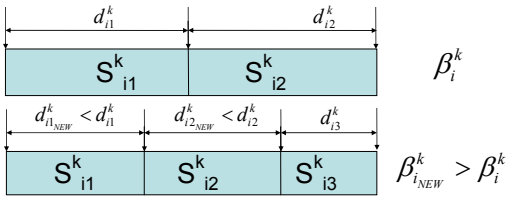

Fig. 1. Reducing latency with the control server

VI. Example

To illustrate the benefits of the previous approach, we consider a computational unit in charge of the control of two jet engine compressors. The processor is also executing a (hard) periodic task in addition to several (soft) aperiodic tasks. A simple FIFO queue handles the soft tasks. The first step in the analysis consists in the design of the controllers. We borrow the following model of a jet engine compressor from [KK95]:

$$\begin{aligned} \dot{\phi} &= -\psi - \frac{3}{2}\phi^2 - \frac{1}{2}\phi^3 \\ \dot{\psi} &= \frac{1}{\beta^2}(\phi - \phi_T) \end{aligned} \qquad \text{(VI.1)}$$

where ϕ is the mass flow, β a constant positive parameter, ψ is the pressure rise and ϕ_T corresponds to the throttle mass flow. A control law $\phi_T = g(\phi, \psi)$ is designed to render the closed loop globally asymptotically stable. The closed loop equations are:

$$\begin{aligned} \dot{\phi} &= -\frac{1}{2}(\phi^2 + 1)(\phi + y) \\ \dot{y} &= -(\phi^2 + 1)y \end{aligned}$$

where we have applied the nonlinear change of coordinates $y = 2\frac{\phi^2+\psi}{\phi^2+1}$. Applying equation (IV.3), we obtain the following formula describing the relative deadlines for the control task:

$$d_{i+1} = \frac{29\phi(t_i) + r^2}{5.36r\phi(t_i)^2 + r^2} \cdot \tau^* \qquad \text{(VI.2)}$$

where r is the norm of the previously measured state $(\phi(t_i), y(t_i))$ and $\tau^* \in [0.3\text{ms}, 9.2\text{ms}]$ (computed from (IV.2)) to preserve the stability of the system. The computation time for each control task is 2ms. The operation region will be a ball of radius 5 centered at the origin. In order to show the effectiveness of the approach, we consider 50 different initial conditions equally distributed along the boundary of the operation region. Let the desired performance be $\sigma = 0.33$ for both systems. This implies that the relative deadlines generated by equation (VI.2) are lower bounded by $\hat{d}_{i+1} \geq 7.63$ms, and thus density are upper bounded by $\hat{\beta} \leq 0.26$. The hard periodic task has period

σ	periodic	self-triggered
0.11	890	119
0.22	506	66
0.33	397	51

TABLE I

NUMBER OF EXECUTIONS OF THE CONTROL TASK FOR TIME = 3S.

$T_p = 5$ms and computation time $C_p = 1$ms. We check the schedulability of this set of tasks:

$$\sum_{C_i} \beta_i^k + \beta_{per} \leq \frac{2}{7.63} + \frac{2}{7.63} + \frac{1}{5} = 0.72$$

Since we still have some spare density $\Delta\Gamma = 0.28$, we can take advantage of the control server properties to reduce the latency in both control tasks. Density for each task can be increased in $\frac{\Delta\Gamma}{2} = 0.14$, leading to a reduction of 34% in the latency.

In Figures 2 and 3 we compare the behaviour of both strategies, periodic and self-triggered. To choose a stabilizing period for our system, we select the worst case relative deadline obtained from (VI.2) (other procedures could be applied, leading to similar values). A disturbance is applied at $t = 0.7s$ to both control systems to check the robustness of our strategy. Both systems exhibit a similar behaviour for the state variables for any initial condition (see Figure 2 for one particular initial condition). Figure 3 shows the evolution of the input for the control system. At the beginning, both the periodic and aperiodic use the same relative deadline, but as the system tends to the equilibrium point the aperiodic policy increases the time between executions, whereas the periodic policy keeps updating the controller at the same rate. The right side of Figure 3 zooms the last part of the simulation, where the inter-execution times for the aperiodic strategy is already 24 times larger than the periodic. Hence the self-triggered implementation leads to a much smaller number of executions, while achieving a similar performance. The number of executions required under the control server strategy for both implementations are shown in Table I, for different values of σ (and averaged over all initial conditions considered): the aperiodic policy executes the controller nearly 8 less times than the periodic for a simulation time of 3s. Finally, Figure 4 shows the schedule for the first second. At the beginning, both control tasks require more CPU time so the queue with the soft tasks is always full; then, inter-execution times tend to enlarge as the system tends to the equilibrium point, giving more CPU time to the soft tasks. At $t = 0.7$ the disturbance steers the system far from the origin, and therefore the CPU reduces the deadlines accordingly to guarantee the required performance at the expense of delaying other soft tasks.

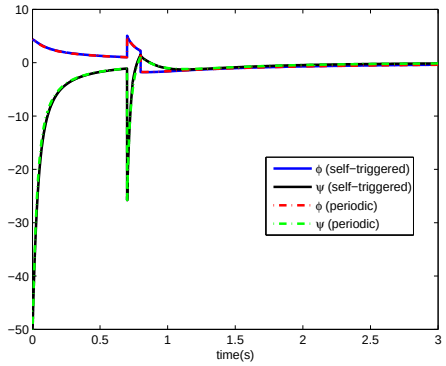

Fig. 2. Evolution of the states for self-triggered and periodic strategies for control task 1

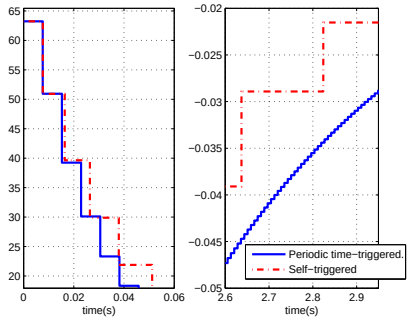

Fig. 3. Control input for periodic and self-triggered implementation

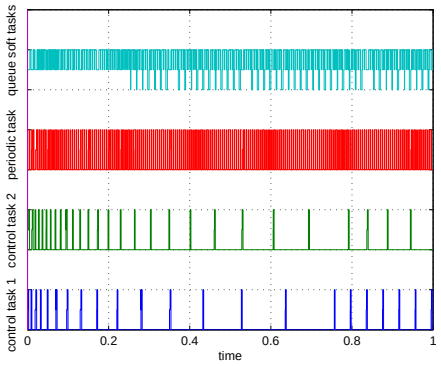

Fig. 4. Scheduling of self-triggered control tasks with the control server

REFERENCES

[AS90] P. Albertos and J. Salt. Digital Regulators Redesign with Irregular Sampling. *11th IFAC World Congress*, 1990.

[AT08] A. Anta and P. Tabuada. Self-triggered stabilization of homogeneous control systems. *To appear in American Control Conference. Available at http://www.ee.ucla.edu/∼adolfo*, 2008.

[CE00] A. Cervin and J. Eker. Feedback scheduling of control tasks. *Conference on Decision and Control*, 2000.

[CE03] A. Cervin and J. Eker. The control server: a computational model for real-time control tasks. *15th Euromicro Conference on Real-Time Systems*, pages 113–120, 2003.

[CHL+03] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.E. Arzen. How does control timing affect performance? *Control Systems Magazine, IEEE*, 23(3):16–30, 2003.

[KK95] M. Krstic and P.V. Kokotovic. Lean backstepping design for a jet engine compressor model. *Proceedings of the 4th IEEE Conference on Control Applications*, 1995.

[LCHZ07] M. Lemmon, T. Chantem, X. Hu, and M. Zyskowski. On Self-Triggered Full Information H-infinity Controllers. *Hybrid Systems: Computation and Control, April*, 2007.

[SLSS96] D. Seto, JP Lehoczky, L. Sha, and KG Shin. On task schedulability in real-time control systems. *17th IEEE RTSS*, 1996.

[Tab07] P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE TAC*, 52(9):1680–1685, 2007.

[VFM03] M. Velasco, J. Fuertes, and P. Marti. The self triggered task model for real-time control systems. *RTSS WIP'03*, 2003.