Adaptive Design of Real-Time Control Systems subject to Sporadic Overruns

Paolo Pazzaglia[1], Arne Hamann[2], Dirk Ziegenbein[2], and Martina Maggio[1]
[1]Saarland University, Department of Computer Science – Email: {pazzaglia, maggio}@cs.uni-saarland.de
[2]Robert Bosch GmbH – Email: {arne.hamann, dirk.ziegenbein}@de.bosch.com

***Abstract*—Most off-the-shelf embedded control systems lack proper mechanisms to handle computational overload conditions. Therefore, delays may accumulate and produce *overruns*, potentially harming the stability and performance of the controlled system. In this paper, we explore a controller implementation in which overrun events are tolerated and tackled with a proper countermeasure, which can be easily plugged into existing controller implementations and in particular commercial off-the-shelf control systems. When an overrun occurs, the control period of the next job is reinitialized and its control parameters are adjusted to counteract the additional delay of the previous job. The main strength of this approach resides in a straightforward applicability and in a high flexibility in deployment. It does neither require a stochastic model of the timing evolution of the system, nor rely on prediction of future delays. We provide an exact tool to determine the system stability, which requires only the knowledge of the worst case response time. The final controlled system exhibits a good trade-off between simplicity and performance, both during nominal and overload conditions.**

I. Introduction

Real-time embedded platforms represent a widespread choice for the implementation of commercial control applications. Those systems are characterized by having multiple functionalities (*tasks*) executing on the same platform, thus competing for the same limited and shared resources, both for computation and communication. As a result, the time required to compute and transfer data can not be considered as negligible. When the computational burden and the amount of data to be transferred vary significantly over time, the system may experience transient *overload conditions*. The underlying causes may differ. They range from variations in execution time due to data-dependent software paths, to preemption from higher priority tasks and bursts of interrupts, or hardware-related effects such as cache misses. During overload conditions, some tasks may not complete before the next periodic instance – a situation commonly called *overrun*. As a consequence, the hypothesis of nearly-perfect periodicity, commonly used in control design, cannot be guaranteed; the occurrence of delayed control commands and aperiodic patterns may undermine the system stability and performance, and the loss of timing determinism makes it challenging to certify the overall system.

Despite being far from ideal, the possibility of transient overloads is actually accepted, even in well-designed implementation [1]. On the one hand, the sporadic nature of such events hardly justify the need for a more powerful embedded platform to host the control application – a choice often primarily constrained by a monetary rationale. On the other hand, the task periods cannot be easily modified, especially when dealing with industrial applications and physical requirements. Thus, the engineers are left with the problem of guaranteeing asymptotic stability and good performance for the controlled system during overloads, as well as in nominal timing condition.

This issue is naïvely solved in most industrial applications by conservatively tuning the controller to withstand worst-case conditions. However, if the system behaves nominally for the vast majority of time, this cautious approach generally leads to poor overall performance [2]. More complex control designs and scheduling strategies proposed in literature rely on a detailed knowledge of the timing behavior like a probabilistic description of the execution time [3], [4], or some levels of prediction of the future behavior [5], [6], [7]. However, this level of detail of the timing behavior is often not available in real applications. Additionally, this kind of characterization is strongly platform- and application-dependent, meaning that every modification or migration of the system would require a new thorough characterization and analysis.

Contribution: Inspired by this challenge, we present an *adaptive* control design to handle sporadic overloads, which (in contrast to most state-of-the-art solutions) does not require prediction of future delays and does not rely on a stochastic characterization of the tasks. Our approach integrates a dynamic management of both (i) the release pattern of the control task, and (ii) the control parameters. The timing strategy is built upon the *continuous stream* model of computation [8]. When an instance (*job*) of the control task experiences an overrun, it is allowed to execute until completion, while the next job release is suspended and its period is reinitialized. The release of the next job may occur only at specific instants where fresh sensor data are acquired, in order to balance both flexibility and timing determinism. Furthermore, the control strategy of each job is adjusted to compensate the amount of the (eventual) overrun experienced by the *previous* job. The proposed strategy is non-intrusive and requires only a small modification to classical control algorithms: the actual implementation on a real device can be obtained by means of just a timer and a table of control parameters. We also provide a deterministic stability analysis of the closed-loop system in all possible dynamical evolutions, using the *joint spectral radius* [9].

This research received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH).

II. Related Work

Sporadic overload conditions represent an inevitable scenario for many real-world embedded control applications, especially in the automotive field [1]. In this context, multiple adaptations of the scheduling policy have been proposed by the research community to counteract the effects of overloads, by preventing uncontrolled delays and performance degradation. In [10], when a job is subject to an overrun, its deadline is postponed and the activation pattern of the control task is locally modified. The authors of [11] and [12] propose to handle overruns by either performing budget replenishment, terminating the job that has late execution or skipping successive jobs. These approaches are successful at handling the overrun conditions, but require a non-standardized implementation of the control task, and make it impossible to use off-the-shelf control implementations.

When available, a probabilistic characterization of the timing can be used to produce optimal designs of both the timing strategy and the controller. In [3], the authors leverage this probabilistic description to find an optimal allocation of bandwidth for the tasks, using the aforementioned continuous stream model [8]. Other works use a probabilistic modelization to build a control design that is robust to the corresponding distribution of delays and overruns [13], [4]. Adaptive control strategies have also been explored, but mostly relying on the hypothesis that the task period is modified in advance, so that the controllers can be adjusted accordingly to the period, by varying the rate and the control action [5], [14], [15]. In this work, we make no assumption of advance knowledge on the probability distribution of delays and overruns.

In other research work, a probabilistic description of the timing of tasks is replaced by a worst-case bound on the number of missed deadlines, in the form of the so-called *weakly hard* model [16]. This model has been leveraged to test the stability [17], [18] and the performance [19] of controllers in overload conditions. More in general, the problem of stability of aperiodic systems has raised much interest in recent years in the control community and a fairly exhaustive survey on the topic can be found in [20]. In [21], the authors propose a sufficient stability test, which consists in building a single dynamic matrix that upper-bounds the system dynamics, leveraging interval algebra. The resulting approach is simple and elegant. However, the obtained results are extremely conservative and the stability condition can easily exceed in pessimism.

In this paper, we propose a joint strategy of modifying both the timing pattern and the controller parameters in case of overruns, without explicit knowledge of the probabilistic behavior of the control task. To the best of our knowledge, this is the first work exploring the combination of timing adjustments and adaptive control that can be applied directly to off-the-shelf control strategies, and their implementation. We can exploit our technique and benefit from a better ratio between performance and complexity.

III. System Model

The plant to be controlled is modelled as a classical linear time-invariant system in continuous time, with multiple input and multiple output signals. We use $x\,(t) \in \mathbb{R}^n$ to indicate the system state vector at time t and $\dot{x}\,(t)$ for its time derivative. Similarly, $u\,(t) \in \mathbb{R}^r$ represents the control signal and $y\,(t) \in \mathbb{R}^q$ the system output (i.e., the sensor measurements), at time t. The plant equations are:

$$\begin{aligned} \dot{x}\,(t) &= A\,x\,(t) + B\,u\,(t)\,. \\ y\,(t) &= C\,x\,(t)\,, \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times r}$ and $C \in \mathbb{R}^{q \times n}$ are constant matrices representing the dynamic evolution of the system. In line with standard assumptions, we assume the system to be controllable and the state to be fully observable. Although our method addresses solely linear time-invariant systems, our proposal could be extended to nonlinear systems via hybridisation of the system dynamics [22].

The control command $u\,(t)$ is computed according to a linear feedback control law, implemented as a real-time task which executes on a single core embedded platform, possibly alongside other concurrent tasks. We refer to an arbitrary control law computation using the term *job*. We denote with a_k and f_k the release time and the finishing time of an arbitrary k-th job of the control task, respectively. We assume that the response time of each job, i.e., $R_k = f_k - a_k$, may assume values ranging in a given interval, $R_k \in [R_{\min}, R_{\max}]$, where $R_{\min}$ is the job's best-case response time and $R_{\max}$ is its worst-case response time. In the absence of worst-case response time knowledge, an upper bound for its value can be used. An arbitrary k-th control job released at time a_k:

(i) makes a copy of the sensor data sampled at a_k, that will use for its entire execution;
(ii) computes the control law for the next update; and
(iii) saves the control command in a local memory slot.

The freshly computed control command will then be updated to the actuator only at the release instant of the *next* job. The update mechanism is performed by a dedicated task with highest priority and regarded as an instantaneous process. The resulting control signal $u\,(t)$ is thus a piece-wise function of t, updated only at specific points in time (a_k) and held constant between two successive updates.

In nominal conditions, the control task is executed with a *basic* periodic pattern of period T and deadline $D = T$. The sensors – producing the vector $y\,(t)$ – are managed by a dedicated hardware task. They sample the plant periodically with period $T_s = T/N_s$, where N_s is a positive integer, i.e., over-sampling the control period with a factor N_s. At each sensors activation, the sensed values are updated in a dedicated memory slot. We make the (reasonable) assumption that this operation occurs with negligible jitter. At the start of our control application, the first activation of the sensor sampling is synchronized with the first release of the control task. Thus, under perfect periodicity conditions, the control task would be released exactly once every N_s sensor samplings.

The choice of the control task period T is a delicate matter that influences the control performance and the computation requirements [23]. We assume that $T \geq R_{\min}$, meaning that the period is always greater than the minimum response time

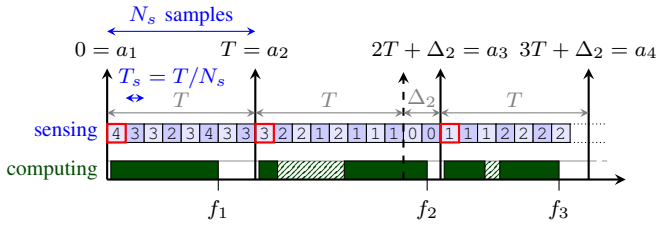

Fig. 1. Example sequence of three jobs with an overrun.

of the task. On the contrary, selecting a period $T \geq R_{\max}$ could possibly be too restrictive in terms of achieved control performance [4], since oftentimes in real applications only a small subset of periods (e.g., *harmonic* periods) can be selected in an application, and the controller would not execute often enough to guarantee a good quality of control. Thus, we consider the case where the chosen period satisfies $T < R_{\max}$. This is in line with many practically relevant cases, such as in the automotive domain, where the selection of T follows the rule of thumb that a small probability of the job response time exceeding the period duration is acceptable and tolerated [1].

Figure 1 shows a timing example, illustrating a timeline for both sensing and computing, in which $N_s = 8$. During the first period, the computation starts with the freshest sensor measurement, and completes within T time units. The second job, on the contrary, is preempted (dashed interval) and is not able to complete within T time units, but terminates after a delay, with finishing time $f_2 > 2\,T$. We can see that the third job is then released only at time $a_3 = 2\,T + \Delta_2$, as soon as the next sensor measurement is available. This is dictated by the chosen strategy in case of overruns, which is discussed in detail in the following section.

IV. Adaptive Control Task Design

When a control task experiences an overrun, the computation of the control signal is not correctly terminated within the control period T. This timing exception requires a proper countermeasure to retain control performance, avoid cascaded delays on the next jobs and inconsistencies in the data flow. In this section we illustrate our proposed strategy that (i) handles the timing exception by relaxing the periodic pattern when an overrun occurs, and (ii) properly adapts the control action to compensate the experienced timing deviation.

A. Period Adaptation

The timing adaptation in case of overruns is realized through a mechanism largely inspired from the so-called *continuous stream* model of computation [8]. In synthesis, the proposed timing adaptation always guarantees the completion of the job that is experiencing an overrun, effectively extending its deadline. This is done by postponing the release of the next job, to happen only after the current job finishes its execution. The release of this new control job is further constrained to occur synchronously with the first available sensor reading, to avoid sampling jitter and consolidate timing determinism in the system. Additionally, the postponed job will then benefit from a "period reset" at its release, i.e., its deadline is set to $D = T$. An example of this mechanism is visible in Figure 1, where $f_2 \neq a_3$ and a_3 is selected as the time in which the first sensor reading after f_2 takes place.

The timing model is here formally described. Let us consider an arbitrary k-th job that is released at time a_k. If the job completes *within* T time units, the next job will be released at $a_{k+1} = a_k + T$. Otherwise (i.e., if $R_k > T$), the next job will be released at $a_{k+1} = a_k + \lceil R_k / T_s \rceil T_s$, that coincides with a new sensor data. We introduce h_k as the time interval between two successive job releases, formally computed as follows:

$$h_k = a_{k+1} - a_k = T + \Delta_k. \tag{2}$$

In other words, h_k is the actual duration of the k-th job and can be equivalently computed as T plus an amount Δ_k that represents the (eventual) additional time interval due to the overrun of the k-th job (inflated with the time to wait for the next sampling instant). Trivially, if no overrun occurs to the k-th job, then $\Delta_k = 0$. We can then express the activation time of the k-th control job as $a_k = (k-1)\,T + \sum_{i=1}^{k-1} \Delta_i$. We denote $\Delta_{\max}$ as the maximum delay experienced by the control task. To ease the notation in the following parts, we introduce the set $\mathcal{H}$ containing all possible values of h_k, i.e., $h_k \in \mathcal{H}$, and defined as follows:

$$\mathcal{H} := \{\, T + i \cdot T_s \,\},\ i \in \mathbb{Z}^{\geq 0} \wedge 0 \leq i \leq \left\lceil \tfrac{R_{\max} - T}{T_s} \right\rceil. \tag{3}$$

The adoption of this period adaptation mechanism produces several advantages:

(i) it prevents cascaded delays, promoting timing independence between the control jobs;
(ii) it is independent from the chosen scheduler, since it only affects the release pattern of the control jobs; and
(iii) restricting the release to specific instants improves time determinism and allows for a more manageable analysis.

Since the control output is communicated only at the beginning of the next job, h_k also coincides with the input-output delay of the k-th job. Thus, a discrete-time model of the plant (1), for the interval $[a_k, a_k + h_k)$ can then be defined as follows:

$$\begin{aligned} x\,[k+1] &= \Phi\,(h_k)\ x\,[k] + \Gamma\,(h_k)\ u\,[k] \\ y\,[k] &= C\,x\,[k]\,, \end{aligned} \tag{4}$$

where $x\,[k]$ is the discrete time representation of the state sampled at time a_k and $x\,[k+1]$ is the state sampled at $a_k + h_k$, while $u\,[k]$ is the control output that is applied from a_k until the end of the interval under analysis. The dynamic matrices are computed using classic control theoretical results [23], as a function of the interval h_k, as follows:

$$\Phi\,(h_k) = e^{A\,h_k} \quad \text{and} \quad \Gamma\,(h_k) = \int_0^{h_k} e^{A\,s} ds\, B. \tag{5}$$

B. Adaptive Control

Our strategy includes also a control adaptation mechanism in case of overruns, alongside the timing pattern adaptation presented above. As introduced in Section III, a control task released at a_k will produce the control command only at the next release instant a_{k+1}. Thus, if the control task was perfectly periodic with period T, one could leverage classic control design to build a controller that is robust to the resulting input-output delay T. However, when an overrun occurs, a

proper correction is required to dynamically adjust for the additional delay. The proposed approach is to assign to the k-th control job the role of compensating the overrun-induced delay of the previous job, namely Δ_{k-1}. This is required in particular to adjust the internal states of the controller (such as the integrator states), that are normally updated only up to the deadline $D = T$ and require a complement for the additional delay, to prevent inconsistencies in the control computation.

In our formulation, we build one control mode for each input-output interval in $\mathcal{H}$. A generic state-space formulation of the controller, for an arbitrary k-th job, will look as follows:

$$\begin{aligned} z[k+1] &= A_c(h_{k-1})\, z[k] + B_c(h_{k-1})\, e[k] \\ u[k+1] &= C_c(h_{k-1})\, z[k] + D_c(h_{k-1})\, e[k]. \end{aligned} \tag{6}$$

Here, $e[k] = r - y[k]$ represents the error between the reference signal r and the plant output sampled at time a_k, while $z[k] \in \mathbb{R}^s$ is the state of the controller. The matrices $A_c(h_{k-1}) \in \mathbb{R}^{s\times s}$, $B_c(h_{k-1}) \in \mathbb{R}^{s\times q}$, $C_c(h_{k-1}) \in \mathbb{R}^{r\times s}$ and $D_c(h_{k-1}) \in \mathbb{R}^{r\times q}$ define the controller dynamics.

In line with classic discrete time modeling, our controller assumes that the error measured at a_k remains constant in the following interval of T time units. If $\Delta_{k-1} \neq 0$ the error is assumed constant also during the delay experienced by the previous job. In Section VI, we show that this approximation is generally acceptable in terms of resulting control performance, in particular in the common cases when $\Delta_{\max}$ is small. The controller matrices A_c, B_c, C_c and D_c will then be designed for a system with plant output sampled at a_k and with input-output interval $\Delta_{k-1} + T = h_{k-1}$. In case of no overruns, the controller defined in (6) works exactly as a classic control designed for an input-output delay equal to its basic period T.

Depending on the controller type, different design techniques can be leveraged. For the case of a state-feedback controller synthesized using linear quadratic regulator theory, there are well-understood design techniques to deal with delays [24]. In this case, we consider $e[k] = x[k]$ and the matrices A_c, B_c, C_c are set to zero (as the controller has no internal state). The matrix $D_c(h_{k-1})$ is set to implement the Linear Quadratic Gaussian (LQG) controller that is optimal with respect to the delay h_{k-1}. If the state is not measurable, an observer is added, and the controller state and matrix reflect the observer behavior. The parameter $z[k]$ represents then an estimate of the plant state, and the control signal is computed using the estimate (matrix C_c), rather than the state measurement. This design provides optimal guarantees for the specific sampling rate.

Another widespread option is the Proportional and Integral (PI) controller, representing more than 90% of all industrial controllers [2]. In this case, the controller will look as follows:

$$\begin{aligned} z[k+1] &= z[k] + h_{k-1} \cdot e[k] \\ u[k+1] &= \bar{K}_P(h_{k-1})\, e[k] + \bar{K}_I(h_{k-1})\, z[k] \end{aligned} \tag{7}$$

where $z[k]$ here represents the discrete time integral of the error, computed with the forward Euler method for the interval $\Delta_{k-1} + T = h_{k-1}$. The gains $\bar{K}_P$ and $\bar{K}_I$ can be optimized as functions of h_{k-1} following standard heuristic procedures.

Our computational model corresponds to the following implementation of the control job.

```
1  while(true) {
2   if (new_data) { // new sensor data
3    t_start = get_time();
4    y = read_data(); u = compute_ctl(y, h);
5    h = get_time() - t_start;
6    if (h < period) sleep(period - h);
7  } }
```

In the code, the function `compute_ctl` uses the measured output `y` and the previous input-output interval `h` to select a controller that computes the value of the control signal that is going to be applied during the following period. The execution of the control function is conditional to the arrival of a new sensor sample. Note that the usage of the `sleep` primitive (line 9) is not ideal in this context, but is however extremely common, especially in industrial and off-the-shelf controllers. In fact, the `sleep_until` primitive would be a better choice to increase timing precision while still avoiding polling, but it is not always available for many operating systems and programming languages.

V. Stability Analysis

Studying the stability of the controlled system requires checking that, for all possible sequences of ordered control jobs, the dynamics of the plant always asymptotically converges to the stable state. In the particular case under analysis, however, the system dynamics can be effectively defined as a *switching* discrete time system. As a consequence, we are required to collect the dynamic matrices of the closed loop for each possible input-output value $h_i \in \mathcal{H}$ and test all feasible combinations of these matrices. Since the plant (5) is function of h_k while the controller (6) is function of h_{k-1}, a naïve formulation of the closed loop system would produce a dynamic matrix that is function of both h_k, h_{k-1}. This however yields to additional complexity, since we would end up with $\#\mathcal{H}^2$ different closed loop matrices, together with ordering constraints involving the pair (h_k, h_{k-1}).

To reduce the complexity of the stability analysis algorithm, we propose here an alternative modelization, where the closed loop dynamics is defined as function of h_k only. First of all, two auxiliary variables are introduced, namely $\tilde{u}[k] = u[k+1]$ and $\tilde{z}[k] = z[k+1]$. We then define the lifted vector $\xi(k) \in \mathbb{R}^{(n+s+2r)}$ as $\xi(k) = [x[k]^T, \tilde{z}[k]^T, \tilde{u}[k]^T, u[k]^T]^T$. With some algebraic manipulation of equations (5) and (6), and considering $r = 0$ without loss of generality, the resulting closed loop system can then be written as follows:

$$\xi(k+1) = \Omega(h_k)\, \xi(k), \tag{8}$$

with $\Omega(h_k)$ being the closed loop dynamic matrix, defined as

$$\Omega(h_k) = \begin{bmatrix} \Phi(h_k) & 0 & 0 & \Gamma(h_k) \\ B_c(h_k)\, C\, \Phi(h_k) & A_c(h_k) & 0 & B_c(h_k)\, C\, \Gamma(h_k) \\ D_c(h_k)\, C\, \Phi(h_k) & C_c(h_k) & 0 & D_c(h_k)\, C\, \Gamma(h_k) \\ 0 & 0 & I & 0 \end{bmatrix}$$

where I represents the identity matrix of appropriate size. Solving the stability problem for such a system means that,

for each possible sequence σ_m of any $m \geq 1$ consecutive jobs, where $\sigma_m = \{h_1, h_2, \ldots, h_m\}$, the resulting dynamic matrix

$$\Omega_{\sigma_m} = \Omega(h_m) \cdot \Omega(h_{m-1}) \cdots \Omega(h_2) \cdot \Omega(h_1) \quad (9)$$

is stable. Here, Ω_{σ_m} is the left matrix product of the new dynamic formulation over a sequence σ_m.

A. Stability Analysis Using Joint Spectral Radius

To determine the asymptotic stability of the closed loop system in (8), we leverage the *Joint Spectral Radius* [9] (JSR) of the set of matrices $\mathcal{A} := \{\Omega(h_i)\}, \forall h_i \in \mathcal{H}$. The JSR of $\mathcal{A}$, classically denoted as $\rho(\mathcal{A})$, is a generalization of the spectral radius of a matrix [23]. It represents the largest asymptotic one-step-average contraction rate of the system trajectories and is formally defined as follows:

$$\rho(\mathcal{A}) = \lim_{m\to\infty} \rho_m(\mathcal{A}), \quad \rho_m(\mathcal{A}) = \max_{\sigma_m} \| \Omega_{\sigma_m} \|^{\frac{1}{m}}, \quad (10)$$

where $\| \cdot \|$ is any matrix norm. The system (8) is asymptotically stable if and only if $\rho(\mathcal{A}) < 1$. On the practical side, since computing the exact value requires checking infinitely long sequences, we are interested in finding good upper and lower bounds for the true value of $\rho(\mathcal{A})$. To this end, we introduce another related quantity, the so called *generalized spectral radius* (GSR), defined as:

$$\hat{\rho}(\mathcal{A}) = \lim_{m\to\infty} \sup \hat{\rho}_m(\mathcal{A}), \ \hat{\rho}_m(\mathcal{A}) = \max_{\sigma_m} \rho(\Omega_{\sigma_m})^{\frac{1}{m}}, \quad (11)$$

where $\rho(\Omega_{\sigma_m})$ is the spectral radius of Ω_{σ_m}. From the well-known Gel'fand-Berger-Wang formula [25], the JSR and GSR are equivalent, while for sequences of generic length m the following bounds always hold:

$$\max_{\ell \leq m} \hat{\rho}_\ell(\mathcal{A}) \leq \rho(\mathcal{A}) \leq \min_{\ell \leq m} \rho_\ell(\mathcal{A}). \quad (12)$$

This latter correlation can be leveraged to build an algorithm that computes the upper and lower bound for the joint spectral radius, as presented, e.g., in [26], [27]. The JSR can then be estimated with acceptable precision by testing "long enough" sequences. Then, if the upper bound on the JSR is below 1, we can safely conclude that the system is asymptotically stable.

B. Design Parameters and their Effects on Stability

In the proposed approach, the choice of the sensor period T_s represents an important design variable. Indeed, it determines the *granularity* of the possible values for h_k, thus also the cardinality of the set $\mathcal{H}$ (Equation (3)). Tuning this parameter requires solving a careful trade-off: on the one hand, a larger T_s greatly mitigates the complexity of the stability analysis algorithm, while, on the other hand, a value T_s such that $R_{max} - (T + \Delta_{max}) \approx 0$ promotes an efficient resource management. Clearly, the chosen value must also guarantee the stability of the closed system and a satisfactory performance.

In the particular case of $N_s = 1$, i.e., when $T = T_s$, the timing adaptation effectively matches the so called *skip-next* strategy [4], [11], [18], where, in case of overruns, the next jobs are *skipped* until the current job completes its execution. This choice is inevitably coarser and possibly reduces the stability margins, but benefits of a more regular pattern (the job releases occur at time instants that are always multiples of T).

The approach proposed in this paper benefits from a net and effort-minimal decoupling of control design and platform integration. In fact, once a controller has been designed to be stable for a particular set $\mathcal{H}$, it is sufficient to check that the set $\tilde{\mathcal{H}}$ of input-output intervals in the actual implementation, satisfies $\tilde{\mathcal{H}} \subseteq \mathcal{H}$, i.e., that the actual worst-case response time of control task namely $\tilde{R}_{max}$, satisfies $\tilde{R}_{max} < \lceil (R_{max}/T_s) \rceil T_s$. This property makes it resilient to modifications of the other tasks of the application, or to future deployments in different platforms with other project-specific functions, without the need of retuning the controller.

As a final note, the proposed stability analysis is also independent from the actual probability of experiencing overruns.

VI. Experimental Evaluation

In this section, we show the results of applying the proposed method to two dynamic systems and controllers. We test the closed-loop systems in various configurations of $R_{\max}$ and T_s. For each configuration, we built a set of 50000 random sequences σ_m, each with $m = 50$ jobs and their corresponding response times. Here, we intentionally limit the comparisons to approaches that do not use probabilistic scenarios, since our approach targets systems where the delay pattern is unknown. We check the stability of the closed-loop systems, and compute the worst-case performance as $J_w = \max_{\sigma_m} \{\sum_0^m e[k]^2\}$.

In the first example, we design a PI controller for an unstable system. The purpose of this example is to show that an adaptive control strategy can achieve better quality of control than a fixed counterpart. We sample the system with period $T = 10\,ms$, and test our adaptive control strategy against two fixed alternatives. In these alternatives the control parameters $\bar{K}_P$ and $\bar{K}_I$ are fixed, and selected based on Equation (7), as if the control period was given – either T or $R_{\max}$. However, the control job activation pattern is not regular every T time units, but subject to delays. Table I shows the results when considering different possible values of $R_{\max}$ and T_s. In terms of stability, the joint spectral radii computation gives us the guarantee that all the systems are stable for all the values of delays up to $R_{\max}$. In terms of performance, the adaptive control behaves better than its fixed counterparts, in which the parameters are set for a fixed delay (lower numbers for J_m are better).

TABLE I
Worst-case performance J_m for different control strategies with Adaptive Periods for an unstable system, controlled with a PI controller and sampling time $T = 10\,m\,s$.

$R_{\max}$	T_s	**Adaptive Control**	Fixed T	Fixed $R_{\max}$
$1.1 \cdot T$	$T/2$	0.4233	0.4270	0.4306
	$T/5$	0.4418	0.4431	0.4468
$1.3 \cdot T$	$T/2$	0.4233	0.4270	0.4371
	$T/5$	0.4270	0.4272	0.4370
$1.6 \cdot T$	$T/2$	0.3907	0.3929	0.4099
	$T/5$	0.4099	0.4119	0.4291

In the second example, we design an LQG regulator for a permanent magnet synchronous motor, borrowing the plant equations from [18, Example 2]. The controller is built as a

TABLE II
STABILITY AND WORST-CASE PERFORMANCE FOR A PERMANENT MAGNET SYNCHRONOUS MOTOR WITH PERIOD $T = 50\mu\,s$.

$R_{\max}$	T_s	Stability (JSR Adaptive) [LB, UB]	Cost with No Overruns	**Adaptive Period Adaptive Control**	Adaptive Period Fixed Control (T)	Adaptive Period Fixed Control ($R_{\max}$)	Fixed Period Fixed Control ($R_{\max}$)
$1.1 \cdot T$	$T/2$	$[0.872408, 0.872414]$	0.0375	0.0584	0.0852	0.0764	0.0385
	$T/5$	$[0.796620, 0.796623]$		0.0418	0.0431	0.0421	
$1.3 \cdot T$	$T/2$	$[0.872408, 0.872414]$	0.0375	0.0584	0.0852	0.0639	0.0439
	$T/5$	$[0.848062, 0.848068]$		0.0497	0.0603	0.0505	
$1.6 \cdot T$	$T/2$	$[0.980760, 0.981873]$	0.0375	0.1904	unstable	0.2785	0.0663
	$T/5$	$[0.895828, 0.895943]$		0.0667	0.1041	0.0614	

collection of optimal linear quadratic regulators, designed for each interval in $\mathcal{H}$. We test the stability and performance of the closed loop system when applying our strategy.

Table II shows our results. The bounds in the stability column show that the adaptive control design is stable and tolerates the delays. We compare the obtained cost to the cost of running the system when no overrun occurs (0.0375). If the controller is designed to be optimal for period T and executed with a varying delay pattern, when the maximum delay increases the closed-loop system becomes unstable. If the controller is designed to be optimal for period $R_{\max}$, on the contrary, the performance decreases (i.e., the cost increases) when the delay is limited (small $R_{\max}$).

A final comparison point is an optimal controller designed and executed with fixed period $R_{\max}$. If the value of $R_{\max}$ is known, it is in fact possible to design an optimal LQG controller to be executed with period $R_{\max}$. This knowledge is rarely available in real systems, and we consider this number ideal in the same way the cost with no overrun is ideal. While the worst-case performance for the adaptive controller is worse than this number, under the (realistic) hypothesis of sporadic overload, the cost for running the adaptive controller presented in this paper will assume values closer to the cost with no overrun most of the time. Moreover, the worst-case performance values obtained with the adaptive control strategy and fine grained sampling are close enough to the $R_{\max}$ fixed period cost, which we consider a point in favor of the proposed technique.

VII. CONCLUSIONS

This paper presents an adaptive design approach for real-time control systems subject to sporadic overruns, which leverages both a dynamic adaptation of the timing pattern and of the controller parameters to counteract the negative effects of the overruns on the system stability and performance. The implementation does not rely on stochastic or predictive delay models, and simply uses a timer and a table of control parameters. This makes it a viable solution to modify off-the-shelf controllers and preserve formal control guarantees in the presence of overruns and transient faults.

REFERENCES

[1] S. Tobuschat, R. Ernst, A. Hamann, and D. Ziegenbein, "System-level timing feasibility test for cyber-physical automotive systems," in *SIES*, 2016.

[2] M. Jelali, "An overview of control performance assessment technology and industrial applications," *Control engineering practice*, 2006.

[3] D. Fontanelli, L. Greco, and L. Palopoli, "Optimal mean square control using the continuous stream model of computation," in *CDC*, 2015.

[4] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin, "DMAC: Deadline-Miss-Aware Control," in *ECRTS*, 2019.

[5] M. Schinkel, W.-H. Chen, and A. Rantzer, "Optimal control for systems with varying sampling rate," in *ACC*, vol. 4, 2002, pp. 2979–2984.

[6] L. Mirkin, Z. J. Palmor, and D. Shneiderman, "H^2 optimization for systems with adobe input delays: A loop shifting approach," *Autom.*, vol. 48, no. 8, pp. 1722–1728, 2012.

[7] Y. Xu, A. Cervin, and K.-E. Årzén, "Jitter-robust lqg control and real-time scheduling co-design," in *ACC*, vol. 2018-June, 2018, pp. 3189–3196.

[8] D. Fontanelli, L. Palopoli, and L. Abeni, "The continuous stream model of computation for real-time control," in *RTSS*, 2013.

[9] G.-C. Rota and W. Strang, "A note on the joint spectral radius," 1960.

[10] M. Caccamo, G. Buttazzo, and L. Sha, "Handling execution overruns in hard real-time control systems," *IEEE Transactions on Computers*, vol. 51, no. 7, 2002.

[11] A. Cervin, "Analysis of overrun strategies in periodic control tasks," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 219–224, 2005.

[12] A. Cervin and J. Eker, "Control-scheduling codesign of real-time systems: The control server approach," *Journal of Embedded Computing*, 2005.

[13] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzén, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *RTCSA 2004*, 2004.

[14] M. Kauer, D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. M. Annaswamy, "Fault-tolerant control synthesis and verification of distributed embedded systems," in *DATE 2014*, 2014.

[15] S. K. Ghosh, S. Dey, D. Goswami, D. Mueller-Gritschneder, and S. Chakraborty, "Design and validation of fault-tolerant embedded controllers," in *DATE 2018*, 2018.

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

[17] S. Linsenmayer and F. Allgower, "Stabilization of networked control systems with weakly hard real-time dropout description," in *CDC*, 2017.

[18] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein, "Control-system stability under consecutive deadline misses constraints," in *ECRTS*, 2020.

[19] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. Di Natale, "Beyond the weakly hard model: Measuring the performance cost of deadline misses," in *ECRTS*, 2018.

[20] L. Hetel, C. Fiter, H. Omran, A. Seuret, E. Fridman, J.-P. Richard, and S. I. Niculescu, "Recent developments on the stability of systems with aperiodic sampling: An overview," *Automatica*, vol. 76, 2017.

[21] M. Velasco, P. Martí, R. Villa, and J. M. Fuertes, "Stability of networked control systems with bounded sampling rates and time delays," in *IECON*, 2005.

[22] M. G. Soto and P. Prabhakar, "Hybridization for stability verification of nonlinear switched systems," in *RTSS*, 2020.

[23] K. J. Åström and B. Wittenmark, *Computer-controlled systems: theory and design*, 2013.

[24] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen, "How does control timing affect performance?" *IEEE control systems magazine*, vol. 23, no. 3, pp. 16–30, 2003.

[25] M. A. Berger and Y. Wang, "Bounded semigroups of matrices," *Linear Algebra and its Applications*, vol. 166, 1992.

[26] V. D. Blondel and Y. Nesterov, "Computationally efficient approximations of the joint spectral radius," *SIAM Journal on Matrix Analysis and Applications*, vol. 27, no. 1, pp. 256–272, 2005.

[27] F. Dercole and F. Della Rossa, "Tree-based algorithms for the stability of discrete-time switched linear systems under arbitrary and constrained switching," *IEEE Transactions on Automatic Control*, vol. 64-9, 2018.