Robust co-synthesis of embedded control systems with occasional deadline misses

Amir Behrouzian[1], Dip Goswami[1], Twan Basten[1,2]
[1]Eindhoven University of Technology, The Netherlands
[2]ESI, TNO, The Netherlands

***Abstract*—Feedback control applications are robust to occasional deadline misses. This opens up the possibility of saving scarce (computation and communication) resources on embedded platforms. Stability and performance requirements of a control loop impose restrictions on acceptable patterns of deadline misses (e.g., not too many misses in a row). Such requirements are captured by (m,k)-firmness conditions. That is, at least m control computation jobs must meet deadlines in any k consecutive jobs. (m,k)-firm design requires (i) representation of stability and performance requirements in terms of (m,k)-firm deadlines (ii) controller synthesis taking into account the (m,k)-firmness parameters (iii) schedule analysis to verify guarantees on meeting the firmness conditions. We present a co-synthesis framework for these three design components and illustrate its applicability with examples.**

I. (m,k)-FIRMNESS IN CONTROLLER DESIGN

Allowing deadline misses in feedback control implementations is increasingly getting attention due to its potential to save (computation and communication) resources in embedded settings [1], [2]. This is an interesting use case since control systems are inherently robust to occasional deadline misses. The design of a feedback controller is mainly concerned about stability (e.g., system states always stay bounded) and performance (e.g., how fast the system reaches the reference signal). Both stability and performance are influenced by the number of deadline misses over a window of consecutive executions of control computation jobs. Typically, such timing requirements are well captured by (m,k)-firm deadline specifications implying that at least m jobs must meet deadlines in any window of k consecutive jobs. Generally, a lower k, a short time window, is required for stable controller synthesis. Performance is evaluated over a longer window, i.e., a higher k. We represent both stability and performance (m,k)-firmness conditions in controller design and present a scheduling analysis framework for guaranteeing both these conditions in an implementation.

II. EMBEDDED CONTROL AND DEADLINES

A. Linear state-feedback control

Feedback control applications *regulate* the behavior of a dynamical system (plant). A controller is realized by performing three sequential operations – sensing (reading plant states using sensors), computing (computing control actions) and actuating (applying a control signal to the plant). In an embedded implementation, these operations are performed as tasks – T_s (sensing), T_c (control computing) and T_a (actuating). A control algorithm is realized by periodically repeating these three operations as shown in Fig. 1(a). Sampling period h is defined as the time interval between the start of execution of two consecutive T_s jobs. Sensor-to-actuator delay τ is defined as the time from the start of T_s to the end of T_a within one sampling period. In standard control, both h and τ should be constant over the sampling periods. Overall system dynamics with sampling period h and delay τ (for linear systems) is given by [3],

$$S_h : x(k+1) = A_{h,\tau}x(k) + B_{h,\tau}u(k), \tag{1}$$

where $A_{h,\tau}$ and $B_{h,\tau}$ are system and input matrices capturing delay and $x(k)$ are system states in the k^{th} sampling period. The controller is $u(k)$ and the general form of a state-feedback controller is given by,

$$u(k) = Kx(k), \tag{2}$$

where K is the feedback gain to be designed.

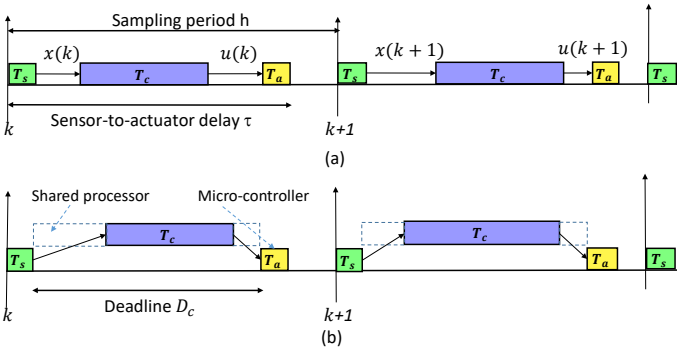

Fig. 1. Control tasks and deadlines: (a) Sequential execution of T_s (sensing), T_c (computing) and T_a (actuating) tasks. (b) Control execution deadline D_c.

B. Implementation architecture

Constants h and τ are typically achieved by time-triggered executions of the T_s and T_a tasks. That is, T_s and T_a are triggered strictly periodically with period h and the offset between them is equal to deadline D_c of the task T_c – illustrated in Fig. 1(b). T_s and T_a can be executed on a separate micro-controller or a dedicated core in a time-driven fashion. T_c runs on a shared processor with other tasks. The deadline D_c can be computed as,

$$D_c = \tau - e_s - e_a - \epsilon, \tag{3}$$

where τ is the sensor-to-actuator delay, e_s and e_a are execution times of T_s and T_a, and ϵ is any other delay caused by the communication between the tasks. Such setting is common in distributed implementations [4].

C. Deadline misses

In the occasion of a deadline miss (by a T_c job), a control application may adopt various policies and the controller $u(k)$ should be designed accordingly [5]. There are mainly two choices for a deadline-missed control computation job: (i) setting $u(k) = 0$ (ii) using the old input, i.e., $u(k) = u(k-1)$. We denote a control computation job that misses its deadline by M and one that meets (hits) its deadline by H. If an M job can be detected immediately after it is activated, the control computation job can be aborted, the old control signal can be held using a Zero-Order-Hold mechanism and thus, the processor utilization can be reduced. Detecting an M job is for example possible if the control computation task T_c is executed on a shared processor with time-division multiple access (TDMA) scheduling. If the T_c (estimated) execution time e_c exceeds the (remaining) allocated computation budget on the shared processor until the deadline D_c, then T_c will miss its deadline. Firmness analysis for detectable M and H jobs is presented in recent work [6], [7]. We consider such a task model where an M is detected and aborted as soon as T_c is released and the old control input is held. When an H job is detected immediately after its activation, the execution of T_c is continued in the upcoming sampling period. Subsequently, the control signal $u(k)$ is updated and passed on to T_a. On the other hand, in the case of an M job, the T_c job is aborted in the following sampling period, the processor is freed up and the old $u(k)$ is held. This is illustrated in Fig. 2.

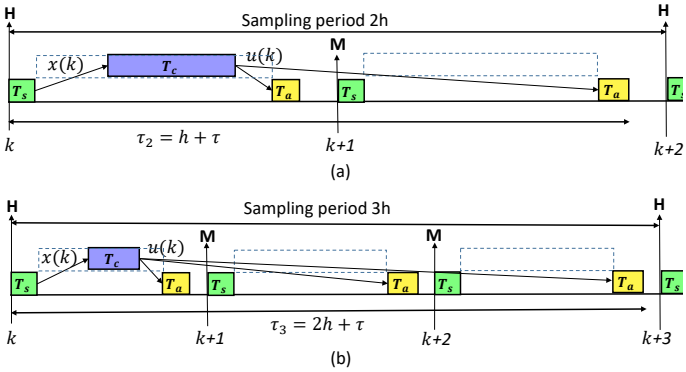

Fig. 2. Control task with deadline misses. h and τ are same as shown in Fig. 1. Execution patterns are (a) $H \rightarrow M \rightarrow H$ (b) $H \rightarrow M \rightarrow M \rightarrow H$.

D. Modeling deadline misses

Fig. 2(a) shows an example of three executions of control computation jobs, $H \rightarrow M \rightarrow H$. Since the M job is not executed and the old control signal is held, the effective sampling period is the time between the start of two consecutive H jobs. Hence, the sampling period for this pattern of deadline misses is $2h$ and the delay is given by $\tau_2 = h + \tau$. Overall system dynamics is denoted by,

$$S_{m,1} : x(k+1) = A_{2h,\tau 2}x(k) + B_{2h,\tau 2}u(k), \qquad (4)$$

where $A_{2h,\tau 2}$ and $B_{2h,\tau 2}$ are system and input matrices for the $H \rightarrow M \rightarrow H$ pattern with sampling period $2h$ and delay τ_2. Fig. 2(b) shows an example of four executions of control computation jobs with an $H \rightarrow M \rightarrow M \rightarrow H$ pattern. Similar to the previous example, the effective sampling period is $3h$ and the delay is $\tau_3 = 2h + \tau$ in this case. The resulting model is given by,

$$S_{m,2} : x(k+1) = A_{3h,\tau 3}x(k) + B_{3h,\tau 3}u(k). \qquad (5)$$

The general model for β consecutive M jobs can be derived similarly considering a sampling period of βh and a delay of $(\beta - 1)h + \tau$.

III. Stable controller synthesis and performance

A. Switched control

We consider state-feedback control of the form shown in Eq. 2. Since an M job is not executed, the control action $u(k)$ is updated only at the H jobs. The following timing knowledge is assumed for stable controller synthesis:

- The maximum number of consecutive occurrences of M jobs β is known at design time.
- At any H job, the hit/miss pattern of the following $\beta + 1$ jobs is known at runtime.

These assumptions are valid for setups with a fixed set of periodic real-time (control) tasks under TDM or static-priority preemptive scheduling.

We illustrate the idea with an example trace with $\beta = 2$ in Fig. 3. There are three possible scenarios of interest:

- $H \rightarrow H$: When an H job is followed by another H job, the system S_h is as shown in Eq. 1. In this scenario, we use the controller in Eq. 2 with feedback gain $K = K_h$.
- $H \rightarrow M \rightarrow H$: In this scenario, the system $S_{m,1}$ is as shown in Eq. 4 with sampling period $2h$ and delay $h + \tau$. We use the controller in Eq. 2 with feedback gain $K = K_{m,1}$.
- $H \rightarrow M \rightarrow M \rightarrow H$: In this scenario, the system $S_{m,2}$ is as shown in Eq. 5 with sampling period $3h$ and delay $2h + \tau$. We use the controller in Eq. 2 with feedback gain $K = K_{m,2}$.

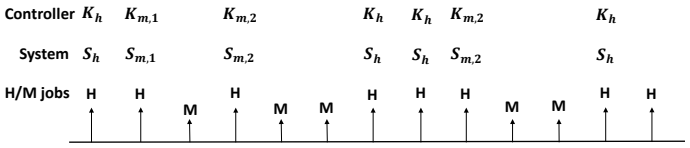

Fig. 3. Controller execution for $\beta = 2$ (maximum two M jobs in a row).

Generally, there are $\beta + 1$ scenarios for a maximum of β consecutive M jobs and each scenario uses different feedback gain K. Depending on the order of M and H jobs, the system switches between S_h, $S_{m,1}$ and $S_{m,2}$ in the example shown in Fig. 3.

B. Stability

Switching as illustrated in Fig. 3 can potentially lead to instability [3]. Design of feedback gains K_h, $K_{m,1}$ and $K_{m,2}$ should ensure overall system stability using analytical tools such as common quadratic Lyapunov functions (CQLF) [8]. Generally, the H jobs occur more frequently than the M jobs and the system mostly runs under S_h. Therefore, in our controller synthesis, we design (i) aggressive gain K_h with a higher performance using standard techniques like pole placement [4] (ii) gains $K_{m,1}$ and $K_{m,2}$ such that a CQLF

exists between S_h, $S_{m,1}$ and $S_{m,2}$ assuring overall switching stability.

The closed-loop dynamics for S_h with K_h designed as described above is given by,

$$x(k+1) = (A_{h,\tau} + B_{h,\tau}K_h)x(k) = A_{cl,h}x(k), \quad (6)$$

where K_h is designed using standard pole placement for high performance. Design of $K_{m,1}$ and $K_{m,2}$ is done using Theorem 1 to ensure switching stability.

Theorem 1: If there exist Z_1, Z_2 and Y with $Y = Y^T > 0$ such that the following Linear Matrix Inequalities hold,

$$Y - A_{cl,h}YA_{cl,h}^T > 0,$$
$$\begin{bmatrix} Y & Y^TA_{2h,\tau 2}^T + Z_1^TB_{2h,\tau 2}^T \\ A_{2h,\tau 2}Y + B_{2h,\tau 2}Z_1 & Y \end{bmatrix} > 0,$$
$$\begin{bmatrix} Y & Y^TA_{3h,\tau 3}^T + Z_2^TB_{3h,\tau 3}^T \\ A_{3h,\tau 3}Y + B_{3h,\tau 3}Z_2 & Y \end{bmatrix} > 0,$$

then $P = Y^{-1}$, $K_{m,1} = Z_1P$, $K_{m,2} = Z_2P$ and the resulting switched system is stable.

Proof: Omitted for space reasons. ■

Theorem 1 can be generalized to any β. The above design imposes the following restriction on the order of M and H jobs – any β M jobs must be followed by at least one H job. Therefore, in any consecutive $\beta + 1$ executions, at least one job must meet its deadline. The above stable controller requires the (1,β + 1)-firm condition to be met. For example, $\beta = 2$ implies a (1,3)-firm condition for stability.

C. Performance

A performance requirement can represented as an (m,k)-firm condition using existing approaches [2], [4]. The performance is evaluated over a longer time window. This implies a value of k significantly larger than $\beta+1$ (i.e., the k value for stability). Thus, we have two (m,k)-firmness conditions for stability and performance.

D. Illustrative example

We consider an unstable second-order mass-spring-damper system with the following state-space model for illustration [2],

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -3 & 1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u,$$

with a sampling period $h = 5ms$ and delay $\tau = 4.95ms$. The stability condition is captured by (1,3)-firm deadlines with $\beta = 2$ (i.e., maximum two M jobs in a row). We design K_h, $K_{m,1}$ and $K_{m,2}$ as described above and obtain,

$$K_h = \begin{bmatrix} -3.4657 & -1.7442 & 0.8252 \end{bmatrix},$$
$$K_{m,1} = \begin{bmatrix} 0.9655 & -1.2021 & 0.0317 \end{bmatrix},$$
$$K_{m,2} = \begin{bmatrix} 0.9655 & -1.2021 & 0.0317 \end{bmatrix}.$$

We further consider a performance requirement that from a given initial condition of $x = \begin{bmatrix} 5 & 1 \end{bmatrix}$ the system should reach $x = \begin{bmatrix} 0 & 0 \end{bmatrix}$ the latest in $2s$. This implies a window of $k = \frac{2s}{5ms} = 400$ samples is of interest for performance. Fig. 4 shows the system response of three examples with different numbers of M jobs in a window of $k = 400$ samples. Clearly, the performance requirement (settling within $2s$) is not met if the number of M jobs goes beyond 80 in a window of 400 samples. Therefore, the performance requirement can be captured by a (320,400)-firmness condition. Generally, such performance requirements can be derived analytically as shown in [2], [4].

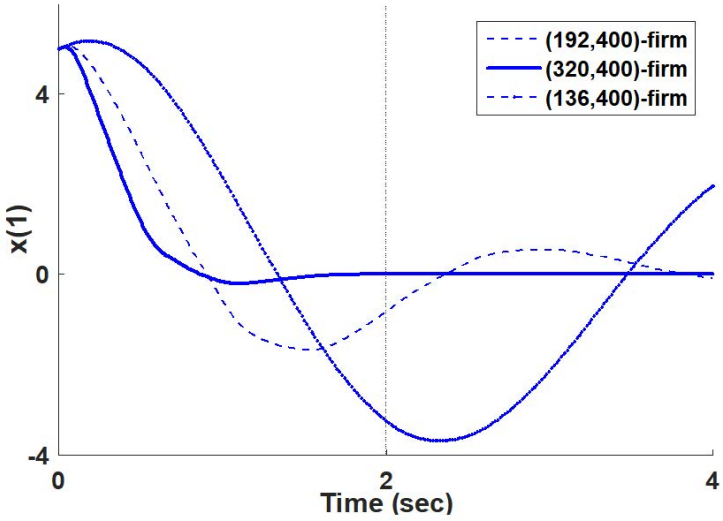

Fig. 4. System response with 264, 208 and 80 M jobs in a window of 400 samples (i.e., with (136,400)-firm, (192,400)-firm and (320,400)-firm deadlines). Only the response with the (320,400)-firmness condition meets the performance requirement of settling down within $2s$.

IV. Schedule Analysis

A. TDMA schedule

We consider a setup illustrated in Fig. 1(b) where the control computation jobs (of task T_c) are periodically activated by the sensing task T_s (running on a dedicated micro-controller). T_c runs on a shared processor under a TDMA policy. The micro-controller and the shared processor are *asynchronous* (i.e., there is no common time base and the alignment is unknown). Under a TDMA policy, task execution is performed in *slots* which are predefined time windows. The length of a slot is denoted s. A TDMA wheel consists of a given number of slots n. Any two slots are separated by a constant context switch time c. Thus, the length of a TDMA wheel is $w = n(s + c)$. Fig. 5 illustrates an example with $n = 8$ where the black blocks indicate c. One or more slots can be allocated to a task running on the processor.

We aim to schedule the control application described in Section III-D on the above architecture. Sampling period $h = 5ms$ implies that T_s is activated every $5ms$ and subsequently, it activates T_c on the shared processor. The execution time of T_c is $e_c = 0.8ms$. The considered delay $\tau = 4.95ms$ and the deadline $D_c = 4.90ms$ for T_c (with total execution time of T_s and T_a $0.05ms$). The TMDA wheel $w = 5.5ms$, $s = 0.6375ms$ and $c = 0.05ms$. The 2^{nd} and 4^{th} slots are allocated to T_c which is indicated by shaded blocks in Fig. 5. Depending on the alignment between the micro-controller and the processor, T_c gets budgets of the processor and misses or meets its deadline. A firmness analysis should verify whether the stability condition (e.g., (1,3)-firmness) and the performance condition (e.g., (320,400)-firmness) are satisfied

in all possible alignments between the micro-controller and the shared processor.

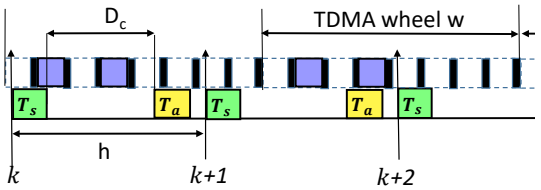

Fig. 5. Scheduling of the control application under a TDMA scheme.

B. Firmness analysis

First, we define *allocation function* $f : \Re_0^+ \rightarrow \{0, 1\}$ which takes 1 when the processor is accessible at time t to the task under consideration and 0 otherwise. Fig. 6 shows $f(t)$ for the TDMA configuration and allocation described in the previous section. $f(t) = 1$ in the 2^{nd} and 4^{th} slot of the TDMA wheel since they are allocated to T_c. Under TDMA, $f(t)$ is periodic as $f(t + w) = f(t)$.

Next, we define *resource availability function* $g(t)$ as,

$$g(t) = \int_{t}^{t+D_c} f(s)ds, \tag{7}$$

where D_c is the deadline of the task. Fig. 6 shows $g(t)$ for our example; the dotted line indicates e_c. Clearly, $g(t) \geq e_c$ leads to an H job. Since $f(t)$ is periodic, $g(t)$ is also periodic with the same period w. The periodicity can be noticed in the example of Fig. 6.

Finally, we define *hit zone function* $hz(t)$ as follows,

$$hz(t) = \begin{cases} 1 & \text{if } g(t) \geq e_c, \\ 0 & \text{if } g(t) < e_c. \end{cases} \tag{8}$$

If a job is activated at time t and $hz(t) = 0$, the job is an M job and is aborted. Using this technique, we can also compute the miss/hit pattern of the following $\beta + 1$ jobs which is required for stable controller synthesis. That is, $hz(t)$, $hz(t + h)$, $\cdots$, $hz(t + \beta h)$ need to be computed every sampling period (at runtime) to obtain the miss/hit pattern of $\beta + 1$ samples based on which the feedback gain K is chosen.

Further, the number of H jobs in k consecutive jobs starting from any t is computed by,

$$ht(t) = \sum_{n=0}^{k-1} hz(t + nh), \tag{9}$$

where h is the sampling period. Since $hz(t)$ is a periodic function, $ht(t)$ is also periodic with same period w. Therefore, the minimum H jobs in any k consecutive jobs can be obtained by,

$$ht_{min} = \min_{0 \leq t < w} ht(t). \tag{10}$$

A given (m,k)-firmness is satisfied if $ht_{min} \geq m$.

C. Finite point method

For the verification of (m,k)-firmness, ht_{min} needs to be computed. This requires computation of $ht(t)$ for all t in the range of $0 \leq t < w$ as shown in Eq. 10. All values of t can be tried by time-discretization with a step-size equal to a clock cycle. Obviously, such a brute force method can be computationally expensive in particular when such analysis

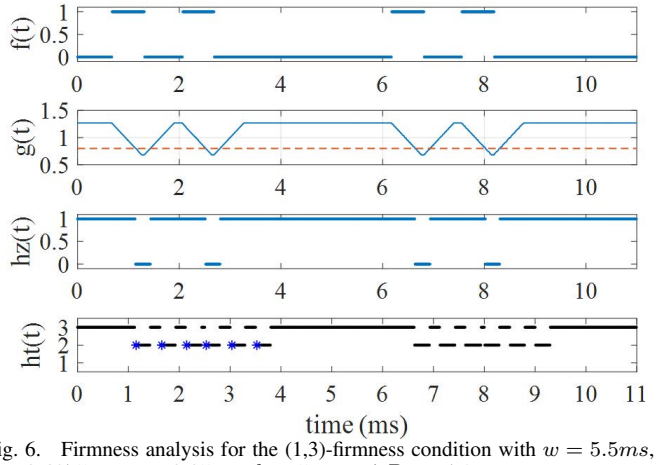

Fig. 6. Firmness analysis for the (1,3)-firmness condition with $w = 5.5ms$, $s = 0.6375ms$, $c = 0.05ms$, $h = 5ms$ and $D_c = 4.9ms$.

is used at runtime or in an iterative design process. The Finite Point (FP) method reported in [6] provides an efficient computation method. The idea is that it is enough to only check for those values of t at which $ht(t)$ decreases and they can be computed from the hit zone function. Those points are indicated by stars in Fig. 6 for (1,3)-firmness. Similarly, the (320,400)-firmness condition for performance can also be verified. Thus, it can be shown that the above allocation meets both stability and performance requirements. The FP method is sufficiently fast to be embedded in an iterative method to find suitable TDMA slot allocations when designing embedded control applications.

V. Conclusions

We presented a co-synthesis framework for controllers which allow for occasional deadline misses specified by (m,k)-firmness conditions. Our method brings together two paradigms – firmness analysis to guarantee given (m,k)-firmness conditions on a given embedded implementation platform and switched control theory for stable controller synthesis in presence of deadline misses specified by (m,k)-firmness. The implication of (m,k)-firmness is investigated with respect to control performance. The potential of co-synthesis will be further investigated in terms of a tighter resource dimensioning.

References

[1] M. Hamdaoui and P. Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines, IEEE Transactions on Computers, 44(12), 1995, pp. 14431451.

[2] E. van Horssen et al. Performance analysis and controller improvement for linear systems with (m, k)-firm data losses, In ECC, 2016.

[3] A. Y. Bhave and B. H. Krogh, Performance Bounds on State-Feedback Controller with Network Delay, In CDC, 2008.

[4] D. Goswami et al. Relaxing Signal Delay Constraints in Distributed Embedded Controllers, IEEE Transaction on Control Systems Technology 22(6), 2014, pp. 2337-2345.

[5] D. Goswami et al. Characterizing feedback signal drop patterns in formal verification of networked control systems, In CACSD, 2013.

[6] A. R. B. Behrouzian et al. Sample-drop firmness analysis of TDMA-scheduled control applications, In SIES 2016.

[7] A. R. B. Behrouzian et al. Firmness analysis of real-time applications under static-priority preemption scheduling, In RTAS 2018.

[8] O. Mason and R. Shorten . On common quadratic Lyapunov functions for stable discretetime LTI systems, IMA Journal of Applied Mathematics, 69(3), 2004, pp. 271-283.