A Framework for Designing Embedded Real-Time Controllers

Yifan Wu, Enrico Bini, Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
{y.wu,e.bini,giorgio}@sssup.it

Abstract

Control systems are typically designed assuming an ideal behavior of the computing infrastructure where controllers execute. In practice, however, in highly loaded computing systems consisting of multiple concurrent controllers, resource constraints may introduce delays and jitter in control loops that may degrade control performance significantly. Hence, taking resource constraints into account since the beginning of the design cycle is crucial for optimizing the performance of a control system.

In this paper, we propose a general framework for evaluating the performance of a control system as a function of multiple timing attributes (e.g., sampling frequencies, delays and jitter) and for selecting the proper control task parameters (e.g., periods and deadlines) taking resource constraints into account. The proposed framework is illustrated using a real control plant.

1. Introduction

The typical approach adopted during the design of a control system is to separate performance requirements from architecture and implementation issues. In a first stage, the control law is designed assuming an ideal behavior of the computing system on which the controller executes, where tasks run smoothly on the processor without considering any kind of interference. This is equivalent of synthesizing a controller in the continuous time domain without delay. When computational resources are taken into account in the design, the limited processing power of the system is considered by assigning a fixed sampling rate to the controller, whereas other types of interference are cumulated by considering a fixed input-output delay in the control loop. In this case, a controller can either be discretized or directly designed in the discrete time domain using sampled-data control theory.

In a second stage, once performance requirements are ensured by the control laws, control loops are mapped into periodic tasks and schedulability analysis is performed to verify whether the timing constraints assumed by the control designer can be met. If so, the system is implemented, otherwise the control laws must be designed by assuming different sampling rates and/or delays, and the process must be repeated.

Even when timing constraints are verified through feasibility analysis (using predicted values), the actual system implementation may reveal overload conditions and longer delays that force further refinement steps in the design process, unless very pessimistic assumptions are considered on the system [8]. Figure 1 illustrates the typical refinement process of the classical design methodology.

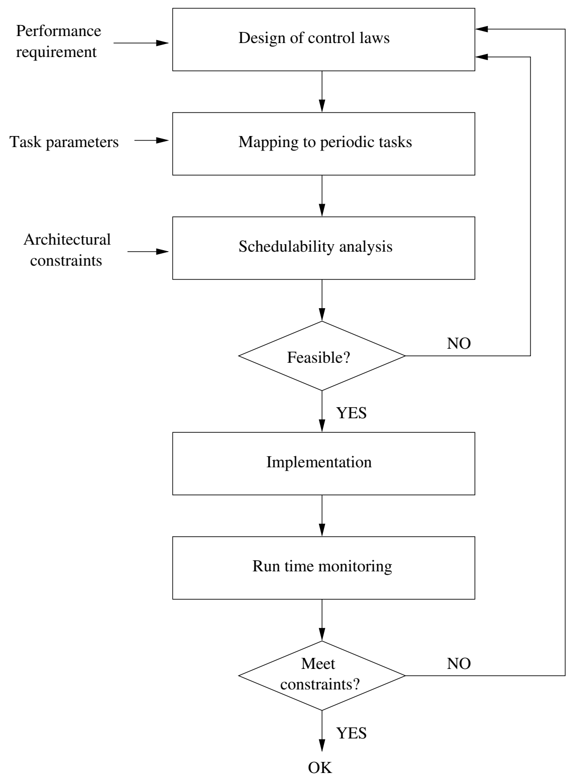

Figure 1. Typical design cycle of a real-time control system.

Such a separation of concerns facilitates both control design and implementation, allowing the system to be developed by teams with different expertise. In fact, control experts can focus on system-level goals, such as stability, robustness, and control performance, whereas computer engineers can concentrate on task mapping, schedulability analysis, resource management and code generation to ensure a reliable support to the application [1].

Unfortunately, however, such a spiral design methodology has the following disadvantages:

- Long and expensive development. Since design is performed following a trial and error strategy, several refinement steps can be required to find a suitable solution, especially when computational resources are scarce and the application consists of several concurrent and interacting activities.

- Suboptimal performance. The myopic search in the space of solutions does not guarantee that the found solution leads to the best performance. A different setting of parameters could guarantee feasibility with a significant increase in the performance.

- Suboptimal use of the resources. Since resource constraints are not taken into account in the design process (except for verifying feasibility), a feasible solution does not guarantee optimal resource exploitation, which would be of crucial importance in embedded systems where resources are scarce. For instance, optimal resource usage would allow to minimize energy consumption while meeting performance requirements.

The major problem in such a design practice is that the assumptions made at the first stage of control design are difficult to meet in the implementation, unless delays are assumed equal to sampling periods [14]. However, it has been shown [7] that, in most cases, a shorter and varying delay leads to a better performance than a fixed but longer delay. Sampled-data control theory usually assumes a negligible or at least constant input-output delay, whereas in resource constrained implementations (as the case of embedded systems and networked control systems) many concurrent tasks competing for computational resources may cause transient or permanent overload conditions, as well as introduce variable input-output latencies in control loops. Such non-deterministic effects can significantly degrade the overall system performance and possibly lead to the violation of some properties achieved during the control design phase, including system stability.

As a result, a trade-off between control performance and resources usage should be wisely considered during the whole design process. In particular, architecture constraints (as processing power, memory size, maximum power consumption) and operating system effects (as runtime overhead, blocking time, response time, intertask interference) should be properly modelled to possibly optimize the design towards a precise control objective.

In recent years, the awareness of schedulability issues has grown significantly in control systems design. As reported by Törngren et al. [23], there are many complex dependencies between control metrics (e.g., rise time, overshoot, and stability) and platform metrics (e.g., task utilization and response time). A convenient way to relate these complex aspects of control and real-time computing was presented by Seto et al. [21], who proposed to translate a co-design problem into an optimization problem. The basic idea of this approach is to use a *Performance Index* to measure the performance of the control system and use it to formulate an optimization problem, where constraints are represented by task parameters, like sampling periods.

Martí et al. [19] chose task utilizations as the variables to minimize a cost function defined as a linear approximation of a quadratic performance loss index. Cervin et al. [11] obtained a cost function with respect to sampling frequency by computing the standard quadratic cost criterion within a certain range of sampling periods. The cost function was later approximated as a linear function with respect to the task frequency. Caccamo et al. [10] introduced a task rate optimization framework to optimize the control performance with constraints on the schedulability of the task set. The involved cost function was the same performance loss index as introduced by Seto et al. [21].

Kim [16] suggested to express the cost as a function of both periods and delays, where periods were found assuming that the delays are given. Then the new delays were computed by simulating the schedule of all the tasks up to the hyperperiod, and iteratively the periods were computed again assuming the new values of delay. However, this method considered only fixed priorities and was extremely time consuming.

In this paper, we propose a general framework to treat the control design as an optimization problem. In order to derive the proper timing attributes of the control tasks that achieve the best performance, we start by evaluating the performance of a control system as a function of multiple timing attributes, like sampling frequencies, delays and jitters. To test the performance of the system under different timing attributes we propose a method for injecting desired delays into task executions using the S.Ha.R.K real-time kernel [13]. Then, resource and architecture limitations are taken into account by deriving the space of admissible design variables [5, 6]. A real control application is presented to validate the proposed approach. Figure 2 illustrates the proposed design methodology, whereas Figure 3 depicts a typical performance function in the space of the design variables. The shadowed region denotes the feasible region where task parameters satisfy the required timing constrains. Notice that the optimal control performance must take such constraints into account and can only be achieved by wisely selecting the task parameters setting.

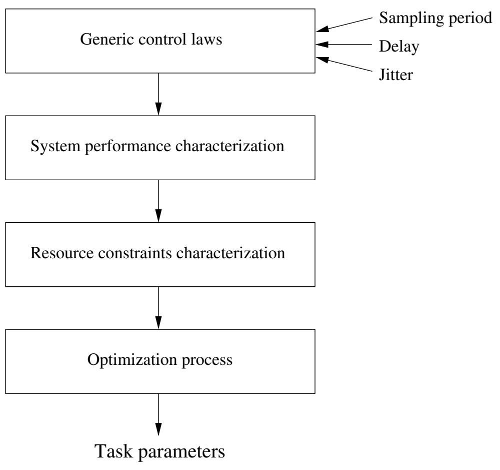

Figure 2. Proposed design methodology.

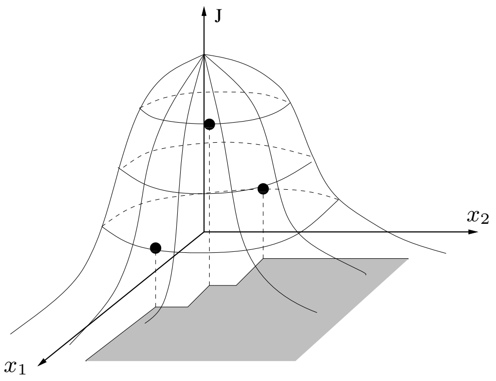

Figure 3. Relation between control performance and task parameters

The rest of the paper is organized as follows. Section 2 presents a general framework of how to construct a performance function with respect to different timing attributes. Section 3 describes how to generate the desired timing attributes in order to evaluate the corresponding performance. Section 4 reviews the feasible EDF deadline space which can be used as the feasibility region in the optimization problem. Section 5 presents some experimental results on a real plant. Finally, Section 6 states our conclusions and future works.

2. The Performance Loss Index

The primary goal of a control system is to meet stability and performance requirements, such as transient response and steady-state accuracy [8]. Beyond such requirements, controller design attempts to minimize the system error, defined as the difference between the desired response of the system and its actual response. The smaller the difference, the better the performance. Hence, performance criteria are mainly based on measures of the system error. Traditional criteria (reported in control text-books, e.g. [9]), such as IAE (Integral of the Absolute Error), ITAE (Integral of Time-weighted Absolute Error), ISE (Integral of Square Error) or ITSE (Integral of Time-weighted Square Error), provide quantitative measures of a control system response and are used to evaluate (and design) controllers.

More sophisticated performance criteria, mainly used in optimal control problems, account for the system error and for the energy that is spent to accomplish the control objective. The higher the energy demanded by the controller, the higher the penalty paid in the performance criterion. In some case, system error and control energy are multiplied by a weight to balance their relative importance. For example, in [18] and [12] the performance criterion is only based on the system error, whereas in [22] and [21] both system error and control energy are considered.

To describe our performance loss index, we start from the **ISE** index which is given in [9] and defined as follows:

$$ISE = \int_0^\infty e^2(t)dt, \tag{1}$$

where the system error $e(t)$ is the difference between system output and equilibrium value. Since the integral upper limit of the **ISE** index is infinity, a closed loop control system with permanent error will give an infinite value. In practical use, the integral upper limit of Eq. (1) could be designated to t_p so that the performance of the control system is evaluated only during the time interval $(0, t_p)$. When **ISE** index is used in discrete time with sampling period of h, and assume the equilibrium value is zero (i.e., $e(t) = y(t)$), Eq. (1) could be written as:

$$ISE(h) = \sum_{k=0}^{t_p/h} \int_0^h (y(k{\cdot}h + t))^2 dt. \tag{2}$$

Eq. (2) expresses the performance loss index as a function of the sampling period and Figure 4 (taken from one of our experimental results in Section 5) illustrates the shape of this function. Note, the monotone and convex properties are not necessary, but they fit for a wide range of control systems.

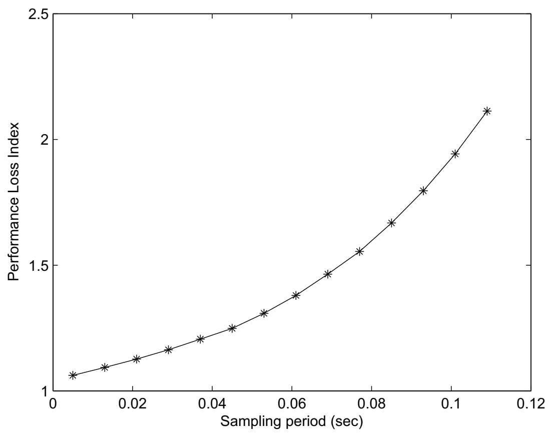

Figure 4. Performance loss index as a function of sampling period.

The performance loss index $ISE(h)$ is then approximated by a linear function of h

$$J(h) = \alpha + \beta h,$$

or by a quadratic function

$$J(h) = \alpha + \beta h + \gamma h^2.$$

In most research papers, the sampling period is the only timing attribute used as a design variable to formulate the optimization problem. This is not sufficient because other non-deterministic factors, such as delays and jitter, introduced by the runtime environment, may cause a performance degradation and even system instability [20, 17].

To model such factors, we consider a typical control task consisting of three stages: Input (I), Calculation (C) and Output (O), corresponding to sampling, calculation and actuation in the control domain. Without loss of generality, it is assumed that the Input occurs at the beginning of each job, whereas the Output occurs at the end of each job. Due to the interference caused by other tasks (typically preemption and/or blocking) two types of delays can be introduced during task execution, as depicted in Figure 5:

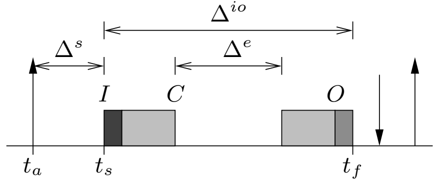

Figure 5. Control loop timing

- The *sampling delay*, which is the time between the arrival time t_a of the job and its start time t_s:

$$\Delta^s = t_a - t_s$$

- The *input-output delay* (or *IO delay*), which is the time between the start time t_s and the finishing time t_f:

$$\Delta^{io} = t_s - t_f$$

Note that Δ^{io} can be prolonged by the extra delay Δ^e induced by real-time scheduling (i.e., preemption from higher priority tasks and/or blocking from lower priority tasks). If C is the worst-case execution time of the Calculate part, we have:

$$\Delta^{io} = \Delta^e + C.$$

Note that the *sampling jitter* is defined by

$$\jmath^s = \max \Delta^s - \min \Delta^s$$

and the *input-output jitter* (or *IO jitter*) is defined by

$$\jmath^{io} = \max \Delta^{io} - \min \Delta^{io}.$$

The amount of delays and jitter experienced by each task depends on several factors, including the running scheduling algorithm, the overall workload, and the task parameters (i.e., computation times, periods, and deadlines). If not properly taken into account, delays and jitter may degrade the performance of the system and even jeopardize its stability.

Therefore the ideal approach to integrate control performance and those timing attributes is to embody all the possible variables as variables of the performance loss index function. Considering n control tasks, $\tau_1 \ldots \tau_n$, running on one processor, the performance loss index of task τ_i with respect to the timing attributes defined above can be written as follows:

$$J_i(h_i, \Delta_i^s, \jmath_i^s, \Delta_i^{io}, \jmath_i^{io}). \tag{3}$$

In real-time systems, such timing attributes are typically enforced by task parameters like periods and deadlines. For instance, a convenient method for reducing delays and jitter is to limit the execution interval of each task by setting a suitable relative deadline [3, 15]. A comparative evaluation with other jitter reduction approaches has been carried out in [7] to show the simplicity and the effectiveness of such an approach. Indeed, there is a direct relation between the control performance and the relative deadline of the control task, in the sense that decreasing the relative deadline reduces delays and jitter, thus increasing the control performance.

To derive a performance loss index as a function of the task parameters the following mapping must be made (T_i, D_i and C_i denote task period, relative deadline and WCET, respectively):

$T_i = h_i$ Task period equal to sampling period.

$\Delta^s = \Delta^{io} = D_i - C_i$ In the worst case, both the sampling delay and the input-output delay can be approximated in this way.

$\jmath^s = \jmath^{io} = D_i - C_i$ In the worst case, both the sampling jitter and the input-output jitter can be approximated in this way.

In the mapping, we assume using the EDF scheduling algorithm, because RM dose not allow to map priorities into delays and jitters.

Once the performance loss index is expressed as a function of periods and relative deadlines, a global performance loss index function can be defined considering all the n tasks:

$$J = \mathcal{F}(J_1, \ldots, J_n), \qquad (4)$$

where $\mathcal{F} : \mathbb{R}^n \rightarrow \mathbb{R}$ is a system-wide function used to combine the individual performance of control tasks into an overall system performance. Note that function $\mathcal{F}$ depends on the user's interest and can be, for instance, a linear combination of all the individual performance loss indexes, or the minimum among the performance loss indexes.

It is worth observing that, by mapping timing attributes into task parameters we cannot derive a single performance loss index like $J_i(T_i, D_i)$ from Eq. (3). In fact, J_i depends not only on its own period T_i and deadline D_i, but also on the periods and deadlines of the other tasks. Therefore, only a global performance loss index can be obtained as a function of all the task periods and deadlines.

3. Deriving Performance using S.Ha.R.K

The performance loss function J_i of Eq. (3) can be derived in an empirical way, by injecting artificial delays in the code of a controller task and computing the corresponding control performance according to Eq. (2). Although the implementation is described considering the S.Ha.R.K real-time kernel [13], the adopted methodology is general to be used in any real-time system. Notice that the deadline here is allowed to be larger than the period in order to assign delays and jitter larger than sampling period.

3.1. Generating configurable timing attributes

We first describe the technique we used to generate configurable delays in task execution. The most intuitive solution to generate a sampling delay is to defer the start time of the controller task by inserting a delay primitive before the input procedure. Similarly, the input-output delay can be introduced by inserting a delay primitive before the output procedure, as shown in the following pseudocode:

Pseudocode 1 PID_Task

```
1: loop
2:     Delay($\Delta^s$)
3:     sampled-data ← Input()
4:     control-signal ← Calculation(sampled-data)
5:     Delay($\Delta^{io}$)
6:     Output(control-signal)
7:     End_of_Job()
```

End_of_Job() is a function that suspends the current job and waits for the next release. This function may vary depending on the real-time kernel on top of which the task is running. In S.Ha.R.K, the corresponding function is Task_Endcycle().

The problem with this implementation is that, when deadlines are larger than periods, delays can be larger than expected, as depicted in Figure 6.

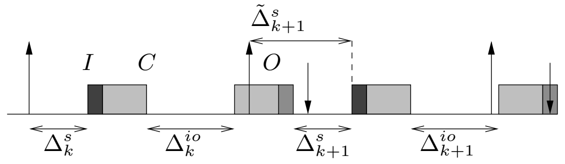

Figure 6. Problem when deadlines are larger than periods

In fact, the sampling delay is generated after the release of the $(k+1)^{th}$ job and, when the end of the k^{th} job occurs after the beginning of the next period, the start time delay is increased (being $\tilde{\Delta}^s_{k+1}$ rather than Δ^s_{k+1}).

To solve this problem, we split the controller task into three subtasks. A periodic subtask and two aperiodic subtasks. At the end of each job of the periodic subtask, a system-level event is posted to activate the first aperiodic subtask after a given amount of time, equal to the specified sampling delay Δ^s. Such an aperiodic subtask performs Input and Calculation, and at the end it posts another system-level event to activate the second aperiodic subtask after the specified input-output delay Δ^{io}. The second aperiodic subtask performs the Output and finishes the control job. The two aperiodic subtasks are scheduled with a lower priority with respect to the periodic task. In the S.Ha.R.K kernel, this can be easily implemented thanks to the configurable hierarchical scheduling architecture. Data is passed from subtask to subtask through a stream communication port. Figure 7 illustrates the activation pattern of the subtasks.

The timeline on the top of the figure shows the equivalent execution of the control task with the proper enforced delays. It can be easily seen that, except for a negligible overhead due to the subtask activation, the specified sampling delay Δ^s_k and input-output delay Δ^{io}_k are not affected

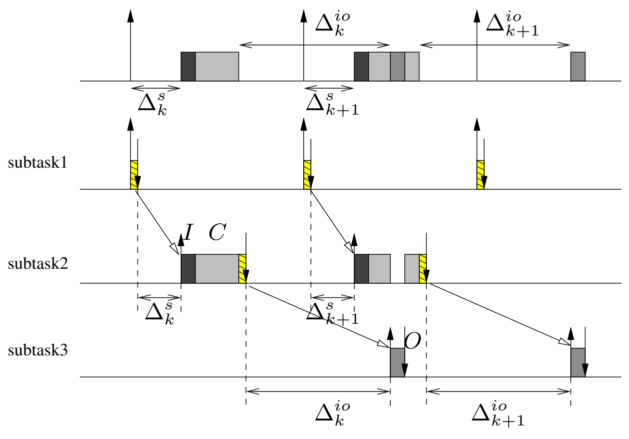

Figure 7. Sequence of subtasks to generate delays larger than periods.

by the task finishing time. It is worth mentioning that the second aperiodic subtask is assigned a priority higher than that of the first aperiodic subtask, because the Output is less time consuming and should not be preempted by the execution of the first aperiodic subtask. This approach also allows to generate tasks with arbitrary jitter, obtained by introducing random activation delays in the subtasks.

3.2. Application in S.Ha.R.K

The pseudocode of the controller task in S.Ha.R.K real-time kernel is listed as below:

Pseudocode 2 Subtask1

```
1: loop
2:     Kernel_Event_Post(
           $t_{cur}$ + $\Delta^s$,
           event_activate_aperiodic_task_1
       )
3:     Job_Finish()
```

Pseudocode 3 Subtask2

```
1: loop
2:     sampled-data ← Input()
3:     control-signal ← Calculate(sampled-data)
4:     Send_Data_To_Port(control-signal)
5:     Kernel_Event_Post(
           $t_{cur}$ + $\Delta^{io}$,
           event_activate_aperiodic_task_2
       )
6:     Job_Finish()
```

Pseudocode 4 Subtask3

```
1: loop
2:     control-signal ← Read_Data_From_Port()
3:     Output(control-signal)
4:     Job_Finish()
```

Here, Kernel_Event_Post(*t*,*e*) is the built-in function of S.Ha.R.K which posts a system-level event *e* after *t* time. t_{cur} is the current system time.

4. Resource Constraints Characterization

Since the system performance J always increases as the period or the deadline of the controllers decrease, the solution of the design problem is to decrease the design variables T_i and D_i as much as possible. Hence it is necessary to study the period and deadline values that are admitted by the available computational resource.

A good starting point to determine the feasible parameters is the EDF necessary and sufficient test [4]. According to this test a task set is schedulable by EDF if and only if:

$$\begin{cases} \sum_{i=1}^n \frac{C_i}{T_i} \leq 1 \\ \forall t \in \mathsf{dlSet}\ \sum_{i=1}^n \max\left\{0, \left\lfloor \frac{t-D_i+T_i}{T_i} \right\rfloor\right\} C_i \leq t \end{cases} \tag{5}$$

where dlSet is an opportune subset of absolute deadlines.

Unfortunately this test does not provide a description of the feasible parameters that is well suited for maximizing the performance. In fact, since periods and deadlines appear within the floor operator, it is not clear what is the shape of the boundary that is necessary to apply constrained optimization techniques such as the Lagrange multipliers.

One possible strategy that can be adopted for the performance optimization consists in the following two steps.

1. Assume for all the tasks $D_i = T_i$ and then use the necessary and sufficient test for EDF

$$\sum_{i=1}^n \frac{C_i}{T_i} \leq 1 \tag{6}$$

   that is linear and it can be used in optimization [21].

2. Fix the task periods as derived at the previous step. Relax then the assumption $D_i = T_i$, and perform the optimization onto the space of feasible deadlines [5].

Due to the regularity of the constrain of Eq. (6), the first step can be made by applying standard convex optimization techniques. If the performance function conforms to a class of some special functions (such as linear, exponential or logarithmic) then a closed solution can also be found [21, 2].

The second step can be accomplished by exploiting the geometric properties of the space of feasible deadlines. Bini and Buttazzo [5] proved that given the computation times $\mathbf{C} = (C_1, \ldots, C_n)$ and the periods $\mathbf{T} = (T_1, \ldots, T_n)$, then the region of the feasible deadline is

$$\mathbb{S} = \bigcap_{\mathbf{k} \in \mathbb{N}^n} \bigcup_{i : k_i \neq 0} \{\mathbf{D} \in \mathbb{R}^n : D_i \geq \mathbf{k} \cdot \mathbf{C} - (k_i - 1) T_i\} \quad (7)$$

To clarify the geometry of the space of feasible deadlines we propose an example. Suppose we have 2 tasks whose parameters are $\mathbf{C} = (2, 6)$ and $\mathbf{T} = (4, 12)$. Then by applying the definition of Eq. (7), the resulting space of feasible deadlines can be drawn (the union of dark and light gray areas in Figure 8).

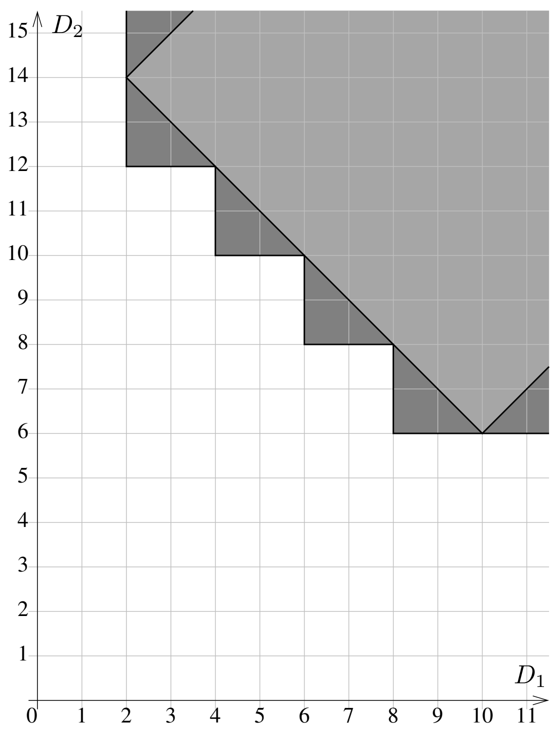

Figure 8. The region of feasible deadlines.

Since the performance always improves for smaller deadlines (i.e. $\frac{\partial J_i}{\partial D_i} \leq 0$) then all the corners of the region of the feasible deadlines are a local optima. An optimization routine should then test the performance value at all these local optima and select the best performing solution. In the example depicted in Figure 8 the local optima occur in the set $\mathbb{S} = \{(8, 6), (6, 8), (4, 10), (2, 12)\}$.

Even if in the simple case of two tasks this enumeration scheme seems to be easily applicable, as the number n of tasks increases, this method becomes extremely more difficult and time consuming. It is then very useful to use a convex subregion of the exact space. In [5] it is proved that if the following set of linear constraints are satisfied

$$\begin{cases} D_i - D_j \leq T_i & \forall i, j \\ D_j \left(1 - \sum_{i=1}^n U_i\right) + \sum_{i=1}^n U_i \, D_i \geq \sum_{i=1}^n C_i & \forall j \end{cases}$$

then the resulting deadline assignment is feasible. Moreover, since in the first step of our optimization procedure we assigned the periods such that the total utilization $\sum_i U_i$ reaches 1, the convex constraint becomes

$$\begin{cases} D_i - D_j \leq T_i & \forall i, j \\ \sum_{i=1}^n U_i \, D_i \geq \sum_{i=1}^n C_i \end{cases} \quad (8)$$

In Figure 8 the convex subregion is depicted in light gray.

Although Eq. (8) provides only a sufficient region, the convexity allows to implement a very efficient algorithm for finding a deadline assignment.

5. Experimental Results

In the experiment we extracted the performance loss index J_i using the technique described in Section 3. The plant is a ball-and-beam system which is controlled by a PID regulator. The system is modelled using the following transfer function:

$$G(s) = \frac{\alpha}{s^2}$$

and the control performance is evaluated according to Eq. (2).

Experimental data is collected by S.Ha.R.K tracer, and then analyzed in Matlab. S.Ha.R.K tracer saves data into memory during the runtime of the real-time operating system. When the system is terminated, all data is saved into a file.

In the first experiment, we varied both the sampling period and the input-output delay (we remind that the input-output delay can exceed the period). The performance loss illustrated in Figure 9 has been scaled by dividing it by the minimum value in one experiment. As expected, the performance loss increases with the period and the input-output delay. It can be noticed that the input-output delay affects significantly the performance of the system. Hence a careful control system co-design with respect to only periods [21] should be enriched by the information about the delay.

In the second experiment we evaluated the performance loss index as a function of sampling period and input-output jitter. The result in Figure 10 shows that input-output jitter does not affect the system performance as significantly as input-output delay. In fact fast sampling frequency is able to tolerate input-output jitter. However, input-output jitter degrades the system performance much or even jeopardizes the stability of the system especially when sampling frequency is slow. Therefore jitter should also be taken into account when making real-time control co-design.

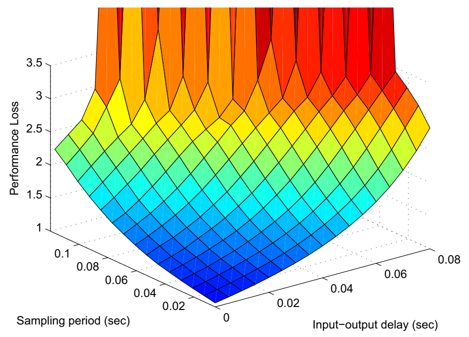

Figure 9. Performance loss with respect to sampling period and IO delay

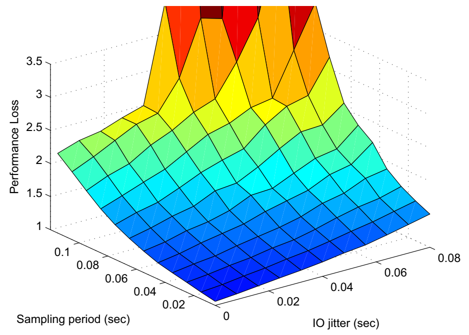

Figure 10. Performance loss with respect to sampling period and IO jitter

Similar experimental results have been obtained considering sampling delay and sampling jitter.

The above results also demonstrate that the control performance with respect to different timing attributes can be approximated as linear, quadratic or convex function if the control system remains stable. This verifies the correctness of our approach in Section 2.

6. Conclusions and Future Works

In this paper we proposed a general framework for treating real-time control design as an optimization problem where the optimal control performance is obtained by setting properly the timing attributes. We presented a method to evaluate the performance of a control system as a function of multiple timing attributes, such as sampling frequencies, delays and jitter. This method is based on the injection of configurable delays into task code using S.Ha.R.K real-time kernel. Resource constraints were then considered by deriving admissible design parameters.

In the future we plan to develop a procedure that optimizes the performance loss index derived within this framework.

References

[1] K.-E. Årzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Proceedings of the* 39th *IEEE Conference on Decision and Control*, Sydney, Australia, Dec. 2000.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alarez. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 50(2):111–130, Feb. 2001.

[3] P. Balbastre, I. Ripoll, and A. Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In *Proceedings of the* 18th *Euromicro Conference on Real-Time Systems*, pages 65–74, Dresden, Germany, July 2006.

[4] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the* 11th *IEEE Real-Time Systems Symposium*, pages 182–190, Lake Buena Vista (FL), U.S.A., Dec. 1990.

[5] E. Bini and G. Buttazzo. The space of EDF feasible deadlines. In *Proceedings of the* 19th *Euromicro Conference on Real-Time Systems*, pages 19–28, Pisa, Italy, July 2007.

[6] E. Bini, G. Buttazzo, and M. Di Natale. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39(1–3):5–30, Aug. 2008.

[7] G. Buttazzo and A. Cervin. Comparative assessment and evaluation of jitter control methods. In *Proceedings of the* 15th *conference on Real-Time and Network Systems*, pages 163–172, Nancy, France, Mar. 2007.

[8] G. Buttazzo, P. Martí, and M. Velasco. Quality-of-control management in overloaded real-time systems. *IEEE Transactions on Computers*, 56(2):253–266, Feb. 2007.

[9] R. C. Dorf and R. H. Bishop. *Modern Control Systems*. Prentice Hall, tenth edition, 2004.

[10] M. Caccamo, G. Buttazzo, and L. Sha. Elastic feedback control. In *Proceedings of the* 12th *Euromicro Conference on Real-Time Systems*, pages 121–128, Stockholm, Sweden, June 2000.

[11] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1–2):25–53, July 2002.

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

[13] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo. A new kernel approach for modular real-time systems development. In *Proceedings of the* 13th *Euromicro Conference on Real-Time Systems*, pages 199–206, Delft, The Nederlands, June 2001.

[14] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Embedded control systems development with giotto. In *Proceedings of The Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 64–72, Snow Bird (UT), U.S.A., June 2001.

[15] H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson. Computing the minimum edf feasible deadline in periodic systems. In *Proceedings of the* 12th *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 125–134, Sydney, Australia, Aug. 2006.

[16] B. K. Kim. Task scheduling with feedback latency for real-time control systems. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*, pages 37–41, Hiroshima, Japan, October 1998.

[17] H. J. Kushner and L. Tobias. On the stability of randomly sampled systems. *IEEE Transactions on Automatic Control*, 14(4):319–324, Aug. 1969.

[18] F.-L. Lian, J. Moyne, and D. Tilbury. Network design consideration for distributed control systems. *IEEE Transactions on Control Systems Technology*, 10(2):297–307, Mar. 2002.

[19] P. Martí, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes. Optimal state feedback based resource allocation for resource-constrained control tasks. In *Proceedings of the* 25th *IEEE Real-Time Systems Symposium*, pages 161–172, Lisbon, Portugal, Dec. 2004.

[20] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Stochastic analysis and control of real-time systems with random time delays. *Automatica*, 34(1):57–64, Jan. 1998.

[21] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proceedings of the* 17th *IEEE Real-Time Systems Symposium*, pages 13–21, Washington (DC), U.S.A., Dec. 1996.

[22] K. G. Shin and C. L. Meissner. Adaptation of control system performance by task reallocation and period modification. In *IEEE Proc. of the* 11th *Euromicro Conference on Real-Time Systems*, pages 29–36, York, England, June 1999.

[23] M. Törngren, D. Henriksson, K.-E. Årzén, A. Cervin, and Z. Hanzalek. Tools supporting the co-design of control systems and their real-time implementation: Current status and future directions. In *Proceedings of the 2006 IEEE Computer Aided Control Systems Design Symposium*, pages 1173–1180, München, Germany, Oct. 2006.