Control-Quality Driven Design of Embedded Control Systems with Stability Guarantees

Amir Aminifar[1], Petru Eles[2], Zebo Peng[2], Anton Cervin[3], Karl-Erik Årzén[3]
[1]Embedded Systems Laboratory, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland
[2]Embedded Systems Laboratory, Linköping University, Sweden
[3]Department of Automatic Control, Lund University, Sweden
amir.aminifar@epfl.ch, {petru.eles,zebo.peng}@liu.se, {anton,karlerik}@control.lth.se

***Abstract*—Today, the majority of control applications in embedded systems, e.g., in the automotive domain, are implemented as software tasks on shared platforms. Ignoring implementation impacts during the design of embedded control systems results in complex timing behaviors that may lead to poor performance and, in the worst case, instability of control applications. This article presents a methodology for implementation-aware design of high-quality and stable embedded control systems on shared platforms with complex timing behaviors.**

Keywords: Control-Scheduling, Co-Design, Control Performance, Stability, Robustness, Embedded Control Systems, Real-Time Control, Cyber-Physical Systems

I. Introduction

Today, many embedded systems, e.g., in the automotive domain, comprise several control applications. These applications are in charge of controlling the physical plants in the systems. Such systems with tight interaction between the physical and the cyber (processing) elements, which together achieve capabilities that cannot be obtained otherwise, are also known as cyber-physical systems. An early example of such systems is computer-controlled automotive engines, which are essential to fuel-efficient and low-emission vehicles. The tight interaction between the physical and cyber (processing) elements renders physical time a fundamental parameter when reasoning about this class of embedded systems.

In the past few years, we have been witnessing a shift from federated architectures to integrated architectures, in which several applications share the same platform, due to the increasing functional complexity and substantial economic savings. This trend is particularly visible in the automotive domain [1]. Today, the majority of control applications in the automotive domain are implemented as software tasks on shared platforms.

Ignoring the implementation impacts during the design of embedded control systems on shared platforms results in design outcomes with underutilized resources, poor control performance, or instability of control applications. In particular, it is well known that such resource sharing leads to complex temporal behaviors that degrade the quality of control and, more importantly, may jeopardize the stability of control applications, if not properly taken into account during design.

Having the platform shared among several tasks, the delay between sampling and actuation not only will be longer than on a dedicated platform, but also varying. This is due to the fact that several tasks compete for execution on the shared platform. The situation only gets more complex if we take into consideration the fact that the computation times of the tasks usually vary due to different input data and different states of the platform, e.g., cache and pipeline. Therefore, as a result of sharing the platform, the control task may experience considerable amount of *latency* (the constant part of the delay) and *jitter* (the varying part of the delay), which affect the control performance and stability of the control application [2–4].

Traditionally, the problem of designing embedded control systems has been dealt with in two independent steps, where first the controllers are synthesized and, second, these controllers are mapped and scheduled on a given platform. However, this approach often leads to either resource underutilization or poor control performance and, in the worst-case, may even lead to instability of control applications, because of the timing problems which can arise due to certain implementation decisions [5], [6]. Thus, in order to achieve high control performance while guaranteeing stability even in the worst-case, it is essential to consider the timing behaviors extracted from the system schedule during control synthesis and to keep in view control performance and stability during system scheduling. The issue of control–scheduling co-design [6] has become an important research direction in recent years.

In order to capture control performance, two kinds of metrics are often used: (1) stochastic control performance and (2) robustness (stability-related metrics). The former identifies the expected (*mathematical expectation*) control performance of a control application, whereas the latter is considered to be a measure of the worst-case control performance. On the one hand, considering solely the expected control performance may result in solutions exhibiting high expected performance that, however, do not necessarily satisfy the stability requirements in the worst-case scenario. On the other hand, considering merely the worst-case stability, often results in a system with poor expected control performance. This is due to the fact that the design is solely tuned to a scenario that occurs very rarely. Thus, even though the overall design optimization goal should be the expected control performance, taking the worst-case control stability into consideration during design space exploration is indispensable for a large class of safety critical applications.

Previous work has mainly focused on one of the two metrics, e.g., in [7, 8], the authors consider only the expected control performance, while, in [9, 10], the authors consider merely the worst-case control performance. Exceptions are the

proposed approaches in [11, 12] that are, however, restricted only to static-cyclic and time-triggered scheduling.

This article discusses the design and optimization of high-quality and stable embedded control systems running on shared platforms, while taking the timing impacts of the implementation into consideration during the design process.

The remainder of this article is organized as follows. In Section II, we illustrate the different metrics considered in design of embedded control systems. Section III discusses the different timing interfaces and their relation with the different control performance metrics. In Section IV, we illustrate the interdependency between the control and scheduling processes and the importance of control–scheduling co-design. In Section V, we formulate the control-scheduling problem to optimize control performance, while guaranteeing stability. In Section VI, we propose a methodology to address this problem and evaluate the efficiency of this methodology in Section VII. Finally, in Section VIII, we conclude that it is essential to consider the interplay between real-time scheduling and control synthesis during the design of embedded control systems on shared platforms, taking into account both expected control performance and worst-case stability.

II. Control-Quality versus Stability

A correct design methodology for embedded control systems should target the optimization of the overall control performance of the system as its main objective, while also guaranteeing the stability of the system, in the worst-case.

Figure 1 illustrates the relation between the expected control performance and worst-case control performance. The red region shows the unstable (unrobust) area, where the worst-case control cost is not finite. The yellow region is robust (low worst-case control cost), but low-quality area (high expected control cost). The green region is the high quality (low expected control cost) and robust and stable (finite worst-case control cost) area. Observe that there is an inherent trade-off between the expected control cost and worst-case control cost, hence the white region. That is, it is possible to minimize the worst-case control cost at the expense of increasing the expected control cost, and vice versa [13].

The overall performance (i.e., control quality) of the system is captured by the expected control performance, which should be the main optimization objective in designing embedded control systems. However, a design methodology targeting only the expected control cost may end up in the red region and with a high-quality but unstable design solution. Therefore, considering only the expected control cost is not sufficient to guarantee the stability of the embedded control system, in the worst case.

It is also essential to guarantee the stability of the system at all times and even for the worst-case scenario and considering only the expected control performance does not necessarily guarantee the stability of the system. The stability of the system is captured by the worst-case control performance metric, which should be considered as a constraint during the design process. However, a design methodology targeting

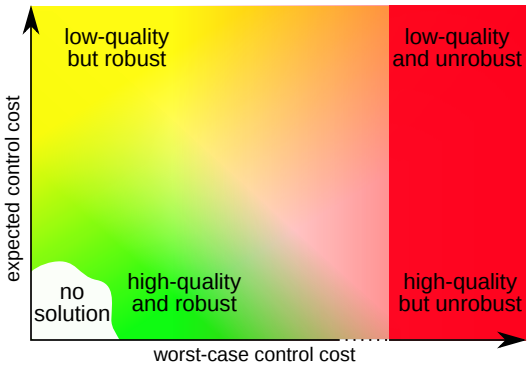

Fig. 1. The expected and worst-case control performance and the corresponding regions

only the worst-case control cost as its main optimization objective may end up in the yellow region and with a low-quality, but robust, design solution. Therefore, while it is important to guarantee that the system remains stable even in the worst-case scenario, such metrics should not be used as optimization objective. This is essentially because the worst-case scenario does not capture the overall behavior of the system and often involves significant amount of pessimism. A design methodology driven merely by the worst-case scenarios is over-provisioning, since the design solutions are tuned to the worst-case scenario, which does not capture the most probable scenarios.

A correct design methodology is devised towards the majority of cases, while also taking into consideration the worst-case scenario. In the case of embedded control systems, such a design methodology optimizes the expected control performance, while also ensuring the worst-case stability, and finds solutions in the green region.

Let us assume each plant is modeled by a continuous-time system of equations [2]

$$\begin{aligned} \dot{\boldsymbol{x}}_i &= \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{u}_i + \boldsymbol{v}_i, \\ \boldsymbol{y}_i &= \boldsymbol{C}_i \boldsymbol{x}_i + \boldsymbol{e}_i, \end{aligned} \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and control signal, respectively. The additive plant disturbance $\boldsymbol{v}_i$ is a continuous-time white-noise process with zero mean and a given covariance matrix. The plant output is denoted by $\boldsymbol{y}_i$ and is sampled periodically with some delays at discrete time instants—the measurement noise $\boldsymbol{e}_i$ is a discrete-time Gaussian white-noise process with zero mean and a given covariance. The control signal $\boldsymbol{u}_i$ will be updated periodically, according to the control law, with some delays at discrete time instants and is held constant between two updates by a hold-circuit in the actuator. The control law determines the control input $\boldsymbol{u}_i$ for the plant state $\boldsymbol{x}_i$.

We shall now elaborate on the metrics to quantify the expected and worst-case control performance.

A. Expected Control Performance

We quantify the expected control performance of a system using a standard quadratic cost function [2]

$$J_i^{\mathrm{e}} = \lim_{T\to\infty} \frac{1}{T} \int_0^T \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix}^{\mathsf{T}} Q_i \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} dt \qquad (2)$$

Here, the positive semi-definite weight matrix Q_i is given by the designer and captures the relative importance of plants states and control signals. Having the sensor–actuator delay distribution, we use the Jitterbug toolbox [4, 14] to compute the expected control cost.

While appropriate as a metric for the average quality of control, the above cost function cannot provide a *hard guarantee* of stability in the worst case. Using Jitterbug, the stability of a plant can be analyzed in the mean-square sense *if* all time-varying delays are assumed to be independent stochastic variables. However, by their nature, task and message delays do not behave as independent stochastic variables. Therefore, the stability results based on the above quadratic cost are not valid as worst-case guarantees.

B. Worst-Case Stability

We quantify the worst-case control performance of a system by computing an upper bound on the worst-case gain from the plant disturbance d to the plant output y. The plant output is then guaranteed to be bounded by

$$\|y\| \leq J_i^{\mathrm{w}} \|d\|.$$

If $J_i^{\mathrm{w}} = \infty$, then stability of the system cannot be guaranteed. A smaller value of J_i^{w} implies a higher degree of robustness. The worst-case control cost J_i^{w} is computed by the Jitter Margin toolbox [3] and depends on the plant model P_i, the control application Λ_i with associated control law, sampling period h_i, the nominal sensor–actuator (input–output) latency L_i, the worst-case sensor (input) jitter Δ_{is}, and the worst-case actuator (output) jitter Δ_{ia}. The nominal sensor–actuator latency and worst-case sensor and actuator jitters are computed using response-time analysis.

While appropriate as a metric for the worst-case stability, the above metric cannot capture the overall control performance of the system. This is because the worst-case stability is calculated based on the extreme values of the delay experienced by a control application. Therefore, the above metric does not capture the expected control performance.

III. Timing Interfaces

A. Delay Distribution

Delay distribution is the timing interface between real-time scheduling and control performance. Essentially, delay distribution captures the frequency of the delays experienced by a control task. That is, the expected control performance is calculated based on the the entire delay distribution, and not only the extreme values. This indicates that the expected control performance captures the overall performance of the systems and the quality of control.

Delay distribution, however, does not capture the order and dependencies among the delays experienced by control applications. Therefore, delay distribution cannot be used for guaranteeing safety in the worst-case scenario. Nevertheless, due to its richness and simplicity, delay distribution is one of the most extensively-used timing interfaces for (expected) quality assessment. To obtain the delay distribution experienced by each controller, we perform an event-driven system simulation (see Section VI).

B. Latency–Jitter

The latency–jitter interface is a considerably less expressive timing interface compared to the delay distribution. The latency–jitter interface abstracts the exact delay patterns by considering only the extreme values. Note that the latency–jitter interface captures very little about the distribution of the delay. Therefore, it is not an appropriate metric for measuring the expected control quality. In other words, similar values of latency and jitter might lead to completely different control qualities, depending on the actual delay distribution. However, to provide hard stability guarantees, we have to consider the worst-case scenario and this interface is simple enough to capture sufficient conditions for stability.

In order to compute the worst-case control performance, we shall compute the nominal sensor–actuator latency L_i, worst-case sensor jitter Δ_{is}, and worst-case actuator jitter Δ_{ia} for each control application Λ_i as follows,

$$\begin{aligned} \Delta_{is} &= R_{is}^{\mathrm{w}} - R_{is}^{\mathrm{b}}, \\ \Delta_{ia} &= R_{ia}^{\mathrm{w}} - R_{ia}^{\mathrm{b}}, \\ L_i &= \left(\frac{R_{ia}^{\mathrm{b}} + R_{ia}^{\mathrm{w}}}{2}\right) - \left(\frac{R_{is}^{\mathrm{b}} + R_{is}^{\mathrm{w}}}{2}\right), \end{aligned} \qquad (3)$$

where R_{is}^{w} and R_{is}^{b} denote the worst-case and best-case response times for the sensor task τ_{is} of the control application Λ_i, respectively. Analogously, R_{ia}^{w} and R_{ia}^{b} are the worst-case and best-case response times for the actuator task τ_{ia} of the same control application Λ_i. We perform worst-case and best-case response-time analysis [15] to obtain R_{is}^{w}, R_{is}^{b}, R_{ia}^{w}, and R_{ia}^{b}.

IV. Control–Scheduling Co-Design

The design of embedded control systems running on shared platforms comprises two main processes: control synthesis and real-time scheduling. Traditionally, the controllers were first designed by the control engineers and, only then, could the computer engineers schedule these controllers on shared platforms. This design technique, however, leads to suboptimal design solutions, essentially because the interdependency between control synthesis and real-time scheduling has been ignored. In other words, the decisions made in one of these processes, to a great extent, determines and limits the possible choices in the other process.

To address the shortcomings of the traditional techniques, there is a need for a co-design methodology to consider the

control synthesis and real-time scheduling processes in an integrated fashion. We shall now elaborate on the interdependency between control synthesis and real-time scheduling.

Given a control plant and the performance and stability requirements, let us identify the space of all stable solutions (or solutions with certain performance requirements) for the control synthesis problem. This design solution space, where the stability (and performance) requirements are satisfied for the given plant, has four abstract dimensions:[1]

- control law: determines the control signals that can be applied to the plant.
- execution pattern: determines when and how often a controller executes.
- execution-time profile: determines how long one execution of a controller takes.
- delay profile: determines the characteristics of the delay experienced by a controller.

Let us imagine the space of all possible design solutions that satisfy the stability (and performance) requirements for the given plant, in this four dimensional space. Each stable design solution is identified by a control law, an execution pattern, an execution-time profile, and a delay profile in this abstract space.

Similarly, given the number of tasks, let us identify the space of all schedulable solutions for the real-time scheduling problem. In abstract terms, this design solution space, where the schedulability requirements are satisfied for the given number of tasks, has three abstract dimensions:

- execution pattern: determines when and how often a task executes.
- execution-time profile: determines how long one execution of a task takes.
- delay profile: determines the characteristics of the delay that is experienced by each task.

Let us imagine the space of all possible design solutions that satisfy the schedulability requirements for the given task set, in this three dimensional space. Each schedulable[2] design solution is identified by an execution pattern, an execution-time profile, and a delay profile in this abstract space.

A valid control-scheduling solution should be both in the space of stable solutions and in the space of schedulable solutions. Note that the space of stable solutions and the space of schedulable solutions have three dimensions in common. Therefore, a valid design solution should be in the intersection of these two spaces.

Let us now focus on a simplified case with only the execution patterns and delay profiles, for the sake of presentation. Figure 2 shows the space of stable solutions (in yellow), determined by the control synthesis constraints, and also the space of the schedulable solutions (in red), determined by the real-time scheduling constraints. The intersection, shown in orange, identifies the space of all stable and schedulable solutions.

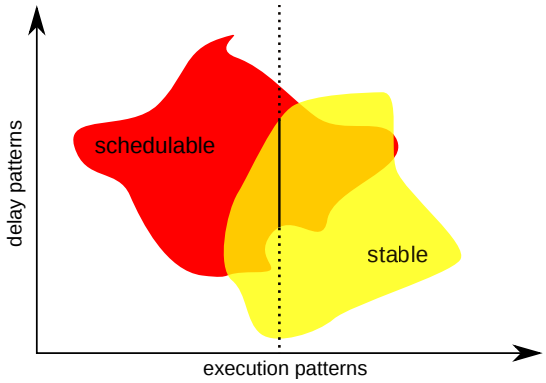

Fig. 2. The intersection of the design space for stable solutions (determined by the control synthesis process) and the design space for schedulable solutions (determined by real-time scheduling process)

Inevitably, fixing one of the dimensions, e.g., the execution pattern, reduces the space of all valid design solutions drastically. This scenario is also shown in Figure 2, where the execution pattern is considered to be fixed. The solid black line (the intersection of stable and schedulable space with the fixed execution patten) depicts the space of stable and schedulable delay profiles that can be explored in the search space. This search space (solid black line) is, therefore, substantially smaller than the original space of stable and schedulable solutions (shown in orange). Clearly, limiting the design space exploration to the space covered by the solid black line may lead to suboptimal design solutions. This is essentially the drawback with the traditional techniques based on the principle of separation of concerns. In traditional approaches, the control law and execution-time profile are fixed by the control engineers, leaving only the space of execution patterns and delay profiles for design exploration by the computer engineers.

In summary, the interdependency between control synthesis and task scheduling motivates the need for a co-design methodology. The traditional approaches treat these two processes separately and often obtain suboptimal solutions.

V. Problem Formulation

Given a set of plants $\mathbf{P}$, the goal is to determine the scheduling and control parameters, having the expected control performance as the optimization objective, while guaranteeing stability and robustness requirements. Hence, the optimization problem is formulated as:

$$\begin{aligned} \min \quad & \sum_{P_i \in \mathbf{P}} w_i J_i^{\mathrm{e}} \\ \text{s.t.} \quad & J_i^{\mathrm{w}} < \bar{J}_i^{\mathrm{w}}, \quad \forall P_i \in \mathbf{P}, \end{aligned} \tag{4}$$

where the weights w_i are determined by the designer. To guarantee stability, the worst-case control cost J_i^{w} must have

[1] Indeed, the actual dimensionality of the search space is much larger than what is explained here. For the simplicity of presentation, here, we assume control laws, execution patterns, execution-time profiles, and delay profiles can each be captured in one dimension.

[2] A schedulable design solution is a design solution in which all instances of all tasks have finite response times. This schedulability definition is tailored to the case of control applications, which do not enforce hard deadlines [16].

a finite value. However, in addition to worst-case stability, the designer may require an application to satisfy a certain degree of robustness in the worst case ($\bar{J}_i^{\mathrm{w}}$). If the worst-case requirement for an application Λ_i is only to be stable, the constraint on the worst-case control cost J_i^{w} is to be finite. The optimization parameters are the controllers, and scheduling parameters (e.g., the sampling period and the priority) for each control application.

VI. Design of Embedded Control Systems

The overall flow of our design methodology [17, 18] is shown in Figure 3. Given the plant model and system specification, in each iteration, each control application is assigned new real-time parameters, e.g., sampling periods. We consider the coordinate and direct search methods [19] from the class of derivative-free optimization technique, where the derivative of the objective function is not available or it is time consuming to obtain. These methods are desirable for our optimization since the objective function is the result of an inside optimization loop (i.e., the objective function is not available explicitly) and it is time consuming to approximate the gradient of the objective function using finite differences. This search method, iteratively, assigns shorter periods to controllers which violate their worst-case robustness requirements or provide poor expected control performance since shorter period often leads to better control performance.

For a certain real-time parameter assignment, we shall now proceed with control synthesis. For a given sampling period and a given, constant sensor–actuator delay (i.e., the time between sampling the plant output and updating the controlled input), it is possible to find the control-law that minimizes the expected cost J_i^{e} [2]. Thus, the optimal controller can be designed if the delay is considered constant at each execution (for each instance) of the control application. Since the overall performance of the system is determined by the expected control performance, the controllers should be designed for the expected average behavior of the system. Therefore, we design the Linear-Quadratic-Gaussian (LQG) controllers to compensate for the expected sensor–actuator delay, using MATLAB and the Jitterbug toolbox [14].

In order to compute the worst-case control performance, we shall first compute the nominal sensor–actuator latency L_i, worst-case sensor jitter Δ_{is}, and worst-case actuator jitter Δ_{ia} for each control application Λ_i. Towards this, we shall first perform response-time analysis to obtain the best-case and worst-case response times for sensor and actuator tasks. Then, we proceed with computing the latency and jitter values, based on Equation (3) in Section III-B.

Having computed the latency and jitter values, we shall now check if all control applications are guaranteed to be stable, even in the worst-case scenarios, using the Jitter Margin toolbox [3]. If any of the control applications is unstable with the given real-time parameters and synthesized controllers (see the "Stable?" block), then we shall explore (inner loop in Figure 3) new real-time parameters; otherwise, we proceed with

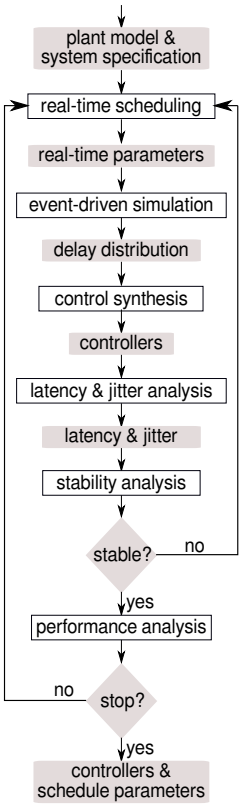

Fig. 3. Overall flow of our embedded control systems design methodology

computing the expected control performance which captures the overall performance of the system.

As explained above, each controller is designed for a constant (expected) sensor–actuator delay. However, the sensor–actuator delay is, in reality, not constant at runtime due to interference from other applications competing for the shared resources. The quality of the constructed controller is degraded if the sensor–actuator delay distribution is different from the constant one assumed during the control-law synthesis. For the given real-time parameter assignment, we proceed with an event-driven system simulation to obtain the distribution of the delay experienced by each controller. Having found the sensor–actuator delay distribution for each control application, it is now possible to use the Jitterbug toolbox [4, 14] to compute the expected control cost. The optimization (outer loop in Figure 3) terminates (see the "Stop?" block) once the search method cannot find a higher quality design solution in several consecutive iterations.

VII. Experimental Evaluation

To support the previous discussions, we compare our proposed methodology, which optimizes the expected control performance while providing stability guarantees, against three other techniques (see Table I for the results). We consider 125 benchmarks with varying number of plants, from 2 to 15. The plants are taken from a database with inverted pendulums, ball and beam processes, DC servos, and harmonic

oscillators [2]. Such benchmarks are representative of realistic control problems and are used extensively for experimental evaluations.

As for the first comparison, we run the same algorithm as our proposed approach, however, it terminates as soon as it finds a stable design solution. Therefore, this approach, called NO_OPT, does not involve any performance optimization but guarantees worst-case stability. We calculate the relative expected control cost improvements $\frac{J^{\text{e}}_{\text{NO\_OPT}} - J^{\text{e}}_{\text{EXP–WST}}}{J^{\text{e}}_{\text{NO\_OPT}}}$, where $J^{\text{e}}_{\text{EXP–WST}}$ and $J^{\text{e}}_{\text{NO\_OPT}}$ are the expected control costs produced by our approach and the NO_OPT approach, respectively. Our proposed approach produces solutions with guaranteed stability and an overall control quality improvement of $53 \pm 11\%$ on average, compared to an approach which only guarantees worst-case stability and does not involve any expected control performance optimization.

The second comparison is made with an optimization approach driven by the worst-case control performance. The approach, called WST, is exactly the same as our approach but the objective function to be optimized is the worst-case control cost. Similar to the previous experiment, we are interested in the relative expected control cost improvements $\frac{J^{\text{e}}_{\text{WST}} - J^{\text{e}}_{\text{EXP–WST}}}{J^{\text{e}}_{\text{WST}}}$, where $J^{\text{e}}_{\text{WST}}$ is the expected control cost of the final solution obtained by the WST approach. Our proposed approach, while still guarantees worst-case stability, has an average improvement of $26 \pm 9\%$ in terms of the expected control cost, compared to an approach driven by the worst-case control performance.

The third comparison is performed against an optimization approach, called EXP, which only takes into consideration the expected control performance. Since the worst-case control performance constraints are ignored, the search space is larger, and the algorithm should be able to find a superior design solution in terms of the expected control performance. The comparison has been made considering the relative expected control cost difference $\frac{J^{\text{e}}_{\text{EXP}} - J^{\text{e}}_{\text{EXP–WST}}}{J^{\text{e}}_{\text{EXP}}}$, where $J^{\text{e}}_{\text{EXP}}$ is the expected control cost of the final solution found by the EXP approach. Since the worst-case control performance constraints are relaxed, the final solution of this approach can turn out to be unstable. Therefore, in addition to the relative expected control cost comparison, we also report the percentage of designs produced by the EXP approach for which the worst-case stability cannot be guaranteed. The first observation is that, on average, for $44 \pm 13\%$ of the benchmarks this algorithm ended up with a design solution for which the stability could not be guaranteed. The second observation is that our approach is on average $-2.3 \pm 3.7\%$ away from the relaxed optimization approach exclusively guided by expected control performance. This states that we are able to guarantee worst-case stability with a very small loss on expected control quality.

Finally, we measure the runtime of our proposed approach on a PC with a quad-core CPU running at 2.83 GHz with 8 GB of RAM and Linux operating system. The expected runtime of our approach is 182 ± 138 seconds. For large systems (15 control applications), our approach could find a high-quality stable design solution in less than 7 minutes, while it takes 178 minutes for the EXP approach, which is based on a genetic algorithm similar to [20], to terminate. This includes real-time parameters optimization, control synthesis and delay compensation, latency and jitter analysis based on real-time response-time analysis, worst-case stability analysis using the Jitter Margin toolbox, event-driven system simulation, and expected control performance analysis using the Jitterbug toolbox.

TABLE I
EXPERIMENTAL RESULTS: COMPARISON AGAINST NO_OPT, WST, EXP.

	NO_OPT	WST	EXP	
	Improvement (%)	Improvement (%)	Difference (%)	Invalid (%)
Mean ($\mu \pm \sigma$)	$53 \pm 11\%$	$26 \pm 9\%$	$-2.3 \pm 3.7\%$	$44 \pm 13\%$

VIII. CONCLUSIONS

In this paper, we highlight the importance of taking implementation aspects into consideration during the design of embedded control systems. Ignoring these implementation details leads to over-provisioned, low-quality, or unstable design solutions. We further illustrate that a correct design methodology targets the expected control performance as its main objective, while also guaranteeing worst-case stability. Finally, based on these principles, we propose a methodology for implementation-aware design of high-quality and stable embedded control systems on shared platforms.

ACKNOWLEDGMENTS

This research has been partially supported by the Swedish national strategic research area (project eLLIIT), the Swedish Research Council, the Hasler Foundation (project no. 15048), the ONR-G through the Award Grant No. N62909-17-1-2006, and the BodyPoweredSenSE (grant no. 20NA21-143069) RTD project evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

REFERENCES

[1] M. Di Natale and A. L. Sangiovanni-Vincentelli. "Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools". In: *Proceedings of the IEEE* 98.4 (2010), pp. 603–620.

[2] Karl-Johan Åström and Björn Wittenmark. *Computer-Controlled Systems*. 3rd ed. Prentice Hall, 1997.

[3] Anton Cervin. "Stability and Worst-Case Performance Analysis of Sampled-Data Control Systems with Input and Output Jitter". In: *Proceedings of the 2012 American Control Conference (ACC)*. 2012.

[4] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl Erik Årzén. "How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime". In: *IEEE Control Systems Magazine* 23.3 (2003), pp. 16–30.

[5] Björn Wittenmark, Johan Nilsson, and Martin Törngren. "Timing Problems in Real-Time Control Systems". In: *Proceedings of the American Control Conference*. 1995, pp. 2000–2004.

[6] 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*. 2000, pp. 4865–4870.

[7] 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*. 1996, pp. 13–21.

[8] E. Bini and A. Cervin. "Delay-Aware Period Assignment in Control Systems". In: *Proceedings of the* 29^{th} *IEEE Real-Time Systems Symposium*. 2008, pp. 291–300.
[9] Fumin Zhang, Klementyna Szwaykowska, Wayne Wolf, and Vincent Mooney. "Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems". In: *Proceedings of the* 29^{th} *IEEE Real-Time Systems Symposium*. 2008, pp. 47–56.
[10] Pratyush Kumar et al. "A Hybrid Approach to Cyber-Physical Systems Verification". In: *Proceedings of the* 49^{th} *Design Automation Conference*. 2012.
[11] H. Rehbinder and M. Sanfridson. "Integration of Off-Line Scheduling and Optimal Control". In: *Proceedings of the* 12^{th} *Euromicro Conference on Real-Time Systems*. 2000, pp. 137–143.
[12] Dip Goswami, Martin Lukasiewycz, Reinhard Schneider, and Samarjit Chakraborty. "Time-Triggered Implementations of Mixed-Criticality Automotive Software". In: *Proceedings of the* 15^{th} *Conference for Design, Automation and Test in Europe (DATE)*. 2012.
[13] Michael Eisenring, Lothar Thiele, and Eckart Zitzler. "Conflicting criteria in embedded system design". In: *IEEE Design & Test of Computers* 17.2 (2000), pp. 51–59.
[14] B. Lincoln and A. Cervin. "Jitterbug: A Tool for Analysis of Real-Time Control Performance". In: *Proceedings of the* 41^{st} *IEEE Conference on Decision and Control*. 2002, pp. 1319–1324.
[15] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. "A General Framework for Analysing System Properties in Platform-Based Embedded System Designs." In: *DATE*. Vol. 3. 2003.
[16] Amir Aminifar. "Analysis, Design, and Optimization of Embedded Control Systems". PhD thesis. Linköping Studies in Science and Technology, 2016.
[17] Amir Aminifar, Soheil Samii, Petru Eles, Zebo Peng, and Anton Cervin. "Designing High-Quality Embedded Control Systems with Guaranteed Stability". In: *Proceedings of the* 33^{th} *IEEE Real-Time Systems Symposium (RTSS)*. 2012.
[18] Amir Aminifar, Petru Eles, Zebo Peng, and Anton Cervin. "Control-quality driven design of cyber-physical systems with robustness guarantees". In: *Proceedings of the* 16^{th} *Conference for Design, Automation and Test in Europe (DATE)*. 2013.
[19] J. Nocedal and S.J. Wright. *Numerical Optimization*. 2nd ed. Springer, 1999.
[20] S. Samii, A. Cervin, P. Eles, and Z. Peng. "Integrated Scheduling and Synthesis of Control Applications on Distributed Embedded Systems". In: *Proceedings of the Design, Automation and Test in Europe Conference*. 2009, pp. 57–62.

Amir Aminifar received his PhD from the Swedish National Computer Science Graduate School (CUGS), Linköping University, Sweden, in 2016. He is currently a research scientist at the Embedded Systems Laboratory of Swiss Federal Institute of Technology Lausanne (EPFL). His current research interests are centered around embedded and cyber-physical systems.

Petru Eles is Professor of Embedded Computer Systems with the Department of Computer and Information Science (IDA), Linköping University. He received his PhD in Computer Science, Politehnica University Bucharest. Petru Eles' current research interests include embedded systems, real-time systems, electronic design automation.

Zebo Peng is a Professor of Computer Systems, Director of the Embedded Systems Laboratory, and Vice-Chairman of the Department of Computer and Information Science at Linkping University. He has published more than 350 technical papers and five books in various topics related to embedded and cyber-physical systems.

Anton Cervin received his PhD in Automatic Control in 2003. He is currently an associate professor at Lund University, Sweden, where he does research on event-based control, autonomous systems, real-time systems, and controller-scheduling co-design.

Karl-Erik Årzén received his PhD in Automatic Control from Lund University in 1987. Since 2000 he is Professor in Automatic Control at Lund University. He is co-director for the Wallenberg Autonomous Systems and Software Program (WASP), a member of IEEE and of the Swedish Academy of Engineering Sciences. His research interests are control of computer systems, cloud computing, and embedded real-time control.