# Runtime Trade-Offs Between Control Performance and Resource Usage in Embedded Self-Triggered Control Systems

Soheil Samii[1], Petru Eles[1], Zebo Peng[1], Anton Cervin[2]
[1]Department of Computer and Information Science, Linköping University, Sweden
[2]Department of Automatic Control, Lund University, Sweden

## Abstract

*During the recent years, researchers in control engineering have proposed more resource-efficient control strategies than the traditional periodic control paradigm, resulting in two main control approaches: event-based and self-triggered control. Such nonperiodic, state-based control methods, although subject to many open research problems, can provide similar control performance as periodic control implementations but with less use of computation and communication resources. An important research direction, which is the central part of this paper, is to find new runtime scheduling techniques to trade off control performance with resource usage in self-triggered control systems. We present in this paper the current state of our research on scheduling of self-triggered control tasks under the consideration of the control-performance versus resource-usage trade-off.*

## 1 Introduction and Related Work

Embedded control systems have traditionally been designed and implemented as periodic tasks that periodically sample and read sensors, compute control signals, and write the computed control signals to actuators [18]. Modern embedded control systems comprise several control loops (several physical plants are controlled concurrently) that share execution platforms with limited computation and communication bandwidth [6]. Such resource sharing is not only due to multiple control loops but also due to other (noncontrol) application tasks that execute on the same platform as the control tasks and, moreover, have time-varying resource needs. The design goal is not only to provide high control performance for the multiple control loops by an efficient use of the resources, but also to accomodate resources for the execution of other application tasks. Since the resource requirements of the application tasks are inherently varying during execution, the system needs to implement resource management and runtime adaptation techniques that find proper trade-offs between the resource usage and control performance of the multiple control tasks. Research efforts have been made recently towards such resource management of periodic control systems with mode changes [4, 13] and overload scenarios [10, 3].

Periodic control tasks, which are common in many embedded control systems, use the system resources to sample and compute control signals independently of the states of the controlled plants, leading to inefficient resource usage in two extreme cases: (1) the resources are used unnecessarily much when a plant is stable, and (2) depending on the period, the resources might be used too little to provide good control when a plant is close to instability (the two inefficiencies also arise in situations with small and large plant disturbances, respectively).

Event-based control [16] is an increasingly popular approach that can achieve similar control performance as periodic control but with less use of CPUs and communication components [17, 14, 7, 5, 8]. In such approaches, plant states are measured continuously to generate control events when needed, which then activate the control tasks that perform sampling, computation, and actuation (periodic control systems can be considered to constitute a class of event-based systems that generate control events with a constant time-period independent of the states of the controlled plant). An important characteristic of event-based control is that other tasks executing on the same platform as the control tasks can use the resources more often than if periodic tasks are used to implement the control loops. While reducing CPU and communication requirements, event-based control loops typically include specialized hardware for continuous measurement or very high-rate sampling of plant states to generate control events.

Self-triggered control [1, 15] is another resource-efficient alternative that does not rely on specialized, hardware-based event-generators but results in similar levels of resource usage. Instead of reacting to control events, a self-triggered control task uses the sampled states and a model of the controlled plant to compute at runtime a deadline on when the task needs to execute again to achieve specified control requirements (e.g., stability requirements). Since the deadline of the next task execution is computed already at the end of the task, a resource manager has, compared to event-based control systems, a larger time window and more options for task scheduling and optimization of control performance and resource usage; in event-based control systems, a control event usually implies that the control task has immediate or very urgent need to execute, imposing tight constraints on the resource manager and scheduler.

The goal of our research is to develop a software-based approach for online scheduling and control-performance optimization for monoprocessor systems running multiple self-triggered control tasks, in conjunction with other application tasks with time-varying CPU requirements. This approach must be able to at runtime find proper trade-offs between the control tasks' control performance and CPU usage, as well as to consider the varying CPU requirements of other running application tasks. In addition, to be able to make use of this approach, a worst-case analysis method must be developed to verify at design time that the computation resources are sufficient to achieve stability of all control loops in worst-case execution scenarios, assuming in such situations that the resources are used only by the control tasks. To our knowledge, the resource-management problem at hand has not been elaborated for event-based and self-triggered control in the literature of real-time and control systems.

The remainder of this paper is organized as follows. Sections 2 and 3 describe the system model and self-triggered control paradigm, respectively. Our problem formulation is presented in detail in Section 4. We present in Section 5 a brief outline of our scheduling and resource management technique. Subsequently, we outline in Section 6 our remaining research work and expected results. Section 7 concludes the paper.

## 2 System Model

Let us in this section introduce the system model and components that we consider in this paper. Figure 1 shows an example of a control system with a CPU hosting several application tasks depicted with white circles. The tasks are divided in two partitions: a *control* partition and a *best-effort* partition. The control partition comprises a scheduler and three control tasks $\tau_1$, $\tau_2$, and $\tau_3$ that implement feedback-control loops for the three physical plants $P_1$, $P_2$, and $P_3$, respectively. The outputs of a plant are connected to A/D converters and sampled by the corresponding control task. The produced control signals are written to the actuators through D/A converters and are held constant until the next execution of the task. The tasks are scheduled on the CPU according to some scheduling policy, priorities, and deadlines.

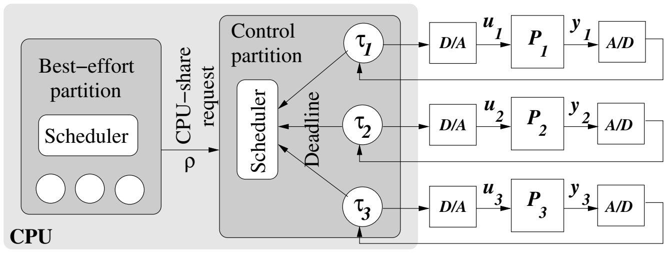

**Figure 1. Control System Architecture**

| | | |
|---|---|---|
| Sampling Computation Actuation | Update states | Compute deadline |

**Figure 2. Components of the Self-Triggered Control Task**

The best-effort partition in Figure 1 consists of three tasks that do not have strict timing constraints but rather execute according to some scheduling policy when there are computation resources available (i.e., when the control partition is not executing a control task). Although the control partition has highest priority, the best-effort partition can provide a request on the amount of CPU currently needed by its tasks. This request can be used by the scheduler in the control partition to achieve a proper trade-off between the provided control performance and CPU usage of the control tasks. The best-effort partition receives its requested CPU share only if the control tasks can achieve sufficient levels of control performance with the remaining CPU share—if not, the control tasks will use more CPU time and give less to the best-effort partition.

## 3 Self-Triggered Control

We shall in this section present the self-triggered control paradigm [1, 15]. The general structure of the control tasks we shall discuss in this paper is shown in Figure 2. The first execution segment comprises three sequential parts: (1) sampling of the plant outputs, (2) computation of the control signal, and (3) writing the computed control signal to actuators (the plant in-

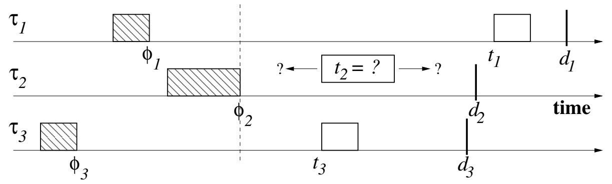

**Figure 3. Scheduling example**

puts). The second segment updates controller states to be used for the next control computation (the next execution of the control task). These two first execution segments constitute the traditional periodic control task, which can be scheduled with traditional real-time schedulers for periodic tasks (e.g., rate-monotonic or earliest-deadline-first scheduling [9]).

A self-triggered control task is responsible for triggering its next execution time, which is decided by the task itself depending on the latest measurements of the plant outputs and the plant dynamics. This triggering is done by providing a deadline on the next execution to the operating system (the scheduler of the control partition in Figure 1), which in turn uses this deadline to schedule the next execution. Thus, in addition to the first two execution segments in Figure 2, a self-triggered control task computes in the third execution segment a deadline on when it must sample the outputs and update the control signal again (i.e., a deadline on when the first segment of the task must start to execute again). This deadline is computed to satisfy the specified control objectives, which usually are requirements related to stability of the controlled plants. Contributions to the computation of this deadline in two different control contexts have been made by Anta and Tabuada [1, 2] and by Wang and Lemmon [15].

The computed deadline is used as a constraint for the scheduling of task executions in the context of resource sharing among multiple control tasks. The scheduling is done by the operating system, which in addition to the deadline considers the CPU request of the best-effort partition and a proper control-performance trade-off for the control-task partition. This scheduling component of the control partition is the main topic of this paper.

Figure 3 shows an example of the schedule at some point in execution of three control tasks. At time $\phi_2$, task $\tau_2$ finishes its execution and prior to that it must schedule its next execution by computing its deadline $d_2$ and start time $t_2$. We can see that tasks $\tau_1$ and $\tau_3$ have already executed before time $\phi_2$ (the dashed rectangles) and have consequently already scheduled their next executions with start times $t_1$ and $t_3$, respectively. The tasks in the best-effort partition are scheduled in the time intervals in which no control task is scheduled for execution. We shall come back to Figure 3 in the next section.

## 4 Problem Formulation

Let us in this section have a detailed discussion of the resource-management and optimization problem at hand. Given is a set of self-triggered control tasks $\mathbf{T}$ with index set $\mathcal{I}_\mathbf{T}$. Each task $\tau_i \in \mathbf{T}$ ($i \in \mathcal{I}_\mathbf{T}$) implements a given feedback controller of a plant

$$P_i: \begin{array}{l} \dot{\boldsymbol{x}}_i = A_i \boldsymbol{x}_i + B_i \boldsymbol{u}_i \\ \boldsymbol{y}_i = C_i \boldsymbol{x}_i \end{array}$$

where the vectors $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and controlled input, respectively. The continuous-time output $\boldsymbol{y}_i$ is measured and sampled in the first execution segment of $\tau_i$. The controlled input $\boldsymbol{u}_i$ is updated and is held constant until the next execution of the control task. Such linear models are commonly used in controller design and synthesis. The controlled plant can have additive and bounded state disturbances, which can be taken into account by a self-triggered control task when computing deadlines for its future execution [11]. The (worst-case) execution time of task $\tau_i$ is denoted $c_i$.

When a control task $\tau_i \in \mathbf{T}$ has executed the control segments related to sampling, control-signal computation, actuation, and state update, the task computes a deadline $d_i$ that represents the latest start time of the next execution of $\tau_i$. Upon completion of a control task, the scheduler of the control partition is triggered to schedule the next execution of that task, given the current schedule and the computed deadline. This scheduling decision aims to achieve a proper trade-off between control performance and the amount of CPU usage of the control partition. The trade-off is not only done between the multiple control tasks but also between the best-effort partition that generates varying CPU requests during execution. The CPU cost of a control-task execution relative to the provided control performance is thus decided by the CPU requests from the other running tasks in the best-effort partition (Figure 1 shows that the best-effort partition communicates its CPU need to the scheduler of the control partition).

Let us now formulate the problem central to this paper. Let us denote with $\phi_i$ the scheduled completion time (decided by the start time and the worst-case execution time) of the current execution of a task $\tau_i$. Having computed the deadline $d_i$ in the third execution segment of $\tau_i$ (Figure 2), a scheduling constraint for $\tau_i$ is that $\phi_i \leqslant t_i \leqslant d_i$. This constraint is to be taken into consideration by the scheduler when placing the next execution into the current schedule. This placement must not only consider the computed deadline, but also the CPU request from the best-effort partition and the control-performance trade-off between the task and the other already scheduled control-task executions. The placement of the next execution of $\tau_i$ in the current schedule can be done in the idle times of the CPU or by moving an already scheduled task forwards or backwards in time. This trade-off among the control performance of the multiple control tasks must be considered together with the resource usage, which has a varying cost due to the varying CPU request of other running application tasks. For each $\tau_j \in \mathbf{T} \setminus \{\tau_i\}$, we have the following properties:

- We have an already computed deadline $d_j$.
- We have a start time $t_j$ ($t_j \leqslant d_j$).
- For each $\tau_k \in \mathbf{T} \setminus \{\tau_i, \tau_j\}$, either $t_k + c_k \leqslant t_j$ or $t_k \geqslant t_j + c_j$ hold.

The placement of $\tau_i$ in the schedule (i.e., the determination of $t_i$) might involve changing the current schedule (i.e., changing $t_j$ for some $j \in \mathcal{I}_\mathbf{T} \setminus \{i\}$) and must therefore satisfy the following constraints for each $\tau_j \in \mathbf{T}$:

- $\phi_i \leqslant t_j \leqslant d_j$
- For each $\tau_k \in \mathbf{T} \setminus \{\tau_j\}$, either $t_k + c_k \leqslant t_j$ or $t_k \geqslant t_j + c_j$ hold.

For example, in Figure 3, when deciding the schedule of $\tau_2$, the scheduler considers the schedule after time $\phi_2$. If another control task must be moved due to control-performance trade-offs, it must be moved to or after $\phi_2$.

The cost that we adopt for each control task $\tau_i \in \mathbf{T}$ is a linear combination $J_i(t_i) = w_i J_i^\mathrm{c}(t_i) + \rho J_i^\mathrm{r}(t_i)$, where $w_i J_i^\mathrm{c}$ is a weighted control cost (smaller cost means higher control performance), $J_i^\mathrm{r}$ is the cost of executing on the CPU (executing a task later in time leads to a smaller resource cost), and $\rho$ is a time-varying parameter that the other noncontrol tasks on the CPU use to communicate to the scheduler the desired trade-off between control performance and resource usage. The parameter $\rho$ is sampled by the control-partition scheduler before any scheduling decision is taken. The given weights $w_i$ are used to transform the control costs to similar magnitudes as well as to indicate relative importance of the multiple control loops. In Figure 1, the best-effort partition changes the parameter $\rho$ continuously depending on the varying workload generated by the running application tasks in that partition. This means that the best-effort partition continuously provides a request on the current control-performance and resource-usage trade-off. The costs are all functions of the start time $t_i$ of the next execution of $\tau_i$. The control cost for task $\tau_i \in \mathbf{T}$ is defined in the time interval $[\phi_i, d_i]$ as the quadratic state cost

$$J_i^\mathrm{c}(t_i) = \frac{1}{d_i - \phi_i} \int_{\phi_i}^{d_i} \boldsymbol{x}_i^\mathsf{T}(t) Q_i \boldsymbol{x}_i(t) dt,$$

where $Q_i$ is a weight matrix (usually a diagonal matrix) that can be used by the designer to give different weights to the individual state components in $\boldsymbol{x}_i$. Such quadratic state costs are common in the literature of control systems [18]. The start time $t_i$ of the next execution of $\tau_i$ determines the time when the first segment (Figure 2) of $\tau_i$ executes (i.e., when the next sample is taken and consequently the next actuation is made to update the control input $\boldsymbol{u}_i$). The start time $t_i$ thus affects the dynamics of the state $\boldsymbol{x}_i$ in the considered time interval $[\phi_i, d_i]$. The CPU cost for task $\tau_i \in \mathbf{T}$ is defined in the same time interval as the linear cost

$$J_i^\mathrm{r}(t_i) = \frac{d_i - t_i}{d_i - \phi_i},$$

which models a linear decrease between a resource cost of 1 at $t_i = \phi_i$ and a cost of 0 at $t_i = d_i$ (postponing the next execution gives a small resource cost and hence CPU time to potential executions of tasks in the best-effort partition). Since the placement of the next execution of a certain task $\tau_i \in \mathbf{T}$ might involve moving some of the other tasks, the objective of the control-task scheduler is to find a schedule that minimizes the cumulative cost of all self-triggered control tasks. This cost is defined as

$$J = \sum_{i \in \mathcal{I}_\mathbf{T}} J_i(t_i) = \sum_{i \in \mathcal{I}_\mathbf{T}} \left( w_i J_i^\mathrm{c}(t_i) + \rho J_i^\mathrm{r}(t_i) \right).$$

Thus, an already scheduled task execution can be moved if this results in an improvement in the overall cost of all control tasks. For example, in Figure 3, the scheduled execution of $\tau_3$ is moved to accomodate execution of $\tau_2$, if that move results in a smaller overall control and resource cost than if $\tau_2$ is scheduled in the idle time intervals given by the current schedule at in the time interval $[\phi_2, d_2]$.

## 5 Approach Outline

We shall in this section propose our scheduling method for self-triggered control tasks. Let us consider the execution of control task $\tau_i \in \mathbf{T}$ and assume that it is scheduled to complete at time $\phi_i$ (as an example, this could be task $\tau_2$ in Figure 3). The most recent measurement of the outputs of plant $P_i$ is $\boldsymbol{y}_{\phi_i} = \boldsymbol{y}_i(\phi_i - c_i)$.

The cost function $J_i^{\mathrm{c}}$ can be written as a function of this last sample and the start time of the next execution of $\tau_i$ as $J_i^{\mathrm{c}}(t_i) = \boldsymbol{y}_{\phi_i}^{\mathsf{T}} M_i(t_i) \boldsymbol{y}_{\phi_i}$. The matrix $M_i$ depends on $t_i$ and has a complex closed-form mathematical expression involving several matrix integrals. We consider that this matrix is approximated numerically by another matrix $\tilde{M}_i(t_i)$ of polynomials, which can be evaluated fast at runtime. Such numerical approximations are done at design time. Thus, the approximated overall cost to be considered at runtime by the control-task scheduler is

$$\tilde{J}_i(t_i) = w_i \boldsymbol{y}_{\phi_i}^{\mathsf{T}} \tilde{M}_i(t_i) \boldsymbol{y}_{\phi_i} + \rho \frac{d_i - t_i}{d_i - \phi_i}.$$

Our scheduling approach is based on a golden-section search [12]. For task $\tau_i$, this search evaluates $\tilde{J}_i(t_i)$ for several values of $t_i \in [\phi_i, d_i]$ towards the best solution found by the golden-section search. The granularity of the search can be chosen depending on the constraints on execution-time overhead of the scheduler.

When the golden-section search has completed, we need to consider how its results affect the existing schedule for the other control-task executions. If the best $t_i$ is in conflict with the existing schedule, the scheduler must decide whether to modify the current schedule in order to accomodate the next execution of $\tau_i$ at the best start time $t_i$, or to use another value of $t_i$ that is not in conflict with the existing schedule. This trade-off is enabled by the fact that the scheduler has already performed the search for the currently scheduled control-task executions. Thus, for each $\tau_j \in \mathbf{T} \setminus \{\tau_i\}$, the scheduler knows at runtime the value of $\tilde{J}_j(t_j)$ for several values of $t_j \in [\phi_j, d_j]$, out of which the scheduler only needs to consider those evaluated values in $[\phi_i, d_j]$. Some of these values are then compared to find the best overall cost trade-off possible with this proposed scheduling heuristic.

## 6 Further Work

We are currently working on finalizing the proposed scheduling heuristic and implementing it in a simulation platform. We plan to use this platform to validate and evaluate our proposed approach. In the following list, we summarize the remaining work to be done and our expected research results:

- Assess the execution-time overhead of the scheduling mechanism (deciding the start time of the next task execution) and compare this to a periodic control strategy.

- Develop a design-time analysis method that can determine whether the available computation resources are sufficient to achieve stable control for all controlled plants in worst-case scenarios. This analysis does not need to consider the execution requirements of the other application tasks in the best-effort partition as the control tasks are allowed to use the full capacity of the CPU to achieve stable control in worst-case scenarios. In addition, it must be guaranteed offline that, in every execution scenario, the scheduler is able to schedule the control-task executions before their respective deadlines—that is, to achieve stability—imposed in the last execution segment of each self-triggered control task.

- Perform experiments on various benchmarks without a best-effort partition to measure the control performance achieved by our proposed scheduling heuristic. We expect to achieve similar control performance as periodic control but with less use of computation resources, including the execution-time overhead induced by the proposed control-task scheduler.

- Perform experiments on various benchmarks with a best-effort partition with different CPU requirements. It is interesting to investigate how the control-performance is degraded when the control tasks share the CPU with other application tasks. We expect to have no or little degradation for scenarios with some CPU request from the best-effort partition. When $\rho$ is increased even more, it is natural to expect a larger degradation, but still satisfactory control performance. Even if $\rho$ is very large, stability must be guaranteed at the cost of not giving the requested CPU bandwidth to the application tasks in the best-effort partition.

## 7 Conclusions

Resource-constrained embedded control systems, traditionally implemented with periodic sampling and actuation, are emerging towards such adaptive implementation methodologies as event-based and self-triggered control in which the control intervals are dynamic and governed by the system state. Scheduling and resource-management in such systems with several control loops must be efficient both in terms of the provided control performance and the amount of system resources used at runtime. We have in this paper presented the current status of our research towards software-based runtime resource-management and optimization techniques for embedded self-triggered control systems. We have outlined the remaining work to be completed and the expected results of our research.

## Acknowledgments

The authors thank Prof. Paulo Tabuada and the other members of Cyber-Physical Systems Laboratory at University of California, Los Angeles, for hosting Soheil Samii during a visit in 2009. The ideas presented in this paper originated during that visit.

## References

[1] A. Anta and P. Tabuada. Self-triggered stabilization of homogeneous control systems. In *Proceedings of the American Control Conference*, pages 4129–4134, 2008.

[2] A. Anta and P. Tabuada. On the benefits of relaxing the periodicity assumption for networked control systems over CAN. In *Proceedings of the* 30th *IEEE Real-Time Systems Symposium*, pages 3–12, 2009.

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

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

[5] A. Cervin and T. Henningsson. Scheduling of event-triggered controllers on a shared network. In *Proceedings of the* 47th *Conference on Decision and Control*, pages 3601–3606, 2008.

[6] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. E. Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, 2003.

[7] A. Cervin and E. Johannesson. Sporadic control of scalar systems with delay, jitter and measurement noise. In *Proceedings of the* 17th *IFAC World Congress*, 2008.

[8] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. van den Bosch. Analysis of event-driven controllers for linear systems. *International Journal of Control*, 81(4):571–590, 2008.

[9] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):47–61, 1973.

[10] P. Martí, J. Yépez, M. Velasco, R. Villà, and J. Fuertes. Managing quality-of-control in network-based control systems by controller and message scheduling co-design. *IEEE Transactions on Industrial Electronics*, 51(6):1159–1167, 2004.

[11] M. Mazo Jr. and P. Tabuada. Input-to-state stability of self-triggered control systems. In *Proceedings of the* 48th *Conference on Decision and Control*, pages 928–933, 2009.

[12] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1988.

[13] S. Samii, P. Eles, Z. Peng, and A. Cervin. Quality-driven synthesis of embedded multi-mode control systems. In *Proceedings of the* 46th *Design Automation Conference*, pages 864–869, 2009.

[14] P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52(9):1680–1685, 2007.

[15] X. Wang and M. Lemmon. Self-triggered feedback control systems with finite-gain l2 stability. *IEEE Transactions on Automatic Control*, 45(3):452–467, 2009.

[16] K. J. Åström. Event based control. In *Analysis and Design of Nonlinear Control Systems: In Honor of Alberto Isidori.* Springer Verlag, 2007.

[17] K. J. Åström and B. Bernhardsson. Comparison of periodic and event based sampling for first-order stochastic systems. In *Proceedings of the* 14th *IFAC World Congress*, volume J, pages 301–306, 1999.

[18] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems.* Prentice Hall, 1997.