Dynamic Scheduling and Control-Quality Optimization of Self-Triggered Control Applications

Soheil Samii[1], Petru Eles[1], Zebo Peng[1], Paulo Tabuada[2], Anton Cervin[3]
[1]Department of Computer and Information Science, Linköping University, Sweden
[2]Department of Electrical Engineering, University of California, Los Angeles, USA
[3]Department of Automatic Control, Lund University, Sweden

Abstract**—Time-triggered periodic control implementations are over provisioned for many execution scenarios in which the states of the controlled plants are close to equilibrium. To address this inefficient use of computation resources, researchers have proposed self-triggered control approaches in which the control task computes its execution deadline at runtime based on the state and dynamical properties of the controlled plant. The potential advantages of this control approach cannot, however, be achieved without adequate online resource-management policies. This paper addresses scheduling of multiple self-triggered control tasks that execute on a uniprocessor platform, where the optimization objective is to find trade-offs between the control performance and CPU usage of all control tasks. Our experimental results show that efficiency in terms of control performance and reduced CPU usage can be achieved with the heuristic proposed in this paper.**

I. Introduction and Related Work

Control systems have traditionally been designed and implemented as tasks that periodically sample and read sensors, compute control signals, and write the computed control signals to actuators [1]. Many systems comprise several such control loops (several physical plants are controlled concurrently) that share execution platforms with limited computation bandwidth [2]. Moreover, resource sharing is not only due to multiple control loops but can also be due to other (noncontrol) application tasks that execute on the same platform. In addition to optimizing control performance, it is important to minimize the CPU usage of the control tasks, in order to accommodate several control applications on a limited amount of resources and, if needed, provide a certain amount of bandwidth to other noncontrol applications. For the case of periodic control systems, research efforts have been made recently towards efficient resource management with additional hard real-time tasks [3], mode changes [4], [5], and overload scenarios [6], [7].

Control design and scheduling of periodic real-time control systems have well-established theory that supports their practical implementation and deployment. In addition, the interaction between control and scheduling for periodic systems has been elaborated in literature [2]. Nevertheless, periodic implementations can result in inefficient resource usage in many execution scenarios. The control tasks are triggered and executed periodically merely based on the elapsed time and not based on the states of the controlled plants, leading to inefficiencies in two cases: (1) the resources are used unnecessarily much when a plant is in or close to equilibrium, and (2) depending on the period, the resources might be used too little to provide good control when a plant is far from the desired state in equilibrium (the two inefficiencies also arise in situations with small and large disturbances, respectively). Event-based and self-triggered control are the two main approaches that have been proposed recently to address inefficient resource usage in control systems.

Event-based control [8] is an approach that can result in similar control performance as periodic control but with more relaxed requirements on CPU bandwidth [9], [10], [11], [12], [13]. 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). While reducing resource usage, event-based control loops typically include specialized hardware (e.g., ASIC or FPGA implementations) for continuous measurement or very high-rate sampling of plant states to generate control events.

Self-triggered control [14], [15], [16], [17] is an alternative that achieves similar reduced levels of resource usage as event-based control. A self-triggered control task computes deadlines on its future executions, by using the sampled states and the dynamical properties of the controlled system, thus canceling the need of specialized hardware components for event generation. The deadlines are computed based on stability requirements or other specifications of minimum control performance. Since the deadline of the next execution of a task is computed already at the end of the latest completed execution, 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

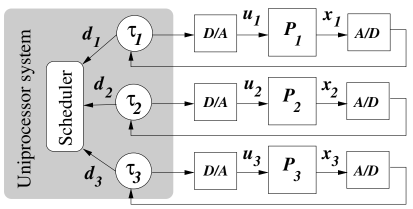

Figure 1. Control-system architecture. The three feedback-control loops include three control tasks on a uniprocessor computation platform. Deadlines are computed at runtime and given to the scheduler.

systems, a control event usually implies that the control task has immediate or very urgent need to execute, thus imposing very tight constraints on the resource manager and scheduler.

The contribution of this paper is a software-based middleware component for scheduling and optimization of control performance and CPU usage of multiple self-triggered control loops on a uniprocessor platform. To our knowledge, the resource management problem at hand has not been treated before in literature. Our heuristic is based on cost-function approximations and search strategies. Stability of the control system is guaranteed through a design-time verification and by construction of the scheduling heuristic.

The remainder of this paper is organized as follows. We present the system and plant model in Section II. In Section III, we discuss the temporal properties of self-triggered control. Section IV shows an example of the execution of multiple self-triggered tasks on a uniprocessor platform. The example further highlights the scheduling and optimization objectives of this paper. The scheduling problem is defined in Section V and is followed by our scheduling heuristic in Section VI. Experimental results with comparisons to periodic control are presented in Section VII. The paper is concluded in Section VIII.

II. 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 three control tasks (depicted with white circles) τ_1, τ_2, and τ_3 that implement feedback-control loops for the three 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. The contribution of this paper is a scheduler component for efficient CPU usage and control performance.

The set of self-triggered control tasks and its index set are denoted with $\mathbf{T}$ and $\mathcal{I}_{\mathbf{T}}$, respectively. Each task $\tau_i \in \mathbf{T}$ ($i \in \mathcal{I}_{\mathbf{T}}$) implements a given feedback controller of a plant. The dynamical properties of this plant are given by a linear, continuous-time state-space model

$$\dot{\boldsymbol{x}}_i(t) = A_i \boldsymbol{x}_i(t) + B_i \boldsymbol{u}_i(t) \tag{1}$$

in which the vectors $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and controlled input, respectively. The plant state is measured and sampled by the control task τ_i. The controlled input $\boldsymbol{u}_i$ is updated at time-varying sampling intervals according to the control law

$$\boldsymbol{u}_i = K_i \boldsymbol{x}_i \tag{2}$$

and is held constant between executions of the control task. The control gain K_i is given and is computed by control design for continuous-time controllers. The design of K_i typically addresses some costs related to the plant state $\boldsymbol{x}_i$ and controlled input $\boldsymbol{u}_i$. The plant model in Equation 1 can include additive and bounded state disturbances, which can be taken into account by a self-triggered control task when computing deadlines for its future execution [18]. The worst-case execution time of task τ_i is denoted c_i and is computed at design time with tools for worst-case execution time analysis.

III. Self-Triggered Control

A self-triggered control task [14], [15], [16], [19], [17] uses the sampled plant states not only to compute control signals, but also to compute a temporal bound for the next task execution, which, if met, guarantees stability of the control system. The self-triggered control task comprises two sequential execution segments. The first execution segment consists of three sequential parts: (1) sampling the plant state $\boldsymbol{x}$ (possibly followed by some data processing), (2) computation of the control signal $\boldsymbol{u}$, and (3) writing it to actuators. This first execution segment is similar to what is performed by the traditional periodic control task.

The second execution segment is characteristic to self-triggered control tasks in which temporal deadlines on task executions are computed dynamically. As shown by Anta and Tabuada [16], [19], the computation of the deadlines are based on the sampled state, the control law, and the plant dynamics in Equation 1. The computed deadlines are valid if there is no preemption between sampling and actuation (a constant delay between sampling and actuation can be taken into consideration). The deadline of a task execution is taken into account by the scheduler and must be met to guarantee stability of the control system. Thus, in addition to the first execution segment, which comprises sampling and actuation, a self-triggered control task computes in the second execution segment a completion deadline D on the next task execution, relative to the completion time of the task execution. The exact time instant of the next task

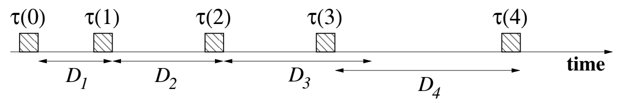

Figure 2. Execution of a self-triggered control task. Each job of the task computes a completion deadline for the next job. The deadline is time-varying and state-dependent.

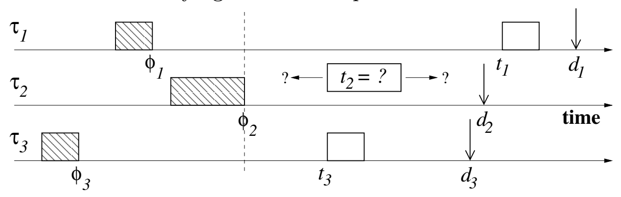

Figure 3. Scheduling example. Tasks τ_1 and τ_3 have completed their execution before ϕ_2 and their next execution instants are t_1 and t_3, respectively. Task τ_2 completes execution at time ϕ_2 and its next execution instant t_2 has to be decided by the scheduler. The imposed deadlines must be met to guarantee stability.

execution, however, is decided by the scheduler based on optimizations of control performance and CPU usage.

Figure 2 shows the execution of several jobs $\tau(q)$ of a control task τ. After the first execution of τ (i.e., after the completion of job $\tau(0)$), we have a relative deadline D_1 for the completion of the second execution of τ (D_1 is the deadline for $\tau(1)$, relative to the completion time of job $\tau(0)$). Observe that the deadline between two consecutive job executions is varying, thus reflecting that the control-task execution is regulated by the dynamically changing plant state, rather than by a fixed period. Note that the fourth execution of τ (job $\tau(3)$) starts and completes before the imposed deadline D_3. The reason why this execution is placed earlier than its deadline can be due to control-quality optimizations or conflicts with other control tasks. The deadline D_4 of the successive execution is relative to the completion time and not relative to the previous deadline.

For a control task $\tau_i \in \mathbf{T}$, it is possible to compute a lower and upper bound $D_i^{\min}$ and $D_i^{\max}$, respectively, for the deadline of a task execution relative to the completion of its previous execution. The minimum relative deadline $D_i^{\min}$ bounds the CPU requirement of the control task and is computed at design time based on the plant dynamics and control law [16], [10]. The maximum relative deadline is decided by the designer to ensure that the control task executes with a minimum rate (e.g., to achieve some level of robustness or a minimum amount of control quality).

IV. Motivational Example

Figure 3 shows the execution of three self-triggered control tasks τ_1, τ_2, and τ_3. The time axes show the scheduled executions of the three tasks, respectively. A dashed rectangle indicate a completed task execution of execution time given by the length of the rectangle. The white rectangles show executions that are scheduled after time moment ϕ_2. The scenario is that task τ_2 has finished its execution at time ϕ_2 and computed its next deadline d_2. The scheduler is activated at time ϕ_2 to schedule the next execution of τ_2, considering the existing scheduled executions of τ_1 and τ_3 (the white rectangles) and the deadlines d_1, d_2, and d_3. Prior to time ϕ_2, task τ_3 finished its execution at ϕ_3 and its next execution was placed at time t_3 by the scheduler. In a similar way, the start time t_1 of τ_1 was decided at its most recent completion time ϕ_1. Other application tasks may execute in the time intervals in which no control task is scheduled for execution.

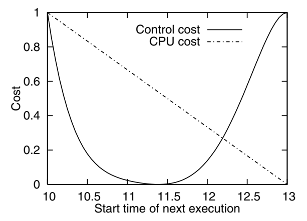

Figure 4. Control and CPU costs. The two costs depend on the next execution instant of the control task and are in general conflicting objectives.

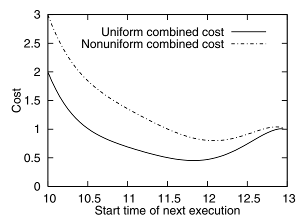

Figure 5. Combined control and CPU costs. Two different combinations are shown with different weights between control and CPU costs.

The objective of the scheduler at time ϕ_2 in Figure 3 is to schedule the next execution of τ_2 (i.e., to decide the start time t_2) before the deadline d_2. Figure 4 shows an example of the control and CPU costs (as functions of t_2) of task τ_2 with solid and dashed lines, respectively. Note that a small control cost in the figure indicates high control quality, and vice versa. For this example, we have $\phi_2 = 10$ and $d_2 - c_2 = 13$, which bound the start time t_2 of the next execution of τ_2. By only considering the control cost, we observe that the optimal start time is 11.4. The intuition is that it is not good to schedule a task immediately after its previous execution (early start times), since the plant state has not changed much by that time. It is also not good to execute the task very late, because this leads to a longer time in which the plant is in open loop between actuations.

By only considering the CPU cost, the optimal start time is 13, which means that the execution will complete exactly at the imposed deadline, if the task experiences its worst-case execution time. As we have discussed, the objective is to consider both the control cost and CPU cost during scheduling. The two costs can be combined together with weights that are based on the required level of trade-off between control performance and resource usage, as well as the characteristics and temporal requirements of other noncontrol applications that execute on the uniprocessor platform. The solid line in Figure 5 shows the sum of the control and CPU cost, indicating equal importance of achieving high control quality and low CPU usage. The dashed line indicates the sum of the two costs in which the CPU cost is included twice in the summation. By considering the cost shown by the dashed line during scheduling, the start time is chosen more in favor of low CPU usage than high control performance. For the solid line, we can see that the optimal start time is 11.8, whereas it is 12.1 for the dashed line. The best start time in each case might be in conflict with an already scheduled execution (e.g., with task τ_3 in Figure 3). In such cases, the scheduler can decide to move an already scheduled execution, if this degradation of control performance and resource usage for that execution is affordable.

V. Problem Formulation

We shall in this section present the specification and objective of the runtime-scheduler component in Figure 1. The two following subsections shall discuss the scheduling constraints that are present at runtime, as well as the optimization objectives of the scheduler.

A. Scheduling constraints

Let us first define nonpreemptive scheduling of a task set $\mathbf{T}$ with index set $\mathcal{I}_{\mathbf{T}}$. We shall consider that each task $\tau_i \in \mathbf{T}$ ($i \in \mathcal{I}_{\mathbf{T}}$) has a worst-case execution time c_i and an absolute deadline $d_i = \phi_i + D_i$, where D_i is computed by the second execution segment of the control task and is the deadline relative to the completion time of the task (Section III). A schedule of the task set under consideration is an assignment of the start time t_i of the execution of each task $\tau_i \in \mathbf{T}$ such that there exists a bijection (also called one-to-one correspondence) $\sigma : \{1, \ldots, |\mathbf{T}|\} \longrightarrow \mathcal{I}_{\mathbf{T}}$ that satisfies the following properties:

$$t_{\sigma(k)} + c_{\sigma(k)} \leqslant d_{\sigma(k)} \qquad \text{for } k \in \{1, \ldots, |\mathbf{T}|\} \tag{3}$$

$$t_{\sigma(k)} + c_{\sigma(k)} \leqslant t_{\sigma(k+1)} \qquad \text{for } k \in \{1, \ldots, |\mathbf{T}| - 1\} \tag{4}$$

The bijection σ gives the order of execution of the task set $\mathbf{T}$ (i.e., the tasks are executed in the order $\tau_{\sigma(1)}, \ldots, \tau_{\sigma(|\mathbf{T}|)}$). Thus, task τ_i starts its execution at time t_i and is preceded by executions of $\sigma^{-1}(i) - 1$ tasks (σ^{-1} is the inverse of σ). Equation 3 models that the start times are chosen such that each task execution meets its imposed deadline, whereas Equation 4 models that the scheduled task executions do not overlap in time (i.e., the CPU can execute at most one task at any time instant).

Having introduced the scheduling constraints, let us proceed with the problem definition. The initial schedule (the schedule at time zero) of the set of control tasks $\mathbf{T}$ is given and determined offline. At runtime, when a task completes its execution, the scheduler is activated to schedule the next execution of that task by considering its deadline and the trade-off between control quality and resource usage. Thus, when a task $\tau_i \in \mathbf{T}$ completes at time ϕ_i, we have at that time a schedule for the task set $\mathbf{T}' = \mathbf{T} \setminus \{\tau_i\}$ with index set $\mathcal{I}_{\mathbf{T}'} = \mathcal{I}_{\mathbf{T}} \setminus \{i\}$. This means that we have a bijection $\sigma' : \{1, \ldots, |\mathbf{T}'|\} \longrightarrow \mathcal{I}_{\mathbf{T}'}$ and an assignment of the start times $\{t_j\}_{j \in \mathcal{I}_{\mathbf{T}'}}$ such that $\phi_i \leqslant t_{\sigma'(1)}$ and that Equations 3 and 4 hold, with $\mathbf{T}$ replaced by $\mathbf{T}'$. At time ϕ_i, task τ_i has a new deadline d_i and the runtime scheduler must decide the start time t_i of the next execution of τ_i to obtain a schedule for the entire set of control tasks $\mathbf{T}$. The scheduler is allowed to change the current order and start times of the already scheduled tasks $\mathbf{T}'$. Thus, after scheduling, each task $\tau_j \in \mathbf{T}$ has a start time $t_j \geqslant \phi_i$ such that all start times constitute a schedule for $\mathbf{T}$ according to Equations 3 and 4. The next subsection presents the optimization objectives that are taken into consideration when determining the start time of a task.

B. Optimization objectives

Our optimization objective at runtime is twofold: to minimize the control cost (a small cost indicates high control quality) and to minimize the CPU cost (the CPU cost indicates the CPU usage of the control tasks). We remind that ϕ_i is the completion time of task τ_i and d_i is the deadline of its next execution. Since the task must complete before its deadline and we consider nonpreemptive scheduling, the start time t_i is allowed to be at most $d_i - c_i$. Let us therefore define the control and CPU cost for task τ_i in the time interval $[\phi_i, d_i - c_i]$, which is the scheduling time window of τ_i. The overall cost to be minimized follows thereafter.

1) Control cost: The (quadratic) *state cost* in the considered time interval $[\phi_i, d_i - c_i]$ is defined as

$$J_i^{\mathrm{x}}(t_i) = \int_{\phi_i}^{d_i} \boldsymbol{x}_i^{\mathsf{T}}(t) Q_i \boldsymbol{x}_i(t) dt, \tag{5}$$

where $t_i \in [\phi_i, d_i - c_i]$ is the start time of the next execution of τ_i. The weight matrix Q_i (usually a diagonal or sparse matrix) is used by the designer to assign weights to the individual state components in $\boldsymbol{x}_i$. It can also be used to transform the cost to a common baseline or to specify importance relative to other control loops. Quadratic state costs are common in the literature of control systems [1] as a metric for control performance. Note that a small cost indicates high control performance, and vice versa. The dependence of the state cost on the start time t_i is implicit in Equation 5. The start time decides the time when the control signal is updated and thus affects the dynamics of the plant state $\boldsymbol{x}_i$ according to Equation 1. In some control problems (e.g., when computing the actual state-feedback law in Equation 2), the cost in Equation 5 also includes a term penalizing the controlled input $\boldsymbol{u}_i$. We do not include this term since the input is determined uniquely by the state through the given control law $\boldsymbol{u}_i = K_i \boldsymbol{x}_i$.

The design of the actual control law, however, typically addresses both the state and the control-input costs.

Let us denote the minimum and maximum value of the state cost J_i^{x} in the time interval $[\phi_i, d_i - c_i]$ with $J_i^{\mathrm{x,min}}$ and $J_i^{\mathrm{x,max}}$, respectively. We define the *control cost* $J_i^{\mathrm{c}} : [\phi_i, d_i - c_i] \longrightarrow [0, 1]$ as

$$J_i^{\mathrm{c}}(t_i) = \frac{J_i^{\mathrm{x}}(t_i) - J_i^{\mathrm{x,min}}}{J_i^{\mathrm{x,max}} - J_i^{\mathrm{x,min}}}. \tag{6}$$

Note that this is a function from $[\phi_i, d_i - c_i]$ to $[0, 1]$, where 0 and 1, respectively, indicate the best and worst possible control performance.

2) CPU cost: The CPU cost $J_i^{\mathrm{r}} : [\phi_i, d_i - c_i] \longrightarrow [0, 1]$ for task τ_i is defined in the same time interval as the linear cost

$$J_i^{\mathrm{r}}(t_i) = \frac{d_i - c_i - t_i}{d_i - c_i - \phi_i}, \tag{7}$$

which models a linear decrease between a CPU cost of 1 at $t_i = \phi_i$ and a cost of 0 at the latest possible start time $t_i = d_i - c_i$ (postponing the next execution gives a small CPU cost since it leads to lower CPU load). An example of the control and CPU costs, is shown in Figure 4, which we discussed in the example in Section IV.

3) Overall trade-off: There are many different possibilities for the trade-off between control performance and CPU usage of the control tasks. The approach taken in this paper is that the specification of the trade-off is made offline in a static manner by the designer. Specifically, we define the cost $J_i(t_i)$ of the task τ_i under scheduling as a linear combination of the control and CPU costs according to

$$J_i(t) = J_i^{\mathrm{c}}(t_i) + \rho J_i^{\mathrm{r}}(t_i), \tag{8}$$

where $\rho \geqslant 0$ is a design constant that is chosen offline to specify the required trade-off between achieving a low control cost versus reducing the CPU usage.[1] For example, by studying Figure 5 again, we note that the solid line shows the sum of the control and CPU costs in Figure 4 with $\rho = 1$. The dashed line shows the case for $\rho = 2$.

At each scheduling point, the optimization goal is to minimize the overall cost of all control tasks. The cost to be minimized is defined as

$$J = \sum_{j \in \mathcal{I}_{\mathbf{T}}} J_j(t_j), \tag{9}$$

which models the cumulative control and CPU cost of the task set $\mathbf{T}$ at a given scheduling point.

[1]The problem statement and our heuristic are also relevant for systems in which the background computations have time-varying CPU requirements. In such systems, the parameter ρ is changed dynamically to reflect the current workload of other noncontrol tasks and is read by the scheduler at each scheduling instant.

VI. Scheduling Heuristic

Our approach is divided into both offline and online activities. The offline activity, which is described in Section VI-A, comprises two parts: (1) to approximate the control cost $J_i^{\mathrm{c}}(t_i)$ for each task $\tau_i \in \mathbf{T}$, and (2) to verify that the platform has sufficient computation capacity to achieve stability in all possible execution scenarios. The online activity, which is implemented in the scheduler component in Figure 1, comprises a search that finds several scheduling alternatives and chooses one of them according to the desired trade-off between control performance and CPU usage. We shall discuss this online heuristic in Section VI-B in which we also elaborate on how the scheduling is made to guarantee stability.

A. Design-time activities

To support the runtime scheduling, two main activities are to be performed at design time. The first aims to reduce the complexity of computing the state cost in Equation 5 at runtime. This is addressed by constructing approximate cost functions, which are affordable to evaluate at runtime optimization. The second activity aims to provide stability guarantees in all possible execution scenarios. This is achieved by a verification at design time and by construction of the runtime scheduling heuristic.

1) Cost-function approximation: We consider that a task τ_i has completed its execution at time ϕ_i at which its next execution is to be scheduled and completed before its imposed deadline d_i. Thus, the start time t_i must be chosen in the time interval $[\phi_i, d_i - c_i]$. The most recent known state is $\boldsymbol{x}_{i,0} = \boldsymbol{x}_i(t_i')$, where t_i' is the start time of the just completed execution of τ_i. The control signal has been updated by the task according to the control law $\boldsymbol{u}_i = K_i \boldsymbol{x}_i$ (Section II). By solving the differential equation in Equation 1 with the theory presented by Åström and Wittenmark [1], we can describe the cost in Equation 5 as

$$J_i^{\mathrm{x}}(\phi_i, t_i) = \boldsymbol{x}_{i,0}^{\mathsf{T}} M_i(\phi_i, t_i) \boldsymbol{x}_{i,0}.$$

The matrix M_i includes matrix exponentials and integrals and is decided by the plant, controller, and cost parameters. It further depends on the difference $d_i - \phi_i$, which is bounded by D_i^{min} and D_i^{max} (Section III). Each element in the matrix $M_i(\phi_i, t_i)$ is a function of the completion time ϕ_i of task τ_i and the start time $t_i \in [\phi_i, d_i - c_i]$ of the next execution of τ_i. An important characteristic of M_i is that it depends only on the difference $t_i - \phi_i$. Due to time complexity, the computation of the matrix $M_i(\phi_i, t_i)$ is not practical to perform at runtime. To cope with this complexity, our approach is to use an approximation $\widehat{M_i}(\phi_i, t_i)$ of

$M_i(\phi_i, t_i)$. The scheduler presented in Section VI-B shall thus consider the approximate state cost

$$\widehat{J_i^{\mathrm{x}}}(t_i) = \boldsymbol{x}_{i,0}^{\mathsf{T}} \widehat{M_i}(\phi_i, t_i) \boldsymbol{x}_{i,0} \tag{10}$$

in the optimization process. The approximation of $M_i(\phi_i, t_i)$ is done at design time by computing M_i for a number of values of the difference $d_i - \phi_i$. The matrix $M_i(\phi_i, t_i)$, which depends only on the difference $t_i - \phi_i$, is computed for equidistant values of $t_i - \phi_i$ between 0 and $d_i - c_i$ (the granularity is a design parameter). The precalculated points are all stored in memory and are used at runtime to compute $\widehat{M_i}(\phi_i, t_i)$.

2) Offline stability guarantee: Before the control system is deployed, it must be made certain that stability of all control loops is guaranteed. This certification is twofold: (1) to make sure that there is sufficient computation capacity to achieve stability, and (2) to make sure that the scheduler, in any execution scenario, finds a schedule that guarantees stability by meeting the imposed deadlines. The first step is to verify at design time that the condition

$$\sum_{j \in \mathcal{I}_{\mathbf{T}}} c_j \leqslant \min\{D_j^{\min}\}_{j \in \mathcal{I}_{\mathbf{T}}} \tag{11}$$

holds. The second step, which is guaranteed by construction of the scheduler, is described in Section VI-B3. To understand Equation 11, let us consider that a task $\tau_i \in \mathbf{T}$ has finished its execution at time ϕ_i and its next execution is to be scheduled. The other tasks $\mathbf{T} \setminus \{\tau_i\}$ are already scheduled before their respective deadlines. The worst-case execution scenario from the point of view of scheduling is that the next execution of τ_i is due within its minimum deadline $D_i^{\min}$, relative to time ϕ_i (i.e., its deadline is $d_i = \phi_i + D_i^{\min}$) and that each scheduled task $\tau_j \in \mathbf{T} \setminus \{\tau_i\}$ has its deadline within the minimum deadline $D_i^{\min}$ of τ_i (i.e., $d_j \leqslant d_i = \phi_i + D_i^{\min}$). In this execution scenario, every task must execute exactly once within a time period of $D_i^{\min}$ (i.e., in the time interval $[\phi_i, \phi_i + D_i^{\min}]$). Equation 11 follows by considering that τ_i is the control task with the smallest possible relative deadline. In Section VI-B3, we describe how the schedule is constructed, provided that Equation 11 holds.

The time overhead of the runtime scheduler described in the next section can be bounded by computing its worst-case execution overhead at design time (this is performed with tools for worst-case execution time analysis). For simplicity of presentation in Equation 11, we consider this overhead to be included in the worst-case execution time c_j of task τ_j. Independently of the runtime scheduling heuristic, the test guarantees not only that all stability-related deadlines can be met at runtime but also that a minimum level of control performance is achieved. The runtime scheduling heuristic presented

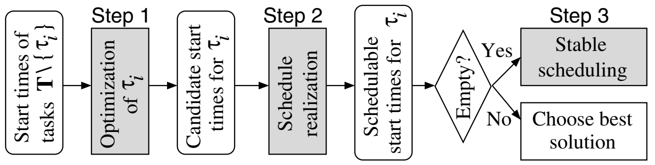
Start times of tasks $\mathbf{T} \setminus \{\tau_i\}$
Step 1
Optimization of τ_i
Candidate start times for τ_i
Step 2
Schedule realization
Schedulable start times for τ_i
Empty?
Yes
No
Step 3
Stable scheduling
Choose best solution

Figure 6. Flowchart of the scheduling heuristic. The first step finds candidate start times that, in the second step, are evaluated with regard to scheduling. If needed, the third step is executed to guarantee stability.

in the following subsection improves on these minimum control-performance guarantees.

B. Runtime heuristic

We shall in this section consider that task $\tau_i \in \mathbf{T}$ has completed its execution at time ϕ_i and that its next execution is to be scheduled before the computed deadline d_i. The other tasks $\mathbf{T} \setminus \{\tau_i\}$ have already been scheduled and each task $\tau_j \in \mathbf{T} \setminus \{\tau_i\}$ has a start time $t_j \geqslant \phi_i$. These start times constitute a schedule for the task set $\mathbf{T} \setminus \{\tau_i\}$, according to the definition of a schedule in Section V-A and Equations 3 and 4. The scheduler must decide the start time t_i of the next execution of τ_i such that $\phi_i \leqslant t_i \leqslant d_i - c_i$, possibly changing the start times of the other task $\mathbf{T} \setminus \{\tau_i\}$. The condition is that the resulting start times $\{t_j\}_{j \in \mathcal{I}_{\mathbf{T}}}$ constitute a schedule for the task set $\mathbf{T}$.

Figure 6 shows a flowchart of our proposed scheduler. The first step is to optimize the start time t_i of the next execution of τ_i. In this step, we do not consider the existing start times of the other tasks $\mathbf{T} \setminus \{\tau_i\}$ but merely focus on the cost J_i in Equation 8. The optimization is based on a search heuristic that results in a set of *candidate start times* $\boldsymbol{\Xi}_i = \{t_i^{(1)}, \ldots, t_i^{(n)}\} \subset [\phi_i, d_i - c_i]$. After this step, the cost $J_i(t_i)$ has been computed for each $t_i \in \boldsymbol{\Xi}_i$. In the second step, we check for each $t_i \in \boldsymbol{\Xi}_i$, whether it is possible to schedule the execution of task τ_i at time t_i, considering the existing start times t_j for each task $\tau_j \in \mathbf{T} \setminus \{\tau_i\}$. This check involves, if necessary, a modification of the starting times of the already scheduled tasks to accommodate the execution of τ_i at the candidate start time t_i. If the start times cannot be modified such that all imposed deadlines are met, then the candidate start time t_i is not feasible. The result of the second step (schedule realization) is thus a subset $\boldsymbol{\Xi}_i' \subseteq \boldsymbol{\Xi}_i$ of the candidate start times. This means that, for each $t_i \in \boldsymbol{\Xi}_i'$, the execution of τ_i can be accommodated at that time, possibly with a modification of the start times $\{t_j\}_{j \in \mathcal{I}_{\mathbf{T}} \setminus \{i\}}$ such that the scheduling constraints in Equations 3 and 4 are satisfied for the whole task set $\mathbf{T}$. For each $t_i \in \boldsymbol{\Xi}_i'$, the scheduler computes the total control and CPU cost, considering all control tasks (Equation 9). The scheduler chooses the start time $t_i \in \boldsymbol{\Xi}_i'$ that leads to the best overall cost. If $\boldsymbol{\Xi}_i' = \emptyset$, meaning that none of the candidate start times in $\mathbf{\Xi}_i$ can be scheduled, the scheduler resorts to the third step (stable scheduling), which guarantees to find a solution that meets all imposed stability-related completion deadlines. Let us in the following three subsections discuss the three steps in Figure 6 in more detail.

1) Optimization of start time: As we have mentioned, in this step, we consider the minimization of the cost $J_i(t_i)$ in Equation 8, which is the combined control and CPU cost of task τ_i. Let us first, however, consider the approximation $\widehat{J}_i^{\mathrm{x}}(t_i)$ (Equation 10) of the state cost $J_i^{\mathrm{x}}(t_i)$ in Equation 5. We shall perform a minimization of this cost by a golden-section search [20]. The search is iterative and maintains, in each iteration, three points $\omega_1, \omega_2, \omega_3 \in [\phi_i, d_i - c_i]$ for which the cost $\widehat{J}_i^{\mathrm{x}}$ has been evaluated. The initial values of the end points are $\omega_1 = \phi_i$ and $\omega_3 = d_i - c_i$. The middle point ω_2 is initially chosen according to the golden ratio as $(\omega_3 - \omega_2)/(\omega_2 - \omega_1) = (1 + \sqrt{5})/2$. The next step is to evaluate the function value for a point ω_4 in the largest of the two intervals $[\omega_1, \omega_2]$ and $[\omega_2, \omega_3]$. This point ω_4 is chosen such that $\omega_4 - \omega_1 = \omega_3 - \omega_2$. If $\widehat{J}_i^{\mathrm{x}}(\omega_4) < \widehat{J}_i^{\mathrm{x}}(\omega_2)$, we update the three points ω_1, ω_2, and ω_3 according to

$$(\omega_1, \omega_2, \omega_3) \longleftarrow (\omega_2, \omega_4, \omega_3)$$

and then repeat the golden-section search. If $\widehat{J}_i^{\mathrm{x}}(\omega_4) > \widehat{J}_i^{\mathrm{x}}(\omega_2)$, we perform the update

$$(\omega_1, \omega_2, \omega_3) \longleftarrow (\omega_1, \omega_2, \omega_4)$$

and proceed with the next iteration. The cost $\widehat{J}_i^{\mathrm{x}}$ is computed efficiently for each point based on the latest sampled state and the precalculated values of M_i, which are stored in memory before system deployment and runtime (Section VI-A1). The search ends after a number of iterations given by the designer. We shall consider this number in the experimental evaluation.

The result of the search is a set of visited points $\mathbf{\Omega}_i = \{t_i^{(1)}, \ldots, t_i^{(n)}\}$ ($n - 3$ is the number of iterations) for which we have $\{\phi_i, d_i - c_i\} \subset \mathbf{\Omega}_i \subset [\phi_i, d_i - c_i]$. The search has evaluated $\widehat{J}_i^{\mathrm{x}}(t_i)$ for each $t_i \in \mathbf{\Omega}_i$. Let us introduce the minimum and maximum approximate state costs $\widehat{J}_i^{\mathrm{x,min}}$ and $\widehat{J}_i^{\mathrm{x,max}}$, respectively, as

$$\widehat{J}_i^{\mathrm{x,min}} = \min_{t_i \in \mathbf{\Omega}_i} \widehat{J}_i^{\mathrm{x}}(t_i) \text{ and } \widehat{J}_i^{\mathrm{x,max}} = \max_{t_i \in \mathbf{\Omega}_i} \widehat{J}_i^{\mathrm{x}}(t_i).$$

We define the approximate control cost $\widehat{J}_i^{\mathrm{c}}(t_i)$ (compare to Equation 6) for each $t_i \in \mathbf{\Omega}_i$ as

$$\widehat{J}_i^{\mathrm{c}}(t_i) = \frac{\widehat{J}_i^{\mathrm{x}}(t_i) - \widehat{J}_i^{\mathrm{x,min}}}{\widehat{J}_i^{\mathrm{x,max}} - \widehat{J}_i^{\mathrm{x,min}}}. \tag{12}$$

Let us now extend $\{\widehat{J}_i^{\mathrm{c}}(t_i^{(1)}), \ldots, \widehat{J}_i^{\mathrm{c}}(t_i^{(n)})\}$ to define $\widehat{J}_i^{\mathrm{c}}(t_i)$ for an arbitrary $t_i \in [\phi_i, d_i - c_i]$. Without loss of generality, we assume that $\phi_i = t_i^{(1)} < t_i^{(2)} < \cdots < t_i^{(n)} = d_i - c_i$. For any $q \in \{1, \ldots, n - 1\}$, we use linear interpolation in the time interval $(t_i^{(q)}, t_i^{(q+1)})$, resulting in $\widehat{J}_i^{\mathrm{c}}(t_i) =$

$$\left(1 - \frac{t_i - t_i^{(q)}}{t_i^{(q+1)} - t_i^{(q)}}\right) \widehat{J}_i^{\mathrm{c}}(t_i^{(q)}) + \frac{t_i - t_i^{(q)}}{t_i^{(q+1)} - t_i^{(q)}} \widehat{J}_i^{\mathrm{c}}(t_i^{(q+1)}) \tag{13}$$

for $t_i^{(q)} < t_i < t_i^{(q+1)}$. Equations 12 and 13 define, for the complete time interval $[\phi_i, d_i - c_i]$, the approximation $\widehat{J}_i^{\mathrm{c}}$ of the control cost in Equation 6. As opposed to an equidistant sampling of the time interval $[\phi_i, d_i - c_i]$, the golden-section search gives a better approximation of $J_i^{\mathrm{x}}(t_i)$ close to the minimum start time, as well as better estimates $\widehat{J}_i^{\mathrm{x,min}}$ and $\widehat{J}_i^{\mathrm{x,max}}$ in Equation 12.

We can now define the approximation $\widehat{J}_i$ of the overall cost J_i in Equation 8 as

$$\widehat{J}_i(t_i) = \widehat{J}_i^{\mathrm{c}}(t_i) + \rho J_i^{\mathrm{r}}(t_i).$$

To consider the twofold objective of optimizing the control quality and CPU usage, we perform the golden-section search in the time interval $[\phi_i, d_i - c_i]$ for the function $\widehat{J}_i(t_i)$. The cost evaluations are in this step merely based on Equations 12, 13, and 7, which do not involve any computations based on the sampled state or the precalculated values of M_i, hence giving time-efficient cost evaluation. This last search results in a finite set of candidate start times $\mathbf{\Xi}_i \subset [\phi_i, d_i - c_i]$ to be considered in the next step.

2) Schedule realization: We shall consider the given start time t_j for each task $\tau_j \in \mathbf{T} \backslash \{\tau_i\}$. These start times have been chosen under the consideration of the scheduling constraints in Equations 3 and 4. We have thus a bijection $\sigma : \{1, \ldots, |\mathbf{T}| - 1\} \longrightarrow \mathcal{I}_{\mathbf{T}} \setminus \{i\}$ that gives the order of execution of the task set $\mathbf{T} \backslash \{\tau_i\}$ (Section V-A). We shall now describe the scheduling procedure to be performed for each candidate start time $t_i \in \mathbf{\Xi}_i$ of task τ_i obtained in the previous step. The scheduler first checks whether the execution of τ_i at the candidate start time t_i overlaps with any existing scheduled task execution. If there is an overlap, the second step is to move the existing overlapping executions forward in time. If this modification also satisfies the deadline constraints (Equation 3), or if no overlapping execution was found, we declare this start time as schedulable.

Let us now consider a candidate start time $t_i \in \mathbf{\Xi}_i$ and discuss how to identify and move overlapping executions of $\mathbf{T} \setminus \{\tau_i\}$. The idea is to identify the first overlapping execution, considering that τ_i starts its execution at t_i. If such an overlap exists, the overlapping execution and its successive executions are pushed forward in time by the minimum amount of time required to schedule τ_i at time t_i such that the scheduling constraint in Equation 4 is satisfied for the entire task set $\mathbf{T}$. To find the first

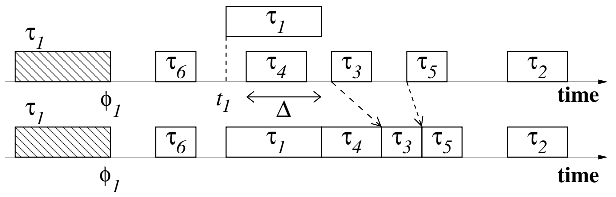

Figure 7. Schedule realization. The upper schedule shows a candidate start time t_1 for τ_1 that is in conflict with the existing schedule. The conflict is solved by pushing the current schedule forward in time by an amount Δ, resulting in the schedule shown in the lower part.

overlapping execution, the scheduler searches for the smallest $k \in \{1, \ldots, |\mathbf{T}| - 1\}$ for which

$$\left[t_{\sigma(k)}, t_{\sigma(k)} + c_{\sigma(k)}\right] \cap [t_i, t_i + c_i] \neq \emptyset. \tag{14}$$

If no such k exists, the candidate start time t_i is declared *schedulable*, because the execution of τ_i can be scheduled at time t_i without any modification of the schedule of $\mathbf{T} \setminus \{\tau_i\}$. If on the other hand an overlap is found, we modify the schedule as follows (note that the schedule of the task set $\{\tau_{\sigma(1)}, \ldots, \tau_{\sigma_{k-1}}\}$ remains unchanged). We first compute the minimum amount of time

$$\Delta = t_i + c_i - t_{\sigma(k)} \tag{15}$$

to shift the execution of $\tau_{\sigma(k)}$ forward. The new start time of $\tau_{\sigma(k)}$ is thus

$$t'_{\sigma(k)} = t_{\sigma(k)} + \Delta. \tag{16}$$

This modification can introduce new overlapping executions or change the order of the schedule. To avoid this situation, we consider the successive executions $\tau_{\sigma(k+1)}, \ldots, \tau_{\sigma(|\mathbf{T}|-1)}$ by iteratively computing a new start time $t'_{\sigma(q)}$ for task $\tau_{\sigma(q)}$ according to

$$t'_{\sigma(q)} = \max\left(t_{\sigma(q)}, t'_{\sigma(q-1)} + c_{\sigma(q-1)}\right), \tag{17}$$

where q ranges from $k+1$ to $|\mathbf{T}| - 1$ in increasing order. Note that the iteration can be stopped at the first q for which $t_{\sigma(q)} = t'_{\sigma(q)}$. The candidate start time t_i is declared to be schedulable if, after the updates in Equations 16 and 17, $t'_{\sigma(q)} + c_{\sigma(q)} \leqslant d_{\sigma(q)}$ for each $q \in \{k, \ldots, |\mathbf{T}| - 1\}$. We denote the set of schedulable candidate start times with Ξ'_i.

Let us with Figure 7 discuss how Equations 16 and 17 are used to schedule a task τ_1 for a given candidate start time t_1. The scheduling is performed at time ϕ_1 at which the execution of the tasks $\tau_2, \ldots, \tau_6$ are already scheduled. The upper chart in the figure shows that the candidate start time t_1 is in conflict with the scheduled execution of τ_4. In the lower chart, it is shown that the scheduler has used Equation 16 to move τ_4 forward by Δ (indicated in the figure and computed with Equation 15 to $\Delta = t_4 + c_4 - t_1$). Tasks τ_3 and τ_5 are moved iteratively according to Equation 17 by an amount less than or equal to Δ. Task τ_2 is not affected because the change in execution of τ_4 does not introduce an execution overlap with τ_2.

If $\Xi'_i \neq \emptyset$, for each schedulable candidate start time $t_i \in \Xi'_i$, we shall associate a cost $\Psi_i(t_i)$, representing the overall cost (Equation 9) of scheduling τ_i at time t_i and possibly moving other tasks according to Equations 16 and 17. This cost is defined as

$$\Psi_i(t_i) = \sum_{q=1}^{k-1} \widehat{J}_{\sigma(q)}(t_{\sigma(q)}) + \widehat{J}_i(t_i) + \sum_{q=k}^{|\mathbf{T}|-1} \widehat{J}_{\sigma(q)}(t'_{\sigma(q)}),$$

where the notation and new start times $t'_{\sigma(q)}$ are the same as our discussion around Equations 16 and 17. The cost $\widehat{J}_{\sigma(q)}(t'_{\sigma(q)})$ can be computed efficiently, because the scheduler has, at a previous scheduling point, already performed the optimizations in Section VI-B1 for each task $\tau_{\sigma(q)} \in \mathbf{T} \setminus \{\tau_i\}$. The final solution chosen by the scheduler is the best schedulable candidate start time in terms of the cost $\Psi_i(t_i)$. The scheduler thus assigns the start time t_i of task τ_i as

$$t_i \longleftarrow \underset{t \in \Xi'_i}{\arg\min}\, \Psi_i(t).$$

If an overlapping execution exists, its start time and the start times of its subsequent executions are updated according to Equations 16 and 17. In that case, the update

$$t_{\sigma(q)} \longleftarrow t'_{\sigma(q)}$$

is made iteratively from $q = k$ to $q = |\mathbf{T}| - 1$, where $\tau_{\sigma(k)}$ is the first overlapping execution according to Equation 14 and $t'_{\sigma(q)}$ is given by Equations 16 and 17. If $\Xi'_i = \emptyset$, none of the candidate start times in Ξ_i could be scheduled such that all tasks meet their imposed deadlines. In such cases, the scheduler guarantees to find a schedulable solution according to the procedure described in the following subsection.

3) Stable scheduling: The scheduling and optimization step can fail to find a valid schedule for the task set $\mathbf{T}$. In such cases, in order to ensure stable control, the scheduler must find a schedule that meets the imposed deadlines, without considering any optimization of control performance and resource usage. Thus, the scheduler is allowed in such critical situations to use the full computation capacity in order to meet the stability requirement. Let us describe how to construct such a schedule at an arbitrary scheduling point.

We shall consider a schedule given for $\mathbf{T} \setminus \{\tau_i\}$ as described in Section VI-B. We thus have a bijection $\sigma : \{1, \ldots, |\mathbf{T}| - 1\} \longrightarrow \mathcal{I}_{\mathbf{T}} \setminus \{i\}$. Since the start time of a task cannot be smaller than the completion time of its preceding task in the schedule (Equation 4), we have

$$t_{\sigma(k)} \geqslant \phi_i + \sum_{q=1}^{k} c_{\sigma(q)}.$$

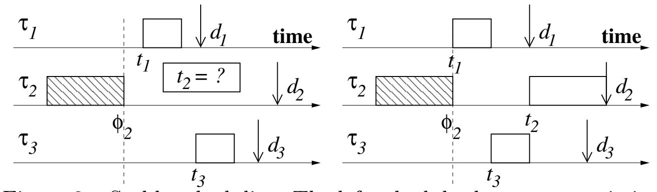

Figure 8. Stable scheduling. The left schedule shows a scenario in which, at time ϕ_2, the scheduler must accommodate CPU time to the next execution of τ_2. In the right schedule, CPU time for this execution is accommodated by moving the scheduled executions of τ_1 and τ_3 to earlier start times.

This sum models the cumulative worst-case execution time of the $k-1$ executions that precede task $\tau_{\sigma(k)}$. Note that the deadline constraints (Equation 3) for the task set $\mathbf{T} \setminus \{\tau_i\}$ are satisfied, considering the given start times. Important also to highlight is that the deadline of a task $\tau_j \in \mathbf{T} \setminus \{\tau_i\}$ is not violated by scheduling its execution earlier than the assigned start time t_j. To accommodate the execution of τ_i, we shall thus change the existing start times for the task set $\mathbf{T} \setminus \{\tau_i\}$ as

$$t_{\sigma(k)} \longleftarrow \phi_i + \sum_{q=1}^{k} c_{\sigma(q)}. \tag{18}$$

The start time t_i of task τ_i is assigned as

$$t_i \longleftarrow \phi_i + \sum_{q=1}^{|\mathbf{T}|-1} c_{\sigma(q)}. \tag{19}$$

This completes the schedule for $\mathbf{T}$. With this assignment of start times, the worst-case completion time of τ_i is

$$t_i + c_i = \phi_i + \sum_{q=1}^{|\mathbf{T}|-1} c_{\sigma(q)} + c_i = \phi_i + \sum_{j \in \mathcal{I}_\mathbf{T}} c_j,$$

which, if Equation 11 holds, is smaller than or equal to any possible deadline d_i for τ_i, since

$$t_i + c_i = \phi_i + \sum_{j \in \mathcal{I}_\mathbf{T}} c_j \leqslant \phi_i + D_i^{\min} \leqslant d_i.$$

With Equations 18 and 19, and provided that Equation 11 holds (to be verified at design time), the scheduler can with the described procedure meet all task deadlines in any execution scenario.

Let us consider Figure 8 to illustrate the scheduling policy given by Equations 18 and 19. In the left schedule, task τ_2 completes its execution at time ϕ_2 and the scheduler must find a placement of the next execution of τ_2 such that it completes before its imposed deadline d_2. Tasks τ_1 and τ_3 are already scheduled to execute at times t_1 and t_3, respectively, such that the deadlines d_1 and d_3 are met. In the right schedule, it is shown that the executions of τ_1 and τ_3 are moved towards earlier start times (Equation 18) to accommodate the execution of τ_2. Since the deadlines already have been met by the start times in the left schedule, this change in start times t_1 and t_3 does not violate the imposed deadlines of τ_1 and τ_3, since the order of the two tasks is preserved. Task τ_2 is then scheduled immediately after τ_3 (Equation 19) and its deadline is met, provided that Equation 11 holds.

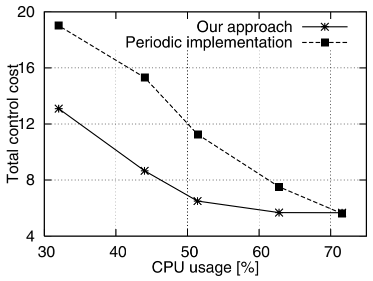

Figure 9. Scaling of the control cost. Our approach is compared to a periodic implementation for different CPU-usage levels. Periodic control uses more CPU bandwidth to achieve the same level of control performance as our approach with reduced CPU usage.

VII. Experimental Results

We have evaluated our proposed runtime scheduling heuristic with simulations of 50 benchmark systems comprising 2 to 5 control tasks that control unstable plants with given initial conditions of the state equations in Equation 1. We have run experiments for several values of the design constant ρ in Equation 8 (the trade-off between control quality and CPU usage) in order to obtain simulations with different amounts of CPU usage. For each simulation, we computed the total control cost (compare to Equation 5) of the entire task set $\mathbf{T}$ as

$$J^{\mathrm{c,sim}} = \sum_{j \in \mathcal{I}_\mathbf{T}} \int_0^{t^{\mathrm{sim}}} \boldsymbol{x}_j^\intercal(t) Q_j \boldsymbol{x}_j(t) dt, \tag{20}$$

where t^{sim} is the amount of simulated time. This cost indicates the control performance during the whole simulated time interval (i.e., smaller values of $J^{\mathrm{c,sim}}$ indicate higher control performance). For each experiment, we recorded the amount of CPU usage of all control tasks, including the time overhead of the scheduling heuristic. The baseline of comparison is a periodic implementation in which the periods were chosen to achieve the measured CPU usage. For this periodic implementation, we computed the corresponding total control cost $J_{\mathrm{per}}^{\mathrm{c,sim}}$ in Equation 20.

Figure 9 shows on the vertical axis the total control costs $J^{\mathrm{c,sim}}$ and $J_{\mathrm{per}}^{\mathrm{c,sim}}$ for our runtime scheduling approach and a periodic implementation, respectively. On the horizontal axis, we show the corresponding CPU usage. The main message conveyed by the results in Figure 9 is that the self-triggered implementation with our proposed scheduling approach can achieve a smaller total control cost (i.e., better control performance) compared to a periodic implementation that uses the same amount of CPU. The designer can tune the CPU usage of the control tasks within a wide range (30 to 60 percent of CPU usage) and achieve better control performance with the proposed scheduling approach, compared to its periodic counterpart. For example, when the CPU usage is 44 percent, the total control costs of our approach and a periodic implementation are 8.7 and 15.3, respectively (in this case, our approach improves on the control performance by 43 percent relative to the periodic implementation). The average cost reduction of our approach, relative to the periodic implementation, is $(J_{\text{per}}^{\text{c,sim}} - J^{\text{c,sim}})/J_{\text{per}}^{\text{c,sim}} = 41$ percent for the experiments with 30 to 60 percent of CPU usage. Note that for very large CPU-usage levels in Figure 9, a periodic implementation samples and actuates the controlled plants very often, which in turns leads to similar control performance as a self-triggered implementation.

The time overhead has been included in the simulations by scaling the measured execution time of the scheduling heuristic relative to the execution times of the control tasks. The main parameter that decides the time overhead of the scheduler is the number of iterations to be implemented by the golden-section search in Section VI-B1. We have found empirically that a relatively small number of iterations are sufficient to achieve good results in terms of our two optimization objectives (our experiments have been conducted with four iterations in the golden-section search). The number of iterations further decides the number of candidate solutions to consider in the scheduling step (Section VI-B2). The results presented in this section show that the proposed solution, including its runtime overhead, outperforms a periodic solution in terms of control performance and CPU usage.

VIII. Conclusions

We presented a framework for dynamic scheduling of multiple control tasks on uniprocessor platforms. The self-triggered control tasks compute their CPU needs at runtime and are scheduled to maximize control performance and minimize resource usage. Our results show that high control performance can be achieved with reduced CPU usage.

References

[1] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems*, 3rd ed. Prentice Hall, 1997.

[2] 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*, vol. 23, no. 3, pp. 16–30, 2003.

[3] 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.

[4] A. Cervin, J. Eker, B. Bernhardsson, and K. E. Årzén, "Feedback–feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1–2, pp. 25–53, 2002.

[5] 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*, 2009, pp. 864–869.

[6] 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*, vol. 51, no. 6, pp. 1159–1167, 2004.

[7] G. Buttazzo, M. Velasco, and P. Martí, "Quality-of-control management in overloaded real-time systems," *IEEE Transactions on Computers*, vol. 56, no. 2, pp. 253–266, February 2007.

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

[9] 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*, vol. J, 1999, pp. 301–306.

[10] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[11] T. Henningsson, E. Johannesson, and A. Cervin, "Sporadic event-based control of first-order linear stochastic systems," *Automatica*, vol. 44, no. 11, pp. 2890–2895, 2008.

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

[13] 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*, vol. 81, no. 4, pp. 571–590, 2008.

[14] M. Velasco, J. Fuertes, and P. Marti, "The self triggered task model for real-time control systems," in *Proceedings of the 23rd Real-Time Systems Symposium, Work-in-Progress Track*, 2003.

[15] M. Velasco, P. Martí, and E. Bini, "Control-driven tasks: Modeling and analysis," in *Proceedings of the 29th IEEE Real-Time Systems Symposium*, 2008, pp. 280–290.

[16] A. Anta and P. Tabuada, "Self-triggered stabilization of homogeneous control systems," in *Proceedings of the American Control Conference*, 2008, pp. 4129–4134.

[17] X. Wang and M. Lemmon, "Self-triggered feedback control systems with finite-gain L2 stability," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 452–467, 2009.

[18] 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*, 2009, pp. 928–933.

[19] 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*, 2009, pp. 3–12.

[20] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. Cambridge University Press, 1999.