This ia a preliminary version of a paper that is due to appear in the 29th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2023

Safety-Aware Implementation of Control Tasks via Scheduling with Period Boosting and Compressing

Shengjie Xu*, Bineet Ghosh*, Clara Hobbs*, P. S. Thiagarajan*†, Prachi Joshi‡, Samarjit Chakraborty*
*The University of North Carolina at Chapel Hill, USA
†Chennai Mathematical Institute, India ‡General Motors, USA

***Abstract*—A crucial requirement for control tasks in safety-critical systems like automotive is that *all* deadlines be met. This is becoming increasingly difficult when several tasks share common resources. One main reason for this lies in obtaining *tight* WCET estimations, especially as software and processor architectures continue to become more complex. Using safe but not necessarily *tight* WCET estimates and meeting all deadlines come at the expense of very pessimistic and inefficient implementations. In this paper, we show that by focusing on "higher-level" properties like control safety, instead of trying to meet all deadlines, it is possible to achieve more efficient implementations of control tasks on shared resources. This has considerable benefits in cost-sensitive domains like automotive. The core of our technique follows the AUTOSAR paradigm where groups of control computations with the same period constitute units of scheduling. Towards this, we suitably increase (boost) or decrease (compress) the sampling periods of control tasks and schedule them in a manner that is cognizant of their high-level safety constraints, but does not necessarily meet all deadlines. Our results for several standard controllers from the automotive domain illustrate the benefits of our approach.**

***Index Terms*—Real-time and embedded systems, Dynamic systems and control, safety**

I. Introduction

Proposed scheme: We propose a new technique for maximizing the number of control tasks that may be "packed" on a processor or resource. Our main novelty is a scheduling technique that satisfies "system-level" properties like control safety, described later in the paper, instead of aiming to meet *all* task deadlines. This shift in focus buys considerable implementation efficiency, which has not been explored in the past in the manner that we do. Figure 1 provides an overview of our approach. Given (1) a set of control tasks $T_1, T_2, \ldots$ with distinct sampling periods $P_1, P_2, \ldots$, our first step is to determine a *common* sampling period P^C by suitably increasing (boosting) or reducing (compressing) each period $P_1, P_2, \ldots$. This results in a new task set (2) that is then scheduled in a time-triggered manner on a resource where time is partitioned into slots of the chosen period size P^C. Such a schedule (3) only allows a subset of tasks from the set $T_1, T_2, \ldots$ to be executed in each slot. The ones not scheduled *miss* their deadlines. In the example shown in Figure 1, the schedule for the task T_1 is $110110\ldots$, that of T_2 is $010010\ldots$, T_3 is $101101\ldots$, and finally, that of T_4 is $001001\ldots$, where a 1 denotes the deadline being met and a 0 a deadline miss. Here, each slot is only large enough to execute at most two of the four tasks. Although there are several deadline misses, the schedule is derived in a manner that system-level properties of relevance, like the safety of the physical system being controlled, is not violated. If *all* the deadlines were to be met, then the full task set (T_1 to T_4) would not have been implementable on this single resource and more processors would be necessary. Given pre-designed controllers (1) and a safety property associated with each task T_i, how to obtain (3) while guaranteeing all the safety properties, is the main technical contribution of this paper. While there is a large volume of literature on scheduling safety-critical tasks to meet safety properties [1], [2], they rather attempt to *meet all* deadlines and thereby ensure safety, rather than focus on safety with deadline misses.

This work was supported by the NSF grant #2038960.

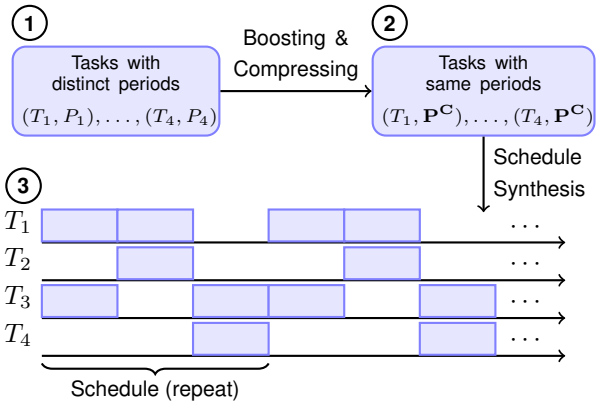

Fig. 1. Proposed implementation scheme for control tasks $T_1, \ldots, T_4$.

Background and motivation: Efficient implementation of software is a key to success in many cost-sensitive domains like automotive. Today, a modern car has several hundred million lines of software code implemented on different *electronic control units* (ECUs). The core functionality implemented by such code consists of various feedback control loops, *e.g.*, engine control, brake control, cruise control, motor control, and suspension and vibration control. Here, the traditional implementation workflow consists of the control strategy being designed first, followed by implementing it as a software task, that is scheduled to meet the deadline determined during the controller design phase. This ensures a separation of concerns that enables control theorists and embedded systems engineers to only communicate via the *deadlines* that needed to be met.

However, meeting *all* deadlines—which is assumed in the above workflow—is turning out to be increasingly challenging. With growing software and processor architecture

complexity, estimating safe *and* tight *worst case execution time* (WCET) estimates of software tasks is becoming a losing proposition [3]. For WCET estimates to be safe, they are increasingly overestimated. Meeting all task deadlines with such overestimated WCET values leads to pessimistic or infeasible implementations. Further, automotive in-vehicle architectures are rapidly moving away from "one function per ECU" or federated, to multiple functions sharing resources, *viz.*, "integrated" architectures [4]. The clear trend is that future architectures will be less "static" than before, as indicated by developments like AUTOSAR Adaptive [5] and service-oriented paradigms [6]. Such trends necessitate decoupling the software from the underlying hardware architecture, to attain flexibility and ease of task migration across architectures. However, "architecture-independent" WCET estimates exacerbate the pessimism even further. As a result, all downstream scheduling techniques that rely on *safe* WCET estimates of tasks are becoming too pessimistic to remain useful in practice.

Deadlines are only a means to an end: This paper, therefore, asks the question—can implementation pessimism be reduced by not having to meet *all* deadlines? In other words, can the focus be shifted to satisfying properties of consequence, instead of deadlines? In the context of our problem, they are "safety" properties defined on the physical system being controlled by the software tasks to be implemented and scheduled. We define them as follows: Given a plant and a suitable controller, let τ_{nom} be a trajectory in the state space of the closed-loop system (plant + controller) when the control task meets *all* deadlines. This is referred to as the *ideal* or *nominal* behavior. Any other trajectory τ is referred to as "safe" if it is at most a specified d^{safe} distance away from τ_{nom}, under a suitably defined distance metric. This is mathematically defined later in the paper. The intuition here is that occasional deadline misses will result in a different but acceptable state space trajectory, as long as its deviation is not too much from the ideal one. As outlined at the start of this section, how to schedule multiple control tasks with different sampling periods and safety properties, in order to reduce implementation pessimism, is the key technical contribution of this paper.

For simplicity of exposition, we consider implementations on a single processor only. But the general scheme derived here can be extended to cases where control tasks are partitioned and implemented on multiple computation and communication resources. Our proposed scheme is also compliant with OSEK and AUTOSAR, where *runnables* or tasks with the same period are grouped together for scheduling [7], [8].

Organization: The rest of this paper is organized as follows: in Section II, we review the existing literature relevant to our work. In Section III, we introduce the necessary mathematical preliminaries and the models used in this work. We formally state the problem in Section IV, followed by our proposed approach in Section V. Numerical results showing the benefits of this approach are presented in Section VII. Finally, Section VIII concludes the paper followed by an outline of possible extensions.

II. Related work

How to implement multiple control applications on resource-constrained embedded systems is being actively studied over the past couple of years [9]. These studies have been motivated by the increasing volumes of software in domains like automotive [10], where the algorithmic core of the software is a feedback control algorithm. Exploiting the robustness of such control algorithms to adaptively allocate resources, *e.g.,* by switching between time-triggered and event-triggered communication, has been studied in the past [11]. Whether to schedule a control task or not (in which case it misses its deadline) can be considered to be a form of resource allocation in a similar vein.

There is indeed a considerable volume of literature on *checking* where control safety properties (including stability) are satisfied under a given deadline hit/miss pattern. Some of the representative recent literature on this include [12]–[16]. Work in this domain stems from the difficulty in timing analysis of control software [17], which is attributed to both – the challenges in WCET analysis and the modeling of the code structure [18]. But there has been much less work on how to *synthesize* task schedules to meet control safety properties, and especially properties that are more general than stability, as we do in this paper. The work in [19] investigated scheduling with safety constraints, but the applicability of their methods is limited to controllers with the same period. Our work is also closely related to scheduling using *weakly-hard constraints* (that specify deadline hit/miss patterns) [20] and its applications [14], [21], [22], as we discuss in the next section. It is worth noting that while the focus of this paper is on scheduling, *viz.,* determining when a control task meets its deadline and when not, the underlying principle of *focussing in the system-level property* and not focusing on "secondary" properties like timing behavior is applicable more generally. For example, when messages are not fully encrypted or authenticated for security [23], [24], it might be shown that a safety property of the form studied in this paper cannot be violated even if the system is under attack. Similar results may also be established in the case of ensuring system reliability [25].

III. System Modelling

In this section, we introduce the necessary background on system modeling that will be utilized throughout the remainder of the work. We first wish to lay out the fundamentals of modeling control systems, then move on to the discussion about the characterization of deadline hit/miss patterns and how they affect system behavior such as control safety.

A. The State-Space Model

Control systems are dynamic in nature and are often described using differential equations. One common representation of control systems is the state-space model, where the *state* of the system is represented by a state vector $x(t) = [x_1(t)\ x_2(t)\ \dots\ x_n(t)]^T$ and the *input* to the system by $u(t) = [u_1(t)\ u_2(t)\ \dots\ u_p(t)]^T$. Using these notations,

the state-space model of a continuous linear time-invariant system is given by

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{n\times p}$. Eq. (1) shows that the rate of change of the system state ($\dot{x}(t)$) depends both on the current state ($x(t)$) and the control input ($u(t)$). When implemented on a processor, the state-space model is discretized, with a discretization period of P, and assumes the form of

$$x[t+1] = A_d x[t] + B_d u[t]. \tag{2}$$

In this work, we assume that the system model is closed-loop, where the measurement of the current state is used to determine the control input of the next actuation. In practice, the control input u is computed by a periodic real-time task running on a processor and is assumed to be of the form

$$u[t] = Kx[t-1], \tag{3}$$

where $K \in \mathbb{R}^{p\times n}$ is the *feedback gain*. We follow the logical execution time (LET) paradigm, where the deadline equals the sampling period. A new control input is always applied at the deadline of the control job, *i.e.,* the system state is sampled at time $t-1$ and used to compute the control input for time t, where the state and control input are computed according to Eqs. (2) and (3).

B. Characterizing Deadline Hit/Miss Patterns

The feedback gain matrix K in Eq. (3) is designed in tandem with the discretization period P of the system, and the correct behavior of the control system relies on the timely completion of the computation of the control input u by the end of each period. If the periodic real-time task computing the control input misses its deadline, then control performance suffers, and the system may deviate from its ideal behavior and become unsafe. However, not all deadline misses have the same impact, and many control systems can tolerate a certain amount of deadline misses before the system is considered unsafe. Many works have studied the characterization of deadline misses and how they impact control performance. Notably, the work in [20] proposes a systematic method for characterizing deadline hit/miss patterns. These so-called weakly-hard constraints—especially the $\overline{\binom{m}{k}}$ model, which demands that in any k consecutive invocations of a task, there can be *at most* m deadline misses—have been studied in a number of settings, including schedulability analysis, formal verification, and runtime monitoring, with [14], [21], [22] being some recent examples. In this work, we focus on constraints of the type $\binom{m}{k}$ [20] which states that there are *at least* m deadline hits in any k consecutive invocations of the task. This is equivalent to the constraint $\overline{\binom{k-m}{k}}$. Finally, as outlined in Section I, if we represent the hit/miss patterns using a bit string, where 0 represents a deadline miss and 1 a deadline hit, then all hit/miss patterns that comply with the constraint $\binom{m}{k}$ will be a *regular* language over the alphabet $\{0,1\}$ [26]. We shall denote this language as $\mathcal{L}_{(m,k)}$ and exploit the properties of regular languages to synthesize task schedules.

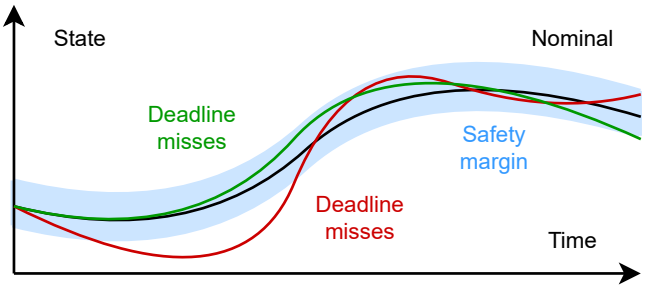

Fig. 2. An example of deviations as a result of deadline misses.

C. System Behavior under Deadline Misses

We now characterize the effects of deadline misses on system behavior. Suppose $x[t]$ is the plant state and $u[t]$ is the control input at time t. When $x[t-1]$ is read, a software job corresponding to the control task is released to compute $u[t]$, which is then applied to the physical plant at time t if the job completes within its deadline. If the job is not scheduled (see Figure 1) then no new control input is computed and the previous control input continues to hold.

We consider the behavior of the plant only over a finite time horizon H. Thus the states of the plant will be recorded at time points $0, 1, \ldots, H$. For ease of exposition, we also assume that the initial state of the system is $z[0] \in \mathbb{R}^n$. With the initial state $z[0]$, we define the *nominal trajectory* of the plant as the trajectory resulting from no deadline misses, denoted as τ_{nom}. Formally, it is the sequence of states of length $H+1$ of the form $x[0], x[1] \ldots, x[H]$ with $x[0] = z[0]$, where $x[t+1]$ is computed with Eq. (2) and $u[t]$ is computed with Eq. (3). As an example, the black-colored trajectory in Fig. 2 is the nominal trajectory. We next wish to define the set of trajectories that do not deviate from the nominal trajectory τ_{nom} by more than a safety bound d^{safe}, shown as the light blue envelope in Fig. 2. Let $\mathcal{T} = \{\tau\}$ be the set of sequences of length $H+1$ over $\mathbb{R}^n$ where $\tau = (\tau[0], \tau[1] \ldots, \tau[H])$ with $\tau[i] = x[i]$, $x[0] = z[0]$ and $i = 0, \ldots, H$. Intuitively, $\mathcal{T}$ denotes the set of *all* possible trajectories of length $H+1$ in the state space that starts from $z[0]$, including ones where some task deadlines are missed. Clearly, the nominal trajectory—where all deadlines are met—is also a member of $\mathcal{T}$.

To quantify deviations from the nominal trajectory, we first define the distance between two points in $\mathbb{R}^n$ using a distance metric $\mathrm{dis}(\cdot)$. This can be any metric such as the Euclidean distance. We then define the distance between a pair of trajectories (τ, τ'), also denoted as $\mathrm{dis}(\cdot)$, given by:

$$\mathrm{dis}(\tau, \tau') = \max_{0\le t\le H} \mathrm{dis}(\tau[t], \tau'[t]). \tag{4}$$

We now fix a safety margin $d^{safe} > 0$. This leads to the set of safe trajectories $\mathcal{T}_{safe} \subset \mathcal{T}$, defined as

$$\mathcal{T}_{safe} = \{\tau \mid \mathrm{dis}(\tau, \tau_{nom}) \le d^{safe}\}. \tag{5}$$

Intuitively, this is the set of trajectories that do not exceed the safety margin around the nominal trajectory, *i.e.*, trajectories

that do not deviate more than d^{safe} from the nominal trajectory. For example, the green trajectory in Fig. 2 is a member of $\mathcal{T}_{safe}$, while the red one is not. Clearly, the nominal trajectory is also a member of $\mathcal{T}_{safe}$.

Finally, suppose $\gamma \in \{0,1\}^H$ is a sequence of length H representing a pattern of deadline hits and misses. Then starting from $z[0]$ we can compute the sequence of plant states with Eq. (2) and control inputs with Eq. (3) if $\gamma[t] = 1$, or hold the previous control input if $\gamma[t] = 0$. We denote the resulting plant trajectory as τ_γ. This leads to

$$\mathcal{T}_{(m,k)} = \{\tau_\gamma \mid \gamma \in \mathcal{L}_{(m,k)}\}. \tag{6}$$

In other words, $\mathcal{T}_{(m,k)}$ is the set of all state space trajectories resulting from deadline hit/miss patterns in the regular language $\mathcal{L}_{(m,k)}$. We call the plant *safe* under $\binom{m}{k}$ if and only if $\mathcal{T}_{(m,k)} \subseteq \mathcal{T}_{safe}$.

IV. Problem Statement

As we have established in Section III, we are interested in leveraging the relationship between deadline misses and control performance to produce a safe schedule for all systems. However, it is difficult to relate the deadline miss patterns of different control tasks if they are running at *distinct* sampling periods. In contrast, if control tasks were to run at the same sampling period, then the resource contention problem would be identical within each period and become much more tractable. We wish to study the interactions between changing periods and scheduling while still focusing on the properties of consequences, *viz.,* control safety. Towards this, we formulate the problem as follows.

Problem 1 (Safe Schedule Synthesis): Given a set of controller tasks $\mathbb{T}$ of distinct periods, each with parameters WCET (C_i), period (P_i), and safety margin (d_i^{safe}), determine if a schedule can be obtained by increasing (boosting) or reducing (compressing) the sampling periods of the control tasks while still maintaining their respective safety properties. Synthesize one if such schedules exist.

Note that we are assuming the sampling periods of control tasks are *reasonably* close to each other (*e.g.* within the same order of magnitude), as large changes in sampling periods are often accompanied by undesirable shifts in system behavior.

V. Proposed Approach

We propose a two-stage solution to Problem 1 as follows:

1) Determine suitable common sampling periods (scheduling slot size) P^C for all control tasks.
2) Synthesize a time-triggered schedule on the computation resource where time is partitioned into slots of size P^C.

The overview of our approach is outlined in Fig. 3. While converting all control tasks to the same sampling periods makes the safety analysis and scheduling problem much simpler, modifying sampling periods (either by boosting or compressing) changes the dynamics of the closed-loop system. This is further complicated by the fact that the tasks are implemented on a resource-constrained platform, and cannot be scheduled to meet all deadlines. This leads to some unintuitive

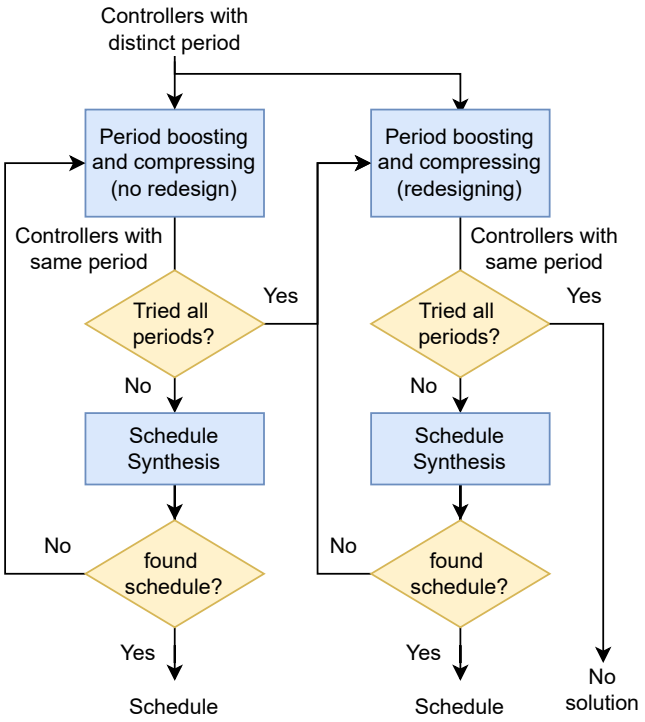

Fig. 3. An overview of the proposed period compression/boosting and schedule synthesis scheme.

effects on the schedulability of the system. For example, while a shorter sampling period generally enhances—or at least does not deteriorate—the control performance of a single system, it also means that fewer tasks can fit in that shorter period and will miss more deadlines. Therefore, the control period must be carefully determined to trade off the control performance of the systems with the number of tasks that fit within the period (*i.e.*, not causing too many deadline misses). Despite the complications, we argue that when done carefully, the combined effect of the two can be used to produce a safe and more efficient implementation of control tasks. We now present the details of period compression and boosting and schedule synthesis in the next two subsections.

A. Period Compression and Boosting

Choosing a common period for all control tasks is a delicate issue. We proceed by considering the WCETs of each control task. Assume we are given a set of controller tasks. For each task T_i, its WCET, period, and safety margin are denoted by C_i, P_i, and d_i^{safe}, respectively. We assume that the task set is not schedulable on the shared processor when no deadline misses are allowed (*e.g.,* the utilization $U = \sum_i C_i/P_i > 1$), since otherwise, a standard scheduling algorithm such as earliest deadline first (EDF) will suffice. We first sort the execution times from large to small, and define potential common periods or slot sizes for scheduling (see Figure 1) as follows:

$$P_k^C = \sum_{i \le k} C_k. \tag{7}$$

We choose these values to ensure that when we use P_k^C as the common time slot size, any k controller tasks can be executed within it. This is because it is large enough to fit the sum of the k largest WCET values. As an example, for the set of five controller tasks in Table III, the possible P_k^C values would be $15\,\text{ms}$, $28\,\text{ms}$, $40\,\text{ms}$, $50\,\text{ms}$, and $60\,\text{ms}$. As discussed earlier, it is important to note that the chosen P_k^C will determine how much a control task is over- or under-sampled compared to its original design, and this by itself might violate a task's safety property. Missing deadlines on top of it, by not scheduling the task at every slot, *might* only further deteriorate the violation and not correct it. But depending on the underlying dynamics of the closed-loop system, certain patterns of deadline misses might also correct a violation caused by over- or under-sampling, causing the *combination* of the two steps to be important and interesting.

Recomputing controller gains: Finally, we have an additional design dimension: whether or not the controller gain values are recomputed based on the chosen common sampling period [27]. This is because gain values designed for the originally-intended sampling period might not work well for a new period. However, it may not always be realistic to redesign a controller for each implementation architecture, especially when controller design and implementation phases are handled by different teams. We explore both possibilities in Section VII.

B. Schedule Synthesis

After the period compression and boosting process outlined in Section V-A, we now have a set of control tasks with the same period. The next step is to synthesize a safe schedule (if one exists) for the task set. We use the two-step synthesis scheme proposed in [19] to model the connection between deadline miss patterns and system behaviors, as discussed in Sections III-B and III-C. For completeness, this scheme is summarized below:

1) *For each controller*, we find a set of weakly-hard constraints $\binom{m}{k}$ whose resulting trajectories $\mathcal{T}_{(m,k)}$ are safe.
2) We use these sets of constraints and properties of regular languages to synthesize a schedule satisfying the constraints *for all controllers*.

1) Safe constraints: Given a control task T_i and its safety margin d_i^{safe}, we wish to determine the set of weakly-hard constraints under which the system is safe. Checking safety under a particular constraint $\binom{m}{k}$ amounts to checking if $\mathcal{T}_{(m,k)} \subseteq \mathcal{T}_{safe}$, or whether $\mathbf{d}(m,k) \leq d_i^{safe}$. Here, $\mathbf{d}(m,k)$ is the maximum deviation of the trajectories in $\mathcal{T}_{(m,k)}$ from the nominal trajectory of T_i. More precisely,

$$\mathbf{d}(m,k) = \max\big\{\text{dis}(\tau,\tau_{nom}) \;\big|\; \tau \in \mathcal{T}_{(m,k)}\big\}. \tag{8}$$

However, checking this directly is expensive, due to the exponential number of hit/miss patterns of length H. Instead, we adapt the `BoundedRuns` algorithm proposed in [12] to compute an upper bound $\widehat{d}(m,k)$ on $\mathbf{d}(m,k)$. The `BoundedRuns` algorithm divides the total H periods into smaller subsequences, each containing t periods ($t \ll H$). Within each subsequence, the algorithm computes all 2^t potential hit/miss sequences and calculates the trajectory corresponding to each sequence. At the end of each subsequence, it computes a bounding box for the reachable states resulting from the 2^t trajectories and uses it as a single starting condition for the next subsequence. Although the output $\widehat{d}(m,k)$ overestimates $\mathbf{d}(m,k)$, the time complexity is significantly reduced from $O(2^H)$ to $O(2^t \frac{H}{t})$. It then suffices to check that $\widehat{d}(m,k) \leq d^{safe}$ to guarantee the safety of the system.

Equipped with the method to check a single constraint $\binom{m}{k}$, we move on to generate the set of safe constraints by iterating through all weakly-hard constraints $\binom{m}{k}$ up to a maximum window size k_{max}, a parameter fixed by the user. For each constraint $\binom{m}{k}$, we compute $\widehat{d}(m,k)$ using the `BoundedRuns` algorithm and compare it with d_i^{safe}. If $\widehat{d}(m,k) \leq d_i^{safe}$, we conclude that the system is safe under $\binom{m}{k}$ and add $\binom{m}{k}$ to the set of safe constraints. Otherwise, no conclusion can be drawn and we do not add it to the set. We additionally apply an optimization scheme that reduces the required number of iterations to generate the list based on the following observations:

$$m_1 \geq m_2 \implies \mathcal{L}_{(m_1,k)} \subseteq \mathcal{L}_{(m_2,k)} \tag{9}$$

$$k_1 \geq k_2 \implies \mathcal{L}_{(m,k_1)} \supseteq \mathcal{L}_{(m,k_2)} \tag{10}$$

These observations imply that

1) $\binom{m_1}{k}$ is safe if $\binom{m_2}{k}$ is safe and $m_1 \geq m_2$, and
2) $\binom{m}{k_1}$ has no safe guarantee if $\binom{m}{k_2}$ has no safe guarantee and $k_1 \geq k_2$

Therefore, we do not need to iterate over every $\binom{m}{k}$ constraint. Instead, we start with $m = 1$ and $k = 2$, increment k when $\widehat{d}(m,k) \leq d_i^{safe}$, and increment m when $\widehat{d}(m,k) > d_i^{safe}$. As a result, the total number of iterations is reduced from $O({k_{max}}^2)$ to $O(k_{max})$. It is important to note that this reduction in computation does not lead to a reduced solution space, as *all* safe contraints will be considered in the next step for schedule synthesis, regardless of the observations made in Eqs. (9) and (10).

2) Synthesizing safe schedules: As introduced in Section III-B, a weakly-hard constraint $\binom{m}{k}$ is a regular language $\mathcal{L}(m,k)$ over $\{0,1\}$, where a string represents a hit/miss pattern satisfying $\binom{m}{k}$. The set of safe constraints generated for controller T_i in the previous step can be represented as regular language as well, by taking the union of the regular languages representing each of the constraints. We use an automaton $\mathbb{A}_i = \langle L^i, \Sigma, T^i, L_f^i, \ell_0^i \rangle$ to represent the weakly-hard constraints for the control task T_i, where L^i is a set of locations (states), $\Sigma = \{0,1\}$ is the input alphabet, T^i is the transition function, L_f^i is the set of accepting locations, and ℓ_0^i is the initial location.

With this construction, an accepting run of $\mathbb{A}_i$ is a hit/miss pattern that satisfies at least one safe weakly-hard constraint for the corresponding controller task T_i. Corresponding to a task set with N control tasks, we will use the set of controller automata $\{\mathbb{A}_i \mid i \in 1,\ldots,N\}$ to construct a *scheduler automaton*, and use it to synthesize a schedule (of the form

shown in Figure 1, where $N = 4$). We consider a time horizon of H time slots and assume that J ($< N$) controller jobs can fit into one time slot. Thus, we shall construct the scheduler automaton as a product of the N controller automata, where accepting runs of length $H + 1$ of the scheduler automaton will constitute the set of safe schedules that we seek.

Example: Consider the following example, where two controller automata $\{\mathbb{A}_1, \mathbb{A}_2\}$ representing a set of weakly-hard constraints on control tasks T_1 and T_2 are given, for which we wish to synthesize a schedule such that only one task can be scheduled in each time slot. Assume that 011011 is an accepting run of $\mathbb{A}_1$, representing a pattern of deadline hit/miss that does not violate the safety constraint of T_1, up to a time horizon of 6. Similarly, assume 100100 is an accepting run of $\mathbb{A}_2$. Clearly, given the two accepting runs, a possible schedule can be given as vectors $\binom{0}{1}\binom{1}{0}\binom{1}{0}\binom{0}{1}\binom{1}{0}\binom{1}{0}$, where each vector denotes the set of tasks that should be scheduled at each time. Control task T_1 should be scheduled in the given time step if and only if there is a 1 in the first position of the vector. Similarly, T_2 should be scheduled if and only if there is a 1 in the second position of the vector—for instance, $\binom{0}{1}$ denotes that controller task T_2 should be scheduled at that time step. We next show how to derive such schedules from a set $\{\mathbb{A}_i\}$.

Definition 1: A **scheduler automaton** $\mathbb{A}^S$ for a set of N control tasks whose constraints are represented by the automata of the form $\mathbb{A}_i = \langle L^i, \Sigma, T^i, L^i_f, \ell^i_0 \rangle$, where at most J controllers can be scheduled in each time slot, is defined as an automaton $\langle L^S, \Sigma^S, T^S, L^S_f, \ell^S_0 \rangle$:

L^S set of locations, $L^S = \prod_i L^i$;
Σ^S input alphabet, $\Sigma^S \subset \{0,1\}^N$. A sequence $\sigma \in \{0,1\}^N$ is in Σ^S if and only if $\sum_i \sigma^i \leq J$;
T^S transition function, $T^S(\ell, \sigma) = \prod_i T^i(\ell^i, \sigma^i)$;
L^S_f accepting locations of the automaton, $L^S_f = \prod_i L^i_f$;
ℓ^S_0 initial location of the automaton, $\ell^S_0 = \prod_i \ell^i_0$.

The new set of locations L^S of the scheduler automaton is obtained by taking a Cartesian product of all the controller automata locations L^i. Similarly, the initial location and the accepting locations are Cartesian products of the individual controller automata's initial locations and accepting locations, respectively. The set of actions $\Sigma^S \subset \{0,1\}^N$ represents the legal actions to take at each time slot, *i.e.*, an action $\sigma \in \Sigma^s$ is valid if and only if $\sum_i \sigma^i \leq J$. The transition function of the scheduler automaton is a modified Cartesian product. Intuitively, it is emulating all the (safe) weakly-hard constraints of all controllers together. Therefore, all the transitions that lead to an accepting state are in the set of valid schedules.

With this construction of the scheduler automaton, an accepting run of the automaton represents a safe schedule of the control tasks. The existence of safe schedules can be checked by running an emptiness check on the scheduler automaton, and schedules can be generated using breadth-first search (BFS).

For simplicity of exposition, we have presented the main steps of our construction explicitly. However, this will not be the most efficient implementation and there is significant scope for optimization using standard formal verification tools [28]. For instance, one might just work with a set of weakly-hard constraints for each controller and begin to explore the state space of the scheduler automaton on the fly while inferring the state spaces of the controller automata (capturing the associated weakly-hard constraints) as needed. We will stop as soon as an accepting run of the scheduler automaton is found. Thus the full price of the construction will be incurred only when there is no feasible schedule.

System	WCET (C_i)	Period (P_i)	Safety Margin (d_i^{safe})
Task 1	10 ms	18 ms	2.5
Task 2	15 ms	20 ms	1.2

TABLE I
THE EXAMPLE TASK SET

Task	**Window Size** (k)	**Minimum Hits** (m) 1	2	3
Task 1	2	✓	—	—
	3	×	✓	—
	4	×	✓	✓
Task 2	2	✓	—	—
	3	✓	✓	—
	4	×	✓	✓

TABLE II
SAFE CONSTRAINTS FOR THE EXAMPLE SYSTEMS IN TABLE I WITH COMMON PERIOD $P^C = 15$ ms.

VI. Illustration of the Proposed Approach

In this section, we illustrate the methods proposed in Section V using a simple task set with two controller tasks (shown in Table I). Note that the utilization of this example test set is $U \approx 1.3 > 1$ and is not schedulable with conventional deadline-based methods. We will first find the suitable common periods for scheduling, and then verify them with schedule synthesis. We refer to Algorithm 1 as a structured guide for this section:

Line 1 The first step is to find suitable common periods so that the methods of schedule synthesis can be applied. Sorting the tasks by their WCET, we obtain the common periods $P^C_1 = 15$ ms, and $P^C_2 = 15 + 10 = 25$ ms. These are the slot sizes that fit one and two tasks, respectively.

Line 3 For each possible common period, we attempt to synthesize a safe schedule of the two systems. The first common period we use for schedule synthesis is $P^C = 15$ ms.

Line 4 Assume that the dynamic of system 1 is described by the following state-space model:

$$\dot{x_1}(t) = \begin{bmatrix} 5 & -2 \\ 0.7 & -1 \end{bmatrix} x(t) + \begin{bmatrix} 2 \\ 0.2 \end{bmatrix} u(t).$$

The system is initially discretized for its original period $P = 18$ ms. To apply the schedule synthesis

methods, we first re-discretize it using the new common period $P_1^C = 15\,\text{ms}$ and obtain the following discrete state-space model:

$$x_1[t] = \begin{bmatrix} 1.0777 & -0.0309 \\ 0.0108 & 0.9850 \end{bmatrix} x(t) + \begin{bmatrix} 0.0311 \\ 0.0031 \end{bmatrix} u(t).$$

We apply the same process to Task 2 and obtain the discretized state-space model for it as well.

Line 5 Assume the time horizon $H = 20$ and maximum window size $k_{max} = 4$. Running `scheduleSynthesis` returns the safe constraints shown in Table II, where a "✓" in position (m, k) means that the weakly-hard constraint $\binom{m}{k}$ is safe for that system, and an "×" indicates that the constraint is not known to be safe. Additionally, a safe schedule is synthesized, shown in Fig. 4. Intuitively, the scheduling strategy is to alternate between scheduling Task 1 and Task 2 in the given 15 ms slots, and it is easy to check that the deadline miss pattern of either task satisfies the $\binom{1}{2}$ constraint, a safe constraint for both systems.

Line 7 In this simple example, our algorithm is able to find a safe schedule without recomputing the gain values and exits. If this is not the case, the algorithm will proceed to the second half and repeat the above process, but with recomputed gains for each potential P^C. The algorithm will return `None` if it reaches the end of the second loop without finding any safe schedule.

Algorithm 1: Illustration of the proposed approach.

input : A set of controller tasks $\mathbb{T}$ with parameters $\{C_i, P_i, d_i^{safe}\}$ (WCET, period, safety margin)
output: A safe schedule with its period P^C; or `None` if no safe schedule can be found

```
ℙ ← getCommonPeriods({C_i}) ;
// Try scheduling without recomputing controller gains ;
for P_k^C ∈ ℙ do
    𝕋_d ← discretize(𝕋, P_k^C) ;
    s ← scheduleSynthesis(𝕋_d) ;
    if s is not None then
        return s
// Try scheduling with recomputing controller gains ;
for P_k^C ∈ ℙ do
    𝕋' ← recomputeGains(𝕋, P_k^C) ;
    𝕋_d ← discretize(𝕋', P_k^C) ;
    s ← scheduleSynthesis(𝕋_d, P_k^C) ;
    if s is not None then
        return s
return None
```

VII. Experimental Results

We implemented our techniques using *Julia* and evaluated them on five standard controllers from the automotive domain. In Section VII-A, we introduce these five controllers along with their parameters (such as WCETs and periods), and the required safety margins that they must satisfy. In Section VII-B, we give an overview of our approach and a summary of our findings. Section VII-C describes the details of the schedule synthesis process, and finally in Section VII-D, we discuss the insights gained from our results.

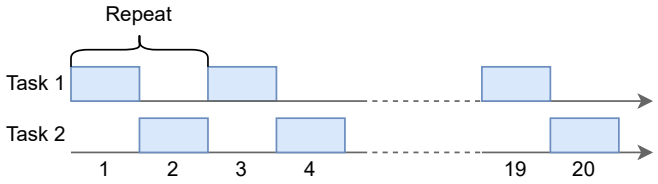

Fig. 4. Safe schedule for the example systems in Table I with common period $P^C = 15\,\text{ms}$.

A. Plant Models

Each of our five controllers is designed for a certain sampling period and WCET, and must be within the given safety margins—Table III provides these parameters.

1) RC Network (RC): Our first model is a resistor-capacitor network [29] with the following model:

$$\dot{x}(t) = \begin{bmatrix} -6.0 & 1.0 \\ 0.2 & -0.7 \end{bmatrix} x(t) + \begin{bmatrix} 5.0 \\ 0.5 \end{bmatrix} u(t).$$

2) F1Tenth Car (F1): Our second model is the linearized motion of an F1Tenth model car [30]:

$$\dot{x}(t) = \begin{bmatrix} 0 & 6.5 \\ 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 19.685 \end{bmatrix} u(t).$$

Our next three plant models are selected from [31] and also represent subsystems from the automotive domain.

3) DC Motor (DC): Our third model is the speed control for DC motor adapted from [32]:

$$\dot{x}(t) = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u(t).$$

4) Car Suspension (CS): Our fourth model is a suspension system adapted from [33]:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -8 & -4 & 8 & 4 \\ 0 & 0 & 0 & 1 \\ 80 & 40 & -160 & -60 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 80 \\ 20 \\ -1120 \end{bmatrix} u(t).$$

5) Cruise Control (CC): Our final model is a cruise control system adapted from [34]:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.0476 & -5.2856 & -0.238 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 2.4767 \end{bmatrix} u(t).$$

System	WCET	Period	Safety Margin
RC	10 ms	23 ms	5.8
F1	13 ms	20 ms	0.37
DC	12 ms	23 ms	0.18
CS	10 ms	27 ms	2.1
CC	15 ms	28 ms	0.48

TABLE III
PARAMETERS FOR THE CONTROLLERS DEFINED IN SECTION VII-A

Period (P^C)	Original gain	Recomputed gain
15 ms (1 task)	Not schedulable	Not schedulable
28 ms (2 tasks)	Schedulable	Schedulable
40 ms (3 tasks)	Not schedulable	Schedulable

TABLE IV
SUMMARY OF EXPERIMENTAL RESULTS

B. Experiment Overview

As Table III shows, the control tasks have different periods and their utilization $U(\approx 2.51)$ is far greater than 1. That is, these tasks are not schedulable on a single processor if no deadline misses are allowed. In this section, we attempt to derive a common period for the controllers and demonstrate that with our proposed schedule synthesis technique they can be scheduled on a single processor while satisfying their safety margins (while missing certain deadlines).

With the WCET values shown in Table III, we derive a number of suitable common periods and convert all the control tasks to one of the common periods so that our schedule synthesis technique may be applied. The common periods we explore are: $P_1^C = 15\,\text{ms}$ (where any one controller task can fit inside one scheduling slot), $P_2^C = 15 + 13 = 28\,\text{ms}$ (any two tasks fit), and $P_3^C = 15 + 13 + 12 = 40\,\text{ms}$ (any three tasks fit). $P_4^C = 50\,\text{ms}$ and $P_5^C = 60\,\text{ms}$ are too long for some systems to stay within their safety margins, and therefore are not discussed here. For each common period, we explore both—using the original and recomputed controller gain values as discussed in Section V-A. A summary of the final results is shown in Table IV. In particular, for the common period $P_1^C = 15\,\text{ms}$, no safe schedule can be obtained; for $P_2^C = 28\,\text{ms}$, a safe schedule can be obtained either with the original gain values or recomputed gain values using the new period; for $P_3^C = 40\,\text{ms}$ a safe schedule is only obtained when the control gain values are recomputed, *i.e.*, the controllers are recomputed. The results demonstrate that recomputing the controller gain values is not always required to obtain a safe schedule. We present the details of our schedule synthesis scheme in the next section.

C. Schedule Synthesis

For each common period, we first discretize all plant models using the selected common period. Using either the original or recomputed gain values, we apply our schedule synthesis technique to determine a safe schedule. We demonstrate this process with the common period $P_2^C = 28\,\text{ms}$. In this example, we use the original gain values and do not recompute them using this common period. The results of this schedule synthesis are presented below.

Model	**Window Size (k)**	**Minimum Hits (m)** 1	2	3	4	5
RC	2	✓	—	—	—	—
	3	×	✓	—	—	—
	4	×	×	✓	—	—
	5	×	×	×	✓	—
	6	×	×	×	×	✓
F1	2	✓	—	—	—	—
	3	×	✓	—	—	—
	4	×	×	✓	—	—
	5	×	×	×	✓	—
	6	×	×	×	×	✓
DC	2	✓	—	—	—	—
	3	✓	✓	—	—	—
	4	×	✓	✓	—	—
	5	×	×	✓	✓	—
	6	×	×	×	✓	✓
CS	2	✓	—	—	—	—
	3	✓	✓	—	—	—
	4	✓	✓	✓	—	—
	5	×	✓	✓	✓	—
	6	×	✓	✓	✓	✓
CC	2	✓	—	—	—	—
	3	✓	✓	—	—	—
	4	×	✓	✓	—	—
	5	×	✓	✓	✓	—
	6	×	×	✓	✓	✓

TABLE V
SYNTHESIZED CONSTRAINTS FOR $P^C = 28\,\text{ms}$, NO RECOMPUTING

Safe constraints: For each controller, we discretize the state-space model using the common period $P_2^C = 28\,\text{ms}$ and use our constraint synthesis technique outlined in Section V-B. The time horizon H is set to 100, and the maximum window size k_{max} is set to 6. The results are shown in Table V, where a "✓" in position (m, k) means that the weakly-hard constraint $\binom{m}{k}$ is safe for that system. *Viz.,* a hit/miss pattern satisfying the weakly-hard constraint $\binom{m}{k}$ would guarantee that the evolution of the plant stays within its safety margin. An "×" mark indicates that the constraint is not known to be safe.

Synthesize safe schedules: After we select the common period and compute the list of safe weakly-hard constraints for each controller, we attempt to schedule all the controller tasks using the automata-based approach outlined in Section V-B. With the common period $P_2^C = 28\,\text{ms}$, two controller tasks can fit within one period/slot. We are able to derive a schedule for the five controllers, where each controller satisfies its corresponding safety constraints despite missing some deadlines and some of them running with a longer period than what they were designed for.

As shown in Fig. 5, one example of a valid schedule is as follows: From $t = 1$ to $t = 9$, the tasks to be scheduled in each slot (Step(t)) are shown in the table. From $t = 10$ to $t = 100$, the schedule is repeated from $t = 4$ to $t = 9$, *viz.*, the schedule at time $t \in [10, 100]$ can be given as $\sigma[((t-10) \bmod 6) + 4]$. We note that this is not the *only* valid schedule for

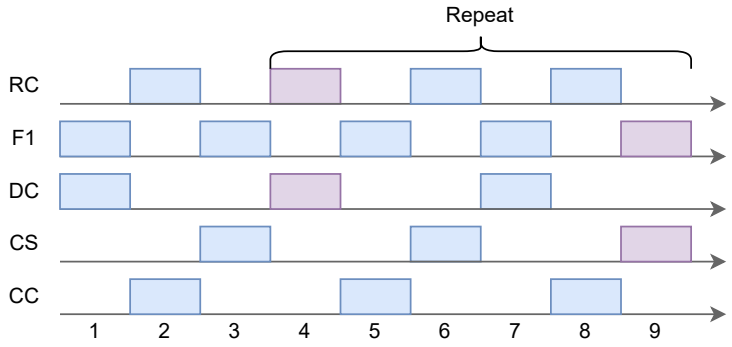

Fig. 5. Synthesized schedule for the five controllers outlined in Section VII-A.

this particular set of control tasks with the common period $P_2^C = 28\,\text{ms}$. The accepting runs of the scheduler automaton represent the set of all the valid schedules. Furthermore, there might be other common periods that can be used to schedule these tasks, but with $P_2^C = 28\,\text{ms}$, we are able to satisfy the safety requirements of all the control tasks.

D. Insights from the results obtained

To understand the effects of period compression and boosting and of deadline misses on system safety, we show the closed-loop dynamics of two specific controllers: the F1tenth Car and the Cruise Control system (see Fig. 6). The black lines denote their nominal trajectories, while the colored lines, except the red ones, represent their evolution under different safe common periods and schedules. There is no feasible schedule for $P_3^C = 40\,\text{ms}$ without recomputing the gain values of all the controllers. To illustrate this, we first obtain a feasible task schedule for $P_3^C = 40\,\text{ms}$ with recomputed gain values for all controllers. When this same schedule is applied to the controllers without recomputing their gain values, we obtain the red trajectories showing safety violations (*viz.*, the trajectories of F1 and CC go outside their safety pipes).

As shown in the 1st subplot of Fig. 6, a system can *diverge* in some cases with a period change. For the F1tenth Car, whose controller was originally designed for a period of $20\,\text{ms}$, when its period is changed to $P_3^C = 40\,\text{ms}$ and its gains are not recomputed, it diverges. This is clearly a safety violation. After recomputing the gain values for $40\,\text{ms}$, however, the system becomes safe under weakly-hard constraint $\binom{5}{6}$. Another type of safety violation is highlighted in the 2nd subplot of Fig. 6. When CC's sampling period is changed to $40\,\text{ms}$ without recomputing its gains, it still remains convergent. However, its trajectory (in red) nevertheless goes outside its safety pipe and is thus deemed unsafe. For common period $P_1^C = 15\,\text{ms}$, all systems have safe weakly-hard constraints that satisfy their safety property. But since only one out of the five tasks can fit in a time slot, the task set cannot be safely scheduled despite the fact that, in theory, a shorter period enhances control performance. This highlights the unintuitive nature of period compression and boosting: what is beneficial to systems on their own may not yield desirable results for the whole task set, and only a joint exploration of common periods and schedules provides the complete picture.

VIII. Concluding remarks

In this paper, we have studied the problem of packing multiple control tasks on a shared processor. The novelty of our approach lies in shifting the focus from ensuring all the deadlines are being met to satisfying system-level safety properties despite some deadline misses. This significantly enlarges the space of schedules that can be explored. To achieve this, we first address the issue of deriving a common sampling period for a set of control tasks that may be originally designed with distinct periods. This turns out to be a delicate problem involving unintuitive tradeoffs between the quality of control, the safety properties, and schedulability. We then use the language of weakly-hard constraints to capture the patterns of deadline misses that may be suffered by the control tasks and use them as the bridge between the system-level safety properties and the valid schedules. More precisely, given a safety property of a control system, we extract a set of weakly-hard constraints such that the system is guaranteed to be safe so long as these weakly-hard constraints are maintained. Then we construct a schedule for the task set such that deadline misses suffered by each control task satisfy its associated set of constraints. Note that increasing a task period makes it easier to meet its deadline, but compromises safety. But reducing the period also compromises safety, while making it easier for the task to meet its deadline. Our work shows how to evaluate this tradeoff using a quantifiable notion of safety.

Our experimental results show that considerably less pessimistic implementations can be obtained—and more schedules become feasible on a shared resource—when the focus is shifted from the usual goal of meeting all deadlines to meeting the safety properties of the systems under control. Furthermore, our approach is in line with the evolving paradigms in the automotive domain such as AUTOSAR Adaptive.

In our future work, we shall further explore the facets of deriving the common period, such as using periods that allow a different number of tasks to be scheduled, depending on their WCETs. We shall also expand on the system properties that must be ensured from the single definition of control safety

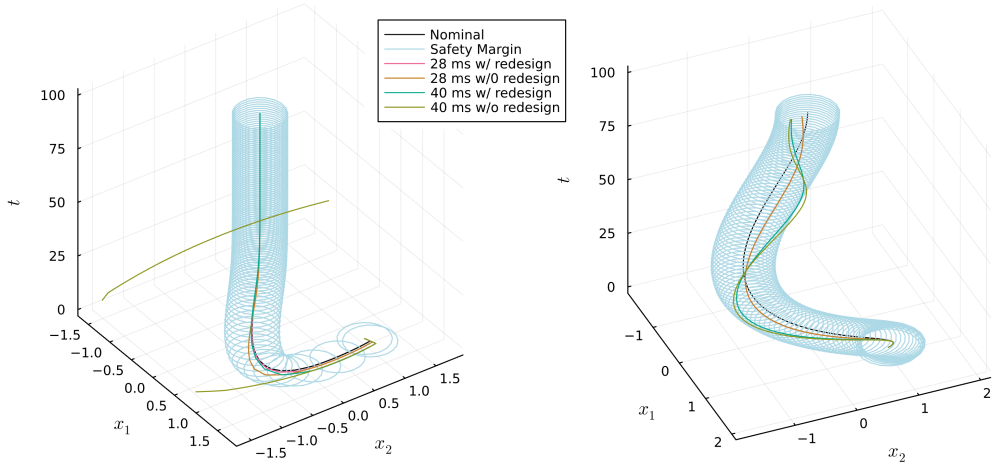

Fig. 6. Dynamics of the F1tenth Car (left) and Cruise Control (right) models

used in this work. For example, multiple levels of criticality can be specified, and the scheduling algorithm will take it into account when making scheduling decisions.

References

[1] A. Masrur *et al.*, "VM-based real-time services for automotive control applications," in *16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2010.

[2] R. Schneider *et al.*, "Multi-layered scheduling of mixed-criticality cyber-physical systems," *J. Syst. Archit.*, vol. 59, no. 10-D, pp. 1215–1230, 2013.

[3] R. Wilhelm, "Determining reliable and precise execution time bounds of real-time software," *IT Professional*, vol. 22, no. 3, pp. 64–69, 2020.

[4] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 28, no. 7, pp. 956–965, 2009.

[5] C. Menard, A. Goens, M. Lohstroh, and J. Castrillon, "Achieving determinism in adaptive autosar," in *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, ser. DATE '20. San Jose, CA, USA: EDA Consortium, 2020, p. 822–827.

[6] A. Arestova, M. Martin, K.-S. J. Hielscher, and R. German, "A service-oriented real-time communication scheme for autosar adaptive using opc ua and time-sensitive networking," *Sensors*, vol. 21, no. 7, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/7/2337

[7] M. Panic, S. Kehr, E. Quiñones, B. Böddeker, J. Abella, and F. J. Cazorla, "Runpar: An allocation algorithm for automotive applications exploiting runnable parallelism in multicores," in *CODES+ISSS*, 2014.

[8] W. Chang *et al.*, "OS-aware automotive controller design using non-uniform sampling," *ACM Trans. Cyber Phys. Syst.*, vol. 2, no. 4, pp. 26:1–26:22, 2018.

[9] W. Chang and S. Chakraborty, "Resource-aware automotive control systems design: A cyber-physical systems approach," *Found. Trends Electron. Des. Autom.*, vol. 10, no. 4, pp. 249–369, 2016.

[10] S. Chakraborty *et al.*, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Des. Test*, vol. 33, no. 4, pp. 92–108, 2016.

[11] D. Goswami, R. Schneider, and S. Chakraborty, "Re-engineering cyber-physical control applications for hybrid communication protocols," in *Design, Automation and Test in Europe (DATE)*, 2011.

[12] C. Hobbs *et al.*, "Safety analysis of embedded controllers under implementation platform timing uncertainties," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4016–4027, 2022.

[13] B. Ghosh *et al.*, "Statistical hypothesis testing of controller implementations under timing uncertainties," in *28th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022.

[14] P. Pazzaglia *et al.*, "Adaptive design of real-time control systems subject to sporadic overruns," in *DATE*, 2021.

[15] M. Maggio *et al.*, "Control-System Stability Under Consecutive Deadline Misses Constraints," in *32nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2020.

[16] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin, "DMAC: Deadline-Miss-Aware Control," in *ECRTS*, 2019.

[17] L. Ju *et al.*, "Context-sensitive timing analysis of esterel programs," in *46th Design Automation Conference (DAC)*, 2009.

[18] S. Chakraborty, T. Erlebach, and L. Thiele, "On the complexity of scheduling conditional real-time code," in *7th International Workshop on Algorithms and Data Structures (WADS)*, 2001.

[19] S. Xu, B. Ghosh, C. Hobbs, P. S. Thiagarajan, and S. Chakraborty, "Safety-aware flexible schedule synthesis for cyber-physical systems using weakly-hard constraints," in *ASPDAC*, 2023.

[20] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50, no. 4, 2001.

[21] C. Huang, W. Li, and Q. Zhu, "Formal verification of weakly-hard systems," in *22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

[22] G. von der Brüggen *et al.*, "Efficiently approximating the probability of deadline misses in real-time systems," in *ECRTS*, 2018.

[23] P. Mundhenk *et al.*, "Security analysis of automotive architectures using probabilistic model checking," in *52nd Annual Design Automation Conference (DAC)*, 2015.

[24] P. Waszecki *et al.*, "Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 36, no. 11, pp. 1790–1803, 2017.

[25] G. Georgakos *et al.*, "Reliability challenges for electric vehicles: from devices to architecture and systems software," in *50th Annual Design Automation Conference (DAC)*, 2013.

[26] N. Vreman, R. Pates, and M. Maggio, "Weaklyhard.jl: Scalable analysis of weakly-hard constraints," in *RTAS*, 2022.

[27] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems (3rd Ed.)*. USA: Prentice-Hall, Inc., 1997.

[28] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, USA: MIT Press, 2000.

[29] R. A. Gabel and R. A. Roberts, *Signals and Linear Systems*, 2nd ed. John Wiley & Sons, 1980.

[30] O'Kelly *et al.*, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," *Proceedings of Machine Learning Research*, vol. 123, 2020.

[31] D. Roy *et al.*, "Multi-objective co-optimization of FlexRay-based distributed control systems," in *RTAS*, 2016.

[32] W. C. Messner and D. M. Tilbury, "Control tutorials for MATLAB and simulink: a web-based approach," 1998. [Online]. Available: http://ctms.engin.umich.edu/CTMS

[33] R. Schneider *et al.*, "Constraint-driven synthesis and tool-support for flexray-based automotive control systems," in *CODES+ISSS*, 2011.

[34] K. Osman, M. F. Rahmat, and M. A. Ahmad, "Modelling and controller design for a cruise control system," *CSPA*, 2009.