# Safety Analysis of Embedded Controllers under Implementation Platform Timing Uncertainties

Clara Hobbs, Bineet Ghosh, Shengjie Xu, Parasara Sridhar Duggirala, and Samarjit Chakraborty, *Fellow IEEE*

***Abstract*—As embedded systems architectures become more complex and distributed, checking the safety of feedback control loops implemented on them becomes a crucial problem for emerging autonomous systems. Towards this, a number of recent papers have addressed the problem of checking *stability* in the presence of deadline misses. In this paper, we argue that analyzing *quantitative properties* like the maximum deviation in system behavior (trajectory in the state space) between an ideal implementation platform and that having timing uncertainties is an equally important problem. We show that different strategies for handling deadline misses (or system overruns), all of which lead to a stable system, might differ considerably when considering such quantitative safety properties. However, analyzing such properties involves reachability analysis that is computationally expensive and hence not scalable. We show that suitable approximation strategies can address this computational bottleneck and such quantitative safety properties can be checked for realistic systems. As a result, we are able to identify best combinations of control and deadline miss handling strategies for individual systems and timing uncertainties.**

***Index Terms*—Control, reachability, real-time, safety, weakly-hard systems.**

## I. Introduction

THE ALGORITHMIC CORE of most autonomous systems, be it in automotive or robotics, is a *feedback control loop*. Such controllers are largely created in a two-step process [1]. In the first, a control strategy is *designed* by resolving choices regarding control laws, sampling periods, and the values of various controller parameters. The second step involves *implementing* this strategy in software running on an embedded platform. This design flow results in a clean separation between control theorists and embedded systems engineers. The former design controllers by *assuming* certain properties of the implementation platform, like fixed sensor-to-actuator delays. The latter must ensure that such assumptions are *guaranteed* in an implementation in order to meet the control performance expected in the design phase.

However, implementation platform architectures are rapidly becoming more distributed and heterogeneous, concurrently running a variety of applications [2]. Hence, such strict *assume/guarantee paradigms*—where the platform characteristics are known a priori—are increasingly becoming infeasible. Therefore, some *uncertainty* in the implementation platform is inevitable. This naturally raises the fundamental question:

> *Are control safety properties satisfied in the presence of implementation platform uncertainties?*

The authors are with the Department of Computer Science in the University of North Carolina at Chapel Hill.

Manuscript received April 07, 2022; revised June 11, 2022; accepted July 05, 2022. This article was presented at the International Conference on Embedded Software (EMSOFT) 2022 and appeared as part of the ESWEEK-TCAD special issue.

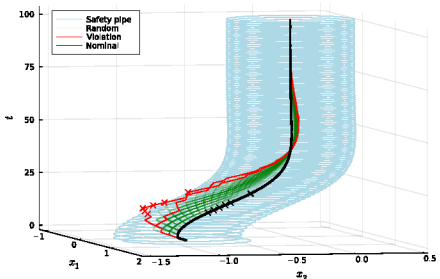



Fig. 1. Different stable trajectories resulting from deadline misses.

Various forms of this question have been addressed in the literature over the past decade [3]. These include how to design controllers in the presence of (variable) delays, and how to *co-design* controllers and their implementation platforms. These approaches can largely be seen as an effort to *close the model-implementation semantic gap*. However, a vast majority of such work has focused on addressing *stability* [4]. In particular, the question is generally of the form: *Given a bound on the number of deadline misses, is the closed-loop system stable?* **Our work in this paper**, while in the same space, deviates considerably in the nature of this question. Instead of a qualitative property like stability, we ask whether *quantitative* safety properties over a bounded time horizon hold in the presence of implementation platform timing uncertainties like deadline misses. Specifically, we are given an ideal system behavior (*e.g.*, when all deadlines are met), and an uncertainty or bound on the number of deadline misses, and would like to estimate the maximum possible deviation in behavior in the presence of such deadline misses from the ideal behavior.

**Illustrative example:** Figure 1 illustrates this problem. It shows how a given closed-loop system evolves in time in the $(x_1, x_2)$ state space. The solid black line shows the *nominal trajectory*, where the control task always meets its deadline. The light blue envelope/spiral around it is a *safety margin*, *i.e.*, it is acceptable to deviate from the nominal behavior because of deadline misses as long as the trajectory stays within this safety envelope. 100 random trajectories—resulting from deadline misses experienced by the control task—are shown in green and red. Although the system is stable even in the presence of deadline misses, the possible behaviors can vary greatly, as can be seen in the figure. The red trajectories violate the specified safety property since they go out of the safety envelope, and where the violations occur have been marked with "×". The corresponding time instants have been marked with a black × on the nominal trajectory, with the distance between a black × and its corresponding red × exceeding the maximum allowed deviation. This figure was obtained using an automotive electric steering system that is evaluated in more detail in Section VII. Here, the goal was to

evaluate if a safety violation occurs if the control task suffers no more than 3 *consecutive* deadline misses. In other words, at most 3 consecutive jobs of this control software fail to compute the control input in time for the necessary actuation. This clearly admits many deadline hit/miss patterns over any reasonable time horizon of interest (*e.g.*, 100 samples of the plant state). It is computationally infeasible to compute the system trajectories for all such deadline hit/miss patterns to check whether or not a safety violation occurs, and how to do this *efficiently* for realistic systems is one of the **contributions of this paper**. The second contribution of this paper relates to evaluating how deadline misses are handled—both from the task scheduling perspective and from the control-theoretic one. Using our proposed schemes it is possible to identify the best combination of control and scheduling (or deadline overrun management) strategies for any given system. From the vantage point of quantitative safety properties, we are able to make a fine-grained distinction between these strategies that was not possible in previous studies that focused on stability analysis. We discuss these below in more detail.

**Relation to prior work:** Our work is motivated by a recent work [5] (and a number of preceding ones on the related problem [6]–[10]) that studied how deadline misses may be handled on an implementation platform and what impact it has on control performance (specifically, stability). Various strategies to handle deadline misses that were studied in this paper include combinations of applying either a *zero* or the *previous* control input to the plant in the case of a deadline miss, and either *killing* the control task that missed its deadline or *letting it complete* its execution beyond the deadline. Killing or letting a task continue execution impacts the load on the system and hence potential future deadline misses, and whether a zero or a previous control input is applied impacts the state space evolution of the closed-loop system. The work in [5] only studies stability, and hence classifies these combinations of strategies into only two classes, *viz.*, whether or not the system is stable. We, on the other hand, show that a much finer and *quantitative* distinction between these strategies can be made when we consider the maximum deviation from a nominal or ideal behavior.

Further, while stability is a necessary property, and the deviation we compute will likely be unbounded if the system is not stable, it by no means ensures *safe* operation in most realistic systems. For example, stability might ensure that an autonomous drone or robot eventually reaches its destination, but would not guarantee that it does not collide with an obstacle on the way. This may be easily visualized in Figure 1, where the black line is the ideal trajectory and the blue safety envelope is specified such that no obstacles lie on the path of the robot; but they might lie just outside the envelope. Hence, too much deviation from the ideal trajectory due to implementation platform timing uncertainties might not guarantee a collision-free path for the robot. The safety property we analyze in this paper helps provide such guarantees.

In addition to more work on checking stability, from a computational perspective it is also easier to do so. This is by relying on techniques like the existence of a Lyapunov function and on results from stability analysis of switched systems [11]. Estimating the maximum deviation is computationally more expensive and hence does not scale because it involves some form of reachability analysis. To get around this, we propose a set of safe approximation techniques for computing such deviations from a nominal behavior in the presence of platform uncertainties (specified by a bound on deadline misses).

We conclude this section by discussing some broader related work. The issue of "implementing" a controller has traditionally not been seen as a problem within control theory. But how to design a controller with sensing-to-actuation delays has been well-known for a number of years [12], [13] and the area of *networked control systems* has studied different aspects of controller design in the presence of delays [14]–[17]. Since *feedback* controllers are inherently tolerant to some model uncertainties, they can also tolerate delay variations to a certain degree. However, if the delays become very large, the controllers are distributed, or multiple control inputs are missed because of deadline violations, then the problem of controller design becomes much more complex and such scenarios are only being looked at more recently [18]–[21]. Modeling the impact of implementation platform (or more specifically *network*) uncertainties on control performance in the form of time-varying or stochastic delays have been studied in [14], [16], [22]. But again, the focus has been on stability. Similarly, time-varying delays and the use of different feedback gains for different delays is closely related to the study of switched systems [11], [23]. While these are abstractly related to our work, both the goals and the techniques used are different—as already outlined, instead of stability we study quantitative properties and use approximation techniques for reachability analysis. Finally, a number of recent papers have addressed various aspects of the control/architecture *co-design* problem [24]–[26]. In contrast to designing a controller to mitigate the impact of sensor-to-actuator delays, as is done in networked control systems, the co-design problem attempts to *jointly* design controllers and network parameters like schedules in order to maximize the "compatibility" between a controller and its implementation platform.

**Paper organization:** The rest of this paper is organized as follows. We propose our system model in Section II, and describe how to encode implementation platform behaviors (*viz.*, deadline hit/miss patterns) in Section III. Section IV provides the notations used in the rest of the paper. We formally define the main problem in Section V and present our solutions in Section VI. We implemented our algorithms and evaluated them on standard benchmarks, presented in Section VII. Finally, we conclude in Section VIII.

## II. System Model

This section outlines the basics of feedback controllers as studied in this paper. We consider linear, multiple-input multiple-output (MIMO) discrete-time models of the form

$$x[t+1] = Ax[t] + Bu[t], \tag{1}$$

$$y[t] = Cx[t] + Du[t]. \tag{2}$$

Here, $x$ is the system state, $y$ is the output, and $A \in \mathbf{R}^{n\times n}$, $B \in \mathbf{R}^{n\times p}$, $C \in \mathbf{R}^{q\times n}$, and $D \in \mathbf{R}^{q\times p}$ are coefficient

matrices. The control input $u$ is computed by a periodic real-time task running on a processor, assumed to be of the form

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

where $K \in \mathbf{R}^{p\times n}$. This follows the *logical execution time* (LET) paradigm, in which the new control input is always applied at the deadline of the control job (assumed to be one sampling period), regardless of when it actually completes. This can alternately be represented using an augmented state space [13] $z[t] = [x[t]^T \;\; u_a[t-1]^T]^T$, giving the model

$$z[t+1] = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} z[t] + \begin{bmatrix} 0 \\ I \end{bmatrix} u_a[t], \tag{4}$$

where $u_a[t] = K_a z[t]$. This augmented form permits standard controller design techniques such as *linear-quadratic regulator* (LQR). Note that when augmented in this way, the feedback gain matrix $K_a \in \mathbf{R}^{p\times(n+p)}$, allows feedback from both the plant state $x[t]$ and the previous control input $u[t-1]$. This can be implemented by saving the previous control input between jobs of the control task. We denote the two blocks of $K_a$ that provide feedback from each of these vectors as $K_x \in \mathbf{R}^{p\times n}$ and $K_u \in \mathbf{R}^{p\times p}$, respectively. Once a controller has been designed, Eq. (4) can be simplified to

$$z[t+1] = \begin{bmatrix} A & B \\ K_x & K_u \end{bmatrix} z[t], \tag{5}$$

allowing the plant and controller to be represented as a single *dynamics matrix*. The system output can also be defined as

$$y[t] = Ez[t], \tag{6}$$

where $E = [C \;\; D]$, giving a more compact representation.

## III. Modeling Implementation Platform Behaviors

To model the behavior of the software task computing the control law—that can potentially hit/meet or miss a deadline at each sampling period (referred to as a *step*)—we propose an automaton-based representation. The initial location of the automaton represents the initial condition of the control task, and a finite run of the automaton represents a possible behavior of the system. This is a sequence of hits and misses from its initial location. In this work we focus on bounded time behaviors, *viz.*, deviations from a nominal behavior over a specified time interval. Hence, any run of this automaton has finite length. Note that the dynamics matrix, capturing the control input to be applied, also changes depending on whether a deadline hit or miss occurs (because it determines the availability of the control input). Therefore, we associate a dynamics matrix with each transition of the automaton. We formally define the transducer automaton that models all possible behaviors of the system as follows.

**Definition 1.** *A transducer automaton $\mathcal{T}$ is defined as a tuple $\langle L, \mathcal{A}, T, \mu, \ell_0\rangle$, where each element is as follows:*

- $L$ *set of locations $\{\ell_1, \ell_2, \ldots, \ell_m\}$;*
- $\mathcal{A}$ *set of scheduler actions $\{hit, miss\}$;*
- $T$ *transition function, where $T : L \times \mathcal{A} \to L$;*
- $\mu$ *dynamics matrix label function for transitions, where $\mu : L \times \mathcal{A} \to \mathbf{R}^{n\times n}$ and $n$ is the dimension of the system under consideration;*

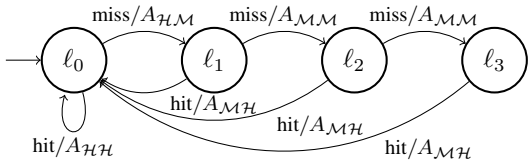



Fig. 2. Transducer automaton capturing 3 maximum consecutive misses.

- $\ell_0$ *initial location of the automaton in $L$.*

As an example, the automaton in Fig. 2 captures all possible deadline hit/miss patterns with at most three consecutive misses. The edges are labeled as $\mathcal{A}/\mu$, *i.e.*, the first part is the scheduler action, and the second is the associated dynamics matrix. Note that in this automaton, each location $\ell_k$ represents the control task having just missed $k$ consecutive deadlines.

### A. Behavior under deadline misses

In order to model the system behavior under a sequence of deadline hits and misses, we use standard techniques, similar to those in [5], to provide the function $\mu$ associating dynamics matrices with the transitions in the automaton. In this model, the logical execution time (LET) paradigm is followed, *i.e.*, a sample of the system state at step $t-1$ is used to compute the control input at time $t$. A software job is released when $x[t-1]$ is read, and has its deadline when $x[t]$ is read. If the job completes on time, the control input is computed as in Eq. (3). If the job misses its deadline, several different actions can be taken, both for how to handle the sequence of released jobs, as well as what control input should be applied.

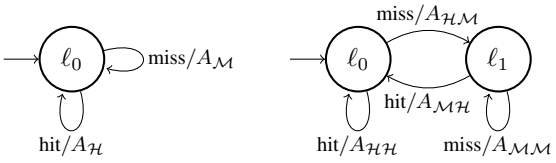
Two strategies are defined in [5] for how to compute a control input when a deadline miss occurs at time $t$. These strategies are called *Zero* and *Hold*. Our goal is to study quantitative safety properties (*viz.*, maximum deviation from a nominal behavior) instead of stability that was studied in [5]. But for the sake of comparison, we consider the same strategies in this paper. The *Zero* strategy says that the control input $u[t]$ is set to 0. The *Hold* strategy instead holds the control input $u[t] = u[t-1]$. The study in [5] also defined strategies for the system-level action on how to handle deadline overruns. In our work, we consider two of these, called *Kill* and *Skip-Next*. The *Kill* strategy simply kills the job that missed its deadline. The *Skip-Next* strategy instead allows the job to continue running, and prevents the release of additional jobs until the job that missed its deadline finishes.

By combining a pair of control input strategy and system-level action, we obtain a strategy for handling deadline misses. Several such models were developed in [5]; we reproduce their models for *Zero&Kill* and *Hold&Kill* next, reformulated in our automata-theoretic setting, and permitting feedback from the previous control input as in Eq. (4).

**Definition 2** (*Zero&Kill* [5])**.** *Given a discrete-time LTI model as in Eq.* (1)*, the Zero&Kill strategy is modeled using the augmented state space $z[k] = [x[k]^T \;\; u_a[k]^T]^T$. The automaton follows the structure in Fig. 3a, with the following matrices:*

$$A_{\mathcal{H}} = \begin{bmatrix} A & B \\ K_x & K_u \end{bmatrix} \qquad A_{\mathcal{M}} = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}$$

Every period, this model computes Eq. (1). When a deadline hit occurs, the next control input is $u_a[k+1] = K_a z[k]$, and when a miss occurs, the control input is instead $u_a[k+1] = 0$.




(a) *Kill* (b) *Skip-Next*

Fig. 3. Transducer automata for different system-level actions.

**Definition 3** (*Hold&Kill* [5])**.** *Given a discrete-time LTI model as in Eq.* (1)*, the Hold&Kill strategy is modeled using the augmented state space* $z[k] = [x[k]^T \ u_a[k]^T]^T$*. The automaton follows the structure in Fig. 3a, with the following matrices:*

$$A_{\mathcal{H}} = \begin{bmatrix} A & B \\ K_x & K_u \end{bmatrix} \qquad A_{\mathcal{M}} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}$$

Similar to the *Zero&Kill* model, this model computes Eq. (1) at every period. When a deadline is met, the next control input is $u_a[k+1] = K_a z[k]$, but now when a miss occurs, the control input is held constant ($u_a[k+1] = u_a[k]$).

There are models described in [5] for the *Zero&Skip-Next* and *Hold&Skip-Next* strategies, but these are limited by requiring a maximum number of consecutive deadline misses. Additionally, their augmented state vectors grow linearly in the number of misses allowed. This leads to scalability issues when computing long trajectories, especially if no limit to the number of consecutive deadline misses is desired. To address these limitations, we propose the following improved models that use a constant-size augmented state vector, imposing no constraints on the allowable sequences of scheduler actions.

**Definition 4** (*Zero&Skip-Next*)**.** *Given a discrete-time LTI model, the Zero&Skip-Next strategy is modeled using the augmented state space* $z[k] = [x[k]^T \ x[save]^T \ u_a[k]^T]^T$*. The automaton is that in Fig. 3b, with the following matrices:*

$$A_{\mathcal{H}\mathcal{H}} = \begin{bmatrix} A & 0 & B \\ 0 & 0 & 0 \\ K_x & 0 & K_u \end{bmatrix} \qquad A_{\mathcal{H}\mathcal{M}} = \begin{bmatrix} A & 0 & B \\ I & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$A_{\mathcal{M}\mathcal{H}} = \begin{bmatrix} A & 0 & B \\ 0 & 0 & 0 \\ 0 & K_x & K_u \end{bmatrix} \qquad A_{\mathcal{M}\mathcal{M}} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

**Definition 5** (*Hold&Skip-Next*)**.** *Given a discrete-time LTI model, the Hold&Skip-Next strategy is modeled identically to Definition 4, with* $A_{\mathcal{H}\mathcal{M}}$ *and* $A_{\mathcal{M}\mathcal{M}}$ *changed as follows:*

$$A_{\mathcal{H}\mathcal{M}} = \begin{bmatrix} A & 0 & B \\ I & 0 & 0 \\ 0 & 0 & I \end{bmatrix} \qquad A_{\mathcal{M}\mathcal{M}} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}$$

We note that these models differ from one another only in the control input applied when a deadline is missed, per the definitions of *Zero* and *Hold* above. The models keep track of the current plant state and control input, and one saved state. Every period, regardless the matrix used, the system dynamics are computed as $x[t+1] = Ax[t] + Bu[t]$, as required by Eq. (1). In normal operation, the $A_{\mathcal{H}\mathcal{H}}$ matrix (identical for both strategies) computes the next control input as $u_a[t+1] = K_a[x[t]^T \ u_a[t]^T]^T$, following the one-period delay of the LET paradigm. When a deadline is missed following a hit, *Skip-Next* semantics let the job continue running, applying its resulting control input upon completion. To implement this, the $A_{\mathcal{H}\mathcal{M}}$ matrices save the system state in $x[save]$ for later use. The strategies differ in their handling of the control input applied on deadline miss: *Hold&Skip-Next* keeps the input constant, while *Zero&Skip-Next* sets the input to zero. On further misses, the $A_{\mathcal{M}\mathcal{M}}$ matrices retain the saved state, and either hold or clear the control input as required. Once the job that overran its deadline completes, the $A_{\mathcal{M}\mathcal{H}}$ matrix (again identical) is applied. This computes a new control input $u_a[t+1] = K_a[x[save]^T \ u_a[t]^T]^T$, following the semantics of *Skip-Next*. Upon further deadline hits, the system returns to normal operation.

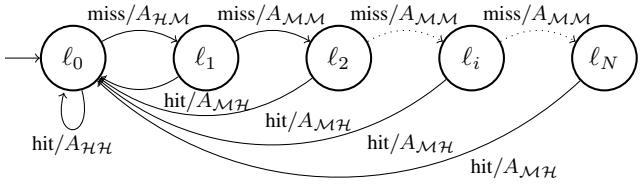



Fig. 4. Transducer automaton capturing $N$ maximum consecutive misses.

### B. Constraints on Deadline Misses

As described above, in this work, the uncertainty in the implementation platform's timing stems from (or results in) some control jobs occasionally missing their deadlines. To give constraints on which deadlines can be missed, we consider any control task to be running on a *weakly-hard real time system* [27]. These systems provide precise bounds on the patterns of deadline misses that can occur. Weakly-hard constraints have been considered in relation to control systems in prior work [5]–[10], [28], and have been found to be a useful abstraction for the complex behavior of real-time task schedulers. In particular, we assume that the scheduler guarantees a maximum of $N$ consecutive deadline misses, a constraint often denoted as $\overline{\langle N \rangle}$ in the literature.

Recalling Fig. 2, we can model such a constraint using transducer automata. In this example, we allow at most three deadline misses, using each location $\ell_k$ in the automaton to represent having missed $k$ consecutive deadlines. The lack of a transition on deadline miss from $\ell_3$ indicates that no further misses are possible. This construction can be generalized to handle any number of consecutive deadline misses $N$ using $N+1$ locations, as shown in Fig. 4. With the matrices from Definition 4 or 5, we can use this construction to model *Zero&Skip-Next* or *Hold&Skip-Next*, respectively, with a weakly-hard constraint. Similarly, by letting $A_{\mathcal{H}\mathcal{H}} = A_{\mathcal{M}\mathcal{H}} = A_{\mathcal{H}}$ and $A_{\mathcal{M}\mathcal{M}} = A_{\mathcal{H}\mathcal{M}} = A_{\mathcal{M}}$ from Definition 2 or 3, we can model the *Zero&Kill* or *Hold&Kill* strategies with a maximum number of consecutive deadline misses.

## IV. Quantitative Safety Properties

In this section, we present the definitions needed to define our main problem, and our approaches that follow. Let the output $y[t]$ (for any sampling period or time step $t$) of the plant—as defined in Eq. (6)—be subsets of the metric space $(M, dis)$, where $M = \mathbf{R}^q$ and $dis : M \times M \to \mathbf{R}$ is a metric on $M$. Note that we do not impose any assumption on $dis$, as long as $(M, dis)$ is a metric space. Note also that this

metric applies only to the system *output*, not necessarily the entire augmented state vector used by a transducer automaton.

Given a transducer automaton $\mathcal{T}$, a possible behavior of the system is defined as a *run*, which intuitively represents a possible trajectory by encoding a sequence of deadline hits and misses and the corresponding locations in the automaton.

**Definition 6** (Run)**.** *A* run $\tau$ *is defined as an alternating sequence of transducer automaton locations and actions,*

$$\tau = \{\ell_0, a_0, \ell_1, \ldots, a_{H-1}, \ell_H\},$$

*where* $a_i \in \mathcal{A}$*, and* $H$ *is the time bound. We denote the set of all runs of* $\mathcal{T}$ *by* $\bar{\tau}$*.*

Given a run $\tau$ and an initial set $z[0]$, the evolution of the run encodes the set of states reached by the plant over the course of the run, and is defined as follows.

**Definition 7** (Evolution of a Run)**.** *Given an initial set* $z[0] \subset \mathbf{R}^n$ *and a run* $\tau = \{\ell_0, a_0, \ell_1, \ldots, a_{H-1}, \ell_H\}$*, the* evolution *of the run* $\tau$ *in an automaton* $\mathcal{T}$ *is defined as*

$$evol(\mathcal{T}, \tau, z[0]) = \big\{z[i+1] = \mu(\ell_i, a_i)z[i] \;\big|\; 0 \le i < H\big\}.$$

*Here,* $z[t]$ *is the augmented state of the system reached at time step* $t$*. We refer to* $evol(\mathcal{T}, \tau, z[0])$ *as* $evol(\tau)$ *when* $\mathcal{T}$ *and* $z[0]$ *are clear from context. Given an evolution of a run* $\tau$*,* $evol(\tau)$*, let the* $evol(\tau)[t] = z[t]$*, where* $1 \le t \le H$*.*

To measure the distance between two sets in the metric space, we use the standard Hausdorff distance, which intuitively gives the longest distance from any point in one set to the closest point in the other set.

**Definition 8** (Hausdorff Distance)**.** *The* Hausdorff distance *between* $S, U \subset \mathbf{R}^n$ *is given by*

$$d_H(S,U) = \max\Big\{\sup_{s\in S}\inf_{u\in U} dis(s,u), \sup_{u\in U}\inf_{s\in S} dis(s,u)\Big\}.$$

Given two runs $\tau_1, \tau_2 \in \bar{\tau}$, we define the deviation between them as the maximum Hausdorff distance between their evolutions at any time step.

**Definition 9** (Deviation)**.** *Given two runs* $\tau_1, \tau_2 \in \bar{\tau}$*, we define their* deviation *as*

$$dev(\tau_1, \tau_2) = \max_{1\le t\le H} d_H\big(E \cdot evol(\tau_1)[t], E \cdot evol(\tau_2)[t]\big).$$

Our methods in Section VI make use of a convex hull, which is a convex set enclosing a given set.

**Definition 10** (Convex Hull)**.** *The* convex hull *of the sets* $S_1, S_2, \ldots, S_p \subset \mathbf{R}^n$ *is denoted by* $hull_{1\le i\le p}\{S_i\}$*.*

## V. Problem Statement

In this section, we define the problem formally as follows.

**Problem 1** (Maximum Deviation)**.** *Given a transducer automaton* $\mathcal{T}$*, an initial set* $z[0] \subset \mathbf{R}^n$*, and a nominal run* $\tau_{nom} \in \bar{\tau}$ *(corresponding to an ideal behavior of the platform,* e.g.*, with no deadline misses), find the maximum deviation* $\mathbf{d}$ *between the nominal run and any other run,*

$$\mathbf{d} = \max_{\tau\in\bar{\tau}} dev(\tau, \tau_{nom}). \tag{7}$$

Note that directly computing $\mathbf{d}$, as in Problem 1, is computationally expensive. To compute an exact value of $\mathbf{d}$ for some time horizon $H$, we need to explore all possible $2^H$ runs of the automaton, assuming two possibilities *viz.*, deadline hit or miss in each step. We must then compute the deviation of each run from the nominal run $\tau_{nom}$. Clearly, such an approach, though feasible for small $H$, becomes intractable very quickly. Instead, we propose to compute a safe bound to $\mathbf{d}$ as follows.

**Problem 2** (Deviation Bound)**.** *Given a transducer automaton* $\mathcal{T}$ *with initial set* $z[0] \subset \mathbf{R}^n$*, and a nominal run* $\tau_{nom} \in \bar{\tau}$*, find a safe upper bound* $\bar{\mathbf{d}}$ *to the maximum deviation between the nominal run and any other run,*

$$\bar{\mathbf{d}} \ge \mathbf{d} = \max_{\tau\in\bar{\tau}} dev(\tau, \tau_{nom}). \tag{8}$$

It is worth pointing out that given an initial state $z[0]$, if a bound $\bar{\mathbf{d}}$ is known, it may be easily possible to compute the maximum deviation for any initial state $cz[0]$, where $c$ is a scalar. In particular, if the metric $dis$ is *absolutely homogeneous*, *i.e.*, $dis(cx, cy) = |c|dis(x, y)$ for $x, y \in M$, then it follows from the definitions that the maximum deviation for $cx[0]$ is at most $|c|\bar{\mathbf{d}}$. Thus, there is no need to perform a computationally expensive procedure to compute several deviation bounds from initial states that are scalar multiples of each other, as this could instead be done once, and the result multiplied by scalars to obtain the other deviations.

In the next section, we propose several solutions to Problem 2 that perform efficiently in practice.

## VI. Computing Upper-bounds on the Deviation

In this section, we present three methods to compute a safe deviation bound $\bar{\mathbf{d}} \ge \mathbf{d}$, as defined in Problem 2, given a transducer automaton. The first method uses reachability analysis of uncertain linear systems (ULSs) to overapproximate any possible behavior of a given transducer automaton, and computes an upper bound $\bar{\mathbf{d}}$ by computing the distance of the reachable set from the nominal trajectory. The second method, given a transducer automaton capturing *at most* $N$ *consecutive misses*, computes reachable sets using a set of recurrence relations. Note that this method does not handle arbitrary automata, but only ones following the $\overline{\langle N \rangle}$ weakly-hard constraint. The third method, given any transducer automaton, computes all possible trajectories up to a small, bounded length. It then computes a convex hull of the obtained trajectories, uses this hull as the next set of initial states, and iterates until the desired time bound. The first method is fundamentally different from the others; the second method, though superficially unrelated to the third, can in fact be viewed as a special case of it. While the second suffices for some systems, the third offers flexibility to produce tighter bounds at the expense of greater execution time. Recommendations for how to use these methods are provided following our experimental results in Section VII.

### A. Using Reachable Sets of Uncertain Linear Systems

In this section, we compute an upper bound $\bar{\mathbf{d}}$, as in Problem 2, using reachable sets of uncertain linear systems

(ULSs) [29], [30]. Before presenting our solution, we first introduce ULSs. Consider the following example from [29],

$$x[t+1] = \begin{bmatrix} 1 & \alpha \\ 0 & 2 \end{bmatrix} x[t],$$

where $\alpha$ represents a parameter, $2 \leq \alpha \leq 3$. Intuitively, a ULS of this form models uncertainty in the system by representing all its possible dynamics matrices.

**Definition 11** (Uncertain Linear Systems)**.** *An uncertain linear dynamical system takes the form*

$$x[t+1] = \Lambda x[t], \tag{9}$$

*where $x[t] \in \mathbf{R}^n$ is the system state at time $t$, and $\Lambda \subseteq \mathbf{R}^{n \times n}$ is the uncertain dynamics matrix.*

Note that the uncertain dynamics matrix is, therefore, capable of representing a set of linear dynamics matrices. Leveraging this modeling richness, we provide an efficient method to compute an upper-bound on $\mathbf{d}$, by modeling the dynamics matrices associated with the transitions of a transducer automaton as an uncertain dynamics matrix. We model all possible sequences of hits/miss of the system (represented by the dynamics matrices of the transitions, *i.e.*, the function $\mu$ in Definition 1) using uncertain dynamics matrix. Note that this method is capable of capturing any possible behavior of the system. Formally, for a given transducer automaton $\mathcal{T}$,

$$\Lambda = \big\{ \mu(\ell, a) \;\big|\; \ell \in L, a \in \mathcal{A} \big\}. \tag{10}$$

Intuitively, $\Lambda$ encodes all possible behaviors of the system at any time step $t$. This method simply over-approximates the behaviors of the transducer automaton by assuming that any dynamics could occur at each time step.

The reachable set of a ULS, at a time step $t$, represents the possible states of the system under any permissible sequence of actions. In our case, Eq. (10) specifies that the reachable set corresponds to an over-approximate set of evolutions of all runs of length $t$. The Hausdorff distance between such a set and the nominal run (as in Definition 9) provides an upper bound on $\mathbf{d}$. Let the one-step reachable set of a ULS, from an initial set $x[0]$, be given as $forward(\Lambda, x[0])$; *i.e.* $x[1] = \Lambda x[0]$, where $x[1] = forward(\Lambda, x[0])$.

*Representing Uncertain Dynamics Matrix:* We represent the uncertain dynamics using an interval matrix [31]. Therefore, we over-approximate $\Lambda$ in Eq. (10) as $\tilde{\Lambda}$ as

$$\tilde{\Lambda}[i,j] = \big[\min\{\Lambda[i,j]\}, \max\{\Lambda[i,j]\}\big], \tag{11}$$

for all $1 \leq i, j \leq n$, where $n$ is the dimension of the system. Clearly, $\tilde{\Lambda} \supseteq \Lambda$. Using this uncertain dynamics matrix, we propose Algorithm 1 to compute an upper bound $\bar{\mathbf{d}} \geq \mathbf{d}$.

*Algorithm 1 and its Safety Proof (Sketch):* The algorithm first computes the uncertain matrix in Line 2. It then computes an upper bound $\bar{\mathbf{d}}$ to the possible deviation in the loop on Lines 4 to 7. In each iteration, we perform the following: 1) Line 5 computes one-step reachability of the ULS. Note that $x[t]$ therefore contains the evolution of all possible runs at time step $t$. 2) In Line 6, we compute the maximum possible deviation between the nominal behavior and the reachable set. 3) We then store the maximum deviation so far on Line 7. Finally, the computed bound $\bar{\mathbf{d}}$ is returned on Line 8. Correctness can be proven by induction. If the reachable set $x[t]$ contains the true reachable set at time $t$, then the $forward(\cdot)$ function on Line 5 computes a superset of the union of $Ax[t]$ for any matrix $A$ in the transducer automaton. Therefore, $x[t+1]$ must also be safe, so the computed deviation is at least the true maximum deviation.

**Algorithm 1:** Computing upper-bound on the deviation as defined in Problem 1.

**input :** A transducer automaton $\mathcal{T}$, initial set $x[0]$, nominal run $\tau_{nom}$, time bound $H$
**output:** An upper bound $\bar{\mathbf{d}} \geq \mathbf{d}$

1 $\bar{\mathbf{d}} \leftarrow -\infty$;
2 $\tilde{\Lambda} \leftarrow$ Compute using Eq. (11); // Represent all possible behaviors (vis-à-vis hits/misses) as an uncertain linear system.
3 $x_{nom} \leftarrow evol(\tau_{nom})$; // Compute the nominal trajectory.
4 **for** $1 \leq t \leq H$ **do**
5 $\quad x[t] \leftarrow forward(\tilde{\Lambda}, x[t-1])$; // Compute reachable set, containing all possible behaviors, at time step $t$.
6 $\quad d_t \leftarrow d_H\big(x_{nom}[t], x[t]\big)$; // Compute the deviation from the nominal trajectory at time step $t$.
7 $\quad \bar{\mathbf{d}} \leftarrow \max\{\bar{\mathbf{d}}, d_t\}$; // Keep track of the maximum deviation.
8 **return** $\bar{\mathbf{d}}$; // Return the maximum deviation.

### B. Using Generalized Recurrence Relations

Consider the transducer automaton in Fig. 2 that captures all possible behaviors with at most three consecutive misses. Note that in this automaton, execution is in location $\ell_k$ if $k$ consecutive deadlines were missed since the last hit.

Let $\Psi^t_\ell$ denote the reachable set of all possible trajectories corresponding to the location $\ell$ at time step $t$. That is, $\Psi^t_{\ell_k}$ denotes the reachable set of all possible trajectories where $k$ consecutive deadlines were missed at time step $t$. Therefore, to compute the reachable set $\Psi^t_\ell$, we need to consider all possible transitions that lead to the location $\ell$, starting from (*i.e.*, initial set) every possible state. Formally, using the policy described in Definition 5, we get the following recurrence relations:

$$\Psi^t_{\ell_0} = hull\big(A_{\mathcal{H}\mathcal{H}} \cdot \Psi^{t-1}_{\ell_0}, A_{\mathcal{M}\mathcal{H}} \cdot \Psi^{t-1}_{\ell_1}, A_{\mathcal{M}\mathcal{H}} \cdot \Psi^{t-1}_{\ell_2}, A_{\mathcal{M}\mathcal{H}} \cdot \Psi^{t-1}_{\ell_3}\big) \tag{12}$$

$$\Psi^t_{\ell_1} = A_{\mathcal{H}\mathcal{M}} \cdot \Psi^{t-1}_{\ell_0} \tag{13}$$

$$\Psi^t_{\ell_2} = A_{\mathcal{M}\mathcal{M}} \cdot \Psi^{t-1}_{\ell_1} \tag{14}$$

$$\Psi^t_{\ell_3} = A_{\mathcal{M}\mathcal{M}} \cdot \Psi^{t-1}_{\ell_2} \tag{15}$$

Where the matrices $A_{\mathcal{H}\mathcal{H}}$, $A_{\mathcal{H}\mathcal{M}}$, $A_{\mathcal{M}\mathcal{H}}$, and $A_{\mathcal{M}\mathcal{M}}$ are defined in Definition 5, with the following initial conditions:

$$\Psi^0_{\ell_0} = x[0]; \qquad \Psi^0_{\ell_p} = \varnothing, \qquad \text{where } 1 \leq p \leq 3 \tag{16}$$

Note that although the above recurrence relations are fixed for a given policy (*Hold&Skip-Next* in this case), this need not be the case. We will provide a generalized set of recurrence relations next, covering any strategy with at most $N$ consecutive misses. Consider the automaton in Fig. 4. Here, the automaton is in location $\ell_k$ if $k$ consecutive misses have just occurred. As in our motivating example, let $\Psi^t_\ell$ denote the reachable set of all possible trajectories in location $\ell$ at time $t$. We construct the following recurrence relations for time step $t \geq 1$:

$$\Psi^t_{\ell_0} = hull_{\ell \in L}\big(\mu(\ell, \text{hit}) \cdot \Psi^{t-1}_\ell\big) \tag{17}$$

$$\Psi^t_{\ell_p} = \mu(\ell_{p-1}, \text{miss}) \cdot \Psi^{t-1}_{\ell_{p-1}}, \qquad \text{where } 1 \le p \le N \quad (18)$$

The initial conditions are as follows:

$$\Psi^0_{\ell_0} = x[0]; \qquad \Psi^0_{\ell_p} = \varnothing, \qquad \text{where } 1 \le p \le N \quad (19)$$

Using the above recurrence relations, we propose Algorithm 2 to compute an upper bound $\mathbf{d}$, as in Problem 1.

**Algorithm 2:** Computing upper-bound on the deviation as defined in Problem 1.

**input :** A transducer automaton $\mathcal{T}$ encoding maximum number of allowed consecutive misses $N$, initial set $\theta$, nominal run $\tau_{nom}$, time bound $H$
**output:** An upper bound $\bar{\mathbf{d}} \ge \mathbf{d}$
/* Each location represents a class of behaviors (vis-à-vis hits/misses) in this algorithm. */
1 $\bar{\mathbf{d}} = d_H\big(evol(\tau_{nom})[0], \theta\big)$;
2 $\Psi^0_{\ell_0} = \theta$ ; // Initialize the initial location (a class of behaviors) with the given initial set.
3 **for** $1 \le k \le N$ **do**
4 $\quad \Psi^0_{\ell_k} = \varnothing$ ; // Initialize the rest of the locations with empty set.
5 **for** $1 \le t \le H$ **do**
6 $\quad \Psi^t_{\ell_0} \leftarrow$ Compute using Eq. (17) ; // Compute the reachable set for the initial location.
7 $\quad$ **for** $1 \le k \le N$ **do**
8 $\quad\quad \Psi^t_{\ell_k} \leftarrow$ Compute using Eq. (18); // Compute the reachable set for rest of the locations.
9 $\quad d_t \leftarrow d_H\big(evol(\tau_{nom})[t], hull_{0 \le l \le N}\{\Psi^t_{\ell_l}\}\big)$; // Compute the deviation from the nominal trajectory at time step $t$.
10 $\quad \bar{\mathbf{d}} \leftarrow \max\{\bar{\mathbf{d}}, d_t\}$; // Keep track of the maximum deviation.
11 **return** $\bar{\mathbf{d}}$; // Return the maximum deviation.

*Algorithm 2 and its Safety Proof (Sketch):* We initialize the recurrence relations in Lines 2 to 4, using Eq. (19). From Lines 5 to 10, we compute the deviation bound $\bar{\mathbf{d}}$. In each iteration of the for loop, we perform the following: 1) In Lines 6 to 8, we compute all possible reachable sets at time step $t$, using Eqs. (17) and (18). 2) In Line 9, we compute the Hausdorff distance between the convex hull of the reachable sets in all locations, and evolution of the nominal run at the same time step. 3) Finally, we store the maximum deviation seen in Line 10. After the loop terminates, the computed upper bound $\bar{\mathbf{d}}$ is returned on Line 11. Correctness can again be proven inductively. If the reachable set is safe at time $t$, then Lines 6 to 8 compute the one-step evolution for each automaton location. This is then overapproximated by a convex hull before computing the Hausdorff distance to the nominal trajectory, leaving a safe reachable set at time $t+1$, so the computed deviation must be an upper bound.

### C. Using Bounded Runs Method

As described in Section V, in the worst case, using a naïve approach to compute the maximum deviation $\mathbf{d}$ involves enumerating all $2^H$ runs for $H$ time steps, and computing the evolution for each such run of the system. Such a brute-force approach is intractable for large $H$, so it cannot be directly used to compute the exact deviation in most cases. However, for a short run length $r \ll H$, this approach can be used quite effectively. By computing a box hull of the reachable sets at the end of each run, we can repeat this process to simulate all $H$ time steps while keeping the execution time low.

Several complexities arise in practice, however, making this approach less straightforward than it may appear from the previous description. If we simply used a single hull of the final state for all runs, we would forget the automaton location in which each run ended. The next tree would then begin all runs from the initial location, making spurious transitions that could potentially cause us to miss some possible evolutions of the system. To avoid this, we instead group the runs by their final location, and return one hull for each location in $L$.

Additionally, if we were to compute the evolution of each run separately, their shared prefixes would create a large amount of redundant work. Since a transducer automaton with $\mathcal{A} = \{\text{hit}, \text{miss}\}$ has $O(2^r)$ runs of length $r$, this would require $O(r \cdot 2^r)$ matrix-vector multiplications. This can be made more efficient by instead performing a depth-first traversal of the trie of all runs of length $r$, keeping partial results in a stack. This reduces the number of matrix multiplications to $O(2^r)$, making an asymptotic runtime improvement that is very impactful in practice. The pseudocode for this is shown in Algorithm 3.

**Algorithm 3:** Computing reachable sets for one iteration of the bounded runs method.

1 **Function** `BoundedRuns` ($\mathcal{T}$, $z[0]$, $r$)
$\quad$ **input :** Transducer automaton $\mathcal{T}$, set of initial states $z[0]$, run length $r$
$\quad$ **output:** Mapping from locations to lists of reachable sets over time
2 $\quad R \leftarrow$ Mapping from locations to lists of reachable sets;
3 $\quad S \leftarrow$ Array of $r$ named triples $\langle z, \ell, a \rangle$;
4 $\quad i \leftarrow 1$;
5 $\quad S[i] \leftarrow \langle z[0], \ell_0, \text{action } 1 \rangle$ ; // Insert initial state, automaton location and scheduler action.
6 $\quad$ **while** $i > 0$ **do**
$\quad\quad$ /* This loop walks the trie of all runs of length $r$. */
7 $\quad\quad$ **if** $S[i]$ *contains a leaf node* **then**
8 $\quad\quad\quad R[S[i].\ell][:] \leftarrow$ Compute the hulls of all states at all time steps;
9 $\quad\quad\quad i \leftarrow i - 1$ ; // Ascend a level
10 $\quad\quad$ **else if** *the last action,* $S[i].a$*, has been tried* **then**
11 $\quad\quad\quad i \leftarrow i - 1$; // Ascend a level
12 $\quad\quad$ **else if** *no transition* $T(S[i].\ell, S[i].a)$ **then**
13 $\quad\quad\quad S[i].a \leftarrow S[i].a + 1$;
14 $\quad\quad$ **else**
15 $\quad\quad\quad S[i+1] \leftarrow \langle \mu(S[i].\ell, S[i].a) S[i].z, T(S[i].\ell, S[i].a), \text{action } 1 \rangle$ ; // Compute the next augmented state as per $S[i]$
16 $\quad\quad\quad S[i].a \leftarrow S[i].a + 1$ ; // Update current working state.
17 $\quad\quad\quad i \leftarrow i + 1$; // Descend to the next level.
18 $\quad$ **return** $R$;

The function uses the array $S$ as a stack of augmented states, automaton locations, and scheduler actions, with $i$ acting as the stack pointer. It assumes the actions $\mathcal{A}$ can be referenced by index, and begins by pushing the initial state, location, and the first action onto $S$ on Line 5. The loop on Line 6 walks the trie of all runs of length $r$. Each iteration, one of four actions is taken. At a leaf node, the hulls are computed on Line 8 and the iteration ascends a level. If the last action for a location has been tried, we ascend a level on Line 11. If the next action to try is missing from the automaton, it is skipped on Line 13. Otherwise, we compute the next augmented state and descend to the next level of the trie. Once the loop finally exits, the list of reachable sets over time for each final location is returned.

We can then run this function iteratively, computing an overapproximation of the reachable set over time. Using this, we compute an upper bound $\bar{\mathbf{d}}$ to the maximum deviation $\mathbf{d}$. Algorithm 4 lists pseudocode that implements this.

**Algorithm 4:** Bounded runs method for computing maximum deviation from a nominal trajectory.

**input :** A transducer automaton $\mathcal{T}$, set of initial states $z[0]$, nominal run $\tau_{nom}$, per-tree run length $r$, number of iterations $J$
**output:** An upper-bound $\bar{\mathbf{d}} \geq \mathbf{d}$

1 $S \leftarrow$ list of $rJ+1$ empty reachable sets;
2 $R \leftarrow \texttt{BoundedRuns}(\mathcal{T}, z[0], r)$; // Compute all (bounded) runs from the initial set $z[0]$.
3 $Q \leftarrow |L| \times |L|$ array of lists of reachable sets over time;
4 $S[0:r] \leftarrow hull_{\ell \in L}(R[\ell])$;
5 **for** $2 \leq i \leq J$ **do**
6  **forall** *locations* $\ell$ *in* $\mathcal{T}$ **do**
7  $\mathcal{T}' \leftarrow \mathcal{T}$ with $\ell_0 = \ell$;
8  $Q[\ell, :] \leftarrow \texttt{BoundedRuns}(\mathcal{T}', R[\ell][r], r)$; // Compute reachable sets over time starting from each location.
9  **forall** *locations* $\ell$ *in* $\mathcal{T}$ **do**
10  $R[\ell] \leftarrow hull_{\ell \in L}(Q[:, \ell])$; // Compute the reachable set for a given location.
11  $S[ir : ir + r] \leftarrow hull_{\ell \in L}(R[\ell])$; // Store reachable sets for the next $r$ steps.
12 **return** $\max_{1 \leq t \leq rJ} \{d_r(evol(\tau_{nom})[t], S[t])\}$;

*Algorithm 4 and its Safety Proof (Sketch):* This algorithm begins by computing all runs from the initial set $z[0]$ on Line 2. This populates $R$ with a mapping from locations to lists of reachable sets for $r$ time steps. Line 4 then computes a convex hull over all locations for each time step, storing the resulting reachable sets in $S$. The loop on Lines 5 to 11 then repeats this procedure $J-1$ times. First, each row of the matrix $Q$ is populated with reachable sets over time, starting from each state, by calling `BoundedRuns` in the inner loop on Line 6. Then, the second inner loop on Line 9 takes a hull of the reachable sets in each column of $Q$, storing the result in $R$. At the end of each iteration, the reachable sets for the next $r$ time steps are stored in $S$ in Line 11. Finally, the algorithm computes $\bar{\mathbf{d}}$ on Line 12 and returns. Correctness can be proven similarly to Algorithm 2. Because the algorithm computes the exact evolution of the reachable set from the previous time steps, then takes convex hulls to over-approximate, the reachable set remains safe. Therefore, the distance computed on Line 12 is also an over-approximation of the true deviation.

## VII. Experimental Evaluation

With several algorithms for computing upper bounds on maximum deviation (Problem 2) presented in Section VI, it is important to give some comparison of these techniques. First, we demonstrate the power given to control designers by being able to prove *safety properties* about control systems under timing uncertainties, rather than just qualitative properties such as stability, in Section VII-B. Next, we describe how our techniques can be used to find the best deadline miss handling strategy for a given control system in Section VII-C. Third, we compare our algorithms against each other, and show that they can be used to solve realistic problems. To this end, we examine both the *scalability* and *degree of overapproximation* of our approaches in Section VII-D. In the case of Algorithm 4, these factors are related: tighter approximation can be achieved at the cost of longer running time. Ultimately, we find that Algorithm 1 generally has too much overapproximation to be practical, and Algorithm 4 never produces worse bounds than Algorithm 2, though the latter can be faster in some cases.

*Implementation and Environment:* We implemented Algorithms 1 and 2 using Python[1], and Algorithm 4 using Julia[2]. We use a minimum bounding box to compute $hull(\cdot)$, and the 2-norm for $dis(\cdot)$. We next introduce the plant models and controllers, then present our experimental results.

### A. Plant Models

In this section, we present four state-space models and controllers that we will consider throughout the experimental evaluation. Three of these, the RC Network, Electric Steering, and F1Tenth Car systems, are two-dimensional; the Aircraft Pitch model is a three-dimensional system.

*RC Network:* Our first model is an RC network [32], represented in discrete time with a period of $h = 100\,\text{ms}$ by

$$x[t+1] = \begin{bmatrix} 0.5495 & 0.07240 \\ 0.01448 & 0.9332 \end{bmatrix} x[t] + \begin{bmatrix} 0.3781 \\ 0.05234 \end{bmatrix} u[t].$$

Using a one-period delay, we use LQR to compute a controller for this system given by

$$u[t] = \begin{bmatrix} 0.09772 & 0.2504 & 0.07805 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}.$$

*Electric Steering:* Our second model is an automotive electric steering system based on a permanent magnet synchronous motor [5]. The plant model represented in discrete time, with a control period of $10\,\mu\text{s}$, is given by

$$x[t+1] = \begin{bmatrix} 0.996 & 0.075 \\ -0.052 & 0.996 \end{bmatrix} x[t] + \begin{bmatrix} 0.100 & 0.003 \\ -0.003 & 0.083 \end{bmatrix} u[t].$$

This system is open-loop stable, having poles inside the unit circle at $0.9957 \pm 0.0626i$. A proportional-integral (PI) controller is designed for this system in [5], but the $K$ matrix given therein appears to make the system unstable. We thus use a new controller given by

$$u[t] = \begin{bmatrix} 0.9067 & 0.07384 & 0 & 0 \\ 0.01041 & 0.9685 & 0 & 0 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}.$$

Note that we have designed this controller assuming no sensing-to-actuation delay, as $K_u$ is a zero matrix. This is done to stress test our techniques with a non-optimal controller.

*Aircraft Pitch:* Our third model is an aircraft pitch model [33], describing the effects of an airplane's elevator deflection angle on the pitch angle. This model is given in discrete time with a period of $h = 100\,\text{ms}$ by

$$x[t+1] = \begin{bmatrix} 0.9654 & 5.457 & 0 \\ -0.001338 & 0.9545 & 0 \\ -0.003842 & 5.544 & 1 \end{bmatrix} x[t] + \begin{bmatrix} 0.02842 \\ 0.001969 \\ 0.005641 \end{bmatrix} u[t].$$

The system's output is simply the pitch angle, represented by the state variable $x_3$. Assuming a sensor-to-actuator delay of $h$, we use LQR to compute a controller given by

$$u[t] = \begin{bmatrix} -0.8551 & 179.2 & 5.999 & 0.3238 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}.$$

[1]https://github.com/bineet-coderep/Jittery-Scheduler
[2]https://github.com/Ratfink/ControlTimingSafety.jl

*F1Tenth Car:* Our final model captures the motion of an F1Tenth [34] model car. At its core, this uses a standard bicycle dynamics model, which is a non-linear system. The controller aims to keep the car driving straight in the positive $x$ direction on a plane. As our algorithms are not able to handle non-linear dynamics, we linearize the model around this condition, and discretize the resulting linear dynamics with a period of $20\,\mathrm{ms}$, arriving at the model

$$x[t+1] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.13 \\ 0 & 0 & 1 \end{bmatrix} x[t] + \begin{bmatrix} 0 \\ 0.02559 \\ 0.3937 \end{bmatrix} u[t] + \begin{bmatrix} 0.13 \\ 0 \\ 0 \end{bmatrix}.$$

Here, the state variables $x_1$ and $x_2$ represent the $x$ and $y$ coordinates of the car, and $x_3$ is the heading angle. Since $x_1$ simply increases proportionally to time in the linearized model, we can remove it for our analysis, leaving the dynamics

$$x[t+1] = \begin{bmatrix} 1 & 0.13 \\ 0 & 1 \end{bmatrix} x[t] + \begin{bmatrix} 0.02559 \\ 0.3937 \end{bmatrix} u[t].$$

The controller for this plant model computes the steering angle of the front wheels. A linear controller for our control objective is given in [35], which we adapt to our setup as

$$u[t] = \begin{bmatrix} 0.2935 & 0.4403 \end{bmatrix} x[t-1].$$

Having introduced our plant models, we next demonstrate our algorithms' ability to verify safety properties of control systems in the presence of timing uncertainties via a case study on the F1Tenth Car model.

### B. Checking Safety Properties

In this section, we illustrate the power of checking deviation using our techniques from Section VI, using a concrete example with the F1Tenth Car model and controller. Consider the scheduling of the controller on a processor, with a period of $20\,\mathrm{ms}$. Some other tasks on the system are provisioned for less than the worst case, so the control task may sometimes miss a deadline. When this occurs, it follows the *Zero&Kill* strategy, *i.e.*, the job is killed and a control input of 0 is applied. The scheduler guarantees that the control task will never miss two deadlines in a row, following the $\overline{\langle 1 \rangle}$ weakly-hard constraint.

Given this information, we can apply the method of [5] to compute an upper bound on the joint spectral radius to check if the system is stable. The upper bound computed is $0.944$, which being less than 1, indicates that system is stable. This is important for control designers to know, since instability would mean the car is unable to follow the intended path, and would certainly not be acceptable. However, this says nothing of the car's possible trajectories, which could potentially include unsafe states. For instance, the car may swerve about the intended path, colliding with an obstacle. Thus, more information is required than stability alone.

Consider now that the initial state of the plant is $x[0] = \left[\begin{smallmatrix} 0.1 \\ 0 \end{smallmatrix}\right]$, *i.e.*, the car is initially $0.1\,\mathrm{m}$ above the $x$ axis, with a heading angle of $0°$. The controller will drive the state to $\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$, ideally along a single *nominal trajectory* in the absence of deadline misses. Following our assumption that timing uncertainties may lead to deadline misses, but no more than one consecutively, the actual trajectory may vary from nominal.

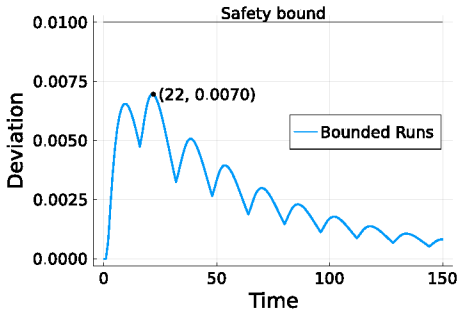



Fig. 5. Deviation bounds from Algorithm 4 with one consecutive deadline miss for the example in Section VII-B.

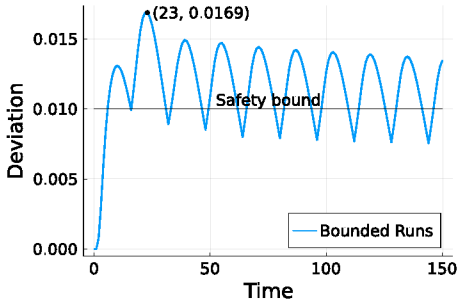



Fig. 6. Deviation bounds from Algorithm 4 with two consecutive deadline misses for the example in Section VII-B.

Let us assume that the designer must ensure that the car's $y$ coordinate never deviates from the nominal trajectory by more than $0.01\,\mathrm{m}$. We next use our algorithms from Section VI to check if this constraint holds.

We begin by using the ULS method (Algorithm 1) to calculate a bound on the maximum deviation. This algorithm returns in $3.83\,\mathrm{s}$, but the deviation bound diverges over time: the maximum is 887 at time $t = 150$. Thus, the safety property is not verified, but may still hold. We proceed to try the recurrence relation method (Algorithm 2), but the deviation bound again diverges. To compute a useful bound, we use the bounded runs iteration method (Algorithm 4), with $r = 16$. The resulting deviation bound at each sampling instant is shown in Fig. 5. Observe that the deviation bound has periodic "spikes" every 16 time steps, resulting from the box hulls taken every iteration. The maximum deviation of $0.0070$ occurs at time $t = 22$. Thus, the safety property holds, and the designer accepts the timing uncertainty from the scheduler.

Later, more functionality is added to the real-time system via several new tasks. As a result, the control task may now miss two consecutive deadlines. This increases the upper bound on the joint spectral radius to $0.959$, meaning that the system is still stable. The deviation bounds for this case are shown in Fig. 6. The bounded runs iteration method, run with $r = 16$, now gives the bound $0.0169$ at time $t = 23$. Thus, we can no longer guarantee that the control system is safe, so the engineer must either redesign the controller, or raise the control task's priority to reduce the number of consecutive deadline misses that are possible. We stress, however, that this does not imply existence of a run violating the safety bound. This is because all our algorithms compute *safe upper bounds* to the deviation, rather than the true maximum deviation.

This example illustrates the importance of analyzing safety properties of control systems that may experience timing un-

TABLE I
NUMBER OF POINTS WHERE EACH STRATEGY MINIMIZES DEVIATION

| Miss Strategy | RC Net | Steering | Aircraft | F1Tenth |
|---|---|---|---|---|
| *Hold&Kill* | **91** | 95 | **100** | **100** |
| *Zero&Kill* | 67 | **100** | 0 | 0 |
| *Hold&Skip-Next* | 0 | 0 | 0 | 0 |
| *Zero&Skip-Next* | 0 | 0 | 0 | 0 |

certainty. The system is stable in both cases, but this does not guarantee that its deviation will be acceptable when deadline misses are possible. In the following sections, we examine the ability of our methods to solve realistic problems, in terms of both deviation bounds and scalability.

### C. Determining the Best Miss Handling Strategy

In the previous section, we demonstrated the value to control designers of being able to compute a bound on the maximum deviation of a control system under deadline misses. Our techniques for solving Problem 2 allow us to determine these bounds for a given initial state and deadline miss handling strategy. Different choices of this strategy will have an impact on deviation, and may impact the safety of the controller. Designers may then wish to use the strategy giving the lowest deviation. Unfortunately, the deviation also depends on the initial state, making this choice less immediately clear.

As observed in Section V, using an absolutely homogeneous metric function allows us to determine deviation bounds for a unit vector initial state $x[0]$ (*i.e.*, $\|x[0]\|_2 = 1$), then use this to easily determine bounds for any state $ax[0]$. We can use this observation to determine the best strategy for any initial state as follows. First, we create a set of unit vector initial states $S \subset \mathbf{R}^n$. These vectors may be drawn uniformly, at random, or in some directions of interest for the control system under consideration. Next, for each state in $S$, we compute deviation bounds for every deadline miss handling strategy using our algorithms from Section VI. We then compare the rankings of the miss handling strategies *vis-à-vis* deviation, and determine the winner for the plurality of states in $S$.

We implemented this technique for the bounded runs iteration method (Algorithm 4). With many initial states in the set $S$, execution could take several days of computation time, but since each run of Algorithm 4 is independent, it is easily parallelized. We ran this implementation for each of our example systems, with a maximum of two consecutive deadline misses, using 100 initial points each. The results are shown in Table I. Note that in some cases, two methods gave the same minimum value of deviation, so the total of each column may be greater than 100. For the RC Network, Aircraft Pitch, and F1Tenth Car models, the *Hold&Kill* strategy gave the lowest deviation bound for the most initial points, whereas *Zero&Kill* was the best strategy found for the Electric Steering model. The *Skip-Next* strategies never gave the lowest deviation bound for any of our example systems. These strategies may prove more useful in systems that are open-loop unstable, unlike our examples. We must stress that these results are dependent on the algorithm used to solve Problem 2, as well as the exact set of initial states. If the exact maximum deviation was known for each scenario under consideration, the results may be different from those shown here. Despite this limitation, this approach is likely still of value to control designers, as it offers new insight into which deadline miss handling strategy works best for an application that may experience timing uncertainties.

### D. Scalability

To further illustrate the value of our methods, we next show their scalability by evaluating them in a variety of situations. We first show how the time taken varies between algorithms, for each of our plant models and miss strategies. Next, we address the runtime growth when varying time horizons, and varying the number of behaviors allowable in the transducer automaton. Finally, we closely examine the bounded runs algorithm, considering how the per-tree run length parameter affects runtime and tightness of the deviation bound.

*Scalability to different systems and strategies:* We conducted a set of experiments to fairly compare our algorithms across the plant models in Section VII-A. To evaluate this, we held the initial state constant across systems as $x_1 = 10$ and $x_2 = 10$, with the remaining state variables set to 0. In all cases, we considered the $\overline{\langle 3 \rangle}$ constraint, and a time horizon $H = 150$. The results are shown in Table II. Configurations where the deviation bound never decreased over the time horizon, *i.e.*, the analysis diverged, are shown by "——".

The ULS method (Algorithm 1) only converged on a useful bound for the RC Network model and the *Zero&Kill* strategy. In all other cases, it diverged, though the longest time taken was only 36.5 s. The generalized recurrence relation method (Algorithm 2) did better, producing bounds for the RC Network model with every strategy considered, but still diverged for the other plant models. The only algorithm that produced bounds for all models was the bounded runs iteration method (Algorithm 4). This is thanks to its per-tree run length parameter $r$, allowing a tradeoff between analysis precision and computation time. We will examine this in more detail later. For several configurations, this algorithm required such a large $r$ to avoid divergence that it exceeded a time limit of 1 h. However, for all other cases, we were able to find a bound, and the $r$ value used is given in parentheses.

*Runtime growth when varying time horizon:* To show the effects of the time horizon on the runtime of our algorithms, we ran all systems with the RC Network model with at most three consecutive misses for 100, 300, and 1000 time steps. The results are shown in Table III. Both Algorithms 2 and 4 can be seen to scale linearly in the number of time steps, as expected. However, the running time of Algorithm 1 appears to grow exponentially due to the behavior of the $forward$ function. The algorithm did not complete for 1000 steps under the *Skip-Next* strategies due to a floating-point overflow resulting from divergent deviation bounds. The time required for the other algorithms is much lower in all cases, making these more attractive from an execution time perspective.

*Number of behaviors allowable:* The structure of a transducer automaton affects the number of behaviors that are allowable for a scheduler, modeled as the language of input strings recognized by the automaton. As the number of permissible behaviors grows, *e.g.*, by allowing more consecutive deadline misses, the runtime of Algorithms 2 and 4 is expected

TABLE II
MAXIMUM DEVIATION BOUNDS, TIME STEP WHERE THEY OCCURRED, AND COMPUTATION TIME FOR THREE CONSECUTIVE MISSES AND 150 STEPS

| Algorithm | Miss Strategy | RC Network | Electric Steering | Aircraft Pitch | F1Tenth Car |
|---|---|---|---|---|---|
| Uncertain Linear Systems (Algorithm 1) | *Hold&Kill* | ——, 4.07 s | ——, 8.89 s | ——, 7.00 s | ——, 3.96 s |
| | *Zero&Kill* | **1.97**, 4, 4.08 s | ——, 8.32 s | ——, 6.99 s | ——, 3.76 s |
| | *Hold&Skip-Next* | ——, 16.8 s | ——, 22.3 s | ——, 36.5 s | ——, 16.6 s |
| | *Zero&Skip-Next* | ——, 16.8 s | ——, 28.0 s | ——, 36.4 s | ——, 16.6 s |
| Generalized Recurrence Relations (Algorithm 2) | *Hold&Kill* | **1.90**, 4, 0.52 s | ——, 0.78 s | ——, 0.79 s | ——, 0.50 s |
| | *Zero&Kill* | **1.90**, 4, 0.50 s | ——, 0.75 s | ——, 0.77 s | ——, 0.49 s |
| | *Hold&Skip-Next* | **1.90**, 4, 0.95 s | ——, 1.28 s | ——, 1.61 s | ——, 0.92 s |
| | *Zero&Skip-Next* | **1.90**, 4, 0.93 s | ——, 1.26 s | ——, 1.59 s | ——, 0.91 s |
| Bounded Runs Iteration (Algorithm 4) | *Hold&Kill* | **1.90**, 4, 0.68 s (4) | **12.37**, 4, 30.8 s (11) | **82.37**, 6, 528 s (19) | **6.01**, 27, 1569 s (20) |
| | *Zero&Kill* | **1.90**, 4, 0.78 s (4) | **13.63**, 8, 685 s (16) | **161.64**, 10, 509 s (19) | *timed out* ($> 1$ h) |
| | *Hold&Skip-Next* | **1.90**, 4, 2.27 s (4) | **12.38**, 5, 2845 s (16) | *timed out* ($> 1$ h) | *timed out* ($> 1$ h) |
| | *Zero&Skip-Next* | **1.90**, 4, 2.37 s (4) | **13.75**, 8, 2864 s (16) | *timed out* ($> 1$ h) | *timed out* ($> 1$ h) |

TABLE III
RUNNING TIME OVER VARYING TIME HORIZON (100, 300, 1000 STEPS), AT MOST THREE CONSECUTIVE MISSES

| Miss Strategy | Algorithm 1 | Algorithm 2 | Algorithm 4 |
|---|---|---|---|
| *Hold&Kill* | 1.9, 15.7, 172 | 0.36, 1.0, 3.6 | 0.096, 0.31, 1.1 |
| *Zero&Kill* | 2.0, 16.4, 178 | 0.35, 1.0, 3.5 | 0.101, 0.34, 1.1 |
| *Hold&Skip-Next* | 7.8, 72.0, —— | 0.66, 1.9, 6.8 | 0.47, 1.4, 4.3 |
| *Zero&Skip-Next* | 8.1, 74.6, —— | 0.65, 1.9, 6.5 | 0.47, 1.4, 4.6 |

TABLE IV
RUNNING TIME OVER VARYING MISSES FOR 150 TIME STEPS, HOLD&SKIP-NEXT

| Algorithm | $\overline{\langle 2 \rangle}$ | $\overline{\langle 4 \rangle}$ | $\overline{\langle 8 \rangle}$ | $\overline{\langle 16 \rangle}$ |
|---|---|---|---|---|
| Algorithm 2 | 0.72 s | 1.20 s | 2.12 s | 3.92 s |
| Algorithm 4 | 0.43 s | 0.87 s | 1.7 s | 3.4 s |

to increase. Algorithm 1 is immune to this effect, since it overapproximates the dynamics matrices of the transducer automaton, ignoring its locations and transitions. To quantify the effect of the number of behaviors on our algorithms' runtime, we ran Algorithms 2 and 4 on the RC Network model using the *Hold&Skip-Next* strategy for 150 time steps, with the weakly hard constraints $\overline{\langle 2 \rangle}$, $\overline{\langle 4 \rangle}$, $\overline{\langle 8 \rangle}$, and $\overline{\langle 16 \rangle}$. As indicated in Table IV, the two algorithms scale similarly in this parameter.

*Varying run length:* The bounded runs iteration method (Algorithm 4) offers a parameter $r$ that controls the number of time steps between bounding box overapproximations of reachable sets. Since the subroutine Algorithm 3 is exponential in this parameter, there is a tradeoff between accuracy of the deviation bound, and required execution time. It is thus pertinent to discuss the choice of this parameter's value.

Because the runtime is exponential in $r$, it may be best to simply use the smallest value possible. This approach was used for the Electric Steering, Aircraft Pitch, and F1Tenth Car models in Table II. A simple linear search determines this minimum $r$ value without wasting much computation time.

It can also be noted that there is a limit to the *highest* value of $r$ that produces any benefit. Since the goal of Problem 2 is to find a bound on the *maximum* deviation, no better bound could be found by setting $r$ greater than the time step at which the bound $\bar{\mathbf{d}}$ occurs. This approach was used for the RC Network in Table II, and can be quite effective for systems where the maximum deviation occurs early. This observation does not imply, however, that there is never a reason to use a larger $r$ than this time step. Take for instance the Electric Steering and Aircraft Pitch models, reported in Table II. It was necessary to increase $r$ to the values listed in parentheses to prevent the reachable set from diverging. However, once this bound was computed, there is no need to use a *larger* value of $r$, as this would only take longer to give the same $\bar{\mathbf{d}}$.

In some cases however, neither of these strategies may be satisfactory, and so the control designer must accept some tradeoff in analysis accuracy and runtime. For example, the F1Tenth Car in Table II gave a deviation bound of $6.01$ at time $27$, greater than $r = 20$. Thus, using a larger $r$ may produce a better deviation bound. However, the execution time is already large, so a designer may decide to accept this result, especially if it meets the required safety bound.

### E. Use of the Algorithms

Having analyzed the scalability of our algorithms, we now provide some general recommendations based on running time and tightness of deviation bounds. When using our algorithms in practice, it is likely best for an engineer to first use Algorithm 2, which sometimes gave good bounds and never required more than 2 seconds in our experiments. Engineers may then switch to Algorithm 4 if Algorithm 2 diverges (*i.e.*, the greatest deviation bound occurs at the end of the time horizon). Due to its poor performance, only finding one bound in Table II, Algorithm 1 is not generally recommended. We present it primarily for its simplicity, which makes it seem like a viable method for solving Problem 2. However, the high amount of over-approximation resulting from combining the matrices of a transducer automaton makes it typically not useful. We note that all our experiments are limited by not examining *how great* of an over-approximation our algorithms produce. This would unfortunately be intractable, as noted in Section V, but since the bounds produced are safe, our methods are still valuable to designers. Finally, it is necessary to discuss the time required for our various algorithms. While a very large run length parameter $r$ is sometimes needed for Algorithm 4 (up to 20 in Table II), leading to long execution times as described in Section VII-D, this is likely not a limitation in practice. Because the analysis is performed offline, running times around an hour as seen in our experiments are likely acceptable, especially since our techniques may reduce the need for lengthy testing cycles.

## VIII. Conclusions and Future Work

Unlike most previous studies on modeling the impact of implementation platform uncertainties on control performance, which focus on stability, in this paper we considered *quantitative* safety properties. In particular, we proposed three approximation techniques to bound the maximum deviation between a nominal behavior and any possible system trajectory resulting from platform timing uncertainties. Our evaluation on four system models shows that we can overcome the computational challenge typically associated with the reachability analysis necessary to analyze such quantitative safety properties. This allows us to choose control and deadline overrun handling strategies for each system to quantitatively optimize system safety. To the best of our knowledge, such a characterization of deadline overrun handling strategies was not studied before.

However, we also see that depending on the system and scheduling strategy, the error accumulated in our approximation strategies may grow very large—to the extent that the estimated deviation becomes unbounded. The focus of our future work will be to reduce such wrapping error. One strategy is to create multiple convex (box) hulls, one for each small cluster of system states, instead of a single one to approximate all reachable states. This could reduce the degree of over-approximation, at the cost of an increase in computational complexity. We also plan to study sampling techniques to provide probabilistic estimates of deviation while improving the scalability of the analysis. In particular, we will explore the use of Jeffreys's Bayes factor testing [36] to obtain deviation bounds with probabilistic guarantees.

**Acknowledgements:** This research has been partially supported by the NSF grant 2038960, AFOSR grant FA9550-19-1-0288, and by an Amazon research award.

## References

[1] S. Chakraborty, M. A. A. Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Des. Test*, vol. 33, no. 4, pp. 92–108, 2016.

[2] T. Sehnke, D. Schwarzmann, M. Schultalbers, and R. Ernst, "Temporal properties in automotive control software," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2017.

[3] 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.

[4] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Breaking silos to guarantee control stability with communication over ethernet TSN," *IEEE Des. Test*, vol. 38, no. 5, pp. 48–56, 2021.

[5] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein, "Control-System Stability Under Consecutive Deadline Misses Constraints," in *32nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2020.

[6] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin, "DMAC: deadline-miss-aware control," in *31st Euromicro Conference on Real-Time Systems (ECRTS)*, 2019.

[7] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. D. Natale, "Beyond the weakly hard model: Measuring the performance cost of deadline misses," in *30th Euromicro Conference on Real-Time Systems (ECRTS)*, 2018.

[8] D. Soudbakhsh, L. T. X. Phan, A. M. Annaswamy, and O. Sokolsky, "Co-design of arbitrated network control systems with overrun strategies," *IEEE Trans. Control. Netw. Syst.*, vol. 5, no. 1, pp. 128–141, 2018.

[9] E. P. van Horssen, A. R. B. Behrouzian, D. Goswami, D. Antunes, T. Basten, and W. P. M. H. Heemels, "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses," in *15th European Control Conference (ECC)*, 2016.

[10] S. Linsenmayer and F. Allgöwer, "Stabilization of networked control systems with weakly hard real-time dropout description," in *56th IEEE Annual Conference on Decision and Control (CDC)*, 2017.

[11] D. Liberzon, *Switching in systems and control*. Springer, 2003.

[12] W. Zhang, M. S. Branicky, and S. M. Phillips, "Stability of networked control systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 84–99, 2001.

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

[14] M. C. F. Donkers, W. P. M. H. Heemels, D. Bernardini, A. Bemporad, and V. Shneer, "Stability analysis of stochastic networked control systems," *Automatica*, vol. 48, no. 5, pp. 917–925, 2012.

[15] K. Okano, M. Wakaiki, G. Yang, and J. P. Hespanha, "Stabilization of networked control systems under clock offsets and quantization," *IEEE Trans. Automat. Contr.*, vol. 63, no. 6, pp. 1618–1633, 2018.

[16] J. P. Hespanha, "Modeling and analysis of networked control systems using stochastic hybrid systems," *Annual Reviews in Control*, vol. 38, no. 2, pp. 155–170, 2014.

[17] A. Masrur, S. Drössler, T. Pfeuffer, and S. Chakraborty, "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.

[18] S. Tseng and J. Anderson, "Deployment architectures for cyber-physical control systems," in *American Control Conference (ACC)*, 2020.

[19] J. Anderson, J. C. Doyle, S. H. Low, and N. Matni, "System level synthesis," *Annu. Rev. Control.*, vol. 47, pp. 364–393, 2019.

[20] Y. Wang, N. Matni, and J. C. Doyle, "A system-level approach to controller synthesis," *IEEE Trans. Autom. Control.*, vol. 64, no. 10, pp. 4079–4093, 2019.

[21] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Trans. Control. Syst. Technol.*, vol. 22, no. 6, pp. 2337–2345, 2014.

[22] M. B. G. Cloosterman, N. van de Wouw, W. P. M. H. Heemels, and H. Nijmeijer, "Stability of networked control systems with uncertain time-varying delays," *IEEE Trans. Automat. Contr.*, vol. 54, no. 7, pp. 1575–1580, 2009.

[23] H. Lin and P. J. Antsaklis, "Stability and stabilizability of switched linear systems: A survey of recent results," *IEEE Trans. Autom. Control.*, vol. 54, no. 2, pp. 308–322, 2009.

[24] S. Chakraborty, J. H. Anderson, M. Becker, H. Graeb, S. Halder, R. Metta, L. Thiele, S. Tripakis, and A. Yeolekar, "Cross-layer interactions in CPS for performance and certification," in *Design, Automation & Test in Europe Conference (DATE)*, 2019.

[25] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-aware integrated routing and scheduling for control applications in Ethernet networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[26] I. Saha, S. K. Baruah, and R. Majumdar, "Dynamic scheduling for networked control systems," in *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2015.

[27] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[28] R. Blind and F. Allgöwer, "Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process," in *54th IEEE Conference on Decision and Control (CDC)*, 2015.

[29] B. Ghosh and P. S. Duggirala, "Robust reachable set: Accounting for uncertainties in linear dynamical systems," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, Oct. 2019.

[30] R. Lal and P. Prabhakar, "Bounded error flowpipe computation of parameterized linear systems," in *Proceedings of the 12th International Conference on Embedded Software (EMSOFT)*, 2015.

[31] R. Farhadsefat, J. Rohn, and T. Lotfi, "Norms of interval matrices," *Institute of Computer Science, Academy of Sciences of the Czech Republic, Tech. Rep*, 2011.

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

[33] 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

[34] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," *Proceedings of Machine Learning Research*, vol. 123, 2020.

[35] K. N. Murphy, "Analysis of robotic vehicle steering and controller delay," in *Fifth International Symposium on Robotics and Manufacturing (ISRAM)*, 1994.

[36] R. Kass and A. Raftery, "Bayes factors," *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 773–795, 1995.