Statistical Hypothesis Testing of Controller Implementations Under Timing Uncertainties

Bineet Ghosh*, Clara Hobbs*, Shengjie Xu*, Parasara Sridhar Duggirala*,
James H. Anderson*, P. S. Thiagarajan†, Samarjit Chakraborty*
*The University of North Carolina at Chapel Hill, NC, USA †Chennai Mathematical Institute, India

***Abstract*—Software in autonomous systems, owing to performance requirements, is deployed on heterogeneous hardware comprising task specific accelerators, graphical processing units, and multicore processors. But performing timing analysis for safety critical control software tasks with such heterogeneous hardware is becoming increasingly challenging. Consequently, a number of recent papers have addressed the problem of *stability analysis* of feedback control loops in the presence of timing uncertainties (*cf.*, deadline misses). In this paper, we address a different class of safety properties, *viz.*, whether the system trajectory deviates too much from the nominal trajectory, with the latter computed for the *ideal* timing behavior. Verifying such *quantitative* safety properties involves performing a reachability analysis that is computationally intractable, or is too conservative. To alleviate these problems we propose to provide statistical guarantees over behavior of control systems with timing uncertainties. More specifically, we present a Bayesian hypothesis testing method based on Jeffreys's Bayes factor test that estimates deviations from a nominal or ideal behavior. We show that our analysis can provide, with high confidence, tighter estimates of the deviation from nominal behavior than using known reachability based methods. We also illustrate the scalability of our techniques by obtaining bounds in cases where reachability analysis fails to converge, thereby establishing the former's practicality.**

***Index Terms*—Control, reachability, real-time systems, safety, weakly-hard systems, statistical hypothesis testing**

I. Introduction

Providing verifiable assurances for autonomous systems is a challenge that has attracted considerable scientific interest [1]–[3]. Traditionally, this involved designing suitable *feedback control loops* and formally verifying their correctness. An emerging challenge in providing such assurances for current generation autonomous systems is providing timing guarantees of the increasingly complex on-board hardware platforms utilizing machine learning (ML) components. Presently, these consist of multiple multicore processors and hardware accelerators like GPUs and FPGAs. As a result, the timing behavior of control software [4] running on them can be highly variable because of complex interference patterns between heterogeneous components. Analyzing the timing behavior of such software consists of two main steps: 1) determining the worst-case execution time (WCET) of the code [5], and 2) using schedulability analysis to validate the timing constraints (or deadlines) assuming which the feedback controllers have been designed. Unfortunately, there is now widespread consensus that on modern hardware platforms, safe WCET estimates cannot be guaranteed without excessive pessimism [6], [7]. This situation is further aggravated by ML components for sensor (camera, radar, lidar) processing that incur content-dependent processing times. Hence, unless very pessimistic WCET bounds are acceptable—which makes designs highly over-provisioned and impractical—a certain non-determinism in the timing behavior of software is unavoidable. This raises the question: *"What performance guarantees can be provided for feedback control loops subjected to certain non-deterministic timing behavior?"* There are various incarnations of this question and the one most widely studied, particularly within *networked control systems*, asks how to ensure *stability* in the presence of uncertainties in network behavior such as delays and dropped packets [8]–[12]. Instead of a *qualitative* property like stability, **in this paper we ask whether *quantitative* properties, such as safety specification over a bounded time horizon, hold in the presence of timing uncertainties**.

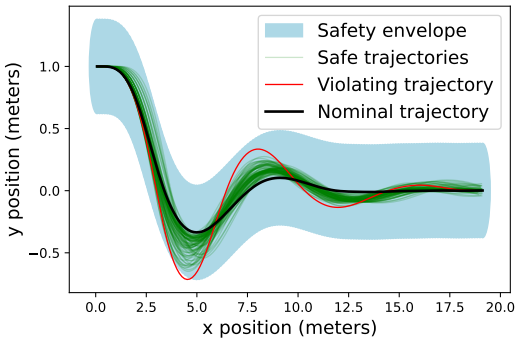


Fig. 1. Deviation in the path of an F1Tenth car due to timing uncertainty.

A. F1Tenth Example

As an example, consider Fig. 1. It shows the trajectory of a lane-following controller for an F1Tenth [13] model car. The car's steering angle and velocity are computed by a feedback controller, designed for the car to follow a predetermined path. Running the controller as designed, with *no* timing uncertainty, results in the *nominal* trajectory shown in black. Around this trajectory, there is a *safety envelope* shown in light blue, representing a safe space for the car to occupy without hitting any obstacles. Due to timing uncertainties in the implementation platform, the software task that computes the control inputs can miss the deadline imposed by the scheduler, resulting in deviation from the nominal trajectory. 100 such trajectories where the control task missed its deadline are shown in the figure—the green trajectories are safe, remaining within the safety envelope for the entire time horizon. However, the

trajectory shown in red deviates too far from the nominal trajectory, briefly leaving the safety envelope near $x = 4$, potentially resulting in a collision with an obstacle. This illustrates the importance of bounding deviations of control systems if timing variations in the control software may occur.

B. Our contributions and related work

Given an ideal system behavior (when the control task always meets its deadline), and a pattern of deadline hits and misses, **we want to estimate the maximum possible deviation from the ideal behavior in the presence of the allowed patterns of deadline misses** over a time horizon of length H. This involves *reachability analysis* of the states visited by the trajectories of the closed-loop system in the time interval $[0, H]$. This is however not a scalable problem and **our contribution** is a *statistical hypothesis testing* (SHT) framework, as shown in Fig. 2. In particular, we use the *Jeffreys's Bayes factor test* [14], [15] to solve our problem.

In our setup, timing behaviors of interest are specified as patterns of *hits* (the deadline is met) and *misses* (deadline is not met) and are assumed to constitute a *regular language* over the alphabet $\{\text{hit}, \text{miss}\}$. We further assume a uniform distribution over these strings of length H (the time horizon of interest). This enables us to implement an efficient sampling method based on the *Recursive RGA* algorithm [16]. However, our method is applicable to other types of languages and distributions as well, provided the runs of the system can be efficiently sampled. In the current setup, we are given a set of initial states of the system, a mathematical model of its (discrete time) dynamics, and a regular language L of strings of length H over the alphabet $\{\text{hit}, \text{miss}\}$. Our goal is to estimate an upper bound $\mathbf{d}_{ub}$ on the deviation of the trajectory induced by any string in L from the nominal trajectory induced by the string consisting of only hits (the ideal timing behavior).

Proposed SHT framework: In our statistical hypothesis testing framework in Fig. 2, to test if a given $\mathbf{d}_{ub}$ is an upper bound for the maximum deviation, our null hypothesis H_0 will assert that with *at most* probability c, a randomly chosen trajectory will have a deviation bounded by $\mathbf{d}_{ub}$ while the alternative hypothesis H_1 will assert that with *at least* probability c, a randomly chosen run will have a deviation that is bounded by $\mathbf{d}_{ub}$. We then use the Jeffreys's Bayes factor test [14] to decide between these two hypotheses. An important consequence of our test is, when the samples we have drawn do not support the alternative hypothesis, they will contain a *counterexample* with a deviation that exceeds the current value of $\mathbf{d}_{ub}$. This will lead to the next iteration of hypothesis testing based on a new, larger $\mathbf{d}_{ub}$. In this sense, our method is driven by a *counterexample guided refinement strategy* to eventually accept the alternative hypothesis (see Fig. 2). When we do so, in addition to the probabilistic guarantee c, we can also bound the type I error rate, (*i.e.*, the probability of inferring the alternative hypothesis when in fact the null hypothesis holds) while type II errors [14] are not relevant in our setting.

Related work: Our work has been influenced by a recent work [17] (and a number of preceding ones on the related

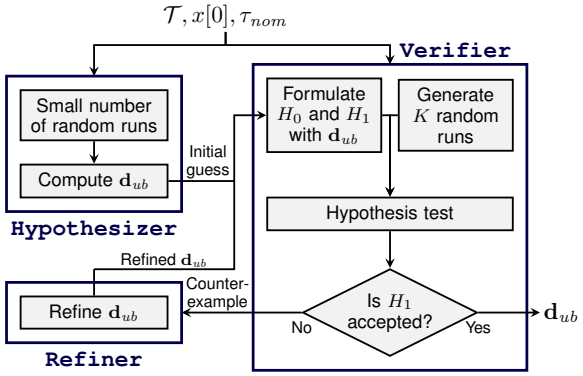


Fig. 2. Proposed statistical hypothesis testing approach.

problem [18]–[20]) that studied how deadline misses may be handled on an implementation platform and what impact it has on control performance; specifically, stability. The strategies studied in [17] 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. We instead study the impact these policies have on the maximum deviation that a trajectory of the closed-loop system can incur over a finite time horizon relative to the nominal trajectory (with no deadline misses).

A sampling based statistical method is also used in [21] to estimate the worst case (*first*) deadline miss probability of tasks scheduled under a static priority scheme in a uniprocessor setting. Since the problem they tackle is quite different from ours, we shall just compare here the two statistical methods. Loosely speaking, in [21], for a given task, the required confidence level δ and the allowed probability to fail ϵ are first fixed. This determines s, the number of runs of the system to be sampled. Depending on the number of samples that are "successes" (*i.e.*, the run encounters a first deadline miss of the task) a probability interval $\ell \leq p \leq r$ is computed such that $|l - r| \leq \delta$ and $P(\ell \leq p \leq r) \geq 1 - \epsilon$. This is an efficient and simple method that scales well. However, in this method, both the number of successes and failures play a crucial role in determining the statistical strength of the test. Hence, it is not clear how this method can be efficiently ported to our setting, primarily due to the following reasons: (i) The deviation bound must be iteratively explored and confidence intervals have to be estimated at each iteration, until the user required confidence is achieved. (ii) This method allows for failure—in our context, failure implies a safety violation.

The related domain of *statistical model checking* [22] often uses sequential hypothesis testing methods such as SPRT (sequential probability ratio test). We have instead chosen the Bayesian hypothesis testing approach, since the test requires the generation of a *fixed* number of samples, which can be done efficiently in our setting. A thorough survey of various statistical model checking methods can be found in [23].

Modeling the impact of *network* uncertainties in the form of time-varying or stochastic delays on control performance has also been studied in [24]–[26]. Here again, the focus

is on *stability*. Finally, a number of recent papers have addressed various aspects of the control/architecture *co-design* problem [27]–[34] in domains such as automotive software design [35], [36]. However, there has been comparatively less work on the use of statistical techniques for the verification of control systems [15], although they perform much better than approximation techniques, as we show in this paper.

Salient features of the proposed method: To conclude this section, we note that the intrinsic nature of hypothesis testing ensures that the number of samples to be drawn depends only on the required strength of the test and not on factors like the length of time horizon H, or the scheduling policy that results in deadline hits and misses. Furthermore, while we characterize *timing uncertainty* using a language of deadline hit/miss patterns, our scheme can be extended to other fine grained types of timing uncertainty as well such as task completion times. It is worth pointing out that verifying quantitative safety properties for control systems is harder than stability, a qualitative safety property. This is because stability analysis can lean on techniques such as the existence of a *Lyapunov function* and on results from stability analysis of switched systems [11].

Organization: The remainder of this paper is organized as follows. Section II explains our system model, followed by the definition of the problem and a discussion of deterministic approaches to the solve the problem in Section III. Our hypothesis testing based framework in presented in Section IV. Before describing our experimental results in Section VI, we illustrate our approach on an example in Section V, where we also provide a rationale for choosing Jeffreys's Bayes factor over other methods. We finally conclude by outlining some directions for future work.

II. System Modelling

We study the state feedback control of discrete time-invariant linear dynamical systems of the form $x[t+1] = Ax[t] + Bu[t]$, where $A \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{n\times p}$. The control input u is computed by a periodic real-time task running on a processor, and is assumed to be of the form $u[t] = Kx[t-1]$, where $K \in \mathbb{R}^{p\times n}$. This can alternately be represented using an augmented state space [37] as $z[t] = \begin{bmatrix} x[t]^T & u_a[t-1]^T \end{bmatrix}^T$, giving the following model:

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

Here, we denote the two blocks of the feedback gain matrix K providing feedback from each of the vectors x and u as $K_x \in \mathbb{R}^{p\times n}$ and $K_u \in \mathbb{R}^{p\times p}$, respectively. This augmented form permits standard controller design techniques such as *linear-quadratic regulator* (LQR) [38]. Further, it allows the plant and controller to be represented as a single *dynamics matrix*.

In order to model the system behavior under a sequence of deadline hits and misses, we use standard techniques, similar to those in [17]. 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 as when $x[t]$ is to be read. If the job completes on time, the control input is computed. If the job misses its deadline, several different actions can be taken—described below—both for generating the missing control input and for handling the task that has missed its deadline [17].

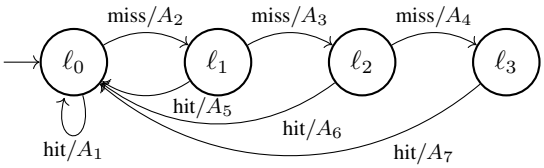


Fig. 3. Transducer automaton capturing 3 maximum consecutive misses.

We specify the behavior of the scheduler as an automaton that maps the allowed patterns of hits and misses to the accompanying plant dynamics and control inputs.

Definition 1: A *transducer automaton* $\mathcal{T}$ is a tuple $\langle L, \mathcal{A}, T, \mu, \ell_{\text{int}} \rangle$, defined as follows:

- $L = \{\ell_1, \ell_2, \ldots, \ell_m\}$: Set of automaton states.
- $\mathcal{A}$: Set of scheduler actions. $\mathcal{A} = \{\text{hit}, \text{miss}\}$
- T: The transition function, where $T : L \times \mathcal{A} \to L$. Let $\mathbb{T} = \left\{ T(\ell_i, a) = \ell_j \;\middle|\; \ell_i, \ell_j \in L, a \in \mathcal{A} \right\}$ denote the set of all transitions of the automaton.
- μ: Associates a dynamics matrix with a transition. Formally, $\mu : \mathbb{T} \to \mathbb{R}^{n\times n}$, where n is the dimension of the system under consideration.
- $\ell_{\text{int}} \in L$: Initial state of the automaton.

We sometimes refer to transducer automata as DFAs. An example of a DFA, capturing all possible behaviors with at most 3 consecutive deadline misses, is shown in Fig. 3. The labels are of the form a_i/A_i, where $a_i \in \mathcal{A}$ and A_i is the dynamics matrix associated to the label using the function μ.

Several policies for handling deadline misses, both in terms of the control input to apply and how to treat the job that has missed its deadline have been proposed in [17]. Many control input strategies can be devised, but any such strategy must be simple enough to be implemented on an actuator. The two we consider here are *Zero*, where a control input of 0 is applied, and *Hold*, where the current control input is used again until a new one can be computed. As for the handling of jobs that have missed their deadlines, there are again multiple ways to handle them. Here, we consider the *Kill* strategy, where the job is killed as soon as its deadline is passed, and the *Skip-Next* strategy, where a job is allowed to run to completion past its deadline, but no new job instance may be released until this happens. By combining a strategy for control input and real-time job scheduling, we arrive at a single *deadline miss strategy*. The resulting combination of policies, named *Zero&Kill*, *Hold&Kill*, *Zero&Skip-Next*, and *Hold&Skip-Next* in [17], will result in different closed loop dynamics under deadline misses. However the nominal behaviors will be identical.

Let the plant states $x[t]$ at time t be subsets of the metric space $(\mathbb{R}^n, dis)$, where dis is a metric on $\mathbb{R}^n$. We do not impose any assumption on dis (but use the Euclidean distance

in this paper). We note that this metric applies only to the *plant state*, not the augmented state vector used by a transducer automaton. Given $\mathcal{T}$, a possible behavior of the system is defined as a *run* consisting of an alternating sequence of locations and actions:

$$\tau = \{\ell_1, a_1, \ell_2, \ldots, a_{H-1}, \ell_H\}$$

where $\ell_1 = \ell_{\text{int}}$, $a_i \in \mathcal{A}$, and H is the time bound. Let the set of all possible runs be $\bar{\tau}$.

Next, the evolution of a run $\tau = \{\ell_1, a_1, \ell_2, \ldots, a_{H-1}, \ell_H\}$, with an initial set $x[0] \subset \mathbb{R}^n$ is denoted as $evol(\tau)$, given by

$$evol(\tau) = \big\{x_0, x[1] = A_1 x[0], x[2] = A_2 x[1], \ldots, x[H] = A_H x[H-1]\big\}.$$

Here, $A_t = \mu(a_t)$ and $x[t]$ are the plant states reached at time step t. Given evolution of a run $evol(\tau)$, let

$$evol(\tau)[t] = x[t], \qquad \text{for } 1 \leq t \leq H.$$

We now define the distance between two sets $S, R \subset \mathbb{R}^n$ using the standard Hausdorff distance, which we denote as

$$\Delta(S, R) = \max\Big\{\sup_{s\in S}\inf_{r\in R} dis(s,r),\ \sup_{r\in R}\inf_{s\in S} dis(s,r)\Big\}.$$

Given two runs $\tau_1, \tau_2 \in \bar{\tau}$, we define deviation between the two runs as the maximum Hausdorff distance between the evolution of the two runs. Formally:

Definition 2 (Deviation): The deviation between two runs τ_1 and τ_2 is given by:

$$dev(\tau_1, \tau_2) = \max_{1\leq t\leq H}\Big\{\Delta\big(evol(\tau_1)[t], evol(\tau_2)[t]\big)\Big\} \qquad (2)$$

Finally, as mentioned in the introduction, we assume a probability distribution $\mathcal{D}$ over the set of runs $\bar{\tau}$. Accordingly, by a random run we shall mean a run drawn from $\bar{\tau}$ according to $\mathcal{D}$. In the present setting, $\mathcal{D}$ is the uniform distribution. However, our analysis method is applicable to any distribution $\mathcal{D}$, provided one can effectively draw samples from $\mathcal{D}$.

III. The Problem and a Deterministic Approach

The analysis problem we wish to solve is as follows.

Problem 1: Given a transducer automaton $\mathcal{T}$, an initial set of plant states $x[0] \subset \mathbb{R}^n$, and a nominal run $\tau_{nom} \in \bar{\tau}$, compute the maximum deviation $\mathbf{d}_{max}$, where:

$$\mathbf{d}_{max} = \max\big\{dev(\tau, \tau_{nom}) \;\big|\; \tau \in \bar{\tau}\big\}. \qquad (3)$$

Assuming a time bound of H, where at each step a deadline can either be met or missed, computing the exact maximum deviation $\mathbf{d}_{max}$ will require computing the deviation of 2^H trajectories from the given nominal trajectory. This becomes intractable for realistic values of H, so more efficient methods must be used to instead *approximate* the value of $\mathbf{d}_{max}$.

To this end, there are many reachability algorithms for linear dynamical systems that can be used to **safely overapproximate** the maximum deviation. We propose one such approach here, which we call **`RS`** (*i.e.*, **r**eachable **s**et), as a baseline against which we will compare our statistical hypothesis testing approach in our experiments in Section VI.

The **`RS`** method begins by fixing a small number of time steps m. Given an axis-aligned n-dimensional interval $x[0]$ as an initial set, the algorithm proceeds iteratively, computing the reachable sets for each successive span of m sampling periods. For the first iteration, all trajectories of length m starting from the corners of the initial set $x[0]$ are computed. We store the minimum bounding box of all such trajectories at each time step, yielding our first m over-approximated reachable sets. At the end of each iteration, we group the runs by their final locations in the automaton, and compute a bounding box for each location.

Using these boxes and their corresponding locations as initial conditions, we compute the over-approximated reachable sets for the following m time steps. This procedure is iterated as many times as required to span the time horizon H (*i.e.*, $\lceil H/m \rceil$ iterations). While the runtime of the **`RS`** algorithm is exponential in the parameter m, it is linear in the number of iterations. Thus by running only a small number of time steps in each iteration, we can compute a *sound* over-approximation of the reachable sets for large time horizons efficiently. From these reachable sets, it is straightforward to compute a safe upper bound $\mathbf{d}_{ub} \geq \mathbf{d}_{max}$.

Unfortunately, this simple reachability-based approach often produces bounds on the maximum deviation that are either quite pessimistic, or require a large amount of execution time (due to a large number of steps per iteration m). Thus, in the next section, we propose the main contribution of this work, a method to *estimate* the value of $\mathbf{d}_{max}$ based on statistical hypothesis testing. As will be seen in Section VI, this method (i) typically outperforms the **`RS`** method in scalability, (ii) while producing much tighter deviation estimates.

IV. The Proposed SHT based Approach

In this section, we present a SHT based approach to solve Problem 1. Specifically, we propose a counterexample guided refinement method to compute an upper bound $\mathbf{d}_{ub}$ for $\mathbf{d}_{max}$ using statistical hypothesis testing. Consequently, our estimate $\mathbf{d}_{ub}$ will be accompanied by a statistical guarantee. We refer to $dev(\tau, \tau_{nom})$ as *deviation of the run* τ.

The inputs to our algorithm that estimates $\mathbf{d}_{ub}$ are a DFA $\mathcal{T}$ that models the behavior of scheduler, the initial set of plant states $x[0]$, a time horizon H, and the nominal behavior τ_{nom}. A network of three modules as illustrated in Fig. 2 will constitute our algorithm. Before we describe each module in detail, we provide a brief overview of the modules constituting the main approach, as follows.

`Hypothesizer`: Using the heuristics described in Section IV-A, we guess an initial $\mathbf{d}_{ub}$. This is then sent to the **`Verifier`** module.

`Verifier`: This module statistically verifies—using K randomly drawn sample runs—whether the given $\mathbf{d}_{ub}$ is indeed an upper-bound with the required probability (say, ≥ 0.99). If $\mathbf{d}_{ub}$ is accepted, it is returned as our final answer. If not, $\mathbf{d}_{ub}$ is sent to the **`Refiner`** module.

The details of the **Verifier** module are presented in Section IV-B.

Refiner: This module pads the deviation bound obtained from the counterexample with slack ϵ, it is set to be the new $\mathbf{d}_{ub}$ and sent to the **Verifier**. This will initiate the next round of hypothesis testing.

A. *Hypothesizer: Guessing an upper-bound on deviation*

To guess an initial upper bound, we observed that a small set of randomly chosen samples can sometimes provide a reasonably good representation of the actual distribution. Thus our initial guess consists of the following steps, with R and ϵ being parameters supplied by the user:

1) Let $\mathcal{S} = \{\tau_1, \tau_2, \ldots, \tau_R\}$ be a set of randomly generated runs sampled according to the given distribution over the set of strings of length H specified by the DFA.
2) Let $\mathbf{d}'_{ub} = \max_{\tau \in \mathcal{S}} \{dev(\tau, \tau_{nom})\}$.
3) Return $\mathbf{d}_{ub} = \mathbf{d}'_{ub} + \epsilon$.

We will refer to this heuristic as $\mathbf{SmallSample}(\cdot)_{[R,\epsilon]}$.

B. *Verifier and Refiner: Validating and refining the bound on deviation*

Here we describe how the hypothesis test is carried for a given value of $\mathbf{d}_{ub}$ and a parameter $c \in (0,1)$ representing the strength (probability) with which the user wishes to assert that $\mathbf{d}_{ub}$ is an upper bound. We use Bayesian hypothesis testing based on Jeffreys's Bayes factor [15]. Accordingly we first formulate the null and alternative hypotheses:

$$H_0 : \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] < c \tag{4}$$

$$H_1 : \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] \geq c \tag{5}$$

Here $\mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right]$ denotes the probability that a randomly drawn sample has a deviation that does not violate the upper bound $\mathbf{d}_{ub}$. Our goal is to determine an upper bound for which the alternative hypothesis is accepted.

First we fix a sufficiently high value (say 10^5) for the *Bayes factor* B which we will soon define. Using the Bayes factor B and probability c we shall compute K, the number of samples needed to choose between the null and alternative hypotheses, whose derivation we will detail below. We next draw K samples $X = \{\tau_1, \tau_2, \ldots, \tau_K\}$ according to the distribution assumed over the set of executions. We then check if *every* member of X satisfies the upper bound $\mathbf{d}_{ub}$. If yes, we accept the alternative hypothesis and return $\mathbf{d}_{ub}$ as the estimated upper bound. If not, we send the first counterexample encountered to the **Refiner** module. Using this counterexample, the **Refiner** module will extract a new bound estimate $\widehat{d}$ (guaranteed to exceed the current value of $\mathbf{d}_{ub}$), set it to be $\mathbf{d}_{ub}$ (after adding a padding factor ϵ) and send it to the **Verifier** for the next round of hypothesis testing.

We will illustrate the relationship between the Bayes factor B, confidence c, and the number of samples to be generated K. Let $X = \{\tau_1, \tau_2, \ldots, \tau_K\}$ be a set of random runs, such that the maximum deviation exhibited by each $\tau_i \in X$ is bounded by $\mathbf{d}_{ub}$. The probability that $\mathbf{d}_{ub}$ is valid, given the null hypothesis, is

$$\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \mid H_0\right] = \int_0^c q^K \, dq. \tag{6}$$

Similarly, the probability that $\mathbf{d}_{ub}$ is valid, given the alternative hypothesis, is

$$\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \mid H_1\right] = \int_c^1 q^K \, dq. \tag{7}$$

The *Bayes Factor* is the ratio of the above two probabilities:

$$\frac{\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \mid H_1\right]}{\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \mid H_0\right]} = \frac{1 - c^{K+1}}{c^{K+1}} \tag{8}$$

Bayes Factor is the strength of evidence favoring alternative hypothesis over null hypothesis. In Jeffreys's Bayes Factor test, we enforce that the ratio computed in Equation 8 is greater than the Bayes factor B provided by the user. Thus a sufficiently high value of the Bayes Factor indicates that the evidence favors the alternative hypothesis over the null hypothesis. Given B, we can now compute the required number of samples K as:

$$\frac{1 - c^{K+1}}{c^{K+1}} > B \iff K > -\log_c(B+1) \tag{9}$$

We call this procedure $\mathbf{Verifier}_B(H_0, H_1)$, where H_0 and H_1 are the null and alternative hypothesis, respectively. Our hypothesis testing procedure coupled to a counterexample guided refinement method is listed in Algorithm 1.

Algorithm 1: Computing upper bound on the deviation as defined in Eq. (3)

input : A transducer automaton $\mathcal{T}$, initial set $x[0]$, nominal run τ_{nom}, time bound H
output: Compute an upper bound $\mathbf{d}_{ub}$ for $\mathbf{d}_{max}$
/* we assume parameters R, ϵ, B and c are provided by the user. */
$\mathbf{d}_{ub} \leftarrow \mathbf{SmallSample}(\mathcal{T}, x[0], \tau_{nom})_{[R,\epsilon]}$; // initial guess
$H_0 \leftarrow \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] < c$; // form H_0 using Eq. (4)
$H_1 \leftarrow \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] \geq c$; // form H_1 using Eq. (5)
$res \leftarrow \mathbf{Verifier}_B(H_0, H_1)$; // perform statistical verification
if res = `True` **then**
  return $\mathbf{d}_{ub}$;
while `True` **do**
  $\mathbf{d}_{ub} \leftarrow \mathbf{Refiner}(res)_\epsilon$; // refine using the counter example
  $H_0 \leftarrow \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] < c$; // refine H_0 with new $\mathbf{d}_{ub}$
  $H_1 \leftarrow \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] \geq c$; // refine H_1 with new $\mathbf{d}_{ub}$
  $res \leftarrow \mathbf{Verifier}_B(H_0, H_1)$; // re-perform statistical verification
  if res = `True` **then**
    return $\mathbf{d}_{ub}$;

We conclude this subsection by bounding type I errors, *i.e.*, where the alternative hypothesis accepted but the null hypothesis actually holds. According to [15], the type I error rate is bounded by:

$$err(B, c) = \frac{c}{c + (1-c)B}.$$

We also note that our iterative procedure terminates only when the alternative hypothesis is accepted. Hence, type II errors (where the null hypothesis is accepted when the the alternative hypothesis actually holds) are not relevant.

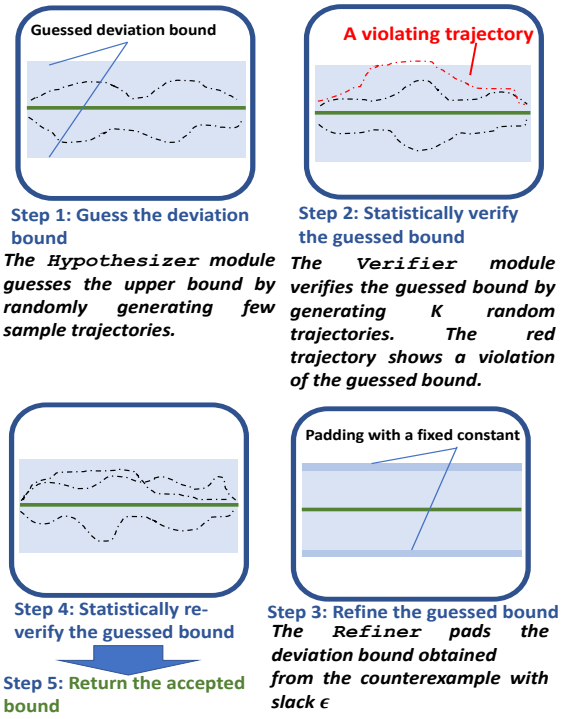


Fig. 4. The steps performed by Algorithm 1 to compute a deviation bound with a desired confidence. The nominal trajectory is shown in green, randomly generated trajectories are shown in black, and $\mathbf{d}_{ub}$ is shown in light blue.

V. Illustration of the Proposed Approach And Advantages of Jeffreys's Bayes Factor

In this section, we demonstrate how the three modules described in Section IV, namely **`Hypothesizer`**, **`Verifier`**, and **`Refiner`**, are used by Algorithm 1 to compute an upper bound on the deviation ($\mathbf{d}_{ub}$) with a probabilistic guarantee. This is to intuitively demonstrate our main approach on the following illustrative example:

$$x[t+1] = \begin{bmatrix} 0.1 & 0.2 \\ 0.2 & 0.1 \end{bmatrix} x[t] + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} u[t]$$
$$u[t] = \begin{bmatrix} 0.05149186 & 0.4189839 \end{bmatrix} x[t]$$

We will compute the maximum deviation from a nominal trajectory with no deadline misses, starting from the initial state $x[0] = [10\ 10]^T$. We assume that no more than two consecutive deadline misses occur up to our time bound $H = 5$. Using the *Hold&Skip-Next* policy, we can accordingly construct a transducer automaton representing this behavior of the scheduler and dynamical system. Given these inputs, Algorithm 1 performs the following steps to compute the maximum deviation, illustrated in Fig. 4.

Step 1: The first module invoked by Algorithm 1 is the **`Hypothesizer`**, which *guesses* an upper bound $\mathbf{d}_{ub}$ to be the maximum deviation. To do so, the module considers the following two random sequences of deadline hit/miss: 01001, 00111 (0 indicates miss, 1 indicates hit). It computes the maximum deviation from the two random samples, $\mathbf{d}_{ub} = 0.1462$.
Step 2: Next, the guessed upper bound $\mathbf{d}_{ub} = 0.1462$ is verified by invoking the module **`Verifier`**. The **`Verifier`** module, pre-tuned with $B = 4.15 \times 10^5$ and $c = 0.99$, returns *False*, *i.e.*, the upper bound $\mathbf{d}_{ub} = 0.1462$ is not verified to be correct with the desired probabilistic guarantees.
Step 3: Since the guessed bound $\mathbf{d}_{ub}$ was not verified, Algorithm 1 next invokes the **`Refiner`** module with the counterexample. **`Refiner`** updates the previous $\mathbf{d}_{ub} = 0.1462$ to $\mathbf{d}_{ub} = 0.2262$ (by padding a fixed constant of 0.001 on top top of the deviation bound obtained from the counterexample).
Step 4: The refined upper bound $\mathbf{d}_{ub} = 0.2262$ is again sent to the **`Verifier`** module for re-checking. This time, the **`Verifier`** module accepts the $\mathbf{d}_{ub} = 0.2262$ as a valid upper bound up-to the desired probabilistic guarantees.
Step 5: The final $\mathbf{d}_{ub} = 0.2262$ is returned as the maximum deviation (with the desired probabilistic guarantees).

Having illustrated the steps performed by our algorithm on a simple example, we argue in the following subsection why we chose Jeffreys's Bayes factor over other methods.

A. Reason for Choosing Jeffreys's Bayes Factor

In theory, it is possible to use other hypothesis testing methods, such as sequential hypothesis testing. However, for this work, Jeffreys's Bayes factor enjoys several advantages. 1) Our method imposes no restriction on the distribution. It merely requires that samples can be drawn from the distribution at random. 2) In sequential hypothesis testing, unlike our method, the number of required samples cannot be fixed *a priori*. 3) Multiple counterexamples can be encountered during random sampling in a round of sequential hypothesis testing, allowing known counterexamples to be explored. In our setting a counterexample represent a violation of a safety property estimate and accordingly Jeffreys's Bayes factor does not allow such known counter examples. 4) Similar to sequential hypothesis testing, our method is independent of the sample space size once the desired confidence (on the answer) is provided by the user.

VI. Experimental Evaluation

We implemented our Algorithm 1 in a Python-based prototype tool, named **`StatDev`**. The tool and the models are available through a public GitHub repository[1]. For $dis(\cdot)$ we use the 2-norm. As mentioned in the introduction, to generate uniform random samples for a given transducer automaton, we implemented the *Recursive RGA* algorithm [16], [39]. We demonstrate the applicability of our method given in Algorithm 1 on four standard examples: an RC network [40], an electric steering application [17], an unstable second-order system [17] and a F1Tenth car model [13]. For these examples we investigate the following questions. **Q1:** What impact do the different scheduling policies have on on the computed deviations? **Q2:** What effect does the probabilistic guarantee c have on the computed deviation? **Q3:** How do our statistically computed deviations compare to the results obtained using the **`RS`** method? Recall that **`RS`** refers to the deterministic reachable set based method described in Section III.

[1]https://github.com/bineet-coderep/StatJitteryScheduler

The following parameters were used in our study: $R = 50$, $\epsilon = 10^{-3}$, $B = 4.15 \times 10^5$, an initial state of $(10, 10)$ and time bound $H = 150$. Since Algorithm 1 is stochastic, we execute it over several trials (50 in this case) and report the mean and the standard deviation (SD) of the obtained $\mathbf{d}_{ub}$ values. For instance, 5 (0.3) means that the mean value of $\mathbf{d}_{ub} = 5$ with SD 0.3, and the reported computation time is the average computation time taken.

The DFAs we consider enforce constraints of the form "at most k consecutive misses". But any other form of constraints, as long as they are regular, could also be used. We denote these DFAs as $\{\mathcal{T}_k\}$ with k ranging over $\{1, 2, 3\}$. For most of the experiments, the DFA $\mathcal{T}_3$ was used.

The main results are summarized in Table I: our statistical method almost always computes tighter bounds than **RS**, except for the F1Tenth example. For unstable systems and and F1tenth, **RS** seems to have an exponential increase in computation time, whereas our statistical method scales much better (note that **RS** fails to compute a bound within an hour).

A. Benchmarks

1) RC network: The RC network is given by the following state space equations.

$$x[t+1] = \begin{bmatrix} 0.5495 & 0.07240 \\ 0.01448 & 0.9332 \end{bmatrix} x[t] + \begin{bmatrix} 0.3781 \\ 0.05234 \end{bmatrix} u[t] \quad (10)$$

Assuming nominal timing behavior of the control software, the feedback controller computed using LQR is

$$u[t] = \begin{bmatrix} 0.09772 & 0.2504 & 0.07805 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}. \quad (11)$$

Using the matrices from Eqs. (10) and (11), we construct a transducer automaton for different deadline miss strategies. We used $\mathcal{T}_3$ to specify the scheduler.

2) Electric steering: The electric steering example is given by the following state space equations.

$$x[t+1] = \begin{bmatrix} 0.5495 & 0.07240 \\ 0.01448 & 0.9332 \end{bmatrix} x[t] + \begin{bmatrix} 0.3781 \\ 0.05234 \end{bmatrix} u[t] \quad (12)$$

Optimal feedback controller for this system under nominal timing behavior computed using LQR is:

$$u[t] = \begin{bmatrix} 0.09772 & 0.2504 & 0.07805 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}. \quad (13)$$

As before, using the matrices from Eqs. (12) and (13), we construct a transducer automaton for different deadline miss strategies. We used $\mathcal{T}_3$ to specify the scheduler.

3) Unstable second-order system: The unstable second-order system example is given by the following state space equations.

$$x[t+1] = \begin{bmatrix} 1.1053 & 0 \\ -0.0209 & 0.99 \end{bmatrix} x[t] + \begin{bmatrix} 0.0526 & 0.0105 \\ 0.0393 & 0.0994 \end{bmatrix} u[t] \quad (14)$$

The control input $u[t]$ is computed as

$$u[t] = \begin{bmatrix} 4.7393 & 0.2430 \\ 0.2277 & -0.8620 \end{bmatrix} x[t-1]. \quad (15)$$

Using the matrices from Eqs. (14) and (15), we construct a transducer automaton $\mathcal{T}_1$ to specify the scheduler.

4) F1Tenth: The model of an F1Tenth car [13] was linearized and the following state space equations were obtained.

$$x[t+1] = \begin{bmatrix} 1 & 0.13 \\ 0 & 1 \end{bmatrix} x[t] + \begin{bmatrix} 0.02559 \\ 0.3937 \end{bmatrix} u[t] \quad (16)$$

The control input $u[t]$ is computed as

$$u[t] = \begin{bmatrix} 0.2935 & 0.4403 \end{bmatrix} x[t-1]. \quad (17)$$

Using the matrices from Eqs. (16) and (17), we construct a transducer automaton $\mathcal{T}_3$ to specify the scheduler.

B. Results

The main results addressing (Q1) and (Q3) are summarized in Table I. For the RC network, electric steering and F1Tenth, $\mathcal{T}_3$ (*i.e.*, the constraint "at most 3 consecutive misses") was used with $c = 0.99$. Since the third system is an unstable open loop system, missing too many deadlines would cause the deviation bound to increase drastically; therefore, we only consider $\mathcal{T}_1$ (*i.e.*, the constraint "at most 1 deadline miss consecutively").

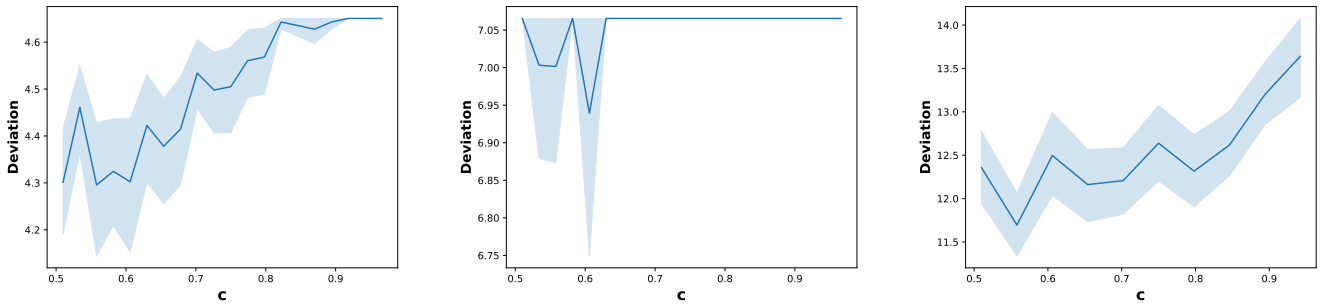
To address (Q2), we fixed $R = 10$ and using $\mathcal{T}_3$ for RC network, electric steering, F1tenth, and $\mathcal{T}_1$ for the unstable second order system, we varied c (the probabilistic guarantee) from 0.51 to 0.99. The resulting mean values of $\mathbf{d}_{ub}$ and 95% confidence interval are shown in Fig. 5. Below we discuss the results in detail for all the examples.

1) RC network: We now discuss the results obtained for the RC network example. **Q1:** Considering at most 3 consecutive deadline misses allowed, we computed the maximum deviation $\mathbf{d}_{ub}$, with scheduling policies *Hold&Kill*, *Zero&Kill*, *Hold&Skip-Next* and *Zero&Skip-Next*, as 1.898 (0), 1.898 (0), 1.819 (0), 1.621 (0) respectively. The details are given in Table I. In this particular example, *Zero&Skip-Next* shows the least deviation bound. Plots for this example is not shown due to space limitations. **Q2:** Considering at most 3 consecutive deadline misses allowed and $R = 10$, we gradually varied c from 0.51 to 0.99. We observe that the mean $\mathbf{d}_{ub}$ increases with increase in c, as expected, and stabilizes at a high probability. Also, note that with low values of c, we witness a wider 95% confidence interval, which narrows down to 0 as c increases. Plots for this example are not shown due to space limitations. **Q3:** The results show (see Table I) that the stochastic method returns tighter bounds than the **RS** method. However, the computation time is higher but still quite small, *i.e*, under 3 s.

2) Electric steering: We now discuss the results obtained for the electric steering example. **Q1:** Similar to our previous example, considering at most 3 consecutive deadline misses allowed, we computed the maximum deviation $\mathbf{d}_{ub}$, with the scheduling policies (detailed results in Table I). The safety envelope is shown in Fig. 6a, using the *Hold&Skip-Next* policy, and $c = 0.99$. Further, in Fig. 6a, we show possible safety violation that might occur when the number of consecutive deadline misses just increases by 1, *i.e.*, considering at most 4 consecutive deadline misses. The red dotted line in Fig. 6 shows the safety violation (with the existing safety envelope)

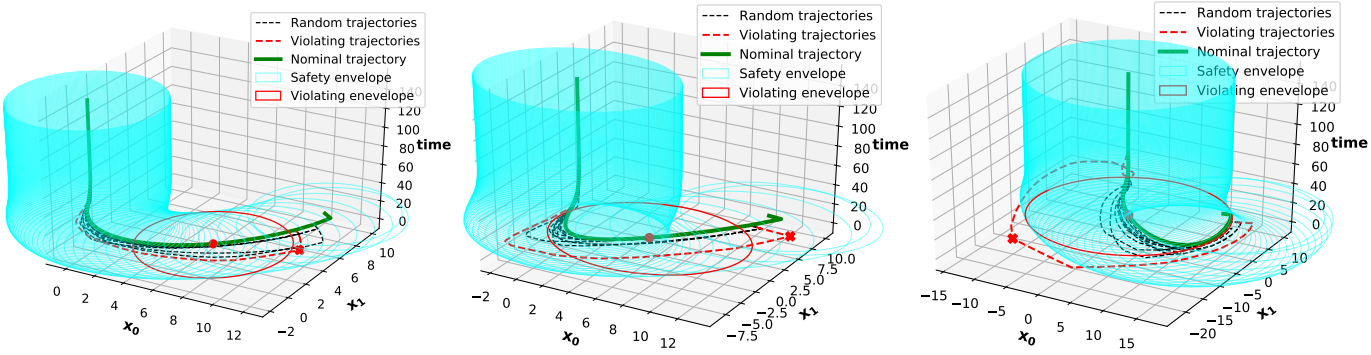
TABLE I
RESULTS COMPARING OUR PROPOSED STATISTICAL METHOD TO THE **RS** METHOD ON FOUR EXAMPLES

Example	Statistic	*Hold&Kill*	*Zero&Kill*	*Hold&Skip-Next*	*Zero&Skip-Next*
RC network	$\mathbf{d}_{ub}$ (Alg. 1)	1.898 (0)	1.898 (0)	1.819 (0)	1.621 (0)
	$\mathbf{d}_{ub}$ (**RS**)	1.90	1.90	1.90	1.90
	Time Taken (Alg. 1)	2.3 s	2.03 s	3 s	2.53 s
	Time Taken (**RS**)	0.68 s	0.78 s	2.27 s	2.37 s
Electric Steering	$\mathbf{d}_{ub}$ (Alg. 1)	3.809 (0)	7.835 (0.15)	4.65 (0)	7.207 (0.203)
	$\mathbf{d}_{ub}$ (**RS**)	12.37	13.63	12.38	13.75
	Time Taken (Alg. 1)	1.68 s	5.29 s	3.26 s	7.03 s
	Time Taken (**RS**)	30.8 s	685 s	2845 s	2864 s
Unstable second-order	$\mathbf{d}_{ub}$ (Alg. 1)	3.552 (0)	11.124 (0.71)	7.065 (0)	9.784 (0.49)
	$\mathbf{d}_{ub}$ (**RS**)	*timed out*	*timed out*	*timed out*	*timed out*
	Time Taken (Alg. 1)	2.28 s	5.55 s	2.17 s	4.05 s
	Time Taken (**RS**)	*timed out*	*timed out*	*timed out*	*timed out*
F1Tenth	$\mathbf{d}_{ub}$ (Alg. 1)	8.763 (0)	15.928 (0.55)	14.2 (0.65)	14.02 (0.5)
	$\mathbf{d}_{ub}$ (**RS**)	6.01	*timed out*	*timed out*	*timed out*
	Time Taken (Alg. 1)	2.02 s	5 s	7.19 s	6.08 s
	Time Taken (**RS**)	1569 s	*timed out*	*timed out*	*timed out*



(a) Electric steering (b) Unstable second-order (c) F1Tenth

Fig. 5. Mean values of $\mathbf{d}_{ub}$ (navy) and 95% confidence interval (light blue) with varying probabilistic guarantee c on the computed bound.



(a) **Electric steering** (b) **Unstable second-order** (c) **F1Tenth**

Fig. 6. Safety envelopes at a distance of $\mathbf{d}_{ub}$ from the nominal trajectory, with random trajectories with one extra consecutive deadline miss.

that might occur if the consecutive deadline misses increases by 1. The safety envelope highlighted in red shows the violating envelope. The behavior would have been safe if the trajectory (in red) was within the highlighted safety envelope (red), whereas it actually stretches outside to the point marked in '$\times$' (red). **Q2:** Considering at most 3 consecutive deadline misses allowed and $R = 10$, we gradually varied c from 0.51 to 0.99. The result is given in Fig. 5a. We observe that the mean $\mathbf{d}_{ub}$ increases with increase in c, as expected, and stabilizes at a high probability. Also, note that with low values of c, we witness a wide 95% confidence interval, which narrows down to 0 as c increases. **Q3:** As given in Table I, we compute tighter values of $\mathbf{d}_{ub}$ and in less time.

3) Unstable second-order system: We now discuss the results obtained for the unstable second-order system example. **Q1:** Unlike to our previous examples, Since the third system is an unstable open loop system, missing too many deadlines would cause the deviation bound to increase drastically; therefore, we only consider $\mathcal{T}_1$ (i.e. the constraint "at most 1 deadline miss consecutively"). We computed the maximum deviation $\mathbf{d}_{ub}$ with the scheduling policies (see Table I). The safety envelope is shown in Fig. 6b, using the *Hold&Skip-Next*

policy, along with a violating trajectory when the constraint changes to $\mathcal{T}_2$. **Q2:** Considering at most 1 consecutive deadline miss allowed and $R = 10$, we gradually varied c from 0.51 to 0.99. The result is given in Fig. 5b. We observe similar behavior as other examples. **Q3:** The **RS** method, for this specification, was not able to compute a bound in an hour. While our proposed method computed the bounds for all scheduling polices under 6 s.

4) F1Tenth: We now discuss the results obtained for the F1tenth example. **Q1:** Similar to the first two examples, we computed the maximum deviation $\mathbf{d}_{ub}$ with the scheduling policies, considering at most 3 consecutive deadline misses. The safety envelope is shown in Fig. 6c, using the *Hold&Skip-Next* policy, and $c = 0.99$. Further, in Fig. 6c, we show possible safety violation that might occur when the number of consecutive deadline misses increases by 1 (at most 4 consecutive deadline misses). **Q2:** Unlike other examples, where the width of the 95% confidence interval decreases sharply, in the F1Tenth example the decrease is not as drastic as other examples. This is shown in Fig. 5c. For F1Tenth, we further increased the value of c to 0.9999, and still witnessed non-zero standard deviation unlike other examples. However, following the method of [17], we computed an upper-bound on the *joint spectral radius* and found the system with at most three deadline consecutive misses is stable. This suggests that even though the overall behavior of the system is stable, the system possibly behaves erratically prior to obtaining stability—that is, the variance of the deviation obtained for various behaviors (w.r.t. deadline hits and misses) is very high. To put it differently, the deviation obtained from two different sequences of deadline hits and misses can be very different. **Q3:** The **RS** method, except for *Hold&Kill* was not able to compute a reasonable bound on the deviation. Whereas our proposed method computed reasonable bound (but the computed bound was, however, tighter than the bound computed using our method), for all policies, under 8 s.

C. General Observations

In this subsection, we draw general observations from our experiments that might help the users to tune this method according to their application. In other words, the following observations are drawn to provide a rule of thumb to choose the parameters and gather insights: 1) Revisit (Q1). 2) Revisit (Q2): The parameter c (confidence on the deviation). 3) Revisit (Q3): When to use a traditional method (**RS**) over our statistical method. 4) How to choose the parameter R.

Revisit (Q1). We observe (from Table I) that there is no one scheduling policy that performs consistently well (in terms of smaller deviation bounds) across all the benchmarks. This suggests that the choice of scheduling policy should be made according to the given application. ***Revisit (Q2): Choice of* c.** A good strategy is to use a high value of c, so that the standard deviation is low and further increases in c will have minimal effect. We note, however, that the unstable second-order and the F1Tenth example with at most 3 consecutive deadline misses fail to achieve a low standard deviation even with $c = 0.99$. ***Revisit (Q3): Choice of method to be used.*** For small dimensional stable systems, like the RC network, traditional methods might work well. However, for unstable systems they are likely to perform poorly due to coarse over-approximations. The exception is the F1Tenth system for which the **RS** method was able to compute a tighter bound while taking a longer time. In such cases our statistical method will perform better and yield tighter bounds in lesser time. ***Choice of* R.** As our method is stochastic, for smaller values of R (and highly chaotic systems), the initial $\mathbf{d}_{ub}$ guess can vary greatly (for different trials). And when such an initial guess is being verified with a low probability c, the chances of the initial guess being accepted is very high. Therefore the SD for low values of c is also high, and the obtained bound $\mathbf{d}_{ub}$ is not guaranteed to be monotonic.

Further, we make general comments on Fig. 5 and Fig. 6 as follows. ***Varying the value of* c (Fig. 5).** We observe that for lower values of c, the mean values of $\mathbf{d}_{ub}$ are lower but the 95% confidence interval are wider, however, as c increases the mean values of $\mathbf{d}_{ub}$ increase whereas the width of the 95% confidence interval decreases sharply, except for F1Tenth where the decrease is not as drastic as other examples. For F1Tenth, the computed *joint spectral radius* showed the system with at most three deadline consecutive misses to be stable, suggesting the system behavior is possibly erratic prior to obtaining stability. Note that we have not shown the plot for RC network due to space limitations, and as its behavior is similar to that of electric steering. We further note that the computed mean deviation bound is not monotonic in any of the examples. At lower values of c, the confidence interval is wider, therefore the variance on the computed bound is much higher at lower values of c. The computed bound is however observed to be stabilizing with narrow 95% confidence interval at only higher values of c, except for the F1Tenth example. Again, the value of c at which the computed mean bound stabilizes is application specific. For instance, in case of electric steering the value of c at which the bound stabilizes is much higher than the one required for unstable second-order system. ***Safety envelope* (Fig. 6).** In this figure, we show the computed deviation bounds using the *Hold&Skip-Next* and $c = 0.99$. In other words, for electric steering and F1Tenth, all the trajectories satisfying the $\mathcal{T}_3$ constraint will be within the computed safety envelope in cyan (with a probabilistic guarantee). Similarly, for the unstable second order system we used the constraint $\mathcal{T}_1$. We observe that for the electric steering example, the violating point (marked with '$\times$' in red) is closer to violating safety envelope (highlighted in red) than the other two examples—this might be due to that fact that electric steering behaves less erratically than the other two examples. Further, we evaluated the robustness of the system *w.r.t* the computed safety envelope, by changing the constraint from $\mathcal{T}_3$ to $\mathcal{T}_4$ for electric steering and F1Tenth, and from $\mathcal{T}_1$ to $\mathcal{T}_2$ for unstable second order system. The violation is shown in red. Note that we timed out while computing a violating trajectory for the RC network example—indicating better robustness than other examples. ***Using Fig. 5 to compute Fig. 6***. Note that

Fig. 5 suggests a heuristic to choose a value for c (at which the mean stabilizes with a narrow 95% confidence interval). Once such a c is chosen, the safety envelope should be computed, as in Fig. 6, using that value of c.

VII. Concluding Remarks

We have shown that quantitative dynamical properties of closed loop control systems can be verified using statistical hypothesis testing. The results obtained are approximate ones (just like traditional deterministic approximation methods are) but are accompanied by probabilistic guarantees. The computational effort required depends mainly on the required confidence level and to a certain extent on the work required to draw the samples. Here, for the sake of being able to compare our method with prior studies, we have restricted ourselves to simple deadline hit-miss patterns as well as low dimensional linear systems. In the future, we plan to consider richer languages as well as high dimensional and non-linear systems. We also plan to study deviations from properties specified using temporal logics like BLTL (Bounded Linear Time Logic) [23]. This will help capture a richer set of quantitative properties that autonomous systems are often required to satisfy—*e.g.*, the system must avoid certain regions but must also visit some other locations with a specified frequency. In such settings too we expect our Bayesian hypothesis testing based method to play a useful role. Finally, the counter example guided statistical hypothesis testing can be applied to black-box systems where obtaining a precise model is challenging.

Acknowledgements: This work is supported by the US NSF grant #2038960. The authors also gratefully acknowledge the feedback from the anonymous reviewers.

References

[1] M. Fisher *et al.*, "Verifying autonomous systems," *Commun. ACM*, vol. 56, no. 9, pp. 84–93, 2013.

[2] J. Wing, "Trustworthy AI," *Commun. ACM*, vol. 64, no. 10, pp. 64–71, 2021.

[3] L. Dennis and M. Fisher, "Verifiable self-aware agent-based autonomous systems," *Proc. IEEE*, vol. 108, no. 7, pp. 1011–1026, 2020.

[4] 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.

[5] P. Axer *et al.*, "Building timing predictable embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4, pp. 82:1–82:37, 2014.

[6] L. Thiele and R. Wilhelm, "Design for timing predictability," *Real-Time Systems*, vol. 28, no. 2-3, pp. 157–177, 2004.

[7] R. Wilhelm, "Real time spent on real time," *Commun. ACM*, vol. 63, no. 10, pp. 54–60, 2020.

[8] P. Pazzaglia *et al.*, "Beyond the weakly hard model: Measuring the performance cost of deadline misses," in *30th Euromicro Conference on Real-Time Systems (ECRTS)*, 2018.

[9] D. Soudbakhsh *et al.*, "Co-design of arbitrated network control systems with overrun strategies," *IEEE Trans. Control. Netw. Syst.*, vol. 5, no. 1, pp. 128–141, 2018.

[10] 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.

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

[12] W. Zhang *et al.*, "Stability of networked control systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 84–99, 2001.

[13] 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, pp. 77–89, 2020.

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

[15] R. Diwakaran *et al.*, "Analyzing neighborhoods of falsifying traces in cyber-physical systems," in *8th International Conference on Cyber-Physical Systems (ICCPS)*, 2017.

[16] O. Bernardi and O. Giménez, "A linear algorithm for the random sampling from regular languages," *Algorithmica*, vol. 62, pp. 130–145, 2010.

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

[18] P. Pazzaglia *et al.*, "DMAC: deadline-miss-aware control," in *31st Euromicro Conference on Real-Time Systems (ECRTS)*, 2019.

[19] 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.

[20] E. Horssen *et al.*, "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses," in *15th European Control Conference (ECC)*, 2016.

[21] S. Bozhko *et al.*, "Monte carlo response-time analysis," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021.

[22] H. Younes and R. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *CAV*, 2002.

[23] A. Legay *et al.*, *Statistical Model Checking*. Springer, 2019.

[24] M. Donkers *et al.*, "Stability analysis of stochastic networked control systems," *Automatica*, vol. 48, no. 5, pp. 917–925, 2012.

[25] M. Cloosterman *et al.*, "Stability of networked control systems with uncertain time-varying delays," *IEEE Trans. Automat. Contr.*, vol. 54, no. 7, pp. 1575–1580, 2009.

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

[27] S. Chakraborty *et al.*, "Cross-layer interactions in CPS for performance and certification," in *Design, Automation & Test in Europe (DATE)*, 2019.

[28] S. Samii *et al.*, "Dynamic scheduling and control-quality optimization of self-triggered control applications," in *31st IEEE Real-Time Systems Symposium (RTSS)*, 2010.

[29] M. Kauer *et al.*, "Fault-tolerant control synthesis and verification of distributed embedded systems," in *Design, Automation & Test in Europe Conference (DATE)*, 2014.

[30] R. Schneider *et al.*, "Multi-layered scheduling of mixed-criticality cyber-physical systems," *Journal of Systems Architecture - Embedded Systems Design*, vol. 59, no. 10-D, pp. 1215–1230, 2013.

[31] D. Roy *et al.*, "Semantics-preserving cosynthesis of cyber-physical systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 171–200, 2018.

[32] R. Mahfouzi *et al.*, "Stability-aware integrated routing and scheduling for control applications in Ethernet networks," in *Design, Automation & Test in Europe Conference (DATE)*, 2018.

[33] D. Roy *et al.*, "Multi-objective co-optimization of flexray-based distributed control systems," in *22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

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

[35] M. Lukasiewycz *et al.*, "System architecture and software design for electric vehicles," in *50th Design Automation Conference (DAC)*, 2013.

[36] W. Chang and S. Chakraborty, "Resource-aware automotive control systems design: A cyber-physical systems approach," *Foundations and Trends in Electronic Design Automation*, vol. 10, no. 4, pp. 249–369, 2016.

[37] K. Åström and B. Wittenmark, *Computer-controlled systems (3rd ed.)*. Prentice-Hall Inc., 1997.

[38] J. P. Hespanha, *Linear Systems Theory: Second Edition*. Princeton University Press, 2018.

[39] P. Flajolet *et al.*, "A calculus for the random generation of labelled combinatorial structures," *Theoretical Computer Science*, vol. 132, no. 1, pp. 1–35, 1994.

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