Statistical Verification of Autonomous System Controllers Under Timing Uncertainties

Bineet Ghosh[1,2*], Clara Hobbs[2], Shengjie Xu[2], Don Smith[2], James H. Anderson[2], P. S. Thiagarajan[2,3], Benjamin Berg[2], Parasara Sridhar Duggirala[2] and Samarjit Chakraborty[2]

[1*]The University of Alabama, AL, USA.
[2]The University of North Carolina at Chapel Hill, NC, USA.
[3]Chennai Mathematical Institute, India.

*Corresponding author(s). E-mail(s): bineet@ua.edu;
Contributing authors: {cghobbs, sxunique, smithfd, anderson, thiagu, ben, psd, samarjit}@cs.unc.edu;

Abstract

Software in autonomous systems like autonomous cars, robots or drones is often implemented on resource-constrained embedded systems with heterogeneous architectures. At the heart of such software are multiple feedback control loops, whose dynamics not only depend on the control strategy being used, but also on the timing behavior the control software experiences. But performing timing analysis for safety critical control software tasks, particularly on heterogeneous computing platforms, is 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 with timing uncertainties deviates too much from the nominal trajectory. 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 the behavior of control systems with timing uncertainties. More specifically, we present a Bayesian hypothesis testing method 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 analysis methods. We also illustrate the scalability of our techniques by obtaining bounds in cases where reachability analysis fails, thereby establishing the practicality of our proposed method.

Keywords: Control, reachability, real-time systems, safety, weakly-hard systems, statistical hypothesis testing

1 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: (i) determining the worst-case execution time (WCET) of the code [5], and (ii) 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 nondeterminism in the timing behavior of software is unavoidable. Further, even with pessimistic WCET bounds, modern autonomous systems are required to run several software processes concurrently, thereby their timing behaviors interfering with each other. Conventional approaches requiring strict adherence to deadlines have become increasingly impractical or require the adoption of costly high-performance implementation platforms.

In this paper we propose techniques that accept the inevitable timing uncertainties that control software in modern autonomous systems will face, but nevertheless provide the necessary safety guarantees. This is done using statistical techniques to overcome the scalability challenges associated with the large state space over which the verification needs to be performed. This raises the question: *"What performance guarantees can be provided for feedback control loops subjected to uncertain timing behaviors?"* 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

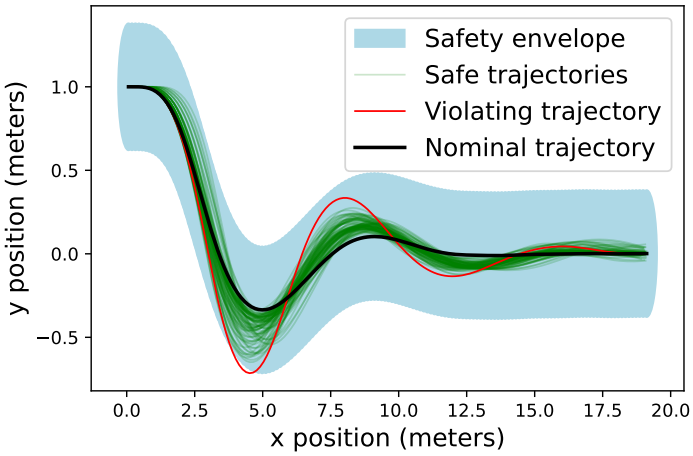


Fig. 1: Deviation in the path of an F1Tenth car due to timing uncertainty.

in network behavior such as delays and dropped packets [8–14]. 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**.

F1Tenth Example

As an example, consider Fig. 1. It shows the trajectory of a lane-following controller for an F1Tenth [15] 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. The distance of any trajectory when subjected to timing uncertainties (deadline misses), from the nominal trajectory (resulting under ideal timing behavior) is used to quantify "safety" in this paper, as illustrated in this example. The goal here is to suitably characterize timing uncertainties and efficiently check whether a given class of timing uncertainties satisfy such a specified safety property.

1.1 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 $[0, H]$. But this is not a scalable problem and **our contribution** is a *statistical hypothesis testing* (SHT) framework to address this scalability issue (see Fig. 3). In particular, we use a Bayesian hypothesis testing approach [16, 17] to address this 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 characterized as 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 [18]. The *Recursive RGA* algorithm offers an efficient solution for generating a random accepting run of a given deterministic finite automaton (DFA) capturing the deadline hit/miss patterns. When provided with a DFA, the *Recursive RGA* algorithm enables the generation of a random word that belongs to the language represented by the DFA. In this paper, we represent the deadline hit/miss patterns using a DFA defined on the alphabet $\langle 0, 1 \rangle$, where 0 signifies a deadline miss and 1 represents a deadline hit. Therefore, by utilizing the *Recursive RGA* algorithm, we can generate a random accepting run of this DFA, which corresponds to a potential observed deadline hit/miss pattern within the system. Considering that the patterns considered in this paper can be represented using a DFA, the *Recursive RGA* algorithm emerged as the most efficient known algorithm to the best of our knowledge for generating random strings (as per uniform distribution) from a given regular language within a polynomial time complexity. Note that our method is also 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

Our statistical hypothesis testing framework formulates two hypotheses, namely null hypothesis H_0 and alternative hypothesis H_1, to test if a given $\mathbf{d}_{ub}$ is an upper bound for the maximum deviation. The null hypothesis H_0 will assert that with *at most* probability c, a randomly chosen trajectory will have a deviation bounded by $\mathbf{d}_{ub}$. 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 $\mathbf{d}_{ub}$. This is illustrated in Fig. 3. We then use a Bayesian

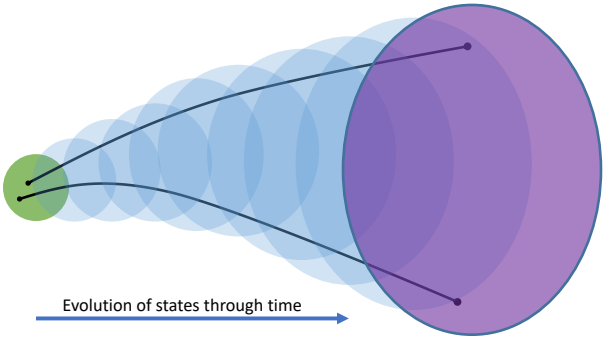


Fig. 2: **Evolution of states through time from an initial set of states.** The figure depicts all the possible sets of states that are reachable when the system starts from a given initial set of states. The set of initial states is given in green, blue represents the set of intermediate states, and purple represents the set of states reached at time t.

hypothesis test to decide between these two hypotheses [16]. 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 $\mathbf{d}_{ub}$. We leverage this counterexample to generate new hypotheses, for the next iteration, based on a new and larger $\mathbf{d}_{ub}$. In this sense, our method is driven by a *counterexample guided refinement strategy* [19] to eventually accept the alternative hypothesis (see Fig. 3). In our statistical hypothesis framework, in addition to the probabilistic guarantee c, we can also bound the so called type I and type II error rates. The type I error rate is defined as the probability of inferring the alternative hypothesis when in fact the null hypothesis holds. Similarly, the type II error rate is defined as the probability of inferring the null hypothesis when in fact the alternate hypothesis holds. We choose the relevant parameters such that the type I error is kept significantly low (as desired), while type II error [16] is not relevant in our setting.

Handling Initial Set of States and Environmental Disturbances

When a system starts its evolution form a single initial state, the resulting trajectory is a sequence of states (or *points*) in time, as was the case in the F1Tenth example. In our previous work [20], the estimation of deviation is computed when the system starts from a given initial state. However, one might want to conduct the analysis for a set of states: either because the initial state is unknown or because the user would like to do a similar analysis for a large collection of initial states. Moreover, for all practical purposes, it is impossible to locate the state from where the evolution of the system (such as a robot) starts its execution from. In such cases, one must consider an initial set of states to perform an analysis, as a single initial state is often useless in

practice. We call the collection of all trajectories starting from the (possibly infinite) initial set as *pipes*.

We consider the example given in Fig. 2 that demonstrates the evolution of the system from initial set of states (marked in green). The set of states reached at time t is given in purple, and intermediate set of states is given in blue. The black lines in the figure demonstrates two random trajectories of the system starting from two random states from the initial set of states. To account for an initial set of states rather than individual states, the analysis conducted in [20] has been expanded to encompass pipes (as depicted in Fig. 2) derived from trajectories. This adjustment is necessary because when initiating from an initial set of states (as opposed to a single state), the system's behavior concerning a sequence of deadline hits and misses is represented by a pipe (infinite sets of trajectories), rather than a trajectory.

This introduces the following two additional research challenges. First, the selection of an appropriate representation for the reachable set becomes crucial. In this regard, we have examined two most widely used representations for reachable sets, Zonotopes and V-polytopes. Zonotopes represent a set by utilizing its center and a set of generator vectors that define the extent of the set from the center. On the other hand, V-polytopes explicitly represent a set by its vertices. Each of these representations offers distinct trade-offs, which will be further explored in Section 5. Second, the computation of distances between pipes, rather than just trajectories, poses a challenge. While certain representations excel at computing set distances, the computation of system evolution becomes challenging (and vice versa). Consequently, a representation must be chosen by carefully considering various factors, as elaborated upon in Section 5. In this work, it is assumed that the initial set of states, that represents the uncertainty around the actual initial set, is pre-computed and given as an input to this method. In practice, the associated uncertainty around an initial state may be estimated based on the error tolerance of the sensors used for measurements, or based on other characteristics of the system. In general, to conduct a conservative yet safe analysis, it is advisable to consider a higher degree of uncertainty regarding the initial state.

In this paper, we extend our previous approach [20] for computing an upper bound from a specific initial state to computing such a bound for an initial set. This generalized analysis requires two main improvements. First, one has to choose an appropriate representation for the reachable set. Second, one has to compute, not just distance between trajectories, but rather compute distances between two (infinite) sets of trajectories. Certain representations can be useful to compute the distance between sets, while computing evolution of the system being hard (and vice versa). Thus, a representation must be chosen trading-off various factors [21, 22]. **Handling pipes, for computing the maximum deviation, is one of the main contributions of this work (Section 5)**. For the clarity of presentation, we use trajectory to refer to pipes in this paper hereon.

A system might also encounter environmental disturbances in its behavior at each time step in addition to uncertainty in its initial state. That is, the system may encounter uncertainty due to environmental disturbances at every time step resulting in a new set of possible system states. Although feedback controllers are frequently used to manage these uncertainties, for computing tight estimates of deviation from nominal trajectory, one has to account for such uncertainties. **In this work, we further discuss how one can adapt our proposed method to compute deviations under the presence of such uncertainties in the behavior of the system (Section 6)**. We note that, in practice, environmental disturbances cannot be measured precisely. As a result, the disturbances can only be represented as a set of states. Consequently, even if the behavior of the system originates from a single initial state, to account for the environmental disturbances, the behavior of the system has to be represented as pipes. Since our previous method is unable to handle such pipes, it is also unable to handle such environmental disturbances.

Related work

The problem of implementing control software on embedded systems has been actively pursued in recent times [23–26], especially in domains like automotive [27, 28]. Here, exploiting the robustness of control algorithms to adaptively allocate resources, *e.g.,* by switching between time-triggered and event-triggered communication, has been studied in the past [29, 30]. Similarly, shielding control software from timing interference of other concurrently running tasks have also been studied [4, 31]. Our work in this paper has been particularly influenced by a recent work by Maggio et al. [32] (and a number of preceding ones on the related problem [33–35]) 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 by Maggio et al. 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). As outlined earlier in this paper, work in this domain stems from the difficulty in timing analysis of control software by Ju et al. [36], which is attributed to both – the challenges in WCET analysis and the modeling of the code structure by Chakraborty et al. [37].

A sampling based statistical method is also used by Bozhko et al. [38] 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 Bozhko et al., 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

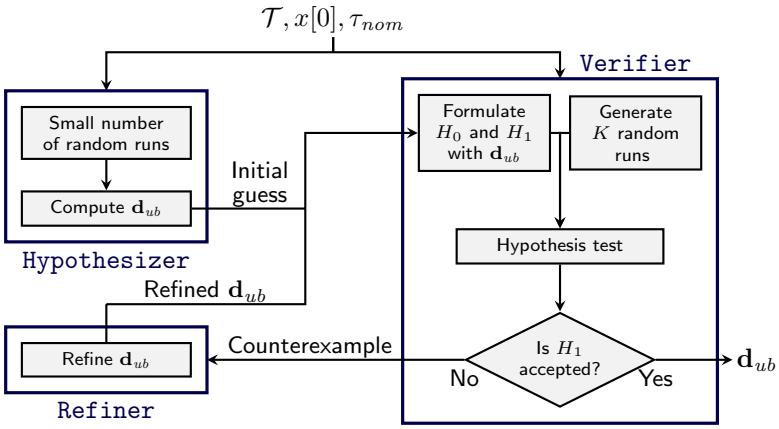


Fig. 3: Proposed statistical hypothesis testing approach.

on the number of samples that are "successes" (*i.e.*, the run encounters a first deadline miss of the task) a probability interval $\ell \leq \rho \leq r$ is computed such that $|l - r| \leq \delta$ and $P(\ell \leq \rho \leq r) \geq 1 - \epsilon$, where ρ is the probability of the task taking place. 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* [39] often uses sequential hypothesis testing methods such as SPRT (sequential probability ratio test). We have instead chosen a Bayesian hypothesis testing approach that requires a *fixed* number of samples, and which can be done efficiently in our setting. A thorough survey of various statistical model checking methods can be found in Legay et al. [40].

Modeling the impact of *network* uncertainties in the form of time-varying or stochastic delays on control performance has also been studied in [41–43]. Here again, the focus is on *stability*. Finally, a number of recent papers have addressed various aspects of the control/architecture *co-design* problem [13, 28, 31, 44–48] in domains such as automotive software design [25, 49]. However, there has been comparatively less work on the use of statistical techniques for the verification of control systems [17], although they perform better than approximation techniques, as we show in this paper.

Salient features of the proposed method

We conclude this section, by noting that the number of samples needed by our procedure to test a given value of $\mathbf{d}_{ub}$ depends only on the strength of the guarantee requested by the user. In particular, the number of samples required does *not* depend on factors such as the length of time horizon H, or the choice of the scheduling policy, even though these factors influence the underlying distribution of trajectories. 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 such as task completion times. It is worth pointing out that verifying quantitative safety properties for control systems is harder than verifying stability, which is a qualitative safety property. Specifically, techniques for stability analysis such as the use of *Lyapunov functions* and results from stability analysis of switched systems [11] are not designed to provide qualitative guarantees.

Organization of this paper

The remainder of this paper is organized as follows. Section 2 explains our system model, followed by the definition of the problem and a discussion of deterministic approaches to the solve the problem in Section 3. Our hypothesis testing based framework in presented in Section 4. Further, we have also provided a detailed discussion of the proposed framework in Appendix A. This discussion delves into the usage and impact of various parameters that are used in the proposed framework. Given our prior work (a preliminary version of this paper appeared in [20]), Sections 5 and 6 are the new technical contributions of this paper over [20], in addition to providing a more complete treatment of this problem. Before describing our experimental results in Section 8, we illustrate our approach on an example in Section 7, where we also provide a rationale for choosing a Bayesian hypothesis testing approach over other methods. The experimental results in Section 8 have also been be redone with the modifications proposed in Section 5. That is, as opposed to [20] which used a single initial state, the results in all the case studies are obtained using an initial set of states. We finally conclude by outlining some directions for future work.

2 System Modeling

We study the state feedback control of discrete time-invariant linear dynamical systems of the form:

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

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:

$$u[t] = Kx[t-1], \tag{2}$$

where $K \in \mathbb{R}^{p \times n}$. Here, Eq. (1) represents the *plant*—the system whose evolution needs to be controlled. The *controller*, given in Eq. (2), issues control

inputs to the plant to control the evolution of the system as desired by the user. Internally, the plant receives the control inputs at some discrete time steps, and controls the evolution of the system based on that input—this part of the system is often termed as the *actuator*. The plant (Eq. (1)) and the controller (Eq. (2)) can alternately be represented using an augmented state space [50] as $z[t] = \left[x[t]^T \ u_a[t-1]^T\right]^T$, giving the following model:

$$z[t+1] = \begin{bmatrix} A & B \\ K_x & K_u \end{bmatrix} z[t] \tag{3}$$

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) [51]. Further, it allows the plant and controller to be represented as a single *dynamics matrix*.

Although this paper proposes a method to calculate the maximum deviation caused by timing uncertainties in linear systems, the same approach can be adjusted to nonlinear systems as follows. When dealing with nonlinear systems, it is common to approximate them with linear models while accepting a certain level of error. In such cases, the error introduced by the linearization process can also affect the computed deviation, depending on various factors. The degree of error amplification relies on the guarantees provided by the linearized model. For instance, if the linearized model ensures that the trajectories of the linear model always stay within a δ-pipe of the actual model (*i.e.*, the linear model's trajectories do not deviate by more than δ compared to the corresponding trajectories of the actual model), it can be guaranteed that the deviation bound calculated using our method with the linear model is at most δ away from the actual deviation bound. In other words, if our method computes a deviation bound of d using the linear model, then the deviation bound of the actual model is $d + \delta$, maintaining the same probabilistic guarantees.

In order to model the system behavior under a sequence of deadline hits and misses, we use standard techniques, similar to those by Maggio et al. 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 [32].

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:

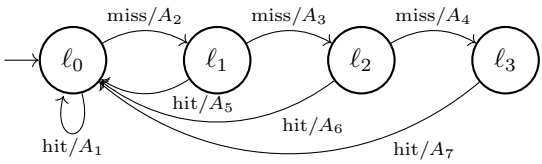


Fig. 4: Transducer automaton capturing 3 maximum consecutive misses.

- $L = \{\ell_1, \ell_2, \ldots, \ell_m\}$: 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. Note that the transitions within the automaton are deterministic. Starting from any state ℓ_i in the automaton, when presented with an action $a \in \{\text{hit}, \text{miss}\}$, the automaton transitions to a unique state ℓ_j that represents how the system handles the occurrence of a deadline hit or miss.
- 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 Deterministic Finite Automata (DFA). An example of a DFA, capturing all possible behaviors with at most 3 consecutive deadline misses, is shown in Fig. 4. 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 by Maggio et al. 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 [32], will result in different closed loop dynamics under deadline misses. However the nominal behaviors will be identical. We note that the policies *Zero & Kill*, *Hold & Kill*, *Zero & Skip-Next*, and *Hold & Skip-Next* only differ in the ways of handling a deadline miss. Conversely, in the event of a deadline hit, the computed controller input is applied in all the policies. Given that the nominal behavior reflects the absence of any

deadline misses, the nominal trajectories for all policies remain unchanged, as they are computed by simply applying the computed controller input at each time step.

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:

$$\tau = \{\ell_1, a_1, \ell_2, \ldots, a_{H-1}, \ell_H\} \tag{4}$$

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

$$evol(\tau) = \big\{x[0], x[1] = A_1 x[0], x[2] = A_2 x[1], \ldots, x[H] = A_H x[H-1]\big\}. \tag{5}$$

Note that since the evolution starts from an initial set of states ($x[0] \subset \mathbb{R}^n$), the states reached at every time step t ($x[t]$) is also a set.

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

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

Note that we must consider distance between sets, not points, because we are now working with the sequence of a set of states (trajectories)—such a concept of distance is Hausdorff. It is important to note that the representation of the sets has a significant impact on the computational complexity of computing the Hausdorff distance—from polytime to NP.

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:

$$dev(\tau_1, \tau_2) = \max_{1\le t\le H}\Big\{\Delta\big(evol(\tau_1)[t], evol(\tau_2)[t]\big)\Big\}. \tag{6}$$

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 $\mathcal{D}$.

3 The Problem Statement 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:*

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

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 8.

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 8, this method (i) typically outperforms the `RS` method in scalability, (ii) while producing much tighter deviation estimates.

4 The Proposed Statistical Hypothesis Testing Based Approach

In this section, we present a statistical hypothesis testing based approach to solve Problem 1. Specifically, we propose a counterexample guided refinement method to estimate 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 for estimating $\mathbf{d}_{ub}$ 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}. Our algorithm is composed of a network of three modules as illustrated in Fig. 3. Before we describe each module in detail, we provide a brief overview of the three modules.

`Hypothesizer`: Using the heuristics described in Section 4.1, we make an initial estimate, $\mathbf{d}_{ub}$. This initial guess is then sent to the `Verifier` module.

`Verifier`: This module statistically verifies — using a Bayesian hypothesis test — whether the current value of $\mathbf{d}_{ub}$ is a sufficiently high percentile (say, $\geq$ 99th percentile) of the distribution of trajectory deviations. If $\mathbf{d}_{ub}$ is accepted, it is returned as our final estimate. If not, we invoke the `Refiner` module, which generates a new, higher value for $\mathbf{d}_{ub}$. The details of the `Verifier` module are presented in Section 4.2.

`Refiner`: This module generates a new guess for $\mathbf{d}_{ub}$ based on the sampled trajectories seen thus far. Specifically, the next value of $\mathbf{d}_{ub}$ is set to be the highest sampled deviation plus some additional slack ϵ. The refiner sends this new value of $\mathbf{d}_{ub}$ back to the verifier for the next round of hypothesis testing.

4.1 `Hypothesizer`: Guessing an upper-bound on deviation

As our proposed counterexample refinement-based technique requires to start with an initial guess of the deviation, we use the following approach. 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. Firstly, we select a predetermined number of samples, denoted as R as specified by the user. Then, we generate R random trajectories that satisfy the given constraint on the deadline hit/miss pattern. This involves generating a sequence of hit/miss patterns (using the RGA algorithm) and subsequently computing trajectories based on these patterns, from the given initial set of states. Next, we calculate the maximum deviation between the

nominal trajectory and the generated random trajectories (with an additional epsilon padding). This maximum deviation serves as our initial guess, but it need not be an accurate bound for the deviation. It simply provides a starting point for our iterative counterexample refinement technique. It is worth mentioning that we choose a small number of samples (determined by the user) because, in some cases, a limited number of random samples can effectively represent 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 $\texttt{SmallSample}(\cdot)_{[R,\epsilon]}$.

4.2 `Verifier`: Testing a value of $\mathbf{d}_{ub}$

Here we describe our use of hypothesis testing to either validate or reject a given estimate of $\mathbf{d}_{ub}$. We first define $\mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right]$ to be the cumulative distribution function (CDF) for the deviation of a randomly chosen trajectory. That is, $\mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right]$ gives the probability that a randomly drawn sample has a deviation less than $\mathbf{d}_{ub}$. Our procedure takes a user-specified parameter $c \in (0, 1)$. In testing an estimate of $\mathbf{d}_{ub}$, the `Verifier` must decide whether $\mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] \geq c$ with high probability.

In addition to specifying c, the user specifies a credibility level for the test. A credibility level of $1 - \alpha$ signifies that $\mathbf{d}_{ub}$ can be accepted when the probability of a type-I error is at most α. For example, a credibility level of $1 - \alpha = 95\%$ signifies that we will accept a given value of $\mathbf{d}_{ub}$ when there is a 95% probability that $\mathbf{d}_{ub}$ is greater than or equal to the c-percentile of distribution of trajectory deviations. As a consequence, the probability of accepting $\mathbf{d}_{ub}$ when it is less than the c-percentile is at most $\alpha = 5\%$.

To perform our test, we formulate the following null and alternative hypotheses:

$$H_0 : \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] < c \tag{8}$$

$$H_1 : \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] \geq c \tag{9}$$

We test our hypotheses by drawing K samples $X = \{\tau_1, \tau_2, \ldots, \tau_K\}$ according to the distribution assumed over the set of trajectories. In order to make a conservative estimate, we will automatically accept H_0 if we find any sample with a deviation higher than $\mathbf{d}_{ub}$. Hence, we only need to consider the case where all samples in X have deviations smaller than $\mathbf{d}_{ub}$. Our goal is to choose the number of samples, K, such that

$$\mathbf{Pr}\left[H_0 \mid X\right] \leq \alpha.$$

We compute K by explicitly calculating this posterior probability of H_0. As with all Bayesian methods, both the results and the complexity of this calculation depend heavily on the choice of the prior probabilities of H_0 and H_1 (see Appendix A). For simplicity, we assume for now that

$$\mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] = \theta \sim \text{Uniform}(0,1).$$

Hence, the prior density $f_\theta(t) = 1$ for $t \in [0,1]$ and the prior probabilities of the hypotheses are

$$\mathbf{Pr}\left[H_0\right] = c \qquad \text{and} \qquad \mathbf{Pr}\left[H_1\right] = 1 - c.$$

Note that $\theta \in \Theta = [0,1]$ because θ is a probability.

Given a set of samples, X, for which every sample obeys $\mathbf{d}_{ub}$, we compute the posterior probability of H_0 as

$$\begin{aligned}\mathbf{Pr}\left[H_0 \mid X\right] &= \frac{\mathbf{Pr}\left[X, H_0\right]}{\mathbf{Pr}\left[X\right]} = \frac{\int_{\theta \in H_0} f_{X|\theta}(t)\ f_\theta(t) dt}{\int_{\theta \in \Theta} f_{X|\theta}(t)\ f_\theta(t) dt} \\ &= \frac{\int_0^c t^k dt}{\int_0^1 t^k dt} \\ &= c^{k+1} \end{aligned} \tag{10}$$

We can now use (10) to determine how large K must be in order to achieve the desired level of credibility. Specifically,

$$c^{K+1} \leq \alpha \implies K \geq \log_c \alpha \implies K \geq \frac{\log \frac{1}{\alpha}}{\log \frac{1}{c}}.$$

So, to test a given pair of hypotheses, we take at most this many samples. If we encounter a counterexample while sampling, we immediately accept H_0 and send the counterexample to the `Refiner` module. If *all sampled deviations* are upper bounded by the current $\mathbf{d}_{ub}$, we accept H_1 and return $\mathbf{d}_{ub}$ as our answer.

We call our hypothesis testing procedure $\texttt{Verifier}_{1-\alpha}(H_0, H_1)$, where H_0 and H_1 are the null and alternative hypothesis, respectively.

4.3 The `Refiner`: Updating our estimate of $\mathbf{d}_{ub}$

The `Refiner` is invoked whenever the `Verifier` rejects a particular value of $\mathbf{d}_{ub}$. This only happens when we have seen a counterexample that violates the current $\mathbf{d}_{ub}$. The `Refiner` module sets the new value of $\mathbf{d}_{ub}$ to be the deviation of this most recent counterexample plus some additional padding term ϵ which is set by the user. This new $\mathbf{d}_{ub}$ is then sent back to the `Verifier`.

The entire iterative hypothesis testing procedure with counterexample guided refinement is listed in Algorithm 1.

Algorithm 1 Computing upper bound on the deviation as defined in Eq. (7)

input : A transducer automaton $\mathcal{T}$, initial set $x[0]$, nominal run τ_{nom}, time bound H ; // $x[0]$ represents initial set of states

output: Compute an upper bound $\mathbf{d}_{ub}$ for $\mathbf{d}_{max}$

/* we assume parameters R, ϵ, α and c are provided by the user. */

$\mathbf{d}_{ub} \leftarrow \texttt{SmallSample}(\mathcal{T}, x[0], \tau_{nom})_{[R,\epsilon]}$; // 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. (8)

$H_1 \leftarrow \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] \geq c$; // form H_1 using Eq. (9)

$res \leftarrow \texttt{Verifier}_{1-\alpha}(H_0, H_1)$; // perform statistical verification

if res = *True* **then**

  return $\mathbf{d}_{ub}$

end

while *True* **do**

  $\mathbf{d}_{ub} \leftarrow \texttt{Refiner}(res)_{\epsilon}$; // 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 $\mathbf{d}_{ub}$

  $res \leftarrow \texttt{Verifier}_{1-\alpha}(H_0, H_1)$; // re-perform statistical verification

  if res = *True* **then**

    return $\mathbf{d}_{ub}$

  end

end

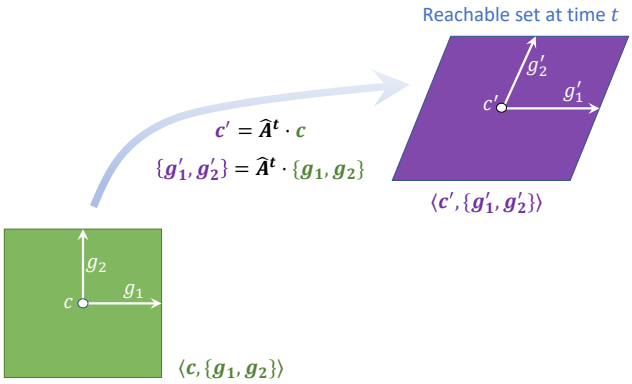


Fig. 5: **Reachability with zonotopes.**

5 Statistical Hypothesis Testing with Sets of Initial States

Computing maximum deviation under timing uncertainties (Problem 1), using our proposed Statistical Hypothesis Testing Approach, requires computing several random trajectories, and computing deviation between those trajectories with the given nominal trajectory. Here, it is worth recalling that trajectories are a sequence of set of states (see Fig. 2), as the evolution starts from an

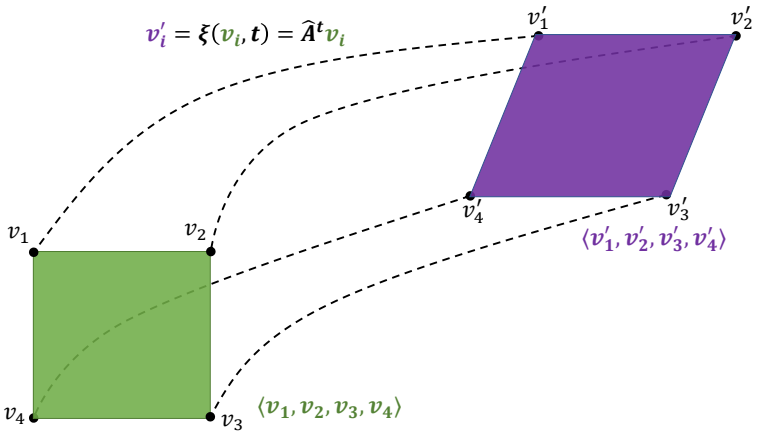


Fig. 6: **Reachability with V-Polytopes.**

initial set of states. That is, these runs indicate a *pipe* (as shown in Fig. 2) originating from the initial set of states, rather than just a single state within the given set. To compute the deviation between two such trajectories, we use Hausdorff distance at every time step (as defined in Eq. (6)). Thus, the representation of sets must be chosen based on its impact on the computational complexity while computing the trajectories and Hausdorff distance. Some representations, such as zonotopes [52] and stars [53], can very efficiently compute trajectories, but computing Hausdorff distance is hard. While some representations, such as V-Polytope [22, 54], can compute Hausdorff distance very efficiently, but computing trajectories is hard. Unfortunately, there is no one representation that can compute both the trajectories and the distance efficiently. Therefore, in this section, we discuss the following choices of representation and its computational complexity *vis-à-vis* computing trajectories and distances. Further note that the techniques mentioned in this section are also incorporated in Algorithm 1. **In other words, Algorithm 1 can handle initial set of states, not just a single state.**

5.1 Representing Sets using Zonotopes

A zonotope is an affine transformation of an unit (hyper-)box, represented by its center (c) and a set of generator vectors (G). Fig. 5 shows two zonotopes (in $\mathbb{R}^2$) in green and purple. A zonotope $\mathcal{Z}$ in $\mathbb{R}^n$ is define formally as follows.

Definition 3 (Zonotopes [52]) A zonotope $\mathcal{Z}$ is tuple $\langle c, G \rangle$; where $c \in \mathbb{R}^n$ is the center, and $G = \{g_1, g_2, \cdots, g_m\}$ is a set of m vectors in $\mathbb{R}^n$ called the generator vectors. The set of states represented by the zonotope $\mathcal{Z}$ is given as

$$[\![\mathcal{Z}]\!] = \{x \mid x = c + \sum_{i=1}^{m} \alpha_i g_i; \text{ such that, for all } i, -1 \leq \alpha_i \leq 1\}.$$

Given a linear system, with dynamics matrix $\hat{A}$, the reachable set at time step t can be computed as follows.

Definition 4 (Reachability using zonotopes) The reachable set at time step t, from an initial set represented using a zonotope $\mathcal{Z}_0$, is given by zonotope $\mathcal{Z}_t$; where

$$\mathcal{Z}_t = \hat{A}^t \cdot \mathcal{Z}_0 = \langle \hat{A}^t \cdot c, \hat{A}^t \cdot G \rangle = \langle c', G' \rangle.$$

This is also illustrated in Fig. 5, where $\mathcal{Z}_0$ and $\mathcal{Z}_t$ are illustrated in green and purple respectively.

Advantage. Using Definition 4, one can easily compute trajectories using $evol(\cdot)$ (see Eq. (5)), where each element of the sequence is a zonotope, if $x[0]$ is also represented using a zonotope.

Disadvantage. Computing Hausdorff distance between two zonotopes, $\mathcal{Z}_1$ and $\mathcal{Z}_2$, requires solving an optimization problem as it relies on vertex enumeration [22].

5.2 V-Polytopes: Representing Set using Vertices

V-polytopes are polytopes represented by their sets of vertices (formally called *extreme points*). Fig. 6 shows two V-Polytopes in green and purple.

Definition 5 (V-polytopes [55]) A V-polytope $\mathcal{V}$ is a convex polytope represented as $\langle v_1, v_2, \cdots, v_m \rangle$; where $v_i \in \mathbb{R}^n$ are the set of extreme points of the polytope (or vertices). The set of states represented by the polytope $\mathcal{V}$ is given as

$$[\![\mathcal{V}]\!] = \{x \mid x \in \texttt{CH}(v_1, v_2, \ldots, v_m); \text{ where } \texttt{CH}(\cdot) \text{ is a convex hull of a set of points}\}.$$

Given a linear system, with dynamics matrix $\hat{A}$, the reachable set at time step t using V-Polytopes can be computed as follows.

Definition 6 (Reachability using V-Polytopes) The reachable set at time step t, from an initial set represented using a V-Polytope $\mathcal{V}_0$, is given by V-Polytope $\mathcal{V}_t$; where

$$\mathcal{V}_t = \hat{A}^t \cdot \mathcal{V}_0 = \langle \hat{A}^t \cdot v_1, \hat{A}^t \cdot v_2, \cdots, \hat{A}^t \cdot v_m \rangle$$

This is also illustrated in Fig. 6, where $\mathcal{V}_0$ and $\mathcal{V}_t$ are illustrated in green and purple respectively.

Hausdorff distance between two V-Polytopes, $\mathcal{V}_1$ and $\mathcal{V}_2$, can be computed as follows.

Definition 7 (Hausdorff distance with V-Polytopes) Given two V-Polytopes, $\mathcal{V}_1 = \langle v_1^1, v_1^1, \cdots, v_p^1 \rangle$ and $\mathcal{V}_2 = \langle v_1^2, v_1^2, \cdots, v_q^2 \rangle$, the Hausdorff distance between $\mathcal{V}_1$ and $\mathcal{V}_2$ can be computed as:

$$\Delta(\mathcal{V}_1, \mathcal{V}_2) = \max \left\{ \max_i \left\{ \min_j \{ dis(v_i^1, v_j^2) \} \right\}, \max_j \left\{ \min_i \{ dis(v_i^1, v_j^2) \} \right\} \right\}$$

Advantage. Computing Hausdorff distance between two V-Polytopes, as given in Definition 7, is straightforward and computationally very efficient.

Disadvantage. Enumerating all potential vertices is required to represent a polytope with its vertices, and this process takes $O(2^n)$ time, where n is the system's dimension. While it introduces computational bottlenecks in high dimensional systems, *this is not a significant problem in low dimensional control applications.* Therefore, using Definition 6, one can compute trajectories using $evol(\cdot)$, where each element of the sequence is a V-Polytope, if $x[0]$ is also represented using a V-Polytope.

V-Polytopes: Our choice of representation

After considering the benefits and drawbacks of zonotopes and V-Polytopes, we chose to *use V-Polytopes in our experiments. However, we wish to note that one can adapt their implementation to use any representation (such as zonotopes, stars, V-Polytopes) according to their application—as our method poses no restriction on the choice of representation of sets.*

1. *Efficient implementation of Hausdorff distance.* Since our method heavily relies on performing random sampling of trajectories to compute its deviation from the nominal trajectory, an efficient computation of Hausdorff distance between trajectories is critical. For a typical case study, a single iteration of our algorithm requires computing deviation of over 1K trajectories, each up to a time bound of 150. Such an iteration involves 150K calculations of the Hausdorff distance. Thus, V-Polytope is an obvious choice in this regard.
2. *Computing Random Trajectories.* While zonotopes are more efficient at computing trajectories than V-Polytopes, this is not a significant problem for low-dimensional control applications. Moreover, given V-Polytopes' advantage of computing Hausdorff distance, we chose to use V-Polytopes in our experiments.

6 Handling Environmental Disturbances

So far we have only considered uncertainties in the initial set, resulting in initial set of states. We now take a closer look at how our proposed approach can be used for systems whose environmental disturbances produce additive noise in the set of states reached at each time step. Such a behavior can be modeled using the following dynamics.

$$x[t+1] = Ax[t] + Bu[t] + Cs[t],$$

where $u[t] = Kx[t-1]$, and $s[t]$ encodes the environmental disturbances experienced by the system at every time step.

Given $s[t]$, for all t, the evolution of the system, therefore, changes as

$$\widehat{evol}(\tau) = \big\{x[0], x[1] = A_1x[0] \oplus Cs[0], x[2] = A_2x[1] \oplus Cs[1], \\ \ldots, x[H] = A_Hx[H-1] \oplus Cs[H-1]\big\}, \tag{11}$$

where τ is a random run, and $\oplus$ denotes Minkowski sum between two sets.

We would like point out that the only change that occurs, as a result of environmental uncertainties, is in the computation of trajectories (*i.e.* $\widehat{evol}$), while the rest of the proposed method remains the same. Further note that the only change in, computation of $\widehat{evol}$, is the Minkowski sum of sets. Therefore, next we discuss how to compute Minkowski sum when the sets are represented as zonotopes and V-Polytopes.

Definition 8 (Minkowski sum of zonotopes) Given zonotopes, $\mathcal{Z}_1 = \langle c_1, G_1\rangle$ and $\mathcal{Z}_2 = \langle c_2, G_2\rangle$, we can compute the Minkowski sum, $\mathcal{Z} = \mathcal{Z}_1 \oplus \mathcal{Z}_2$ as follows. $\mathcal{Z} = \langle c, G\rangle$, where $c = c_1 + c_2$, and G is a list of generator vectors obtained by joining the list of generator vectors G_1 and G_2.

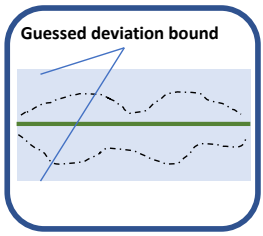
Clearly, when the sets are represented as zonotopes, computing trajectories can be performed very easily using Definition 8. However, note that every Minkowski sum results in adding more generator vectors in the representation. In other words, every Minkowski sum increases the representational complexity of zonotopes—which will have impact on the computation of Hausdorff distance too.

After discussing Minkowski sum of zonotopes, we now briefly discuss Minkowski sum of V-Polytopes. Using the algorithm proposed by Fukuda et al. [54], one can perform Minkowski sum of V-Polytopes in polynomial time—this further justifies our usage of V-Polytopes. In other words, zonotopes may experience computational inefficiencies while computing Hausdorff distance, however employing V-Polytopes does not lead to any significant computational bottleneck. **One can easily incorporate these modifications in Algorithm 1.**

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

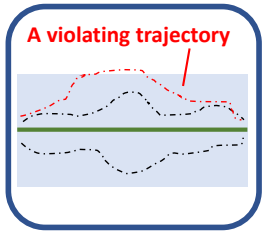
In this section, we demonstrate how the three modules described in Section 4, namely `Hypothesizer`, `Verifier`, and `Refiner`, are used by Algorithm 1 to compute an upper bound on the deviation ($\mathbf{d}_{ub}$) with a probabilistic guarantee. This is to intuitively demonstrate our main approach on the following illustrative example:

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



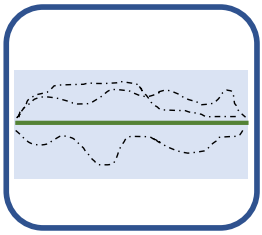
Step 1: Guess the deviation bound

The `Hypothesizer` *module guesses the upper bound by randomly generating few sample trajectories.*



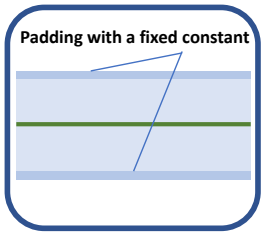
Step 2: Statistically verify the guessed bound

The `Verifier` *module verifies the guessed bound by generating K random trajectories. The red trajectory shows a violation of the guessed bound.*



Step 4: Statistically re-verify the guessed bound

Step 5: Return the accepted bound



Step 3: Refine the guessed bound

The `Refiner` *pads the deviation bound obtained from the counterexample with slack* ϵ

Fig. 7: 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.

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

We will compute the maximum deviation from a nominal trajectory with no deadline misses, starting from the initial state $x[0] = \langle [10 \;\; 10]^T \rangle$ (represented as a V-Polytope). 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. 7.

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, $\mathbf{d}_{ub} = 0.1462$.

The `Verifier` module, pre-tuned with $\alpha = 2.39 \times 10^{-}6$ 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 2: Next, the guessed upper bound $\mathbf{d}_{ub} = 0.1462$ is verified by invoking the module `Verifier`. The `Verifier` module, pre-tuned with $\alpha = 2.39 \times 10^{-}6$ 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 a Bayesian hypothesis testing approach over other methods.

7.1 Reason for Choosing Bayesian Hypothesis Testing

In theory, it is possible to any of a wide variety of statistical techniques, such as sequential hypothesis testing, to derive a safety guarantee. However, the Bayesian hypothesis testing approach discussed in this paper has several advantages.

1. The number of samples required by our procedure is largely independent of the number of possible trajectories. That is, the number of samples we require to perform a hypothesis test does not depend on whether the space of possible trajectories is finite, countably infinite, or uncountably infinite (e.g. trajectories are drawn from a continuous distribution).
2. Our method imposes no restriction on the distribution of trajectories produced by a given system. Our procedure only requires the ability to sample trajectories according to this underlying distribution.
3. Our Bayesian hypothesis testing procedure allows us to compute the exact number of samples required to either reject H_1 or accept H_1 with a specified bound on the probability of a type-I error. Other methods, such as sequential hypothesis testing, cannot compute the number of samples required perform a hypothesis test *a priori*.

4. Our procedure immediately rejects H_1 when a counterexample is found and uses this counterexample to refine our guess for $\mathbf{d}_{ub}$. This makes the bound produced by our procedure more conservative and also prevents us from spending too much time on hypothesis tests where H_1 is unlikely to be accepted. Sequential hypothesis testing may encounter multiple counterexamples during a single round of random sampling.

8 Experimental Evaluation

We extended the tool `StatDev`[1] [20], to implement handling initial set of states (as proposed in Section 5). Our tool is available both in Python[2] and Juila[3]. 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 [18, 56]. We demonstrate the applicability of our method given in Algorithm 1, extended to handle initial set of states as per Section 5, on four standard examples: an RC network [57], an electric steering application [32], an unstable second-order system [32] and a F1Tenth car model [15]. 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 3.

The following parameters were used in our study: $R = 50$, $\epsilon = 10^{-3}$, an initial state of $\langle [10\ \ 10]^T, [12\ \ 10]^T, [12\ \ 12]^T, [10\ \ 12]^T \rangle$ (represented as a V-Polytope) and time bound $H = 150$. For Fig. 9, we use a single initial state $[10\ \ 10]^T$, with rest of the parameters same. 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 1; *our statistical method always computes tighter bounds than `RS`.* 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).

[1] sites.google.com/view/statdev
[2] https://github.com/bineet-coderep/StatJitteryScheduler
[3] https://github.com/Ratfink/ControlTimingSafety.jl

8.1 Benchmarks

8.1.1 RC network

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

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

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

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

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.

8.1.2 Electric steering

The electric steering example is given by the following state space equations.

$$x[t+1] = \begin{bmatrix} 0.996 & 0.075 \\ -0.052 & 0.996 \end{bmatrix} x[t] + \begin{bmatrix} 0.100 & 0.003 \\ -0.003 & 0.083 \end{bmatrix} u[t]. \tag{14}$$

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

$$u[t] = \begin{bmatrix} 0.9067 & 0.07384 & 0 & 0 \\ 0.01041 & 0.9685 & 0 & 0 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}. \tag{15}$$

As before, using the matrices from Eqs. (14) and (15), we construct a transducer automaton for different deadline miss strategies. We used $\mathcal{T}_3$ to specify the scheduler.

8.1.3 Unstable second-order system

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

$$x[t+1] = \begin{bmatrix} 1.1053 & 0 \\ -0.0209 & 0.99 \end{bmatrix} x[t] + \begin{bmatrix} 0.0526 & 0.0105 \\ 0.0393 & 0.0994 \end{bmatrix} u[t] \tag{16}$$

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

$$u[t] = \begin{bmatrix} 4.7393 & -0.2430 \\ -0.2277 & 0.8620 \end{bmatrix} x[t-1]. \tag{17}$$

Using the matrices from Eqs. (16) and (17), we construct a transducer automaton $\mathcal{T}_1$ to specify the scheduler.

8.1.4 F1Tenth

The model of an F1Tenth car [15] was linearized and the following state space equations were obtained.

$$x[t+1] = \begin{bmatrix} 1 & 0.13 \\ 0 & 1 \end{bmatrix} x[t] + \begin{bmatrix} 0.02559 \\ 0.3937 \end{bmatrix} u[t] \tag{18}$$

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

$$u[t] = \begin{bmatrix} 0.2935 & 0.4403 \end{bmatrix} x[t-1]. \tag{19}$$

Using the matrices from Eqs. (18) and (19), we construct a transducer automaton $\mathcal{T}_3$ to specify the scheduler.

8.2 Results

The main results addressing (Q1) and (Q3) are summarized in Table 1. For the RC network, electric steering and F1Tenth, $\mathcal{T}_3$ (*i.e.*, the constraint "at most 3 consecutive misses") was used with $c = 0.99$ (and $\alpha = 2.39 \times 10^{-}6$). 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 0.9 to 0.9999 (with $\alpha \leq 2.41 \times 10^{-6}$ across the interval, maintaining a roughly constant value). We used the *Hold&Skip-Next* strategy for these experiments. The resulting values of $\mathbf{d}_{ub}$ are shown by boxplots and mean values in Fig. 8. Below we discuss the results in detail for all the examples.

8.2.1 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 $\mathbf{d}_{ub}$ to be 2.278 (0) for all the scheduling policies. The details are given in Table 1. The system behavior is shown in Figs. 9a and 10a, using the *Hold&Skip-Next* policy, and $c = 0.99$ (with $\alpha = 2.39 \times 10^{-}6$). Fig. 9a shows the system behavior with a single initial state, whereas Fig. 10a shows the system behavior with an initial set of states. Thus, we see trajectories as a series of points in Fig. 9a, and a series of polytopes in Fig. 10a. We plot Fig. 9a primarily to illustrate the role of safety envelopes, and how trajectories might violate it. The trajectories of the system from a single initial state will have safety envelopes plotted

Table 1: 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)	2.277 (0)	2.277 (0)	2.277 (0)	2.277 (0)
	$\mathbf{d}_{ub}$ (RS)	2.277	2.277	2.277	2.277
	Time Taken (Alg. 1)	1.745 s	1.774 s	1.878 s	1.831 s
	Time Taken (RS)	0.918 s	1.040 s	0.943 s	1.012 s
Electric Steering	$\mathbf{d}_{ub}$ (Alg. 1)	4.568 (0)	9.297 (0.28)	4.573 (0.027)	9.168 (0.28)
	$\mathbf{d}_{ub}$ (RS)	4.795	10.226	8.882	10.625
	Time Taken (Alg. 1)	1.74 s	2.90 s	1.79 s	2.90 s
	Time Taken (RS)	1.569 s	47.43 s	178.0 s	182.8 s
Unstable second-order	$\mathbf{d}_{ub}$ (Alg. 1)	3.959 (0)	14.969 (1.17)	4.632 (0)	12.767 (1.06)
	$\mathbf{d}_{ub}$ (RS)	4.269	—	5.162	—
	Time Taken (Alg. 1)	1.50 s	2.46 s	1.99 s	2.97 s
	Time Taken (RS)	1.068 s	timed out (> 1 h)	3.837 s	timed out (> 1 h)
F1Tenth	$\mathbf{d}_{ub}$ (Alg. 1)	10.42 (0)	19.08 (0.83)	18.53 (1.34)	18.90 (0.74)
	$\mathbf{d}_{ub}$ (RS)	10.425	—	—	—
	Time Taken (Alg. 1)	1.81 s	2.87 s	3.07 s	2.76 s
	Time Taken (RS)	171.1 s	timed out (> 1 h)	timed out (> 1 h)	timed out (> 1 h)

as a circles of radius $\mathbf{d}_{ub}$ (cyan) from the nominal behavior (black), as show in all the subplots of Fig. 9. However, this is not the case when the trajectories are resulting from an initial set of states. Thus, we indicate the violating trajectory—one that deviates more the $\mathbf{d}_{ub}$ from the nominal trajectory—in red, with the maximum and minimum violating time steps marked with arrows. The red trajectory shows safety violation (*i.e.* deviates more than $\mathbf{d}_{ub}$ from the nominal trajectory) by increasing the consecutive deadline from 3 to 4. The red arrow shows the point at which the system violates the safety. That is, at this point, the trajectory deviates more than $\mathbf{d}_{ub}$ from the nominal trajectory. Note that we did not find any safety violation with a single initial state (Fig. 9a).

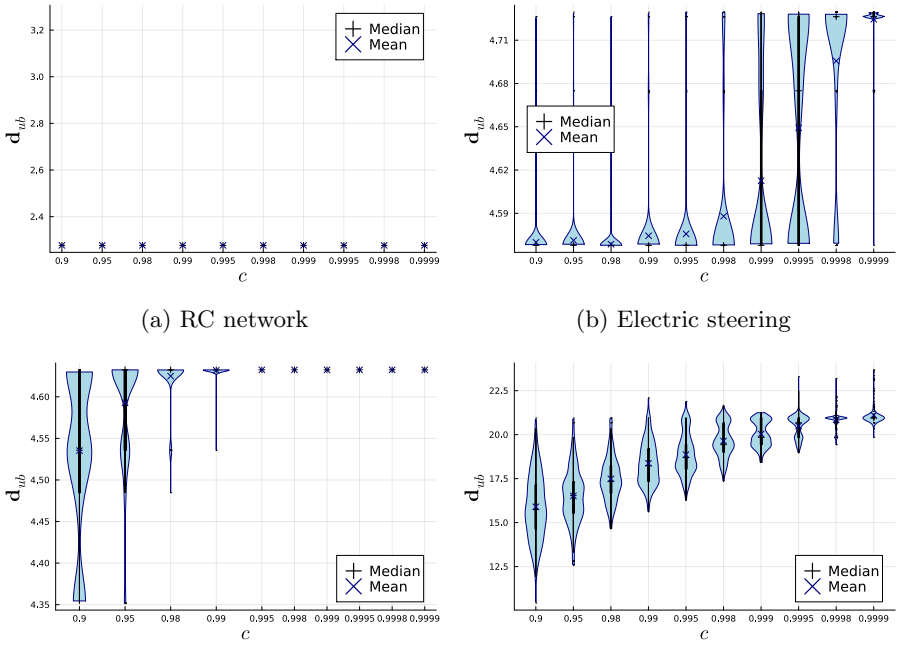
Q2: Considering at most 3 consecutive deadline misses allowed and $R = 10$, we gradually varied c from 0.9 to 0.9999. We observe that the mean $\mathbf{d}_{ub}$ does not change with any further increase in c from $c = 0.9$. This is shown in Fig. 8a.

Q3: The results show (see Table 1) that the stochastic method returns similar bounds as the RS method. However, the computation time is higher but not significantly high, *i.e*, under 2 s.

8.2.2 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 1). The system behavior is shown in Figs. 9b and 10b, using the *Hold&Skip-Next* policy, and $c = 0.99$ (with $\alpha = 2.39 \times 10^{-}6$). Further, the figures also show a possible safety violation that might occur when the number of consecutive deadline misses



(a) RC network

(b) Electric steering

(c) Unstable second-order

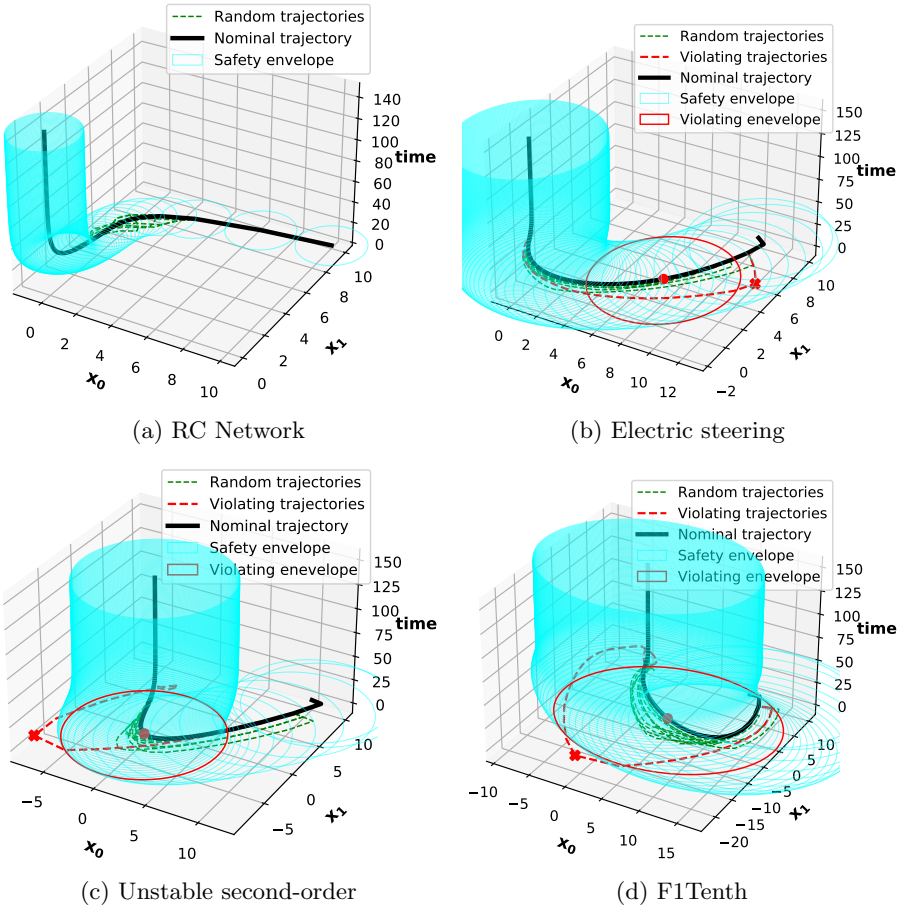
(d) F1Tenth

Fig. 8: Violin and box plots of computed $\mathbf{d}_{ub}$ values over varying probabilistic guarantee c on the computed bound. All boxes in Fig. 8a are single values.

just increases by 1, *i.e.*, considering at most 4 consecutive deadline misses. The red trajectories in Figs. 9b and 10b show a safety violation that might occur if the consecutive deadline misses increases by 1. The red dotted line in Fig. 9b shows the safety violation (with the existing safety envelope) 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 '×' (red). In Fig. 10b, we indicate the maximum and minimum violating time steps marked with arrows (red).

Q2: Considering at most 3 consecutive deadline misses allowed and $R = 10$, we gradually varied c from 0.9 to 0.9999. The result is given in Fig. 8b. We observe that the mean $\mathbf{d}_{ub}$ increases with increase in c, as expected. However, higher outlier deviation values are sometimes returned, unlike in the previous example. Also, note that with low values of c, we witness a wide range, which narrows as c increases.

Q3: As given in Table 1, we compute tighter values of $\mathbf{d}_{ub}$. The computation time for the stochastic method is lower for all strategies, except for *Hold&Kill* where it is slightly higher (≈ 0.2 s).



(a) RC Network

(b) Electric steering

(c) Unstable second-order

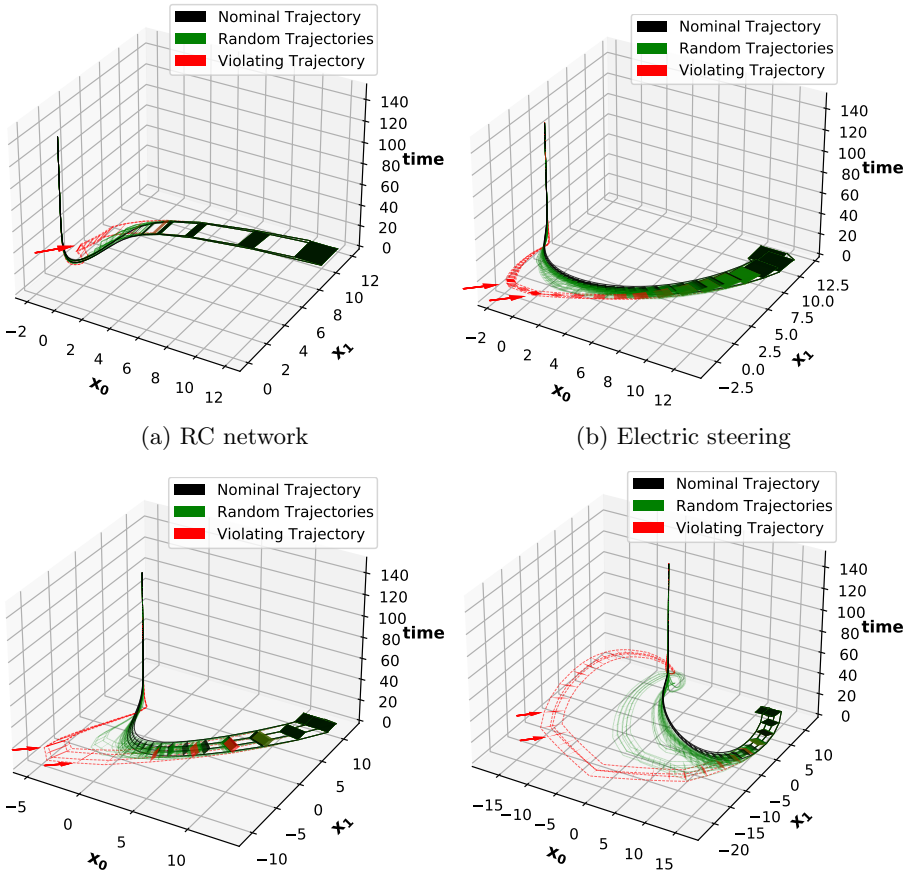
(d) F1Tenth

Fig. 9: **System behavior with a single initial state [20].** Safety envelopes at a distance of $\mathbf{d}_{ub}$ from the nominal trajectory, with random trajectories with one extra consecutive deadline miss.

8.2.3 Unstable second-order system

We now discuss the results obtained for the unstable second-order system example.

Q1: Unlike 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 1). The system behavior is shown in Figs. 9c and 10c, using the *Hold&Skip-Next* policy, along with a violating trajectory when the constraint changes to $\mathcal{T}_2$.



(a) RC network

(b) Electric steering

(c) Unstable second-order

(d) F1Tenth

Fig. 10: System behavior with random trajectories (green) that are within $\mathbf{d}_{ub}$ distance from the nominal trajectory (black), and one violating trajectory (red) with one extra consecutive deadline miss—this trajectory deviates more than $\mathbf{d}_{ub}$ from the nominal trajectory.

Q2: Considering at most 1 consecutive deadline miss allowed and $R = 10$, we gradually varied c from 0.9 to 0.9999. The result is given in Fig. 8c. We observe that the mean $\mathbf{d}_{ub}$ increases with increase in c, as expected. Also, note that with low values of c, we witness a wide range, which narrows (to zero) as c increases—$\mathbf{d}_{ub}$ seems to stabilizing at $c = 0.995$.

Q3: As given in Table 1, we compute tighter values of $\mathbf{d}_{ub}$ in less computation time for all the scheduling policies. Moreover, the `RS` method times out (> 1 h) for *Zero&Kill* and *Zero&Skip-Next* policies.

8.2.4 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 Figs. 9d and 10d, using the *Hold&Skip-Next* policy, and $c = 0.99$. Further, the figures also show a 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 distribution decreases sharply, in the F1Tenth example the decrease is not as drastic. This is shown in Fig. 8d. However, following the method of [32], 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, for all policies, under 3.5 s.

8.3 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 1) 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. Note that our statistical method was able to computer tighter bounds, compared to the RS method, for all the examples. Next, we discuss why the RS method timed out in most cases. By segmenting the time horizon into manageable chunks, the RS technique computes the reachable set by computing all possible trajectories for each segment. It is possible to compute all possible trajectories when each of these parts is small enough. By dividing the time horizon into smaller chunks, we were able to get useful deviation bounds for some of the benchmarks, but not for the majority of them. As a result, we had to increase the number of segments by which we break the time horizon, which made it impossible to compute every possible trajectory in that segment and caused a timeout.

***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.

Comments on Fig. 8 and Fig. 10

Further, we make general comments on Fig. 8 and Fig. 10 as follows.

Varying the value of* c *(Fig. 8). We observe that for lower values of c, the mean values of $\mathbf{d}_{ub}$ are lower but the distributions are wider, however, as c increases the mean values of $\mathbf{d}_{ub}$ increase whereas the width of the distribution 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. At lower values of c, the distribution 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 the distribution narrowing 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.

System Behavior (Fig. 10). In this figure, we show the behavior of the system using the *Hold&Skip-Next* and $c = 0.99$. For RC network, electric steering and F1Tenth, all the trajectories satisfying the $\mathcal{T}_3$ constraint will be within $\mathbf{d}_{ub}$ from the nominal trajectory (with a probabilistic guarantee). Similarly, for the unstable second order system we used the constraint $\mathcal{T}_1$. We observe that for the electric RC network example, the violating trajectory is closer to the random trajectories, compared the other three examples—this might be due to that fact that RC network behaves less erratically than the other three examples. Further, we evaluated the robustness of the system *w.r.t* the computed $\mathbf{d}_{ub}$, by changing the constraint from $\mathcal{T}_3$ to $\mathcal{T}_4$ for RC network, electric steering and F1Tenth, and from $\mathcal{T}_1$ to $\mathcal{T}_2$ for the unstable second order system.

The violating trajectory is shown in red, and the arrows mark the minimum and maximum violating time steps.

Using Fig. 8 to compute Fig. 10. Note that Fig. 8 suggests a heuristic to choose a value for c (at which the mean stabilizes with a narrow distribution). Once such a c is chosen, the safety envelope should be computed, as in Fig. 10, using that value of c.

9 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. Also, we have extended our approach to handle initial set of states, instead of single states. While considering such sets, the representation is critical to the overall complexity of the problem. In this work, we have discussed some standard representations, and how they can used adapted for our work. Further, we have also discussed how our method can be adapted to handle behavioral uncertainties of the system. 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) [40]. 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. The counter example guided statistical hypothesis testing can also be applied to black-box systems, where obtaining a precise model is challenging. Finally, this efficient method for safety verification, as proposed in this paper, may also be used for schedule synthesis [58] where control tasks might miss deadlines, but nevertheless satisfy system-level safety properties. It is also worth noting that while the focus of this paper is on timing uncertainties, the underlying principle of *focussing in the system-level property* and not focusing on "secondary" properties like timing behavior is applicable more generally. For example, when messages are not fully encrypted or authenticated for security [59, 60], it might be shown that a safety property of the form studied in this paper cannot be violated even if the system is under attack. Similar results may also be established in the case of ensuring system reliability [61] under resource constraints.

Acknowledgements: We thank all the reviewers of the RTCSA 2022 version of this paper, as well as the reviewers of this journal version, for their helpful feedback. This work was partially supported by the NSF award #2038960.

References

[1] Fisher, M., *et al.*: Verifying autonomous systems. Commun. ACM **56**(9), 84–93 (2013)

[2] Wing, J.: Trustworthy AI. Commun. ACM **64**(10), 64–71 (2021)

[3] Dennis, L., Fisher, M.: Verifiable self-aware agent-based autonomous systems. Proc. IEEE **108**(7), 1011–1026 (2020)

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

[5] Axer, P., *et al.*: Building timing predictable embedded systems. ACM Trans. Embedded Comput. Syst. **13**(4), 82–18237 (2014)

[6] Thiele, L., Wilhelm, R.: Design for timing predictability. Real-Time Systems **28**(2-3), 157–177 (2004)

[7] Wilhelm, R.: Real time spent on real time. Commun. ACM **63**(10), 54–60 (2020)

[8] Pazzaglia, P., *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] Soudbakhsh, D., *et al.*: Co-design of arbitrated network control systems with overrun strategies. IEEE Trans. Control. Netw. Syst. **5**(1), 128–141 (2018)

[10] Blind, R., Allgöwer, F.: 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] Liberzon, D.: Switching in Systems and Control. Springer, ??? (2003)

[12] Zhang, W., *et al.*: Stability of networked control systems. IEEE Control Systems Magazine **21**(1), 84–99 (2001)

[13] Roy, D., Zhang, L., Chang, W., Mitter, S.K., Chakraborty, S.: Semantics-preserving cosynthesis of cyber-physical systems. Proc. IEEE **106**(1), 171–200 (2018)

[14] Goswami, D., Schneider, R., Chakraborty, S.: Relaxing signal delay constraints in distributed embedded controllers. IEEE Trans. Contr. Sys. Techn. **22**(6), 2337–2345 (2014)

[15] O'Kelly, M., Zheng, H., Karthik, D., Mangharam, R.: F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. Proceedings of Machine Learning Research **123**, 77–89 (2020)

[16] Kass, R., Raftery, A.: Bayes factors. Journal of the American Statistical Association **90**(430), 773–795 (1995)

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

[18] Bernardi, O., Giménez, O.: A linear algorithm for the random sampling from regular languages. Algorithmica **62**, 130–145 (2010)

[19] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) 12th International Conference on Computer Aided Verification (CAV) (2000)

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

[21] Hagemann, W.: Reachability analysis of hybrid systems using symbolic orthogonal projections. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification (CAV) (2014)

[22] Sadraddini, S., Tedrake, R.: Linear encodings for polytope containment problems. In: 2019 IEEE 58th Conference on Decision and Control (CDC) (2019)

[23] Chang, W., Goswami, D., Chakraborty, S., Hamann, A.: OS-aware automotive controller design using non-uniform sampling. ACM Transactions on Cyber-Physical Systems TCPS **2**(4), 26–12622 (2018)

[24] Chang, W., Goswami, D., Chakraborty, S., Ju, L., Xue, C.J., Andalam, S.: Memory-aware embedded control systems design. IEEE Trans. on CAD of Integrated Circuits and Systems **36**(4), 586–599 (2017)

[25] Chang, W., Chakraborty, S.: Resource-aware automotive control systems design: A cyber-physical systems approach. Foundations and Trends in Electronic Design Automation **10**(4), 249–369 (2016)

[26] Chang, W., Roy, D., Hu, X.S., Chakraborty, S.: Cache-aware task scheduling for maximizing control performance. In: Design, Automation & Test in Europe (DATE) (2018)

[27] Chakraborty, S., *et al.*: Embedded systems and software challenges in electric vehicles. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) (2012)

[28] Chakraborty, S., Faruque, M.A.A., Chang, W., Goswami, D., Wolf, M., Zhu, Q.: Automotive cyber-physical systems: A tutorial introduction. IEEE Design & Test **33**(4), 92–108 (2016)

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

[30] Schneider, R., *et al.*: Constraint-driven synthesis and tool-support for flexray-based automotive control systems. In: CODES+ISSS (2011)

[31] Schneider, R., *et al.*: Multi-layered scheduling of mixed-criticality cyber-physical systems. Journal of Systems Architecture - Embedded Systems Design **59**(10-D), 1215–1230 (2013)

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

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

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

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

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

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

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

[39] Younes, H., Simmons, R.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV (2002)

[40] Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R.: Statistical Model Checking, pp. 478–504. Springer, Cham (2019)

[41] Donkers, M., *et al.*: Stability analysis of stochastic networked control systems. Automatica **48**(5), 917–925 (2012)

[42] Cloosterman, M., *et al.*: Stability of networked control systems with uncertain time-varying delays. IEEE Trans. Automat. Contr. **54**(7), 1575–1580 (2009)

[43] Hespanha, J.: Modeling and analysis of networked control systems using stochastic hybrid systems. Annual Reviews in Control **38**(2), 155–170 (2014)

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

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

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

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

[48] Roy, D., *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)

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

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

[51] Hespanha, J.P.: Linear Systems Theory: Second Edition. Princeton University Press, USA (2018)

[52] Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Proceedings of the 8th International Conference on Hybrid Systems:

Computation and Control (HSCC) (2005)

[53] Duggirala, P.S., Viswanathan, M.: Parsimonious, simulation based verification of linear systems. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification (CAV) (2016)

[54] Fukuda, K.: From the zonotope construction to the minkowski addition of convex polytopes. Journal of Symbolic Computation **38**(4), 1261–1272 (2004). Symbolic Computation in Algebra and Geometry

[55] Grünbaum, B., Kaibel, V., Klee, V., Ziegler, G.M.: Convex Polytopes. Springer, New York (2003)

[56] Flajolet, P., *et al.*: A calculus for the random generation of labelled combinatorial structures. Theoretical Computer Science **132**(1), 1–35 (1994)

[57] Gabel, R.A., Roberts, R.A.: Signals and Linear Systems. John Wiley & Sons, USA (1991)

[58] Xu, S., Ghosh, B., Hobbs, C., Thiagarajan, P.S., Chakraborty, S.: Safety-aware flexible schedule synthesis for cyber-physical systems using weakly-hard constraints. In: 28th Asia and South Pacific Design Automation Conference (ASP-DAC) (2023)

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

[60] Waszecki, P., *et al.*: Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **36**(11), 1790–1803 (2017)

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

[62] Ghosh, M.: Objective priors: An introduction for frequentists. Statistical Science **26**(2), 187–202 (2011). Accessed 2023-06-18

[63] Chen, J.J., Novick, M.R.: Bayesian analysis for binomial models with generalized beta prior distributions. Journal of Educational Statistics **9**(2), 163–175 (1984)

Appendix

A Case Study: The Impact of Prior Selection

The methodology presented in Section 4 assumes a uniform prior distribution. Specifically, we assumed

$$\mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}\right] = \theta \sim \text{Uniform}(0, 1).$$

However, the number of samples required to conclude that $\theta \geq 0.99$ with sufficiently high probability depends heavily on the choice of prior. Our choice of a uniform prior feels safe in that it does not assume any prior knowledge of the underlying system. A variety of more nuanced procedures exist for selecting so-called objective priors [62] that avoid some of the pitfalls of using a uniform prior. Plugging these priors into our procedure is straightforward, and a full discussion of objective priors is beyond the scope of this paper.

To illustrate the effect of prior knowledge on our testing procedure, we now consider how the choice of prior impacts the number of samples required by our hypothesis testing procedure. Recall that our hypothesis testing procedure boiled down to estimating the value of θ given that each sampled trajectory would independently obey $\mathbf{d}_{ub}$ with probability θ. Specifically, we take K samples and then try to estimate θ given that $\text{Binomial}(K, \theta) = K$. It is well known that the beta distribution is a *conjugate prior* of binomial likelihood functions — any beta prior distribution and a binomial likelihood function will induce a beta posterior distribution on θ [63]. Hence, to examine the impact of the choice of prior, we consider two alternate beta distributions as priors.

We compare these choices of prior to the uniform prior used in the paper. For this comparison, we again assume that the user-defined parameter $c = 0.99$. That is, we would like to say whether a given $\mathbf{d}_{ub}$ is at least a 99th percentile of the distribution of trajectory deviations. We consider beta prior distributions with modes at $\theta = 0.25$, and $\theta = 0.99$ respectively. [4] These priors reflect two common cases. First, the prior with a peak at $\theta = 0.99$ reflects the case where one strongly believes that θ is *close* to 0.99, but is also agnostic as to whether the true value of θ is above or below 0.99. Second, the prior with a peak a $\theta = 0.25$ reflects the case where one chooses a prior conservatively to ensure the safety of the system. These priors are shown in Figure 11.

Based on each of these priors, we can compute the posterior distribution induced by sampling K trajectories which all obey a given value of $\mathbf{d}_{ub}$. Figure 12 shows the probability of a type-I error, α, as a function of the number of samples, K. We see that the choice of prior significantly impacts the number of samples required by our hypothesis testing procedure. First, we can see that using a prior with a peak at $\theta = 0.25$ has the intended effect. Meeting a given level of α when using this prior requires roughly twice as many samples as were required when using a uniform prior. Surprisingly, the prior with a

[4] Note that $\text{Uniform}(0, 1) \overset{d}{=} \text{Beta}(1, 1)$, so the uniform prior also induces a beta posterior distribution on θ.

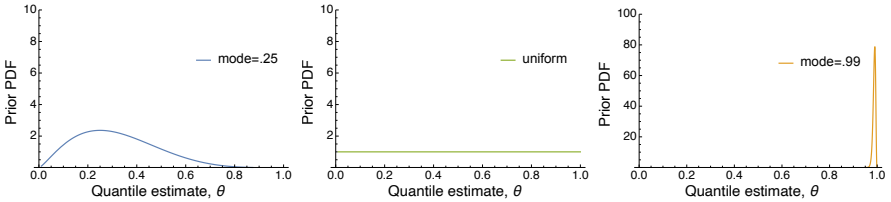


Fig. 11: A variety of prior distributions that can be plugged into our hypothesis testing procedure. We use beta prior distributions because they emit simple posterior distributions when used with a binomial likelihood function. We consider the uniform prior used in Section 4 as well as a prior with a mode of $\theta = 0.25$ and a prior with a mode of $\theta = 0.99$.

peak at $\theta = 0.99$ does not have the opposite effect. Given a strong prior belief that θ is close to 0.99, we might expect to require fewer samples than when using a uniform prior. Figure 12 shows that, although α is slightly lower for small values of K, we actually require *more* samples when using the prior with a peak at .99 than were required when using a uniform prior. The issue is that this prior assigns significant probability density to values slightly above and slightly below 0.99. A large number of samples is then required to decide whether θ is actually above 0.99 or just slightly below 0.99. Hence, while the uniform prior is in some sense non-informative, both of the alternate priors shown here are more conservative in the number of samples they require before allowing us to accept a given value of $\mathbf{d}_{ub}$.

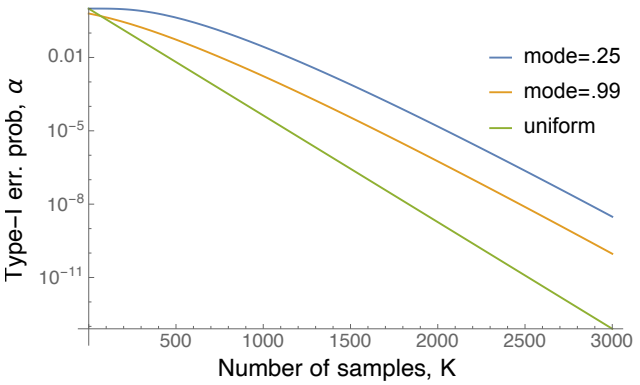


Fig. 12: Type-I error, α, as a function of the sample size K under a variety of beta prior distributions. Using a prior with a mode of 0.25 has the expected effect of requiring more samples than are required when using a uniform prior. Surprisingly, using a prior with a mode of 0.99 also requires more samples than are required when using a uniform prior.