2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

# Probabilistic Rare-Event Verification for Temporal Logic Robot Tasks

Guy Scher[1], Sadra Sadraddini[2] and Hadas Kress-Gazit[1]

***Abstract*— We present a method for calculating the probability that a robot successfully performs a task described using Signal Temporal Logic (STL). We focus on cases where the failure probability is very small, hence a traditional Monte-Carlo method becomes inefficient due to the large number of samples required to observe failures. Using elliptical sliced sampling, normalizing flows, and Bayesian optimization, we develop an algorithm that, under mild assumptions, is applicable to black-box systems, and can be applied to uncertainty sources with non-Gaussian probabilities. We demonstrate the application of our method on three different simulated robots.**

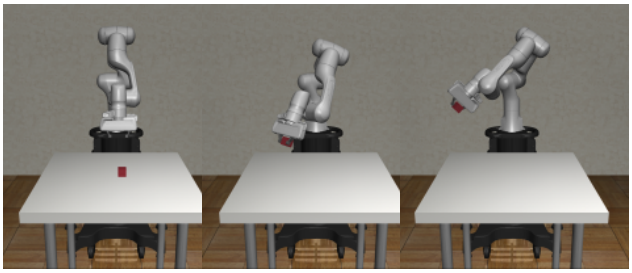
## I. Introduction

As robots become more autonomous and perform longer and more complex tasks around humans or in other sensitive domains, it becomes even more important to ensure their safety (e.g., [1]). Safety must be verified with respect to different environment settings (e.g. weather, lighting), sensor noises, agents with different behaviors that may interact with the robot, and different infrastructure conditions (e.g., roads) that the robot may experience.

In order to certify safety against rare/extreme events, regulatory organizations, such as the U.S. department of transportation (DOT) or the Federal Aviation Administration (FAA), rely on reliability and past performances. For example, for autonomous cars there exists specific braking system tests [2], hardware end-to-end tests, and the requirement of having less than 1 fatality per 100 million driven miles [1], [3], [4]. However, these tests cannot capture all possible scenarios that a car, as a system, can experience in its lifespan; furthermore, such tests may consume many resources [5].

Reasoning about the safety of complex systems operating in environments with uncertainty sources (disturbances, sensor errors, parametric uncertainties, etc.) is often impossible with formal analytical tools. Instead, Simulation-based tools such as statistical model checking [6] are preferred. Given a high-fidelity simulator, these approaches sample executions and analyze them using various statistical methods to determine whether the properties hold [7]. While these methods may not provide accurate results, they do provide confidence bounds. However, when dealing with rare-events, simple Monte-Carlo simulations will not suffice as the number of simulations needed to find a failure event is inversely proportional to the probability of failure. Considering complex systems, high-fidelity models and long simulation time horizons, it may be very expensive to run a lot of simulations.

[1] Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14850, USA {gs679,hadaskg}@cornell.edu
[2] Dexai Robotics, Boston, MA, USA sadra@dexai.com
This work was supported by ONR PERISCOPE N00014-17-1-2699.



(a) (b) (c)

Fig. 1: A simulated robotic manipulator with a policy trained via reinforcement learning fails to lift the cube above 0.5m. Using the methods in this paper, we estimate the failure probability to be around 0.1%, which is too low to be discovered via direct sampling.

In this work we present an algorithm to speed up the process of finding the rare events in a simulation with uncertainties, where the robot needs to perform a task defined by Signal Temporal Logic (STL) [8]. STL is used to reason about complex temporal behavior of real-valued signals such as safety, liveness, and sequentiality, and its quantitative semantics provide a measure to evaluate how close a signal is to satisfaction. We focus on finding the probability of success and its bounds to certify an autonomous robot.

**Contribution:** We build upon the STL-based ESS (Signal Temporal Logic based Elliptical Slice Sampling) verification in [9] for linear systems with Gaussian uncertainties. We relax the linear and Gaussian assumptions to verify non-linear and black-box systems with uncertainties with arbitrary distributions. We combine Bayesian optimization with ESS to develop a method that is able to efficiently sample sets of uncertainty inputs (noises) that would cause the robot to satisfy or falsify a specification written in STL. This allows a user to examine instances of robot failures, in addition to calculating their probability. Fig. 1 depicts an example (Sec. V-D) of a robot whose task it is to pick up a cube from the table and lift it to a height of at least 0.5m. The figure shows one initial cube position, discovered by our algorithm, where the robot fails its task.

**Related work:** There are several sampling techniques in the literature to accelerate the random sampling of Monte-Carlo. In [10], the authors generate a sequence of distributions that would take the simulation from a prior distribution, to the distribution where the system fails a safety function, $f(X) \leq \gamma$ where $X$ is the noise source distribution and $\gamma$ is the safety measure. They use a combination of exploration and optimization to suggest the next distribution to sample from. Using this sequence (ladder) of distributions, they can

estimate the probability of failure $p_\gamma\left(f(X) \leq \gamma\right)$.

In [11], [12], importance sampling and cross-entropy methods are employed to skew the sampling distributions to yield failures in an autonomous driving simulation with a learned model of the world and full perception-based control of the car. Finding failure events using simulation-based falsification are used as well [13]–[15]. These methods, while useful for the design of controllers and identifying failure modes, do not capture the likelihood of a failure to happen, thereby missing a crucial understanding of how critical this mode is to the overall system safety.

## II. Preliminaries

### A. Signal Temporal Logic

Signal Temporal Logic (STL) [16] is a temporal logic that allows writing specifications over continuous signals, capturing complex behaviors for a system to achieve. For example, STL may capture tasks such as "*reach the cup between* 5 *to* 10 *seconds or after* 15 *seconds, and avoid collisions*" for a robotic manipulator.

In this work, we use discrete-time signals $\mathbf{y} = y_0, y_1, \ldots$ where $y_t \in \mathbb{R}^d, \forall t \in \mathbb{N}$. A *predicate* over $\mathbb{R}^d$ is denoted by $\nu = \left(g\left(y\right) \geq 0\right)$ where $g : \mathbb{R}^d \to \mathbb{R}$. STL specifications are defined recursively:

$$\varphi := \top \mid \nu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_{[t_1,t_2]} \varphi_2 \quad (1)$$

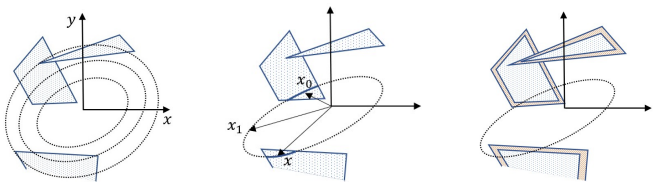
where $\top$ is `True` and $\varphi, \varphi_1$, and $\varphi_2$ are STL formulae. $\vee, \wedge$ are disjunction and conjunction, respectively, $\neg$ is the unary negation operator and $\mathcal{U}$ represents the *until* operator. The *until* operator is bound over the time interval $[t_1, t_2]$. From these operators, one can create *implication* $\varphi_1 \Rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, timed *eventually* $\Diamond_{[t_1,t_2]}\varphi \equiv \top\ \mathcal{U}_{[t_1,t_2]}\varphi$ and timed *always* $\Box_{[t_1,t_2]}\varphi \equiv \neg\left(\Diamond_{[t_1,t_2]}\neg\varphi\right)$. We define the horizon $H_\varphi$ as the minimum time interval needed to correctly evaluate if $\varphi$ is satisfied (see [9]) hence we consider time horizons $T \geq H_\varphi$.

STL has two types of semantics [16]: *Boolean* semantics, where $\varphi$ is either True or False, and *robust* or *quantitative* semantics which captures the closeness of a signal to the satisfaction (violation) of $\varphi$ using the robustness metric, $\rho(\mathbf{y}, \varphi, t) \in \mathbb{R}$ [9]. Positive robustness means $\varphi$ is satisfied. Negative robustness means the specification is violated.

### B. STL-based Elliptical Slice Sampling

This work builds on STL-based ESS (Elliptical Slice Sampling) [9], which provides an efficient computational framework for computing the probability that a linear system with Gaussian noises satisfies any STL specification $\varphi$ with affine predicates $g\left(y\right) = a'y + b$, where $a \in \mathbb{R}^d, b \in \mathbb{R}$. We describe the main approach in [9] for completeness and to highlight the contributions of this paper (removing the linearity and Gaussian assumptions).

Given a linear system with Gaussian noise and an STL formula $\varphi$, [9] first represents the set of possible trajectories of horizon $T$ using a multivariate Gaussian distribution in $\mathbb{R}^{d \cdot T/\Delta t}$, where each point is a trajectory of the system over $T/\Delta t$ time steps where $\Delta t$ is the sampling time. It then creates linear constrained domains $\mathcal{L}$ from the specification constraints using the obstacles, goals or any other predicate in $\varphi$ [9]. Fig. 2a depicts the trajectory Gaussian and the linear domains. Trajectory points that are in the linear domain correspond to trajectories that satisfy (or falsify, depending on the construction) the specification $\varphi$. To find the probability of success (failure), we integrate the probability mass of the trajectory Gaussian over the linear domains.




(a) (b) (c)

Fig. 2: The steps of the STL-based ESS approach [9]. (a) The Gaussian $x_{traj} \sim \mathcal{N}(\mu, \Sigma)$ is represented with the dashed ellipse, and the constrained domains $\mathcal{L}$ are in blue. The objective is to evaluate $p(x_{traj} \in \mathcal{L})$, i.e. the integral of the Gaussian over the blue polygons. (b) The ESS process - given a point $x_0 \in \mathcal{L}$, sample a new point $x \sim \mathcal{N}(\mu, \Sigma) \in \mathcal{L}$. (c) The HDR phase where the domains are enlarged. The probability of the original blue domain: $p(\text{blue}) = p(\text{blue} \,|\, \text{orange} + \text{blue})p(\text{orange} + \text{blue})$.

Since one cannot directly compute the integral, [9] estimates the probability in an iterative, two-step process: First, it uses an Elliptical Slice Sampler (ESS) [17], a Markov Chain Monte-Carlo (MCMC) method, to sample from a posterior where the prior is a multivariate Gaussian $\mathcal{N}(\mu, \Sigma)$. In this case, the posterior distribution is the constrained Gaussian. ESS provides a rejection-free method for sampling a new point $x \in \mathcal{L}$, given a previous point $x_0 \in \mathcal{L}$. It is done by forming an auxiliary ellipse (based on the Metropolis–Hastings method [18] which also gives a richer choice of update steps and is parameter-free) using another point $x_1 \sim \mathcal{N}(\mu, \Sigma)$ (not necessarily in $\mathcal{L}$) and $x_0$, as shown in Fig. 2b. Then, it is possible to sample the new $x$ uniformly from the segments of the ellipse that are in $\mathcal{L}$ (active segments), parameterized only by the scalar $\theta$ that defines the ellipse. There exists an analytical closed-form solution for the intersections of the predicates' hyperplanes (boundaries of $\mathcal{L}$) and the ellipse due to the restriction to linear predicates [19]. The solution allows us to sample new points efficiently and rejection-free from the active segments.

Second, to estimate the probability, [9] uses the Holmes-Diaconis-Ross (HDR) algorithm [20], which is a multi-level splitting algorithm that estimates the probability of sampling from any distribution. In this context, it is enlarging the domain $\mathcal{L}$ to $\mathcal{L}'$ (referred to as a nesting), thus increasing the probability of sampling from it, as shown in Fig. 2c. Then, the probability of sampling from the original domain is a product of conditional probabilities $p(\mathcal{L}) = p(\mathcal{L}|\mathcal{L}')p(\mathcal{L}')$. In each nesting, [9] samples $N_{\text{ESS}}$ points using the ESS algorithm and computes the conditional probability $p(\mathcal{L}|\mathcal{L}') = \sum I(x \in \mathcal{L}) / \sum I(x \in \mathcal{L}')$ where $I(x \in \mathcal{L}) = 1$ if $x \in \mathcal{L}$

and 0 otherwise. This two steps approach allows for sampling from arbitrary small-density functions.

One of the innovations of [9] is that the constrained domains are not computed explicitly, thus the computation is not affected by the combinatorial aspect of a satisfying trajectory (e.g. computing the domain of hitting an obstacle at $t_1$, or at $t_2$ or at $t_1$ and $t_2$, etc.). We can also efficiently sample new points (trajectories) that satisfy the specification. The method exploits the STL robustness score to evaluate if a point (trajectory) is within the domain and its distance from satisfying/violating the specification, allowing for efficient computation for arbitrarily complex STL specifications.

## III. Problem setup

We consider non-linear systems $\mathcal{S}$ of the form

$$x_{t+1} = f(x_t, u_t, w_{t+1}), \quad y_t = g(x_t, u_t, w_{t+1}) \tag{2}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the control input, $y \in \mathbb{R}^d$ is output, and $w \in \mathbb{R}^l$ is the uncertainty input, which can affect both the process as disturbance and output as noise. For simplicity and with a slight abuse of notation, we denote the initial condition $x_0$ as a function of $w_0$.

We assume a controller for the system is given that maps the states (or internal measurements) to control inputs $u_t = h(x_{0:t})$. Given the system and controller, a run of the closed-loop system for $N$-steps (sampling time - $\Delta t$) is a function of the uncertainty inputs:

$$y_{0:N-1} = r(w_{0:N-1}), \tag{3}$$

where $r : \mathbb{R}^{lN} \to \mathbb{R}^{dN}$ is the *run* function.

Given an STL formula $\varphi$ with predicates over $y$ and horizon $N \geq H_\varphi/\Delta t$, the robustness function is $\rho(y_{0:N-1}, \varphi, 0)$, which for simplicity we write as $\varrho^\varphi(y_{0:N-1})$, where $\varrho^\varphi : \mathbb{R}^{dN} \to \mathbb{R}$ maps the run of the system into the real-valued robustness function for formula $\varphi$.

**Assumption 1.** *The map from system run to STL robustness $\varrho^\varphi(y_{0:N-1})$ is Lipschitz continuous.*

**Assumption 2.** *The uncertainty source inputs $w_{0:N-1}$ form a probability space with a* known *distribution $\eta(w_{0:N-1}), \eta : \mathbb{R}^{lN} \to \mathbb{R}_+$.*

Assumption 1 is standard for which a sufficient condition is the system in Eq. (2) and functions appearing in STL predicates being Lipschtiz continuous [21]. Assumption 2 is necessary to draw samples of the random inputs and obtain runs of the system in a Monte Carlo setting. Note that the analytical form of the functions in Eq. (2) may not be fully known and we can treat the simulator as a black box.

**Problem 1.** *Given a system $\mathcal{S}$, a probability distribution $\eta$ for the uncertainties $\boldsymbol{w}$, an STL specification $\varphi$, and time horizon $T \geq H_\varphi$, find the probability $p_s$ that $\mathcal{S}$ satisfies (or violates $p_f$) the specification.*

Note that finding the probability of satisfaction, $p_s$ or violation $p_f$, is interchangeable because we can use negation to convert a desired behavior to an undesired behavior $\psi = \neg\varphi$ and verify $\psi$. Also note that the simulation time horizon $T$ of different simulation runs does not need to be identical; we require that $T \geq H_\varphi$ to ensure the full $\varphi$ is evaluated.

## IV. Robustness-based Verification

To verify non-linear systems with non-Gaussian uncertainties w.r.t to STL specifications (Problem 1), we follow the two step process in [9]; that is, we integrate $\boldsymbol{w} \sim \eta$, as opposed to the trajectory Gaussian in [9], under the constrained domains representing positive robustness of the specification $\varphi$ (no longer required to be affine).

Integrating the trajectory-space Gaussian to find the probability of satisfying an STL specification, as in [9], is not possible in this setting because propagating Gaussian noises in non-linear systems or using non-Gaussian noise distributions as the uncertainty input, will yield a non-Gaussian distribution in the trajectory space. Instead, in this work we use Gaussian uncertainties $\boldsymbol{w} \in \mathbb{R}^{lN}$ as the Gaussian to integrate (restriction lifted in Sec. V-B) - meaning, we no longer deal with the trajectory-space. This boils down to integrating a standard multivariate Gaussian, but now under non-linear, non-convex, and perhaps even very sparse predicates. The noise sources propagate in different ways through the simulation, so we can no longer assume that two "close" points in the uncertainty input space $\mathbb{R}^{lN}$, will have the same sign robustness value; furthermore, we do not have a closed-form solution to the ellipse-predicate boundary intersections in the ESS step, used in [9] to find active segments of the ellipse. Note that approximating the trajectory-level distribution, as opposed to the noise distribution, as a Gaussian is expensive to obtain (many simulations) and also, as is discussed in Sec. IV-B, will poorly describe the true distribution.

### A. Surrogate functions for sampling satisfying simulations

We use the same two-step iterative procedure as described in Sec. II-B: sampling in-domain with ESS and converging to the desired constrained domains with a series of nestings using HDR.

Since we do not have a closed form solution for finding the active segments of the ellipse, we use Bayesian Optimization (BO) [22] based on Gaussian processes (GP) to estimate the robustness function of $\rho\left(r\left(\boldsymbol{w}\left(\theta\right)\right), \varphi, 0\right) = \varrho^\varphi\left(\theta\right)$ (with a slight abuse of notation) in each ESS iteration, where $\theta \in [0, 2\pi]$ parameterize the auxiliary ellipse and $\boldsymbol{w}\left(\theta\right) \in \mathbb{R}^{lN}$ are all the uncertainties in that specific simulation run. By construction, $\boldsymbol{w}_0$ the previously obtained sample from the domain (called $x_0$ in [9]), is at $\theta = 0, 2\pi$ (circular domain, same point). Once we have sampled enough points (simulations with different uncertainties) such that we are satisfied with the confidence bounds that the GP is providing (Fig. 3 in cyan), we can estimate the active segments by estimating a surrogate function and uniformly sample the new point from the active segments. If we fail, e.g. when sampling close to the boundary of the active segment but the sample is not actually in the active segment, we re-sample. In Fig. 3 we see an example of the ESS step using GP for

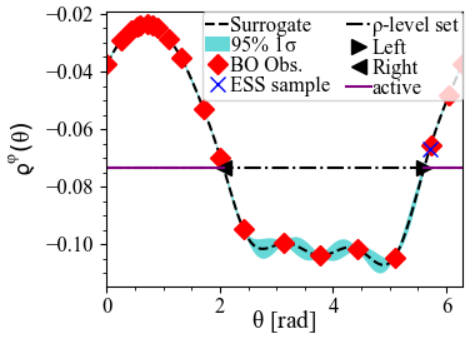



Fig. 3: Example of the robustness values along the ellipse for linear predicates. In red, points sampled in the GP. In black triangles, the boundaries of the true active segments computed with the hyperplane-ellipse closed-form solution. The blue X is the sample from the active segment. Dashed line is the estimated true curve and the dash-dotted line is the shift value (for the HDR phase). Solid purple is the active domain as found with [9]. We can see that the estimated active domain is almost identical to the true one.

a linear system with linear predicates (Sec. V-A), where we can also compare to closed-form solutions as in [9]. Once we successfully sample a new point, within the domain, we continue as in [9] - using HDR and ESS iteratively, sampling points until we reach the domain of interest $\mathcal{L}$ and are able to estimate the probability using the conditional probability of the ladder of nestings.

To accelerate the sampling process, we may use a greedy approach. Instead of sampling $N_{\text{BO}}$ points to estimate a surrogate function, determine the active segments using the intersection with the nesting threshold and sample a $\theta$ uniformly from the active domains, we immediately cease the sampling as soon as we get a point that is active - above the HDR threshold. The process is to sample $N_{\text{BO}}-2$ points and then add the points for $\varrho^{\varphi}(\theta=0)=\varrho^{\varphi}(\theta=2\pi)$ to enforce periodicity in the surrogate function. If after $N_{\text{BO}}$ points, we have no active points other than $\theta=0,2\pi$, we transition to exploitation. When dealing with low probabilities, as we deal with the more advanced nestings in the HDR, the active segments are small and we may have to focus on finding the maxima with GP, rather than trying to estimate the function distribution. This can be tweaked with the GP parameters [22] to prefer exploitation over exploration. We know that $\varrho^{\varphi}(\theta=0,2\pi)$ is the maximum value of our current gathered GP data set. It may not be the global maximum, but they are the only ones over the HDR nesting threshold otherwise we would exit as soon as a point sampled is over the threshold. Switching to exploitation may help finding the active segment near $\theta=0,2\pi$.

### B. Non-Gaussian uncertainties

Noise sources originating in complex sensors often cannot be described by univariate and uni-modal standard distributions such as a Gaussian or a Uniform distribution. For example, the *beam sensor model* [23] is a complex distribution that models typical noise of a range sensor, such as a Lidar. The measurement is affected by random errors, reflections and multi-paths from static and dynamic obstacles, and cross-talk. To accurately find the probability $p_s$ of a system that uses such sensors, we must use a good approximation of the true noise distribution. A Gaussian representation is typically not sufficiently accurate, especially when the true distribution is multimodal or heavy tailed.

Normalizing flows [24]–[26] can construct any probability distribution $\eta$ through a series of volume-preserving transformations on a, usually simpler and parametric, base distribution. This is often done using deep Neural Networks. Specifically in our context, we use these techniques to find a bijective (invertible) function $q(\cdot)$ to transform a variable with a standard multivariate Normal distribution $X\sim\mathcal{N}(0,I)$, to produce a new transformed variable $W=q(X)$ where $W$ follows our noise distribution, $\eta$ and $q:\mathbb{R}^{lN}\to\mathbb{R}^{lN}$. To sample $w\sim W$, we first sample $x\sim X$ and then calculate $w=q(x)$. We use this to transfer any sample $x\sim\mathcal{N}(0,I)$ sampled in the STL-based ESS framework, to a noise input from $w\sim\eta$ needed for the simulation. A challenge of this approach is that the non-Gaussian distribution is warped [25]:

$$\log\left(p_W(w)\right)=\log\left(p_X\left(q^{-1}(w)\right)\right)-\log\left(\left|\frac{dw}{dx}\right|\right) \tag{4}$$

This means that the density at point $w$ is equal to the density at point $x$ plus a term that corrects the density due to the transformation. The corollary is that a point that may be very likely in $X$, may not be as likely in $W$, however, it will be sampled "unintentionally" more frequently. For example, consider $p_x\sim\mathcal{N}(0,1)$ and $p_w\sim 0.5\mathcal{N}(-1,1)+0.5\mathcal{N}(+1,1)$, at $x=0$.

We use the correcting term in Eq. (4) as a weight for the specific sample at nesting $k$, $\alpha_r^k$. When computing the conditional probability for the nesting, we use:

$$p(\mathcal{L}_k|\mathcal{L}_{k-1})=\frac{\sum_{r=1}^{N_{ESS}}\alpha_r^{k-1}I(x_r\in\mathcal{L}_k)}{\sum_{r=1}^{N_{ESS}}\alpha_r^{k-1}I(x_r\in\mathcal{L}_{k-1})} \tag{5}$$

Normalizing flows can be composed in sequence to better learn complex distributions. In this work we use `Pyro` [27] to learn $q(\cdot)$. Given a noise distribution, we learn $q(\cdot)$ once and then reuse it for any specification or system.

### C. Performance

This technique needs to obtain samples that satisfy the specification. Since we consider simulations to be computationally expensive, this technique is best suited for rare-event situations. In a nominal case (without considering the greedy approach or added BO simulations while not finding samples in the domain), the number of simulations would be $N_{\text{sim}}=N_{\text{HDR}}\cdot N_{\text{ESS}}\cdot N_{\text{BO}}\cdot N_{\text{skip}}$, where $N_{\text{HDR}}$ is the number of nestings which is roughly $\log_2 p_s^{-1}$; $N_{\text{ESS}}$ is the user-defined number of samples per nesting (affects the accuracy and confidence bounds); $N_{\text{BO}}$ number of samples needed for the estimation of the surrogate function; and $N_{\text{skip}}$ is the burn-in parameter to decouple dependency between consecutive MCMC samples [9]. Recall that according to the algorithm,

the number $N_{\text{BO}}$ may vary (lower and occasionally even higher).

**Example 1.** *Considering a failing scenario that occurs once in a million simulations, $p_f = 0.0001\%$, we will need at least 1M calls to the simulator to get 1 failure with confidence bound $\sigma = \sqrt{p(1-p)/n} = 10^{-6}$. Using STL-based ESS we need approximately $N_{HDR} = \lceil \log_2 1M \rceil = 20$ nestings; for $N_{ESS} = 32$, $N_{skip} = 4$, and $N_{BO} = 20$ to get total of $N_{sim} = 51,200$ runs (and in practice less) and $\sigma = 8.8 \cdot 10^{-7}$ [9].*

## V. Case Studies

In this section we provide several case studies; A) validates our approach with respect to [9], B) shows a synthetic example with a non-Gaussian noise distribution, C) a system that exhibits hybrid behavior (non-linear), and D) an example with a "black box" RL-based controller. For A, C and D, we chose scenarios where failure is not a true rare event, so that we can compare the results of our method with Monte-Carlo sampling, showing its validity. We comment on the unique advantages of our approach below.

### A. 2-D Holonomic Robot

We re-examine the holonomic robot navigating in a 2D world from [9]; we compare the results of the proposed approach with those of [9] for a system that is linear, with Gaussian uncertainties, to show that they provide similar results. Due to the stochastic nature of the sampling, we do not expect to get the exact same probability for each run of either approach. The system state is $\xi = [x, \dot{x}, y, \dot{y}]'$ and:

$$\begin{aligned} \xi_{t+1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \xi_t + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} \Delta t^2/2m \\ \Delta t/m \end{bmatrix} u_t \\ u_t &= r_t - K_{fb}(\xi_t + w_t) \end{aligned} \tag{6}$$

Where $\otimes$ is the Kronecker product, $m$ is the robot's mass, and the controller $K_{fb}$ is obtained using the discrete Linear Quadratic Regulator [28]. The system and sampled trajectories are shown in Fig. 4 [left]. The task is for the robot to reach the goal in green while avoiding the red obstacles:

$$\varphi_{hol} := \square_{[0,50]} \neg \,\text{Red Obstacles} \wedge \lozenge_{[49,50]} \,\text{Green Goal} \tag{7}$$

The uncertainty inputs are state measurement noises for the entire trajectory $w_t \sim \mathcal{N}(0_4, \text{diag}\left(\left[0.06^2, 0.04^2, 0.06^2, 0.04^2\right]\right))$, $\forall t \in [0, N-1]$.

**Results**: Fig. 3 shows the intermediate result of the GP process where 20 samples (red) are used to construct the surrogate function (dash). In cyan, the estimated uncertainty in the estimation, $1\sigma$ with $95\%$. The black triangles represent the analytical points of intersection obtained with the closed-form hyperplane-ellipse intersection [19]. Fig. 3 shows that our approach can find the boundaries of the active segments in the case of linear systems, and sample the new MCMC point (blue X). Note that while having to sample up to $N_{BO} = 20$ points (in practice, we would find a sample in the active domain sooner), an analytical form solution, as in [9], has to find intersections with thousands-dimensional matrices, so the decrease in performance (computation time) is not significant. The probability to fail this specification calculated with linear STL-based ESS [9] is $p_f = 0.13 \pm 0.05\%$ and with the non-linear approach (this paper) is $p_f = 0.075 \pm 0.032\%$ (using 8056 simulations, $N_{ESS} = 64, N_{HDR} = 10, N_{skip} = 4$). Monte-Carlo simulation yields $p_f = 0.14 \pm 0.05\%$ with 5000 simulations.

### B. Product of Uniform Numbers

In this toy example, we show one of the strengths of our approach, which is to sample from the posterior distribution while Monte Carlo samples from the prior distribution. Let $y = \Pi_{i=1}^{M} w_i$ where $w_i \sim U(0,2)$ (Uniform distribution). The expected value $\mathbb{E}[y] = 1$ is independent of $M$. However, the variance grows larger as $M$ increases. Since $\text{range}(y) = [0, 2^M]$, this mean that $y$'s distribution has a large concentration at $y \gg 1$. When performing Monte-Carlo with large $M$ ($M > 50$), we often will get $\mathbb{E}[y] \ll 1$. Intuitively, as $M$ grows, the chance of sampling $w_i$ that is close to 0 increases which would decrease $y$. We estimate an STL property:

$$\varphi_{PoU} := (y > \alpha), \tag{8}$$

where the scalar $\alpha \in [0, 2^M]$. We can analytically compute $p_s$ by introducing a new auxiliary variable $z_i = -\log\left(\frac{w_i}{2}\right) \sim \exp(1)$. Then we express $-\log(w_i) = -\log(2) + z_i$ so that $-\log(y) = -\log(\prod_1^M w_i) = -\log(x_1) \ldots - \log(x_M) = -M\log(2) + \sum_1^M z_i = -M\log(2) + g_n$. We have $g_n \sim \Gamma(M, 1)$ (Gamma distribution), which means that $p_s = p(-\log(y) < -\log(\alpha)) = p(g_n < -M\log(2) - \log(\alpha))$. It is possible to compute this from the cumulative distribution function of the Gamma distribution and the incomplete gamma function [29].

Following Sec. IV-B, after initially training a neural network to warp samples from $\mathcal{N}(0,1)$ to $U(0,2)$, we use it to obtain $w_i$ and compute $y$ and $\varrho^{\varphi_{PoU}}$. The probability computed with our proposed method for $\alpha = 100, M = 100$ (i.e. the probability that the product of 100 random variables is larger than 100) is $p_s = 4.9\text{e}{-6} \pm 4.3\text{e}{-6}$ using 9540 sampled simulations ($N_{\text{HDR}} = 20, N_{\text{ESS}} = 32, N_{\text{skip}} = 4$). The analytically computed probability is $p_s = 2.86\text{e}{-5}$. Simple Monte-Carlo with $10^5$ simulations found no samples that satisfy Eq. (8), i.e. $p_s = 0$ and $\mathbb{E}_{\text{MC}}[y] = 0.0025 \neq 1$. Using Monte-Carlo and sampling from the prior, misrepresented the distribution of $y$. With our method we can estimate $y$ with fewer samples because we sample from the posterior.

**Remark**: $p_s \equiv 0$ is possible in non-linear systems and bounded noise distributions (consider the setting $\alpha > 2^M$), therefore, practically, after predefined $K$ iterations we would typically stop the algorithm.

### C. Capsule Tosser

In this demonstration, we consider a non-linear system – a mechanism for throwing capsules into bins – in the `Mujoco` simulator [30] as seen in Fig. 4 [right]. The system is hybrid (has discrete modes) due to the two bins and the possibility of the capsule falling in either bin or between them. The specification is:

$$\varphi_{CT} := \neg\lozenge_{[N-10,N]} (\text{Bin}_1 \vee \text{Bin}_2) \tag{9}$$

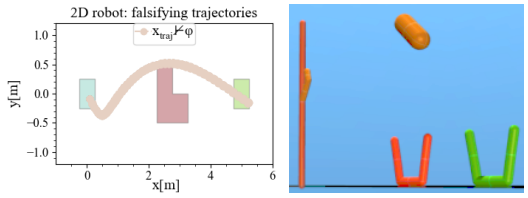



Fig. 4: Simulations environments. [left] 2-D robot navigation example from [9]. [right] Capsule Tosser simulation in Mujoco [30].

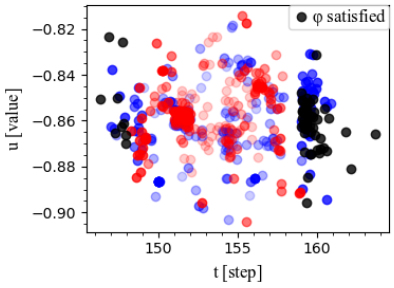



Fig. 5: Capsule Tosser: Black dots represent noise that caused the tosser to fail. Blue and red illustrate the propagation of the sampling, where for sampling from $\mathcal{L}_{k-1}$, blue are in $\mathcal{L}_k$ and red are outside it. The more opaque the color, the closer the simulation is to satisfying $\varphi_{CT}$. The x-axis is the time when the flipping occurs and the y-axis is the control input.

Where $N = T/\Delta t = 1.5\text{s}/0.002\text{s} = 750$ is the simulation horizon. The specification is satisfied if the capsule is not in either bin within the last 10 time steps of the run, capturing failure of the system. In this scenario, the only uncertainties $\boldsymbol{w} \in \mathbb{R}^2$ are the time (step) at which the mechanism starts to operate $t \sim \mathcal{N}\left(154, 2^2\right)$ and the control value ("force"), $u \sim \mathcal{N}\left(-0.86, 0.015^2\right)$.

**Results**: we can visualize the samples on a plot as $\boldsymbol{w} \in \mathbb{R}^2$, Fig. 5. Black points are samples that satisfy the specification, i.e. simulations where the capsule misses the bins. Blue points are points sampled throughout the nestings, from $\mathcal{L}_{k-1}$ that were also within $\mathcal{L}_k$. Red points also sampled from $\mathcal{L}_{k-1}$, but were not in $\mathcal{L}_k$. The more opaque the color, the closer the simulation to satisfying the specification. We can see how the algorithm is advancing the sampling distribution towards the two failure modes (loosely captured as the left and right clusters, having $t < 150$ or $t > 159$). Here, a blue sample, even the more translucent, *may* also be satisfying the specification, i.e. may represent a failed toss. Red samples represent successful tosses. The probability of satisfying $\varphi_{CT}$ (i.e. failed toss) is $p_s = 1.12 \pm 0.38\%$ when sampling 6084 simulations. Monte-Carlo sampling yields $p_s = 0.93 \pm 0.15\%$ using 4096 simulations, showing that our approach is sound for this non-rare-event.

### D. Robotic Manipulator with Reinforcement Learning

Using reinforcement learning [31], we train a neural network controller (D4PG [32]) operating at 20Hz for a Panda arm [33], a 7-DoF robot manipulator, to grasp and lift a cube

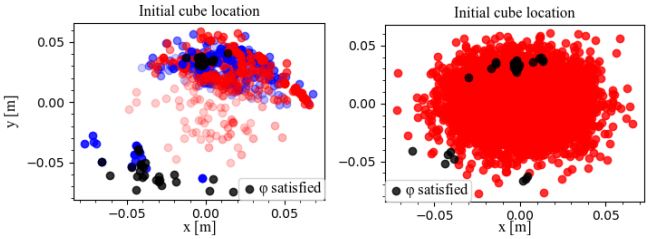



Fig. 6: Panda lift task - uncertainty over $(x, y)$, the cube's initial location. Black dots are initial conditions where the cube failed to reach the height. [left] **Our approach**: For sampling from $\mathcal{L}_{k-1}$, blue points are also in $\mathcal{L}_k$ and red are outside it. [right] **Monte-Carlo**: red are successful runs where the robot lifted the cube to $0.5m$.

off a table. Here the uncertainty is in the position of the cube on the table, $\mathbf{x} \sim \mathcal{N}\left([0,0]', \text{diag}([0.02^2, 0.02^2])\right)$. The specification in Eq. (10) describes the failure mode where the cube is not lifted to a height of at least 0.5m, within 9-10 sec of the execution, see Fig. 1.

$$\varphi_{RL} := \neg\Diamond_{[180,200]}(z_{\text{cube}} - z_{\text{table}} > 0.50[m]) \tag{10}$$

The simulation environment is `robosuite` [34], based on the `Mujoco` [30] physics engine.

**Results**: Our approach yields $p_s = 0.12 \pm 0.05\%$ using 9309 runs ($N_{\text{HDR}} = 9$), while Monte-Carlo finds $p_s = 0.54 \pm 0.1\%$ using 5000 runs. Fig. 6 shows how our sampling moves toward the failing simulation while the Monte-Carlo simulation samples uniformly. The lower group of black dots in Fig. 6 [left] are runs where the failure mode is the cube dropping. The black dots on the top are runs where the arm fails to lift the cube over the threshold, as can be seen in the accompanying video.

## VI. Conclusions and Discussion

We relax the linearity and Gaussian assumption of [9] to enable STL-based ESS to verify non-linear systems and non-Gaussian uncertainties w.r.t complex STL specifications. One of the strengths of our methods is being agnostic to the complexity of the specification. This approach is particularly suited for finding the probability of failure due to rare events. Furthermore, our approach inherently can be used to sample runs that exhibit failures; by sampling from different clusters of failures, as can be seen in Fig. 6 [left], we can provide insight to a designer as to what the failure modes are.

With non-linear systems we cannot guarantee to sample from the active segments in the ESS as it depends on the quality of the GP estimation that depends on the number of samples sampled. This means that some samples may need to be re-sampled. Ill-behaved simulations, where Assumption 1 is not applicable are more prone to suffer from this issue, as the GP is not able to estimate a suitable surrogate function.

In future work, we plan to give formal guarantees and bounds on performance for this algorithm. We will also use this framework for synthesizing new controllers such that the probability of satisfying the task is maximized.

## REFERENCES


[1] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[2] R. Schram, A. Williams, and M. van Ratingen, "Implementation of autonomous emergency braking (aeb), the next step in euro ncap's safety assessment," *ESV, Seoul*, 2013.

[3] M. Cummings, "Adaptation of human licensing examinations to the certification of autonomous systems," in *Safe, autonomous and intelligent vehicles*. Springer, 2019, pp. 145–162.

[4] F. Favarò, S. Eurich, and N. Nader, "Autonomous vehicles' disengagements: Trends, triggers, and regulatory limitations," *Accident Analysis & Prevention*, vol. 110, pp. 136–148, 2018.

[5] H. X. Liu and S. Feng, ""curse of rarity" for autonomous vehicles," *arXiv preprint arXiv:2207.02749*, 2022.

[6] G. Agha and K. Palmskog, "A survey of statistical model checking," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 28, no. 1, pp. 1–39, 2018.

[7] D. Parker, "Verification of probabilistic real-time systems," *Proc. 2013 Real-time Systems Summer School (ETR'13)*, 2013.

[8] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[9] G. Scher, S. Sadraddini, R. Tedrake, and H. Kress-Gazit, "Elliptical slice sampling for probabilistic verification of stochastic systems with signal temporal logic specifications," in *25th ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3501710.3519506

[10] A. Sinha, M. O'Kelly, R. Tedrake, and J. C. Duchi, "Neural bridge sampling for evaluating safety-critical autonomous systems," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6402–6416, 2020.

[11] M. O' Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018.

[12] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 432–442.

[13] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, "Utilizing s-taliro as an automatic test generation framework for autonomous vehicles," in *2016 ieee 19th international conference on intelligent transportation systems (itsc)*. IEEE, 2016, pp. 1470–1475.

[14] L. Yang and N. Ozay, "Synthesis-guided adversarial scenario generation for gray-box feedback control systems with sensing imperfections," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, Sept. 2021. [Online]. Available: https://doi.org/10.1145/3477033

[15] G. Chou, Y. E. Sahin, L. Yang, K. J. Rutledge, P. Nilsson, and N. Ozay, "Using control synthesis to generate corner cases: A case study on autonomous driving," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2906–2917, 2018.

[16] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems*, K. Chatterjee and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–106.

[17] I. Murray, R. Adams, and D. MacKay, "Elliptical slice sampling," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 541–548.

[18] J. Bernardo, J. Berger, A. Dawid, A. Smith, *et al.*, "Regression and classification using gaussian process priors," *Bayesian statistics*, vol. 6, p. 475, 1998.

[19] A. Gessner, O. Kanjilal, and P. Hennig, "Integrals over gaussians under linear domain constraints," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 2764–2774.

[20] P. Diaconis and S. Holmes, "Three examples of monte-carlo markov chains: At the interface between statistical computing, computer science, and statistical mechanics," in *Discrete Probability and Algorithms*, D. Aldous, P. Diaconis, J. Spencer, and J. M. Steele, Eds. New York, NY: Springer New York, 1995, pp. 43–56.

[21] O. Maler, D. Nickovic, and A. Pnueli, "Checking temporal properties of discrete, timed and continuous behaviors," *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, pp. 475–505, 2008.

[22] A. Agnihotri and N. Batra, "Exploring bayesian optimization," *Distill*, 2020, https://distill.pub/2020/bayesian-optimization.

[23] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[24] E. G. Tabak and C. V. Turner, "A family of nonparametric density estimation algorithms," *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 145–164, 2013.

[25] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 3964–3979, 2020.

[26] G. Papamakarios, E. T. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference." *J. Mach. Learn. Res.*, vol. 22, no. 57, pp. 1–64, 2021.

[27] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep Universal Probabilistic Programming," *Journal of Machine Learning Research*, 2018.

[28] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-interscience New York, 1972, vol. 1.

[29] W. Gautschi, "A computational procedure for incomplete gamma functions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 5, no. 4, pp. 466–481, 1979.

[30] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[31] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020. [Online]. Available: https://arxiv.org/abs/2006.00979

[32] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," *arXiv preprint arXiv:1804.08617*, 2018.

[33] Franka Emika, "Panda," 2022. [Online]. Available: https://www.franka.de

[34] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.