A New Simulation Metric to Determine Safe Environments and Controllers for Systems with Unknown Dynamics

Shromona Ghosh*
shromona.ghosh@berkeley.edu

Somil Bansal*
somil@berkeley.edu

Alberto Sangiovanni-Vincentelli
alberto@berkeley.edu

Sanjit A. Seshia
sseshia@berkeley.edu

Claire Tomlin
tomlin@berkeley.edu

ABSTRACT

We consider the problem of extracting safe environments and controllers for reach-avoid objectives for systems with known state and control spaces, but unknown dynamics. In a given environment, a common approach is to synthesize a controller from an abstraction or a model of the system (potentially learned from data). However, in many situations, the relationship between the dynamics of the model and the *actual system* is not known; and hence it is difficult to provide safety guarantees for the system. In such cases, the Standard Simulation Metric (SSM), defined as the worst-case norm distance between the model and the system output trajectories, can be used to modify a reach-avoid specification for the system into a more stringent specification for the abstraction. Nevertheless, the obtained distance, and hence the modified specification, can be quite conservative. This limits the set of environments for which a safe controller can be obtained. We propose SPEC, a specification-centric simulation metric, which overcomes these limitations by computing the distance using only the trajectories that violate the specification for the system. We show that modifying a reach-avoid specification with SPEC allows us to synthesize a safe controller for a larger set of environments compared to SSM. We also propose a probabilistic method to compute SPEC for a general class of systems. Case studies using simulators for quadrotors and autonomous cars illustrate the advantages of the proposed metric for determining safe environment sets and controllers.

KEYWORDS

Simulation metric, Safe environment assumptions, Safe controller synthesis, Model-mismatch, Reach-avoid objectives, Scenario optimization.

1 INTRODUCTION

Recent research in robotics and control theory has focused on developing complex autonomous systems, such as robotic manipulators, autonomous vehicles, and surgical robots. Since many of these systems are safety-critical, it is important to design provably-safe controllers while determining environments in which safety can be guaranteed. In this work, we focus on reach-avoid objectives, where the goal is to design a controller to reach a target set of states (referred to as reach set) while avoiding unsafe states (avoid set). Reach-avoid problems are common for autonomous vehicles in the real world; for example, a drone flying in an indoor setting. Here the reach set could be a desired goal position and the avoid set could be the set of the obstacles. In such a setting, it is important to determine the environments in which the drone can safely navigate, as well as the corresponding safe controllers.

Typically, a mathematical model of the system, such as a physics-based first principles model, is used for synthesizing a safe controller in different environments (e.g., [36, 37]). However, when the system dynamics are unknown, synthesizing such a controller becomes challenging. In such cases, it is a common practice to identify a model for the system. This model represents an abstraction of the system behavior. Recently, there has been an increased interest in using machine learning (ML) based tools, such as neural networks and Gaussian processes, for learning abstractions *directly* from the data collected on the system [5, 6, 28]. One of the many verification challenges for ML-based systems [34] is that such abstractions cannot be directly used for verification, since it is not clear *a priori* how representative the abstraction is of the actual system. Hence, to use the abstraction to provide guarantees for the system, we need to first quantify the differences between it and the system.

One approach is to use model identification techniques that provide bounds on the mismatch between the dynamics of the system and its abstraction both in time and frequency domains (see [17, 23, 29] and references therein). This bound is then used to design a provably stabilizing controller for the system. These approaches have largely been limited to linear abstractions and systems, and the focus has been on designing asymptotically stabilizing controllers.

*Both authors contributed equally to this research.

** All authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.

** This research is supported in part by NSF under the CPS Frontier project VeHICaL project (1545126), by NSF grants 1739816 and 1837132, by the UC-Philippine-California Advanced Research Institute under project IIID-2016-005, by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-17-2-0196, by the DARPA BRASS program under agreement number FA8750-16-C0043, the DARPA Assured Autonomy program under agreement number FA8750-18-C-0101, the iCyPhy center, and by Berkeley Deep Drive.

HSCC '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6282-5/19/04…$15.00
https://doi.org/10.1145/3302504.3311795

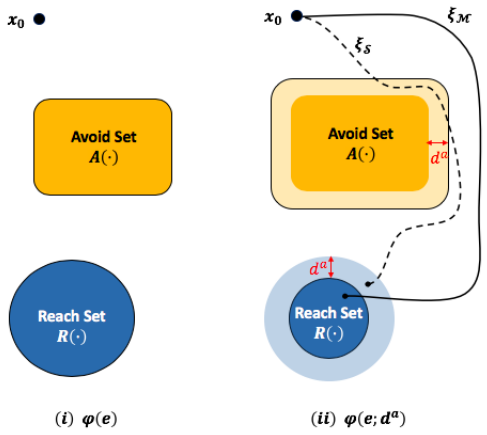


Figure 1. The avoid set is expanded and the reach set is contracted with the simulation metric d^a. If the abstraction trajectory (ξ_M) stays clear of the expanded avoid set and reaches the contracted reach set, the system trajectory (ξ_S) also stays clear of the original avoid set and reaches the original reach set.

Another way to quantify the difference between a general nonlinear system and its abstraction relies on the notion of a (approximate) *simulation metric* [3, 4, 19]. Such a metric measures the maximal distance between the system and the abstraction output trajectories over all finite horizon control sequences. Standard simulation metrics (referred to as SSM here on) have been used for a variety of purposes such as safety verification [20], abstraction design for discrete [27], nonlinear [33], switched [21] systems, piecewise deterministic and labelled Markov processes [14, 35], and stochastic hybrid systems [2, 7, 16, 24], model checking [4, 25], and model reduction [12, 32]. Once computed, the SSM is used to *expand* the unsafe set (or avoid set) in [2]. For reach-avoid scenarios, we additionally use it to *contract* the reach set as shown in Figure 1. If we can synthesize a safe controller that ensures the abstraction trajectory avoids the expanded avoid set and reaches the contracted reach set, then the system trajectory is guaranteed to avoid and reach the original avoid set and reach set respectively. This follows from the property that SSM captures the worst case distance between the trajectories of the system and the abstraction. Consequently, the set of safe environments for the system can be obtained by finding the set of environments for which we can design a safe controller for the abstraction with the modified specification.

Even though powerful in its approach, SSM computes the maximal distance between the system and the abstraction trajectories across all possible controllers. We show in this paper that this is unnecessary and might lead to a conservative bound on the quality of the abstraction for the purposes of controller synthesis. In particular, the larger the distance between the system and the abstraction, the larger the expansion (contraction) of the avoid (reach) set. In many cases, this results in unrealizability wherein there does not exist a safe controller for the abstraction for the modified specification.

In this paper, we propose SPEC, SPEcification-Centric simulation metric, that overcomes these limitations. SPEC achieves this by computing the distance across

(1) only those controllers that can be synthesized by a particular control scheme and that are safe for the abstraction (in the context of the original reach-avoid specification) — these are the only potential safe controllers for the system;
(2) only those abstraction and system trajectories for which the system violates the reach-avoid specification, and
(3) only between the abstraction trajectory and the reach and the avoid sets.

If the reach-avoid specification is changed using SPEC in a similar fashion as that for SSM, it is guaranteed that if a controller is safe for the abstract model, it remains safe for the system. SPEC can be significantly less conservative than SSM, and can be used to design safe controllers for the system for a broader range of reach-avoid specifications. In fact, we show that, among all uniform distance bounds (i.e., a single distance bound is used to modify the specification in all environments), SPEC provides the largest set of environments such that a safe controller for the abstraction is also safe for the system.

Note that a similar metric has been used earlier [18] to find tight environment assumptions for temporal logic specifications. However, it applies in much more restricted settings since it relies on having simple linear representations of the abstraction which can be expressed as a linear optimization problem.

In general, it is challenging to compute both SSM and SPEC when the dynamics of the system are not available. Several approaches have been proposed in the literature for computing SSM [1, 19, 24]; however, restrictive assumptions on the dynamics of the systems are often required to compute it. More recently, a randomized approach has been proposed to compute SSM [2, 16] for finite-horizon properties that relies on "scenario optimization", which was first introduced for solving robust convex programs via randomization [8] and then extended to semi-infinite chance-constrained optimization problems [10]. Scenario optimization is a sampling-based method to solve semi-infinite optimization problems, and has been used for system and control design [9, 11]. In this work, we propose a scenario optimization-based computational method for SPEC that has general applicability and is not restricted to a specific class of systems. Indeed, the only assumption is that the system is available as an oracle, with known state and control spaces, which we can simulate to determine the corresponding output trajectory. Given that the distance metric is obtained via randomization and, hence, is a random quantity, we provide probabilistic guarantees on the performance of SPEC. However, this confidence is a design parameter and can be chosen as close to 1 as desired (within a simulation budget). To summarize, this paper's main contributions are:

- SPEC, a new simulation metric that is less conservative than SSM, and provides the largest set of environments such that a safe controller for the abstraction is also safe for the system;
- a method to compute SPEC that is not restricted to a specific class of systems, and
- a demonstration of the proposed approach on numerical examples and simulations of real-world autonomous systems, such as a quadrotor and an autonomous car.

2 MATHEMATICAL PRELIMINARIES

Let $\mathcal{S}$ be an unknown, discrete-time, potentially non-linear, dynamical system with state space $\mathbb{R}^{n_x}$ and control space $\mathbb{R}^{n_u}$. Let $\mathcal{M}$ be an abstraction of $\mathcal{S}$ with the same state and control spaces as $\mathcal{S}$, whose dynamics are known. We also assume that the bounds between the dynamics of $\mathcal{M}$ and $\mathcal{S}$ are not available beforehand (i.e., we cannot *a priori* quantify how different the two are). $\xi_{\mathcal{S}}(t; x_0, \mathbf{u})$ denotes the trajectory of $\mathcal{S}$ at time t starting from the initial state x_0 and applying the controller $\mathbf{u}$. $\xi_{\mathcal{M}}$ is similarly defined. For ease of notation, we drop $\mathbf{u}$ and x_0 from the trajectory arguments wherever convenient.

We define by $\mathcal{E} := \mathcal{X}_0 \times \mathcal{A} \times \mathcal{R}$ the set of all reach-avoid scenarios (also referred to as *environment scenarios* here on), for which we want to synthesize a controller for $\mathcal{S}$. A reach-avoid scenario $e \in \mathcal{E}$ is a three-tuple, $(x_0, A(\cdot), R(\cdot))$, where $x_0 \in \mathcal{X}_0 \subset \mathbb{R}^{n_x}$ is the initial state of $\mathcal{S}$. $A(\cdot) \in \mathcal{A}, A(\cdot) \subset \mathbb{R}^{n_x}$ and $R(\cdot) \in \mathcal{R}, R(\cdot) \subset \mathbb{R}^{n_x}$ are (potentially time varying) sequences of avoid and reach sets respectively. We leave $\mathcal{A}$ and $\mathcal{R}$ abstract except where necessary. If the sets are not time varying, we can replace $R(\cdot)$ (respectively $A(\cdot)$) by the stationary R (respectively A). Similarly, if there is no avoid or reach set at a particular time, we can represent $A(t) = \emptyset$ and $R(t) = \mathbb{R}^{n_x}$.

For each $e \in \mathcal{E}$, we define a reach-avoid specification, $\varphi(e)$

$$\varphi(e) := \{\xi(\cdot) : \ \forall t \in \mathcal{T} \ \xi(t) \notin A(t) \wedge \xi(t) \in R(t)\}, \tag{1}$$

where $\mathcal{T}$ denote the time-horizon $\{0, 1, \ldots, H\}$. We say $\xi(\cdot)$ satisfies the specification $\varphi(e)$, denoted $\xi(\cdot) \models \varphi(e)$, if $\xi(\cdot) \in \varphi(e)$.

The reader might observe that our use of $R(\cdot)$ in (1) differs somewhat from the intuitive notion of a reach set (depicted, e.g., in Fig. 1). Specifically, in (1) defines the reach-avoid specification such that the output trajectory must remain within $R(t)$ at all times t, while the usual notion involves *eventually* reaching a desired set of states. Note, however, that for the purposes of defining $\varphi(e)$, these notions are equivalent if $R(t)$ in (1) represents the backwards reachable tube corresponding to the desired reach set: if a state is reachable eventually, then the trajectory stays within the backwards reachable tube at all time points. We henceforth use the $R(t)$ in the latter sense since it simplifies the mathematics in the paper.

Finally, we define $\mathcal{U}_\Pi(e) \subset \mathcal{U}$ to be the space of all permissible controllers for e, and $\mathcal{U}$ to be the space of all finite horizon control sequences over $\mathcal{T}$. For example, if we restrict ourselves to linear feedback controllers, $\mathcal{U}_\Pi$ represents the set of all linear feedback controllers that are defined over the time horizon $\mathcal{T}$.

3 PROBLEM FORMULATION

Given the set of reach-avoid scenarios $\mathcal{E}$, the controller scheme $\mathcal{U}_\Pi$, and the abstraction $\mathcal{M}$, our goal is two-fold:

(1) to find the environment scenarios for which it is possible to design a controller such that $\xi_{\mathcal{S}}(\cdot)$ satisfies the corresponding reach-avoid specification $\varphi(e)$,
(2) to find a corresponding safe controller for each scenario in (1).

Mathematically, we are interested in computing the set $\mathcal{E}_{\mathcal{S}}$

$$\mathcal{E}_{\mathcal{S}} = \{e \in \mathcal{E} \ : \exists \mathbf{u} \in \mathcal{U}_\Pi(e) \ , \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}, \tag{2}$$

and the corresponding set of safe controllers $\mathcal{U}_{\mathcal{S}}(e)$ for each $e \in \mathcal{E}_{\mathcal{S}}$

$$\mathcal{U}_{\mathcal{S}}(e) = \{\mathbf{u} \in \mathcal{U}_\Pi(e) \ : \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}. \tag{3}$$

When a dynamics model of $\mathcal{S}$ is known, several methods have been studied in literature to compute the sets $\mathcal{E}_{\mathcal{S}}$ and $\mathcal{U}_{\mathcal{S}}(e)$ for reach-avoid problems [31, 37, 38]. However, since a dynamics model of $\mathcal{S}$ is unknown, the computation of these sets is challenging in general. To overcome this problem, one generally relies on the abstraction $\mathcal{M}$. We make the following assumptions on $\mathcal{S}$ and $\mathcal{M}$:

Assumption 1. *$\mathcal{S}$ is available as an oracle that can be simulated, i.e., we can run an execution (or experiment) on $\mathcal{S}$ and obtain the corresponding system trajectory $\xi_{\mathcal{S}}(\cdot)$.*

Assumption 2. *For any $e \in \mathcal{E}$, we can determine if there exist a controller such that $\xi_{\mathcal{M}} \models \varphi(e)$ and can compute such a controller.*

Assumption 1 states that even though we do not know the dynamics of $\mathcal{S}$, we can run an execution of $\mathcal{S}$. Assumption 2 states that it is possible to verify whether $\mathcal{M}$ satisfies a given specification $\varphi(e)$ or not. Although it is not a straightforward problem, since the dynamics of $\mathcal{M}$ are known, several existing methods can be used for obtaining a safe controller for $\mathcal{M}$.

Under these assumptions, we show that we can convert a verification problem on $\mathcal{S}$ to a verification problem on $\mathcal{M}$. In particular, we compute a distance bound, SPEC, between $\mathcal{S}$ and $\mathcal{M}$ which along with $\mathcal{M}$ allows us to compute a conservative approximation of $\mathcal{E}_{\mathcal{S}}$ and $\mathcal{U}_{\mathcal{S}}(e)$.

4 RUNNING EXAMPLE

We now introduce a very simple example that we will use to illustrate our approach, a 2 state linear system in which the system and the abstraction differ only in one parameter. Although simple, this example illustrates several facets of SPEC. We present more realistic case studies in Section 8.

Consider a system $\mathcal{S}$ whose dynamics are given as

$$x(t+1) = \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(t). \tag{4}$$

We are interested in designing a controller for $\mathcal{S}$ to regulate it from the initial state $x(0) := x_0 = [0, 0]$ to a desired state $x^* = \begin{bmatrix} x_1^*, 0 \end{bmatrix}$ over a time-horizon of 20 steps, i.e, $H = 20$. In particular, we have

$$\mathcal{X}_0 = \{[0, 0]\}, \quad \mathcal{A} = \emptyset, \quad \mathcal{R} = \bigcup_{-4 \le x_1^* \le 4} R(\cdot; x^*),$$

where

$$R(t; x^*) = \mathbb{R}^2, t \in \{0, 1, \ldots, H-1\},$$
$$R(H; x^*) = \{x : \|x - x^*\|_2 < \gamma\}.$$

We use $\gamma = 0.5$ in our simulations. Thus, each $e \in \mathcal{E}$ consists of a final state x^* (equivalently, a reach set $R(H; x^*)$) to which we want the system to regulate, starting from the origin. Consequently, the system trajectory satisfies the reach-avoid specification in this case if $\xi_{\mathcal{S}}(H; x_0, \mathbf{u}) \in R(H; x^*)$.

For the purpose of this example, we assume that the system dynamics in (4) are unknown; only the dynamics of its abstraction $\mathcal{M}$ are known and given as

$$x(t+1) = \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0.1 \end{bmatrix} u(t). \tag{5}$$

In this example, we use the class of linear feedback controllers as $\mathcal{U}_\Pi(e)$, although other control schemes can very well be used. In particular, for any given environmental scenario e, the space of controllers $\mathcal{U}_\Pi(e)$ is given by

$$\mathcal{U}_\Pi(e) = \{LQR(q, x^*) : 0.1 \leq q \leq 100\},$$

where $LQR(q, x^*)$ is a Linear Quadratic Regulator (LQR) designed for the abstraction dynamics in (5) to regulate the abstraction trajectory to x^* [1], with the state penalty matrix $Q = qI$ and the control penalty coefficient $R = 1$. Here, $I \in \mathbb{R}^{2\times 2}$ is an identity matrix. Thus, for different values of q we get different controllers, which affect the various characteristics of the resultant trajectory, such as overshoot, undershoot, and final state. $LQR(q)$ for any given q can be obtained by solving the discrete-time Riccati equation [26]. Our goal thus is to use the dynamics in (5) to find the set of final states to which $\mathcal{S}$ can be regulated and the corresponding regulator in $\mathcal{U}_\Pi(e)$.

5 SOLUTION APPROACH

5.1 Computing approximate safe sets using $\mathcal{M}$ and simulation metric

Computing sets $\mathcal{E}_\mathcal{S}$ and $\mathcal{U}_\mathcal{S}$ exactly can be challenging since the dynamics of $\mathcal{S}$ are unknown *a priori*. Generally, we use the abstraction $\mathcal{M}$ as a replacement for $\mathcal{S}$ to synthesize and analyze safe controllers for $\mathcal{S}$. However, to provide guarantees on $\mathcal{S}$ using $\mathcal{M}$, we would need to quantify how different the two are.

We quantify this difference through a distance bound, d, between $\mathcal{S}$ and $\mathcal{M}$. d is used to modify the specification $\varphi(e)$ to a more stringent specification $\varphi(e; d)$ such that if $\xi_\mathcal{M}(\cdot) \models \varphi(e; d)$ then $\xi_\mathcal{S}(\cdot) \models \varphi(e)$. Thus, the set of safe controllers for $\mathcal{M}$ for $\varphi(e; d)$ can be used as an approximation for $\mathcal{U}_\mathcal{S}(e)$. In particular, if we define the sets $\mathcal{U}_{\varphi(e;d)}$ and $\mathcal{E}_\varphi(d)$ as

$$\begin{aligned} \mathcal{U}_{\varphi(e;d)} &:= \{\mathbf{u} \in \mathcal{U}_\Pi(e) : \xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d)\} \\ \mathcal{E}_\varphi(d) &:= \{e \in \mathcal{E} : \mathcal{U}_{\varphi(e,d)} \neq \emptyset\}, \end{aligned} \tag{6}$$

then $\mathcal{U}_{\varphi(e;d)}$ and $\mathcal{E}_\varphi(d)$ can be used as an approximation of $\mathcal{U}_\mathcal{S}(e)$ and $\mathcal{E}_\mathcal{S}$ respectively. Consequently, a verification problem on $\mathcal{S}$ can be converted into a verification problem on $\mathcal{M}$ using the modified specification.

One such distance bound d is given by the simulation metric, SSM, between $\mathcal{M}$ and $\mathcal{S}$ defined as

$$d^a = \max_{e\in\mathcal{E}} \max_{\mathbf{u}\in\mathcal{U}_\Pi(e)} \|\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) - \xi_\mathcal{M}(\cdot; x_0, \mathbf{u})\|_\infty \tag{7}$$

Here, the ∞-norm is the maximum distance between the trajectories across all timesteps. Typically SSM is computed over the space of all finite horizon controls $\mathcal{U}$ instead of $\mathcal{U}_\Pi(e)$ [20]. Since we are interested in a given control scheme, we restrict this computation to $\mathcal{U}_\Pi(e)$. In general, d^a is difficult to compute, because it requires searching over (the potentially infinite) space of controllers and environments. An approximate technique to compute d^a was presented for systems whose dynamics were unknown with probabilistic guarantees in [2].

However, if d^a can be computed then it can be used to modify a specification $\varphi(e)$ to $\varphi(e; d^a)$ as follows: "expand" the avoid set $A(\cdot)$ to get the augmented avoid set $A(\cdot; d^a) = A(\cdot) \oplus d^a$, and "contract" the reach set $R(\cdot)$ to obtain a conservative reach set $R(\cdot; d^a) = R(\cdot) \ominus d^a$ (see Figure 1). Here, $\oplus$ ($\ominus$) is the Minkowski sum(difference)[2]. Consequently, $\varphi(e; d^a)$ is the set of trajectories which avoid $A(\cdot; d^a)$ and are always contained in $R(\cdot; d^a)$,

$$\varphi(e; d^a) := \{\xi(\cdot) : \xi(t) \notin A(t; d^a), \xi(t) \in R(t; d^a) \forall t \in \mathcal{T}\}. \tag{8}$$

Then it can be shown that any controller that satisfies the specification $\varphi(e; d^a)$ for $\mathcal{M}$ also ensures that $\mathcal{S}$ satisfies the specification $\varphi(e)$.

Proposition 1. *For any $e \in \mathcal{E}$ and controller $\mathbf{u} \in \mathcal{U}_\Pi(e)$, we have $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^a)$ implies $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$.*

The proof of Proposition 1 can be found in the Appendix. Proposition 1 implies that $\mathcal{E}_\varphi(d^a)$ and $\mathcal{U}_{\varphi(e;d^a)}$ can be used as approximations of $\mathcal{E}_\mathcal{S}$ and $\mathcal{U}_\mathcal{S}(e)$ respectively. However, the distance bound in (7) does not take into account the reach-avoid specification (environment) for which a controller needs to be synthesized. Thus, d^a can be quite conservative. As a result, the modified specification can be so stringent that the set of environments $\mathcal{E}_\varphi(d^a)$ for which we can synthesize a provably safe controller for the abstraction (and hence for the system) itself will be very small, resulting in a very conservative approximation of $\mathcal{E}_\mathcal{S}$.

5.2 Specification-Centric Simulation Metric (SPEC)

To overcome these limitations, we propose SPEC,

$$d^b = \max_{e\in\mathcal{E}} \max_{u\in\mathcal{U}_{\varphi(e)}} d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)), \tag{9}$$

where

$$\begin{aligned} d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)) = (\min_{t\in\mathcal{T}}(\min\{h\left(\xi_\mathcal{M}(t; x_0, \mathbf{u}), A(t)\right), \\ - h\left(\xi_\mathcal{M}(t; x_0, \mathbf{u}), R(t)\right)\})\mathbb{1}_{(\xi_\mathcal{S}(\cdot)\not\models\varphi(e))}) \end{aligned} \tag{10}$$

Here $\mathcal{U}_{\varphi(e)} := \{\mathbf{u} \in \mathcal{U}_\Pi : \xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}$ is the set of all controls such that $\mathcal{M}$ satisfies the specification $\varphi(e)$. $\mathbb{1}_l$ represents the indicator function which is 1 if l is true and 0 otherwise, and $h(x, K)$ is the signed distance function defined as

$$h(x, K) := \begin{cases} \inf_{k\in K} \|x - k\|, & \text{if } x \notin K \\ -\inf_{k\in K^C} \|x - k\|, & \text{otherwise.} \end{cases}$$

If for any $e \in \mathcal{E}$, $\mathcal{U}_{\varphi(e)}$ is empty, we define the distance function $d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot))$ to be zero. Similarly, if there is no $A(\cdot)$ or $R(\cdot)$ at a particular t, the corresponding signed distance function is defined to be ∞. There are four major differences between (7) and (9):

(1) To compute the d^b we only consider the feasible set of controllers that can be synthesized by the control policy, $\mathcal{U}_{\varphi(e)} \subseteq \mathcal{U}_\Pi(e)$, as all other controllers do not help us in synthesizing a safe controller for $\mathcal{S}$ (as they are not even safe for $\mathcal{M}$).
(2) To compute the distance between $\mathcal{S}$ and $\mathcal{M}$, we only consider those trajectories where $\mathcal{S}$ violates the specification. This is because a non-zero distance between the trajectories of $\mathcal{S}$

[1] That is, we penalize the trajectory deviation to the desired state x^* in the LQR cost function.

[2] The Minkowski sum of a set K and a scalar d is the set of all points that are the sum of any point in K and $B(d)$, where $B(d)$ is a disc of radius d around the origin.

and $\mathcal{M}$, where the $\xi_{\mathcal{S}} \models \varphi(e)$ does not give us any additional information in synthesizing a safe controller.

(3) Within a falsifying $\xi_{\mathcal{S}}$, we compute the minimum distance of the abstraction trajectory from the avoid and reach sets rather than the system trajectory, as that is sufficient to obtain a margin to discard behaviors that are safe for the abstraction but unsafe for the system.
(4) Finally, a minimum over time of this distance is sufficient to discard an unsafe trajectory, as the trajectory will be unsafe if it is unsafe at any t.

These considerations ensure that d^b is far less conservative compared to d^a and allows us to synthesize a safe controller for the system for a wider set of environments. We first prove that d^b can be used to compute an approximation of $\mathcal{E}_{\mathcal{S}}$.

Proposition 2. *If $\mathcal{U}_{\varphi(e;d^b)} \subseteq \mathcal{U}_{\varphi(e)}$, then $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$ implies $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$ $\forall e \in \mathcal{E}, \mathbf{u} \in \mathcal{U}_{\Pi}(e)$.*

The proof of Proposition 2 can be found in the Appendix. Thus, if we define $\mathcal{U}_{\varphi(e;d^b)}$ and $\mathcal{E}_{\varphi}(d^b)$ as in (6) then they can be used as approximations of $\mathcal{U}_{\mathcal{S}}(e)$ and $\mathcal{E}_{\mathcal{S}}$ respectively. Proposition 2 requires that the set of controllers that satisfy the modified specification, $\mathcal{U}_{\varphi(e;d^b)}$, is a subset of the set of the controllers that satisfy the actual specification, $\mathcal{U}_{\varphi(e)}$. When $\mathcal{U}_{\Pi}(e) = \mathcal{U}$, this condition is trivially satisfied as the modified specification is more stringent than the actual specification. Other control schemes, such as the set of linear feedback controllers and feasibility-based optimization schemes also satisfy this condition. In fact, in such cases, the proposed metric, d^b, quantifies the tightest (largest) approximation of $\mathcal{E}_{\mathcal{S}}$, i.e., $\nexists d < d^b$, such that $\mathcal{E}_{\varphi}(d) \subseteq \mathcal{E}_{\mathcal{S}}$.

Theorem 1. *Let $\mathcal{U}_{\Pi}$ be such that $\mathcal{U}_{\varphi(e;d_1)} \subseteq \mathcal{U}_{\varphi(e;d_2)}$ whenever $d_1 > d_2$. Let $d \in \mathbb{R}^+$ be any distance bound such that*

$$\forall e \in \mathcal{E}\ , \forall \mathbf{u} \in \mathcal{U}_{\Pi}(e)\ , \xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \rightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e). \tag{11}$$

Then $\forall e \in \mathcal{E}, \mathcal{U}_{\varphi(e;d)} \subseteq \mathcal{U}_{\varphi(e;d^b)} \subseteq \mathcal{U}_{\mathcal{S}}(e)$. Moreover, $\mathcal{E}_{\varphi}(d) \subseteq \mathcal{E}_{\varphi}(d^b) \subseteq \mathcal{E}_{\mathcal{S}}$. Hence, $\mathcal{E}_{\varphi}(d^b)$ and $\mathcal{U}_{\varphi(e;d^b)}$ quantify the tightest (largest) approximations of $\mathcal{E}_{\mathcal{S}}$ and $\mathcal{U}_{\mathcal{S}}(e)$ respectively among all uniform distance bounds d.

Theorem 1 states that d^b is the smallest among all (uniform) distance bounds between $\mathcal{M}$ and $\mathcal{S}$, such that a safe controller synthesized on $\mathcal{M}$ is also safe for $\mathcal{S}$. Even though this is a stricter condition than we need for defining $\mathcal{E}_{\mathcal{S}}$, where we care about the existence of at least one safe controller for $\mathcal{S}$, it allows us to use *any* safe controller for $\mathcal{M}$ as a safe controller for $\mathcal{S}$. Formally, $d^b \leq d$, for all $d \in \mathbb{R}^+$ such that $\forall e \in \mathcal{E}_{\varphi}(d)\ , \forall \mathbf{u} \in \mathcal{U}_{\varphi(e;d)}\ , \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$.

Intuitively, to compute (9), we collect all $\xi_{\mathcal{M}}(\cdot), \xi_{\mathcal{S}}(\cdot)$ pairs (across all $e \in \mathcal{E}$ and $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$) where $\xi_{\mathcal{M}}(\cdot) \models \varphi(e)$ and $\xi_{\mathcal{S}}(\cdot) \not\models \varphi(e)$. We then evaluate (10) for each pair and take the maximum to compute d^b. By expanding (contracting) every $A(\cdot) \in \mathcal{A}$ ($R(\cdot) \in \mathcal{R}$) uniformly by d^b, we ensure that none of the $\xi_{\mathcal{M}}(\cdot)$ collected above is feasible once the specification is modified, and hence, $\xi_{\mathcal{S}}(\cdot)$ will never falsify $\varphi(e)$. To ensure this, we prove that d^b is the minimum distance by which the avoid sets should be augmented (or the reach sets should be contracted). Thus, d^b can also be interpreted as the minimum d by which the specification should be modified to ensure that $\mathcal{U}_{\varphi(e;d)} \subseteq \mathcal{U}_{\mathcal{S}}(e)$ for all $e \in \mathcal{E}$.

Corollary 1. *Let $d \in [0, d^b]$ satisfies (11), then $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d)$ implies $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d^b)$.*

We conclude this section by discussing the relative advantages and limitations of SPEC and SSM, and a few remarks.

Comparing SPEC and SSM. SSM is specification-independent (and hence environment-independent); and hence can be reused across different tasks and environments. This is ensured by computing the distance between trajectories across all input control sequences; however, the very same aspect can make SSM overly-conservative. Making SPEC specification-dependent trades in generalizability for a less conservative measure. Although environment-dependent, the set of safe environments obtained using SPEC is larger compared to SSM. This is an important trade-off to make for any distance metric–the utility of a distance metric could be somewhat limited if it is too conservative.

The computational complexities for computing SPEC and SSM are the same since they both can be computed using Algorithm 1. To compute SSM we sample from a domain of all finite horizon controls. To compute SPEC we additionally need to be able to define and sample from the set of environment scenarios, but we believe that some representation of the environment scenarios is important for practical applications.

Remark 1. *The proposed framework can also be used in the scenarios where there is a deterministic controller for each environment. In such cases, $\mathcal{U}_{\Pi}(e)$ (and $\mathcal{U}_{\varphi(e)}$) is a singleton set for every environment e (see Section 8.2 for an example). However, from a control theory perspective, it might be useful to have a set of safe controllers that have different transient behaviors, that the system designer can choose from without recomputing the distance metric.*

Remark 2. *SPEC does not strictly meet the requirements for a metric because of the indicator function within its definition. However, it is possible to remove the indicator function from the definition and constrain the space of feasible environments and controls instead; i.e, we can replace $u \in \mathcal{U}_{\varphi(e)}$ by $u \in \mathcal{U}_{\varphi(e)} \wedge \xi_{\mathcal{S}}(\cdot) \not\models \varphi(e)$ in (9). The two definitions are equivalent, and SPEC would no longer be zero for two distinct input arguments; thus, it satisfies the properties of a metric.*

6 DISTANCE METRIC COMPUTATION

Since a dynamics model of $\mathcal{S}$ is not available, the computation of the distance bound d^b is generally difficult. Interestingly, this computational issue can be resolved using a randomized approach, such as scenario optimization [9]. Scenario optimization has been used for a variety of purposes [10, 11], such as robust control, model reduction, as well as for the computation of SSM [2].

Computing d^b by scenario optimization is summarized in Algorithm 1. We start by (randomly) extracting N realizations of the environment e_i, $i = 1, 2, \ldots, N$ (Line 2). Each realization e_i consists of an initial state x_0^i, and a sequence of reach and avoid sets, $A^i(t)$ and $R^i(t)$. For each e_i, we extract a controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$ (Line 5). If such a controller does not exist, we denote $\mathbf{u}_i$ to be a null controller $\mathbf{u}_{\phi}$. $\mathbf{u}_i$ (if not $= \mathbf{u}_{\phi}$) is then applied to both the system as well as the abstraction to obtain the corresponding trajectories $\xi_{\mathcal{S}}^i(\cdot; x_0^i, \mathbf{u}_i)$ and $\xi_{\mathcal{M}}^i(\cdot; x_0^i, \mathbf{u}_i)$ (Line 6). We next compute the distance between these two trajectories, d_i, using (10) (Line 7). If $\mathbf{u}_i = \mathbf{u}_{\phi}$, no satisfying

controller exists for $\mathcal{M}$, and hence $d(\xi_{\mathcal{S}}(\cdot; x_0^i, \mathbf{u}_n), \xi_{\mathcal{M}}(\cdot; x_0^i, \mathbf{u}_n))$ is trivially 0. The maximum across all these distances, $\hat{d}_\epsilon$, is then used as an estimate for d^b (Line 10).

Although simple in its approach, scenario optimization provides provable approximation guarantees. In Algorithm 1, we have to sample both an $e \in \mathcal{E}$ and a corresponding controller $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$. We define a joint sample space

$$\mathcal{D} = \{(e \times \mathcal{U}_{\varphi(e)}) : e \in \mathcal{E}, \mathcal{U}_{\varphi(e)} \neq \emptyset\} \cup \{(e, \mathbf{u}_\phi) : e \in \mathcal{E}, \mathcal{U}_{\varphi(e)} = \emptyset\} \tag{12}$$

$\mathcal{D}$ contains all feasible $(e, \mathbf{u})$ pairs for $\mathcal{M}$. We create a dummy sample $(e, \mathbf{u}_\phi)$ for all e where a satisfying controller does not exist. We next define a probability distribution on $\mathcal{D}$, $p(e, \mathbf{u}) = p(e) \cdot p(\mathbf{u}|e)$ where $p(e)$ is probability of sampling $e \in \mathcal{E}$ and $p(\mathbf{u} \mid e)$ is the probability of sampling $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$ given e. This distribution is key to capture the sequential nature of sampling $\mathbf{u}$ only after sampling e. For $e \in \mathcal{E}$ where $\mathcal{U}_{\varphi(e)} = \emptyset$, $p(\mathbf{u}_\phi \mid e) = 1$ since $\mathcal{D}$ has only a single entry for e, i.e, $(e, \mathbf{u}_\phi)$. In Algorithm 1, in Line 2, we sample $e_i \sim p(e)$. In Line 5, we sample $\mathbf{u}_i \sim p(\mathbf{u} \mid e_i)$.

Proposition 3. *Let $\mathcal{D}$ be the joint sample space as defined in* (12)*, with the probability distribution $p_{\mathcal{D}} = p(e, \mathbf{u})$. Select a 'violation parameter' $\epsilon \in (0, 1)$ and a 'confidence parameter' $\beta \in (0, 1)$. Pick N such that*

$$N \geq \frac{2}{\epsilon}\left(\ln\frac{1}{\beta} + 1\right), \tag{13}$$

then, with probability at least $1 - \beta$, the solution $\hat{d}_\epsilon$ to Algorithm 1 satisfies the following conditions:

(1) $\mathbb{P}((e, \mathbf{u}) \in \mathcal{D} : d(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}), \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u})) > \hat{d}_\epsilon) \leq \epsilon$

(2) $\mathbb{P}\left((e, \mathbf{u}) \in \mathcal{D} : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\right) > 1 - \epsilon$ *provided* $\mathcal{U}_{\varphi(e, \hat{d}_\epsilon)} \subseteq \mathcal{U}_{\varphi(e)}$.

The proof of Proposition 3 can be found in the Appendix. Intuitively, Proposition 3 states that $\hat{d}_\epsilon$ is a high confidence estimate of d^b, if a large enough N is chosen. If we discard the confidence parameter β for a moment, this proposition states that the size of the violation set (the set of $(e, \mathbf{u}) \in \mathcal{D}$ where the corresponding distance is greater than $\hat{d}_\epsilon$) is smaller than or equal to the prescribed ϵ value. As ϵ tends to zero, $\hat{d}_\epsilon$ approaches the desired optimal solution d^b. In turn, the simulation effort grows unbounded since N is inversely proportional to ϵ.

As for the confidence parameter β, one should note that $\hat{d}_\epsilon$ is a random quantity that depends on the randomly extracted $(e, \mathbf{u})$ pairs. It may happen that the extracted samples are not representative enough, in which case the size of the violation set will be larger than ϵ. Parameter β controls the probability that this happens; and the final result holds with probability $1 - \beta$. Since N in (13) depends logarithmically on $1/\beta$; β can be pushed down to small values such as 10^{-16}, to make $1 - \beta$ so close to 1 to lose any practical importance.

Finally, once we have a high confidence estimate of d^b, we can use it with Proposition 2 to provide guarantees on the safety of a controller for the system, provided that it is safe for the abstraction. (Statement (2) in Proposition 3)

Note that the controller $\mathbf{u}_i$ is extracted randomly from the set $\mathcal{U}_{\varphi(e_i)}$ (Line 5). Obtaining $\mathcal{U}_{\varphi(e_i)}$ and randomly sampling from it can be challenging in itself depending on the control scheme, Π, and the specification, $\varphi(e_i)$. However, one way to randomly extract $\mathbf{u}_i$ is using rejection sampling, i.e., we randomly sample controllers from the set $\mathcal{U}_\Pi$ until we find a controller that satisfies the specification for the model. Since the controller performance is evaluated only on the model during this process, it is often cheap and does not put the system at risk. Nevertheless, choosing a good control scheme makes this process more efficient, as the number of samples rejected before a feasible controller is found will be fewer (see Section 7 for further discussion on this). Rejection sampling, however, poses a problem when $\mathcal{U}_{\varphi(e_i)} = \emptyset$ and there is no way of knowing that beforehand. In such cases, one can impose a limit on the number of rejected samples to make sure the algorithm terminates. This problem can also be overcome easily when there is a single safe controller for each environment, i.e., $\mathcal{U}_{\varphi(e_i)}$ is a singleton set (see Remark 1).

Algorithm 1: Scenario optimization for estimating SPEC

1 set $\hat{d}_\epsilon = 0$
2 extract N realizations of the environment $e_i, i = 1, 2, \ldots, N$
3 **for** $i = 0 : N - 1$ **do**
4 **if** $\mathcal{U}_{\varphi(e_i)} \neq \emptyset$ **then**
5 extract a realization of a feasible controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$
6 run the controller $\mathbf{u}_i$ on $\mathcal{S}$ and $\mathcal{M}$, and obtain $\xi^i_{\mathcal{S}}(\cdot)$ and $\xi^i_{\mathcal{M}}(\cdot)$
7 compute $d_i = d(\xi^i_{\mathcal{S}}(\cdot), \xi^i_{\mathcal{M}}(\cdot))$
8 **else**
9 $\mathbf{u}_i = \mathbf{u}_\phi$ and $d_i = 0$
10 set $\hat{d}_\epsilon = \max_{i \in \{1, 2, \ldots, N\}} d_i$

Remark 3. *Even though we have presented scenario optimization to estimate d^b, alternative derivative free optimization approaches such as Bayesian optimization, simulated annealing, evolutionary algorithms, and covariance matrix adaptation can be used as well. However, for many of these algorithms, it might be challenging to provide formal guarantees on the quality of the resultant estimate of the distance bound.*

Algorithm 1 samples N environment scenarios and corresponding controllers prior to running any executions on $\mathcal{M}$ and $\mathcal{S}$. Imagine at iteration i, we have $d_i > 0$; and if at iteration $(i+1)$, $d_{i+1} < d_i$, then the $(i+1)$th sample is not informative for approximating d^b. A simple way to overcome this issue would be to consider only $\mathcal{U}_{\varphi(e_i; d_i)}$ as the set of feasible controllers at the $(i+1)$th iteration; i.e., consider controllers where $\xi_{\mathcal{M}}(\cdot) \models \varphi(e_i; d_i)$. This variant of Algorithm 1 would reduce the number of executions of the system; and ensure that each execution is informative for estimating d^b. To implement this scheme, we would maintain a running max $d^b_{(i)}$ which contains the maximum of d_i seen till now. In iteration $(i+1)$, instead of sampling from $\mathcal{U}_{\varphi(e)}$ in Line 5, we sample from $\mathcal{U}_{\varphi(e, d^b_{(i)})}$. Further, before the end of loop, in Line 7, we update $d^b_{(i+1)} = \max(d^b_{(i)}, d_{i+1})$.

7 RUNNING EXAMPLE: DISTANCE COMPUTATION

We now apply the proposed algorithm to compute d^b for the setting described in Section 4. $\mathcal{U}_{\varphi(e)}$ in this case is given as

$$\mathcal{U}_{\varphi(e)} = \{\mathbf{u} \in \mathcal{U}_\Pi(e) : \|\xi_\mathcal{M}(H; x_0, \mathbf{u}) - x^*\|_2 < \gamma\},$$

where $\mathcal{U}_\Pi(e)$ is the set of LQR controllers (see Section 4). To illustrate the importance of the choice of distance metric, we compute two different distance metrics between $\mathcal{S}$ and $\mathcal{M}$: d^a in (7) and d^b in (9). To compute d^b, we use Algorithm 1. To compute d^a, we modify Algorithm 1 to sample a random controller from $\mathcal{U}_\Pi(e)$ in Line 5 and compute d_i using (7) in Line 7.

According to the scenario approach with $\epsilon = 0.01$ and $\beta = 10^{-6}$, we extract $N = 2964$ different reach-avoid scenarios (i.e., N different final states to reach). For each $e_i, i \in \{1, 2, \ldots, 2964\}$, we obtain a feasible LQR controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$ using rejection sampling. In particular, we randomly sample a penalty parameter q, solve the corresponding Riccati equation to obtain $LQR(q)$, and apply it on $\mathcal{M}$. If the corresponding $\xi_\mathcal{M}(\cdot)$ satisfies $\varphi(e_i)$, we use $\mathbf{u}_i$ as our feasible controller sample; otherwise, we sample a new q and repeat the procedure until a feasible controller is found. This procedure tends to be really fast and requires simulating only $\mathcal{M}$. A feasible controller was found within 3 samples of q for all e_i in this case. For d^a, we randomly sample a penalty parameter q and use $LQR(q)$ as the controller.

The obtained distance metrics are $d^a = 0.43, d^b = 0$. Since $d^a < \gamma$, it can be used to synthesize a safe controller for $\mathcal{S}$; however, we can synthesize controller only for those reach-avoid scenarios where $\mathcal{M}$ satisfies a much stringent specification: $\xi_\mathcal{M}$ must reach within a ball of radius 0.07 around the target state. Consequently, the set $\mathcal{E}_\varphi(d^a)$ is likely to be very small. In contrast, $d^b = 0$; thus, Proposition 3 ensures that *any* controller designed on $\mathcal{M}$ that satisfies $\varphi(e)$ is guaranteed to satisfy it for $\mathcal{S}$ as well. In particular, the dynamics of $\mathcal{S}$ and $\mathcal{M}$ are same for the state x_1, and state x_2 is uncontrollable for $\mathcal{S}$ and remain 0 at all times. Thus, any controller that reaches within a ball of radius γ around a desired state x_1^* for $\mathcal{M}$, if applied on $\mathcal{S}$, also ensures that the system state reaches within the same ball. Even though this relationship between $\mathcal{S}$ and $\mathcal{M}$ is unknown, d^b is able to capture it only through simulations of $\mathcal{S}$. This example also illustrates that d^b significantly reduces the conservativeness in SSM, and does not unnecessarily contract the set of safe environments.

8 CASE STUDIES

We now demonstrate how SPEC can be used to obtain the safe set of environments and controllers for an autonomous quadrotor and an autonomous car. In Section 8.1, we demonstrate how SPEC provides much larger safe sets compared to SSM. In Section 8.2, we demonstrate how SPEC not only captures the differences between the dynamics of $\mathcal{S}$ and $\mathcal{M}$, but also other aspects of the system, in particular the sensor error, that might affect the satisfiability of a specification.

8.1 Safe Altitude Control for Quadrotor

Our first example illustrates how the proposed distance metric behaves when the only difference between the system and the abstraction is the value of one parameter. However, unlike the running example, the system and the abstraction dynamics are non-linear. Moreover, we illustrate how SPEC can be used in the cases where all safe controllers for $\mathcal{M}$ may not be safe for $\mathcal{S}$.

We use the reach-avoid setting described in [15], where the authors are interested in controlling the altitude of a quadrotor in an indoor setting while ensuring that it does not go too close to the ceiling or the floor, which are obstacles in our experiments.

A dynamic model of quadrotor vertical flight can be written as:

$$\begin{aligned} z(t+1) &= z(t) + T v_z(t) \\ v_z(t+1) &= v_z(t) + T(ku(t) + g), \end{aligned} \tag{14}$$

where z is the vehicle's altitude, v_z is its vertical velocity and u is the commanded average thrust. The gravitational acceleration is $g = -9.8m/s^2$ and the discretization step T is 0.01. The control input $u(t)$ is bounded to $[0, 1]$. We are interested in designing a controller for $\mathcal{S}$ that ensures safety over a horizon of 100 timesteps. In particular, we have $\mathcal{X}_0 = \{(z, v_z) : 0.5 \le z \le 2.5 \wedge -3 \le v_z \le 4\}$, $\mathcal{A} = \{A(\cdot)\}$, and $\mathcal{R} = \mathbb{R}^2$. The avoid set at any time t is given as $A(t) = \{(z, v_z) \in \mathbb{R}^2 : 0.5m \le z(t) \le 2.5m\}$. We again assume that the dynamics in (14) are unknown. Consider an abstraction of $\mathcal{S}$ with same dynamics as (14) except that the value of parameter k in the abstraction dynamics, $k_\mathcal{M}$, is different.

The space of controllers $\mathcal{U}_\Pi(e)$ is given by all possible control sequences over the time horizon (i.e., $\mathcal{U}_\Pi(e) = \mathcal{U}$.) For computing $\mathcal{U}_{\varphi(e)}$, we use the Level Set Toolbox [30] that gives us both the set of initial states from which there exist a controller that will keep the $\xi_\mathcal{M}(\cdot)$ outside the avoid set at all times (also called the reachable set), as well as the corresponding least restrictive controller. In particular, we can apply any control when the abstraction trajectory is inside the reachable set and the safety-preserving control (given by the toolbox) when the trajectory is near the boundary of the reachable set. For computation of the distance bounds, we sample a random controller sequence according to this safety-preserving control law. If any initial state lies outside the reachable set, then it is also guaranteed that $\mathcal{U}_{\varphi(e)} = \emptyset$ so we do not need to do any rejection sampling in this case.

When $k_\mathcal{M} < k$, $\mathcal{M}$ has strictly less control authority compared to $\mathcal{S}$. Thus, any controller that satisfies the specification for $\mathcal{M}$ will also satisfy the specification for $\mathcal{S}$, so $\mathcal{E}_\varphi(0)$ itself is an under approximation of $\mathcal{E}_\mathcal{S}$. SPEC is again able to capture this behavior. Indeed, we computed an estimate for the distance bound using Algorithm 1 and the obtained numbers are $d^a = 0.30$ and $d^b = 0$. Note that not only is d^a conservative, it may not be particularly useful in synthesizing a safe controller for $\mathcal{S}$. d^a computed using Algorithm 1 ensures that a safe controller designed on $\mathcal{M}$ for $\varphi(e; d^a)$ is also safe for $\mathcal{S}$ with high probability, only when this controller is *randomly* selected from the set $\mathcal{U}_\Pi$. However, a random controller selected from $\mathcal{U}_\Pi$ is unlikely to satisfy $\varphi(e; d^a)$ for $\mathcal{M}$ itself, and thus nothing can be said about $\mathcal{S}$ either. Thus, it is hard to *actually* compute an approximation of $\mathcal{E}_\mathcal{S}$. In contrast, d^b samples a controller from the set $\mathcal{U}_{\varphi(e)}$ in Algorithm 1. Therefore, to synthesize a controller, we *randomly* select a controller from the set $\mathcal{U}_{\varphi(e;d^b)}$, which is guaranteed to be safe on both $\mathcal{M}$ and $\mathcal{S}$ with high probability. Therefore, it might be better to compare d^b to d^{a2}, which is

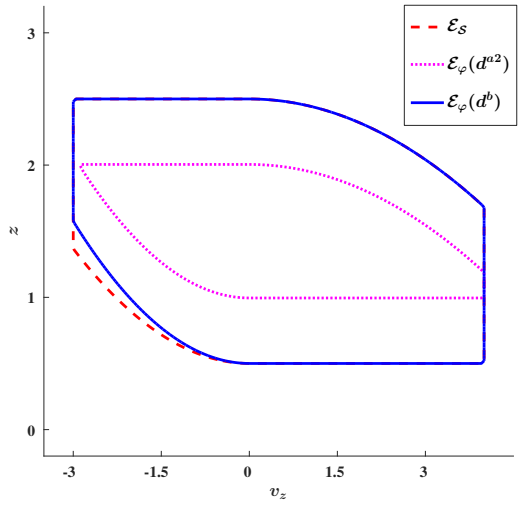


Figure 2. Different reachable sets when the quadrotor abstraction is conservative. The distance metric d^b only considers the distance between trajectories that violates the specification on the system and satisfies it on the abstraction, leading to a less conservative estimate of the distance, and a better approximation of $\mathcal{E}_{\mathcal{S}}$.

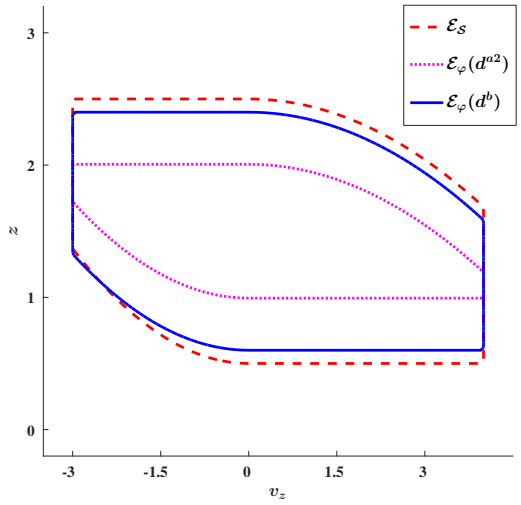


Figure 3. Different reachable sets when the quadrotor abstraction is overly optimistic. The distance metric d^b achieves a far less conservative underapproximation of $\mathcal{E}_{\mathcal{S}}$ compared to the other distance metrics.

defined similar to d^a, except the inner maximum in (7) is computed over $\mathcal{U}_{\varphi(e)}$ instead. d^{a2} in this case turns out to be 0.5.

Note that if we could instead compute the distance metrics exactly, $d^{a2} \leq d^a$, since $\mathcal{U}_{\varphi(e)} \subset \mathcal{U}_{\Pi}$. However, random sampling based estimate of d^{a2} can be greater than that of d^a if the controllers corresponding to a large distance between the $\xi_{\mathcal{S}}(\cdot)$ and $\xi_{\mathcal{M}}(\cdot)$ are sparse in $\mathcal{U}_{\Pi}$ compared to that in $\mathcal{U}_{\varphi(e)}$.

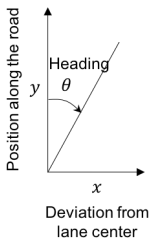
For illustration purposes, we also compute the reachable set $\mathcal{E}_{\varphi}(d^b)$, by augmenting the avoid set by d^b and recomputing the reachable sets using the Level Set Toolbox. As shown in Figure 2, $\mathcal{E}_{\varphi}(0)$ (the area withing the blue contour) is indeed contained within $\mathcal{E}_{\mathcal{S}}$ (the area within the red contour). Here, $\mathcal{E}_{\mathcal{S}}$ has been computed using the system dynamics. Even though $\mathcal{E}_{\varphi}(d^{a2})$ (the area within the magenta contour) is also contained in $\mathcal{E}_{\mathcal{S}}$, it is significantly smaller in size compared to $\mathcal{E}_{\varphi}(d^b)$.

When $k_{\mathcal{M}} > k$, $\mathcal{S}$ has strictly less control authority compared to $\mathcal{M}$. Consequently, there might exist some environments for which it is possible to synthesize a safe controller for $\mathcal{M}$, but the same controller when deployed on $\mathcal{S}$ might lead to an unsafe behavior. We again compute the distance bounds using Algorithm 1 and the obtained numbers are $d^a = 0.30, d^{a2} = 0.49, d^b = 0.1$. The corresponding reachable sets are shown in Figure 3. Even though we start with an overly optimistic abstraction, both d^{a2} and d^b are able to compute an under approximation of $\mathcal{E}_{\mathcal{S}}$; however, the set estimated by d^{a2} is, once again, overly conservative.

8.2 Webots: Lane Keeping

We now show the application of the proposed metric for designing a safe lane keeping controller for an autonomous car.

In this example, we use the **Webots** simulator [39]. The car model within the simulator is our $\mathcal{S}$. For the abstraction $\mathcal{M}$ we consider the bicycle model,



$$\begin{aligned} \dot{x} &= v \cdot \sin\theta \\ \dot{y} &= v \cdot \cos\theta \\ \dot{v} &= a \\ \dot{\theta} &= \frac{v}{l} \tan\omega \end{aligned} \tag{15}$$

where $[x, y, v, \theta]$ is the state, representing perpendicular deviation from the center of the lane, position along the road, speed, and heading respectively. The maximum speed is limited to $v_{max} = 10$ km/hr. We have two inputs, (1) a discrete acceleration control $a = \{-\bar{a}, 0, \bar{a}\}$; and (2) a continuous steering control $\omega \in [-\pi/4, \pi/4] rad/s$. For our experiments, we use $H = 200$, which translates to about 6 seconds of simulated trajectory. The dynamics of $\mathcal{S}$ are typically much more complex than $\mathcal{M}$ and include the physical effects like friction and slip on the road.

In this case, $\mathcal{X}_0 = \{(x_0, \theta_0) : \|x\| \leq 0.2m \wedge \|\theta\| \leq \pi/4 rad\}$; the initial y_0 and v_0 is set to *zero*. $R(t) = \{[x(t), y(t), v(t), \theta(t)] \in \mathbb{R}^4 : \|x(t)\| \leq 0.5m\} \forall t \in \mathcal{T}$. The reach set corresponds to keeping the car within the $0.5m$ of the center of the lane. For keeping the car in the lane, the car is equipped with two sensors, a camera (to capture the lane ahead) and compass (to measure the heading of the car). There is an on board perception module, which first captures the image of the road ahead; and processes it to detect the lane edges and provide an estimate of the deviation of the car from the center of the lane.

There is another car (referred to as the environment car hereon) driving in the front of $\mathcal{S}$, which might obstruct the lane and cause the perception module to incorrectly detect the lane center. For each $e \in \mathcal{E}$, the set of possible initial states of the environment car is given by $\mathcal{P} = \{(x_e, y_e) : \|x_e - x_0\| \leq 2.0m \wedge 6.25m \leq y_e - y_0 \leq 8m\}$. We set the initial speed v_e and heading θ_e of the environment car to $v_{\max}$ and 0 respectively. We want to make sure

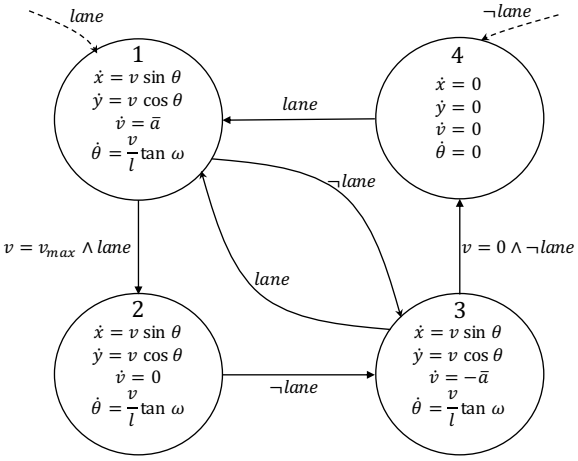
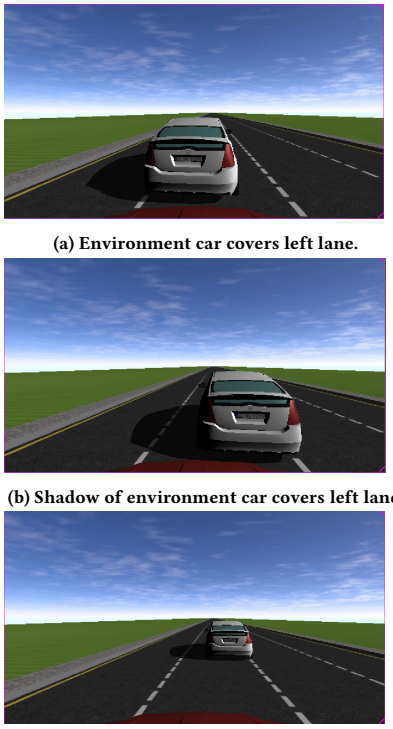


Figure 4. Hybrid controller for lane keeping. *lane* means a lane is detected by the perception system. The dashed line represents the transitions taken on initialization based on the value of *lane*. To closely follow the center of the lane, we synthesize a LQR controller in each mode.

that $\mathcal{S}$ remains within the lane despite all possible initial positions of the environment car. For this purpose, we compute the worst-case d^b across all $p \in \mathcal{P}$.

If the environment car or its shadow covers the lane edges (see Figure 5 for some possible scenarios), then the lane detection fails. Technically speaking, if such a scenario occurs, then $\mathcal{S}$ should slow down and come to stop until the image processing starts detecting the lane again. Consequently, our control scheme $\mathcal{U}_\Pi$, is a hybrid controller shown in Figure 4, where in each mode the controller is given by an LQR controller (with a fixed Q and R matrix) corresponding to the (linearized) dynamics in that mode. In this example, our controller is a deterministic controller since the Q and R matrices are fixed, and hence $|\mathcal{U}_\Pi| = 1$. In Figure 4, in mode (1), the lane is detected and $v(t) < v_{max}$. When the $v(t) = v_{max}$ we transition to mode (2) given the lane is still detected. When the lane is no longer detected, we transition to mode (3) if $v(t) > 0$, or mode (4) if $v(t) = 0$. In modes (3) and (4), the car slows down until the lane is detected again.

By setting $\epsilon = 0.01$ and $\beta = 1e-6$ we get $N \geq 2964$. We used Algorithm 1, to sample N different initial states of the $\mathcal{S}$, $(x_0, \theta_0) \in \mathcal{X}_0$; and environment car in the simulator, $p \in \mathcal{P}$. Since the controller is deterministic, the set of feasible controllers is a singleton set, and hence we do not need to sample a feasible controller (Line 5 in Algorithm 1). Among these environment scenarios, the controller on $\mathcal{M}$ is also able to safely control $\mathcal{S}$ for 2519 scenarios. $\hat{d}_\epsilon$ is determined entirely by the remaining 445 controller, and computed to be 0.34m. We show the application of the the computed $\hat{d}_\epsilon$ for a sample environment scenario in Figure 6. The green lines represent the original reach set. The yellow shaded region represents the contracted reach set for the model computed using $\hat{d}_\epsilon$. The model's trajectory (shown in blue) is contained in the yellow region and hence satisfies the more constrained specification. As a result, even though the system's trajectory (shown in dotted red) leaves the yellow region, it is contained within the original reach set at all times.



(a) Environment car covers left lane.

(b) Shadow of environment car covers left lane.

(c) Lane detected correctly.

Figure 5. The lane detection fails for (a) and (b) and $\mathcal{S}$ car tries to slow down. When lane is correctly detected (c), the LQR controller tries to follow the lane

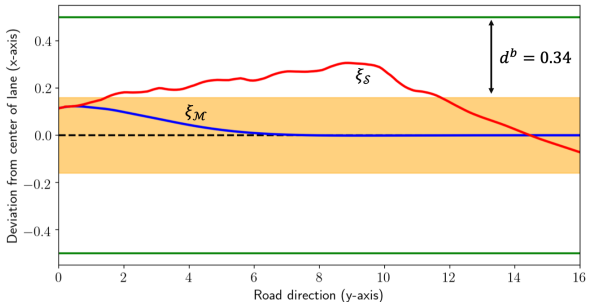


Figure 6. The green lines represent the boundaries of the original reach set. The yellow region is the contracted reach set for the model computed using $\hat{d}_\epsilon$. The model's trajectory shown in blue is entirely contained within the yellow region. Consequently, the system's trajectory (shown in dotted red) leaves the yellow region but is contained within the original reach set at all times.

We also analyze these 445 environmental scenarios that contribute to $\hat{d}_\epsilon$, and notice that the fault lies within the perception module. In Figure 7, we show one such scenario. In this case, $\theta_0 = -\pi/4$. Because of the left rotation of the car, the rightmost lane appears smaller and farther due to the perspective distortion. Furthermore, the presence of the environment car completely cover the rightmost lane in the image. The image processing module now detects the leftmost lane as the center lane and the center lane as the rightmost lane. Consequently, the module returns an inaccurate estimation of the center of the lane, causing $\mathcal{S}$ to go outside the

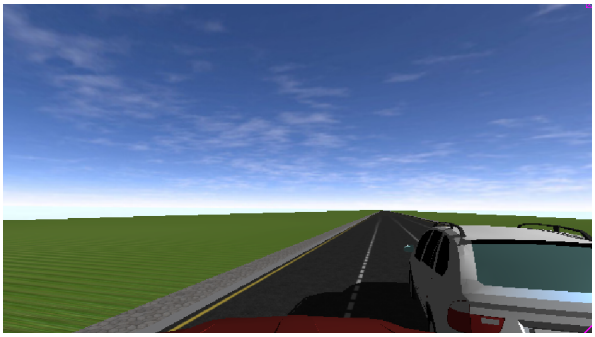


Figure 7. An example of the environment scenario that contributes to the distance between the model and the system. The environment samples used for computing SPEC can be used to identify the reasons behind the violation of the safety specification by the system.

center lane. This example illustrates that the samples in Algorithm 1 that contributed to $\hat{d}_\epsilon$ could also be used to analyze the reasons behind the violation of the safety specification by $\mathcal{S}$.

9 CONCLUSION AND FUTURE WORK

Determining safe environments and synthesizing safe controllers for autonomous systems is an important problem. Typically, we rely on an abstraction of the system to synthesize controllers in different environments. However, when a mathematical model of the system is not available, for example when the abstraction is obtained through data, it becomes challenging to provide safety guarantees for the system based on the abstraction. In this paper, we propose a specification-centric simulation metric SPEC that can be used to determine the set of safe environments; and to synthesize a safe controller using such data-driven abstractions. We also present an algorithm to compute this metric using executions on the system without knowing its true dynamics. The proposed metric is less conservative and allows controller synthesis for reach-avoid specifications over a broader range of environments compared to the standard simulation metric.

In future, it would be interesting to extend the proposed framework for more general specifications and study its application in runtime-assurance frameworks like [22] and [13]. Another interesting direction will be to explore active sampling methods, such as Bayesian Optimization, for the computation of SPEC.

REFERENCES

[1] A. Abate. 2009. A contractivity approach for probabilistic bisimulations of diffusion processes. In *Proceedings of the 48th IEEE Conference on Decision and Control*.

[2] A. Abate and M. Prandini. 2011. Approximate abstractions of stochastic systems: A randomized method. In *Conference on Decision and Control and European Control Conference*.

[3] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. 2000. Discrete abstractions of hybrid systems. In *Proceedings of the IEEE*.

[4] C. Baier, J. Katoen, and K. G. Larsen. 2008. *Principles of model checking*. MIT press.

[5] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin. 2016. Learning quadrotor dynamics using neural network for flight control. In *Conference on Decision and Control*.

[6] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin. 2017. Goal-Driven Dynamics Learning via Bayesian Optimization. In *Conference on Decision and Control*.

[7] M. L. Bujorianu, J. Lygeros, and M. C. Bujorianu. 2005. Bisimulation for general stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*.

[8] G. C. Calafiore and M. C. Campi. 2005. Uncertain convex programs: randomized solutions and confidence levels. In *Mathematical Programming*.

[9] G. C. Calafiore and M. C. Campi. 2006. The scenario approach to robust control design. In *IEEE Transactions on Automatic Control*.

[10] M. C. Campi and S. Garatti. 2011. A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality. In *Journal of Optimization Theory and Applications*.

[11] M. C. Campi, S. Garatti, and M. Prandini. 2009. The scenario approach for systems and control design. In *Annual Reviews in Control*.

[12] T. Dean and R. Givan. 1997. Model minimization in Markov decision processes. In *AAAI/IAAI*.

[13] Ankush Desai, Shromona Ghosh, Sanjit A. Seshia, Natarajan Shankar, and Ashish Tiwari. 2018. SOTER: Programming Safe Robotics System using Runtime Assurance. In *arXiv:1808.07921*.

[14] J. Desharnais, A. Edalat, and P. Panangaden. 2002. Bisimulation for labelled Markov processes. In *Information and Computation*.

[15] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin. 2017. A general safety framework for learning-based control in uncertain robotic systems. In *arXiv:1705.01292*.

[16] S. Garatti and M. Prandini. 2012. A simulation-based approach to the approximation of stochastic hybrid systems. In *Analysis and design of hybrid systems*.

[17] M. Gevers, X. Bombois, B. Codrons, G. Scorletti, and B. D. O. Anderson. 2003. Model Validation for Control and Controller Validation in a Prediction Error Identification framework-Part I: Theory. In *Automatica*.

[18] S. Ghosh, D. Sadigh, P. Nuzzo, V. Raman, A. Donzé, A. L. Sangiovanni-Vincentelli, S. Sastry, and S. A. Seshia. 2016. Diagnosis and Repair for Synthesis from Signal Temporal Logic Specifications. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*.

[19] A. Girard and G. J. Pappas. 2007. Approximate bisimulation relations for constrained linear systems. In *Automatica*.

[20] A. Girard and G. J. Pappas. 2011. Approximate Bisimulation: A bridge between computer science and control theory. In *European Journal of Control*.

[21] A. Girard, G. Pola, and P. Tabuada. 2010. Approximately bisimilar symbolic models for incrementally stable switched systems. In *IEEE Transactions on Automatic Control*.

[22] Sylvia L Herbert*, Mo Chen*, SooJean Han, Somil Bansal, Jaime F Fisac, and Claire J Tomlin. 2017. FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning. *IEEE Conference on Decision and Control*.

[23] H. Hjalmarsson and L. Ljung. 1992. Estimating model variance in the case of undermodeling. In *IEEE Transactions on Automatic Control*.

[24] A. A. Julius and G. J. Pappas. 2009. Approximations of stochastic hybrid systems. In *IEEE Transactions on Automatic Control*.

[25] J. P. Katoen, T. Kemna, I. Zapreev, and D. N. Jansen. 2007. Bisimulation minimisation mostly speeds up probabilistic model checking. In *International Conference on tools and algorithms for the construction and analysis of systems*.

[26] H. Kwakernaak and R. Sivan. 1972. *Linear optimal control systems*.

[27] K. G. Larsen and A. Skou. 1991. Bisimulation through probabilistic testing. In *Information and computation*.

[28] I. Lenz, R. A. Knepper, and A. Saxena. 2015. DeepMPC: Learning Deep Latent Features for Model Predictive Control.. In *Robotics: Science and Systems*.

[29] L. Ljung. 1987. *System identification: theory for the user*.

[30] I. M. Mitchell. 2008. The flexible, extensible and efficient toolbox of level set methods. In *Journal of Scientific Computing*.

[31] S. Mitsch, K. Ghorbal, and A. Platzer. 2013. On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles. In *Robotics: Science and Systems*.

[32] A. V. Papadopoulos and M. Prandini. 2016. Model reduction of switched affine systems. In *Automatica*.

[33] G. Pola, A. Girard, and P. Tabuada. 2008. Approximately bisimilar symbolic models for nonlinear control systems. In *Automatica*.

[34] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. 2016. Towards Verified Artificial Intelligence. In *arXiv:1606.08514*.

[35] S. Strubbe and A. Van Der Schaft. 2005. Bisimulation for communicating piecewise deterministic Markov processes (CPDPs). In *International Workshop on Hybrid Systems: Computation and Control*.

[36] Paulo Tabuada. 2009. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer Science & Business Media.

[37] C. J. Tomlin, J. Lygeros, and S. Sastry. 2000. A game theoretic approach to controller design for hybrid systems. In *Proceedings of the IEEE*.

[38] C. J. Tomlin, G. J. Pappas, and S. Sastry. 1998. Conflict resolution for air traffic management: A study in multiagent hybrid systems. In *IEEE Transactions on automatic control*.

[39] Webots. 1998. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

10 APPENDIX

10.1 Proof of Proposition 1

Proof. Let us consider for a given environment $e \in \mathcal{E}$ and control $\mathbf{u} \in \mathcal{U}_\Pi(e)$, $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^a)$. We would like to prove that $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$.

From (7), we have

$$\|\xi_\mathcal{S}(t) - \xi_\mathcal{M}(t)\| \leq d^a \ \forall t \in \mathcal{T}. \tag{16}$$

From the definition of specification in (8), we have $\xi_\mathcal{M}(\cdot) \models \varphi(e; d^a)$ if and only if $\xi_\mathcal{M}(\cdot) \in \varphi(e; d^a)$. Therefore, $\xi_\mathcal{M}(t) \notin A(t) \oplus d^a$ and $\xi_\mathcal{M}(t) \in R(t) \ominus d^a \ \forall t \in \mathcal{T}$. Since $\xi_\mathcal{M}(t) \notin A(t) \oplus d^a$,

$$\|\xi_\mathcal{M}(t) - a\| > d^a \ , \forall t \in \mathcal{T} \ , \forall a \in A(t). \tag{17}$$

Combining (16) and (17) implies that

$$\|\xi_\mathcal{S}(t) - a\| > 0 \ , \forall t \in \mathcal{T} \ , \forall a \in A(t). \tag{18}$$

Equation (18) implies that $\xi_\mathcal{S}(t) \notin A(t)$ for any $t \in \mathcal{T}$. Similarly, it can be shown that

$$\|\xi_\mathcal{S}(t) - r\| > 0 \ , \forall t \in \mathcal{T} \ , \forall r \in R(t)^c,$$

where $R(t)^c$ denotes the complement of the set $R(t)$. Therefore, $\xi_\mathcal{S}(t) \in R(t) \ \forall t \in \mathcal{T}$.

Since $\xi_\mathcal{S}(t) \notin A(t)$ and $\xi_\mathcal{S}(t) \in R(t)$ for all $t \in \mathcal{T}$, we have $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$. □

10.2 Proof of Proposition 2

Proof. We prove the desired result by contradiction. Suppose there exists an environment $e \in \mathcal{E}$ and a controller $\mathbf{u} \in \mathcal{U}_{\varphi(e, d^b)}$ such that $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$ but $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \not\models \varphi(e)$.

Since $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$, we have that,

$$\forall t \in \mathcal{T} \ \xi_\mathcal{M}(t; x_0, \mathbf{u}) \notin A(t; d^b) = A(t) \oplus d^b \tag{19}$$

$$\forall t \in \mathcal{T} \ \xi_\mathcal{M}(t; x_0, \mathbf{u}) \in R(t; d^b) = R(t) \ominus d^b \tag{20}$$

Since d^b is the solution to (9), and $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$ (as $\mathcal{U}_{\varphi(e; d^b)} \subseteq \mathcal{U}_{\varphi(e)}$) is such that $\xi_\mathcal{M}(\cdot) \not\models \varphi(e)$, (9) and (10) imply that,

$$\min_{t \in \mathcal{T}} \left(\min\{ h\left(\xi_\mathcal{M}(t), A(t)\right), -h\left(\xi_\mathcal{M}(t), R(t)\right) \} \right) \leq d^b. \tag{21}$$

Therefore, $\exists t' \in \mathcal{T}$ such that

$$\min\{ h\left(\xi_\mathcal{M}(t'), A(t')\right), -h\left(\xi_\mathcal{M}(t'), R(t')\right) \} \leq d^b, \tag{22}$$

which implies that either

(1) $h\left(\xi_\mathcal{M}(t'), A(t')\right) \leq d^b$, or
(2) $h\left(\xi_\mathcal{M}(t'), R(t')\right) \geq -d^b$

If $h\left(\xi_\mathcal{M}(t'), A(t')\right) \leq d^b$, $\exists a \in A(t')$ such that

$$\|\xi_\mathcal{M}(t') - a\| \leq d^b.$$

Therefore, $\xi_\mathcal{M}(t') \in A(t') \oplus d^b$, which contradicts (19). Similarly, if $h\left(\xi_\mathcal{M}(t'), R(t')\right) \geq -d^b$, $\exists r \in R(t')^c$ such that

$$\|\xi_\mathcal{M}(t') - r\| \leq d^b,$$

which implies that $\xi_\mathcal{M}(t') \notin R(t') \ominus d^b$, which contradicts (20).

When $\mathcal{U}_{\varphi(e, d^b)} \nsubseteq \mathcal{U}_{\varphi(e)}$, for any controller $\mathbf{u} \in \mathcal{U}_{\varphi(e, d^b)} \setminus \mathcal{U}_{\varphi(e)}$ such that $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^b)$, we can no longer comment on the behavior of the corresponding system trajectory. This is because while computing d^b, these controllers were not taken into account. □

10.3 Proof of Proposition 3

Proof. Statement (1) of Proposition 3 follows directly from the guarantees provided by Scenario Optimization (Theorem 1 in [11]). To use the result in [11], we need to prove: (a) computing d^b can be converted into a standard Scenario Optimization problem and (b) Algorithm 1 samples i.i.d from $\mathcal{D}$ with probability $p_\mathcal{D}$.

(9) can be re-written as $d^b = \max_{(e,\mathbf{u}) \in \mathcal{D}} d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot))$ which can be formalized as the following optimization problem,

$$\begin{aligned} &\min \ g \\ &\text{s.t. } \forall (e, \mathbf{u}) \in \mathcal{D} \ , d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)) \leq g \end{aligned}$$

This is semi-infinite optimization problem where the constraints are convex (in fact, linear) in the optimization variable g for any given $(e, \mathbf{u})$. Statement (1) now follows from Theorem 1 in [11] by replacing $c = 1$, γ by g, Δ by $\mathcal{D}$, and f by $d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)) - g$. Theorem 1 in [11], however, requires that i.i.d samples are chosen from the distribution $p_\mathcal{D}$. This can be proved by noticing that, in Algorithm 1, we first sample $e_i \sim p(e)$ (in Line 2), and then sample $\mathbf{u}_i \sim p(\mathbf{u} \mid e)$ (in Line 4.) Hence, every $(e_i, \mathbf{u}_i)$ is sampled from $p_\mathcal{D} = p(e) \cdot p(\mathbf{u} \mid e)$. Since each $i = 1, \ldots, N$ is sampled randomly and independent of each other, the $(e_i, \mathbf{u}_i)$ pairs are indeed sampled i.i.d from $p_\mathcal{D}$.

Algorithm 1 returns an estimate $\hat{d}_\epsilon$ for d^b. We have already established that $\hat{d}_\epsilon$ satisfies the probabilistic guarantees provided by scenario optimization (Statement (1)). From Proposition 2, we have $\forall (e, \mathbf{u}) \in \mathcal{D}$ where $d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)) \leq \hat{d}_\epsilon$, $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$, provided $\mathcal{U}_{\varphi(e; \hat{d}_\epsilon)} \subseteq \mathcal{U}_{\varphi(e)}$. Therefore,

$$\mathbb{P}\left((e, \mathbf{u}) \in \mathcal{D} : \xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e) \right) > 1 - \epsilon.$$

□

10.4 Proof of Theorem 1 and Corollary 1

Proof. Consider any $d > d^b$. From the statement of Theorem 1, we have that $\mathcal{U}_{\varphi(e; d)} \subseteq \mathcal{U}_{\varphi(e; d^b)}$. Hence, $\mathcal{E}_\varphi(d) \subseteq \mathcal{E}_\varphi(d^b)$ follows from the definition of $\mathcal{E}_\varphi(d)$ in (6). $\mathcal{U}_{\varphi(e; d^b)} \subseteq \mathcal{U}_\mathcal{S}(e)$ and $\mathcal{E}_\varphi(d^b) \subseteq \mathcal{E}_\mathcal{S}$ is already ensured by Proposition 2, and hence Theorem 1 follows.

We now prove that for all $0 < d < d^b$, $\exists \ e \in \mathcal{E}$ such that (11) does not hold, and hence the result of Theorem 1 trivially holds. We prove the result by contradiction. Suppose $0 < d < d^b$ be such that (11) holds. Let $(e^*, \mathbf{u}^*)$ be the environment, controller pair where $d(\xi_\mathcal{M}(\cdot; x_0^*, \mathbf{u}^*), \xi_\mathcal{S}(\cdot; x_0^*, \mathbf{u}^*)) = d^b$. Equation (9) and (10) thus imply that

$$\min_{t \in \mathcal{T}} \left(\min\{ h\left(\xi_\mathcal{M}(t; x_0^*, \mathbf{u}^*), A^*(t)\right), -h\left(\xi_\mathcal{M}(t; x_0^*, \mathbf{u}^*), R^*(t)\right) \} \right) = d^b, \tag{23}$$

and $\xi_\mathcal{S}(\cdot; x_0^*, \mathbf{u}^*) \not\models \varphi(e^*)$. Equation (23) implies that

$$\forall t \in \mathcal{T}, h\left(\xi_\mathcal{M}(t; x_0^*, \mathbf{u}^*), A^*(t)\right) \geq d^b \tag{24}$$

$$\forall t \in \mathcal{T}, h\left(\xi_\mathcal{M}(t; x_0^*, \mathbf{u}^*), R^*(t)\right) \leq -d^b. \tag{25}$$

Equations (24) and (25) imply that

$$\forall t \in \mathcal{T}, \ \xi_\mathcal{M}(t; x_0^*, \mathbf{u}^*) \notin A^*(t) \oplus d, \xi_\mathcal{M}(t; x_0^*, \mathbf{u}^*) \in R^*(t) \ominus d. \tag{26}$$

Consequently, we have $\xi_{\mathcal{M}}(\cdot; x_0^*, \mathbf{u}^*) \models \varphi(e^*; d)$. This contradicts (11) since $\xi_{\mathcal{S}}(\cdot; x_0^*, \mathbf{u}^*) \not\models \varphi(e^*)$. Therefore, for all $0 < d < d^b$, $\exists\ e \in \mathcal{E}$, $\mathbf{u} \in \mathcal{U}_\Pi(e)$, $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \nrightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$.

To prove the corollary, we first prove that if $d_1 > d_2$, then $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_1)$ implies $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_2)$, $\forall e \in \mathcal{E}$. Since $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_1)$, we have

$$\forall t \in \mathcal{T}, \xi_{\mathcal{M}}(t) \notin A(t) \oplus d_1, \xi_{\mathcal{M}}(t) \in R(t) \ominus d_1$$

Since $d_1 > d_2$, the above equation implies that

$$\forall t \in \mathcal{T}, \xi_{\mathcal{M}}(t) \notin A(t) \oplus d_2, \xi_{\mathcal{M}}(t) \in R(t) \ominus d_2.$$

Therefore, $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_2)$. The corollary now follows from noting that for all $0 < d < d^b$, $\exists\ e \in \mathcal{E}$, $\mathbf{u} \in \mathcal{U}_\Pi(e)$, $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \nrightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$. □