

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/323392476>

Verifying Controllers Against Adversarial Examples with Bayesian Optimization

Conference Paper · May 2018

CITATIONS

0

READS

34

5 authors, including:



Felix Berkenkamp

ETH Zurich

14 PUBLICATIONS 164 CITATIONS

[SEE PROFILE](#)



Shaz Qadeer

Microsoft

171 PUBLICATIONS 6,853 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Corral [View project](#)

Verifying Controllers Against Adversarial Examples with Bayesian Optimization

Shromona Ghosh¹, Felix Berkenkamp², Gireeja Ranade³, Shaz Qadeer³, Ashish Kapoor³

Abstract—Recent successes in reinforcement learning have led to the development of complex controllers for real-world robots. As these robots are deployed in safety-critical applications and interact with humans, it becomes critical to ensure safety in order to avoid causing harm. A first step in this direction is to test the controllers in simulation. To be able to do this, we need to capture what we mean by safety and then efficiently search the space of all behaviors to see if they are safe. In this paper, we present an active-testing framework based on Bayesian Optimization. We specify safety constraints using logic and exploit structure in the problem in order to test the system for adversarial counter examples that violate the safety specifications. These specifications are defined as complex boolean combinations of smooth functions on the trajectories and, unlike reward functions in reinforcement learning, are expressive and impose hard constraints on the system. In our framework, we exploit regularity assumptions on individual functions in form of a Gaussian Process (GP) prior. We combine these into a coherent optimization framework using problem structure. The resulting algorithm is able to provably verify complex safety specifications or alternatively find counter examples. Experimental results show that the proposed method is able to find adversarial examples quickly.

I. INTRODUCTION

In recent years, research in control theory and robotics has focused on developing efficient controllers for robots that operate in the real world. Controller synthesis techniques such as reinforcement learning, optimal control, and model predictive control have been used to synthesize complex policies. However, if there is a large amount of uncertainty about the real world environment that the system interacts with, the robustness of the synthesized controller becomes critical. This is particularly true in *safety-critical* systems, where the actions of an autonomous agent may affect human lives. This motivates us to provably verify the properties of controllers in simulation before deployment in the real world.

In this paper, we present an active machine learning framework that is able to verify black-box systems against, or alternatively find, adversarial counter examples to a given set of safety specifications. We test the controller safety under uncertainty that arises from stochastic environments and errors in modeling. In essence, we actively search for

The primary part of the research for this paper was done at Microsoft Research, Redmond while Shromona Ghosh and Felix Berkenkamp were interns.

¹Electrical Engineering and Computer Science Department, University of California, Berkeley, shromona.ghosh@berkeley.edu

²Department of Computer Science, ETH Zürich, Switzerland befelix@inf.ethz.ch

³Microsoft Research, Redmond {giranade, qadeer, akapoor}@microsoft.com

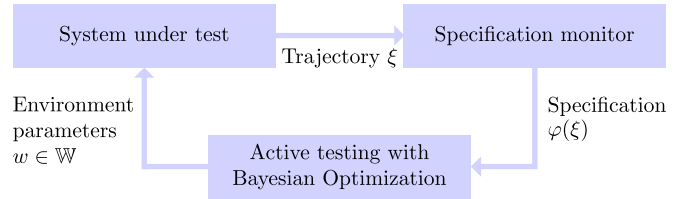


Fig. 1. Framework for active testing. We query the simulation of the system with environment parameters w to obtain trajectories ξ . We test these for safety violations in a specification monitor. The Bayesian optimization framework actively queries new parameters w that are promising candidates to find counter examples that violates the safety specification.

adversarial environments under which the controller could have to operate that lead failure modes in simulation.

Historically, designing robust controllers has been considered in control theory [1], [2]. A common issue with these techniques is that, although they consider uncertainty, they rely on simple linear models of the underlying system. This means that resulting controllers are often either overly conservative or violate safety constraints if they fail to capture nonlinear effects.

For nonlinear models with complex dynamics, reinforcement learning has been successful for synthesizing high fidelity controllers. Recently, algorithms based on reinforcement learning that can handle uncertainty have been proposed [3]–[5], where the performance is measured in expectation. A fundamental issue with learned controllers is that it is difficult to provide formal guarantees for safety in the presence of uncertainty. For example, a controller for an autonomous vehicle must consider human driver behaviors, pedestrian behaviors, traffic lights, uncertainty due to sensors, etc. Without formally verifying that these controllers are indeed safe, deploying them on the road could lead to loss of property or human lives.

Formal safety certificates, i.e., mathematical proofs for safety, have been considered in the formal methods community, where safety requirements are referred to as a *specification*. There, the goal is to verify that the behaviors of a particular model satisfies a specification ([6], [7]). Synthesizing controllers which satisfy a high level temporal specification have been studied in the context of motion planning [8] and for cyber-physical systems [9]. However, these techniques rely on simple model dynamics. For nonlinear systems, reachability algorithms based on level set methods have been used to approximate backward reachable sets for safety verification [10], [11]. However, these methods suffer from two major drawbacks: (1) the curse of dimensionality

of the state space, which limits them to low-dimensional systems; and (2) *a priori* knowledge of the system dynamics.

A dual, and often simpler, problem is *falsification*, which tests the system within a set of environment conditions for adversarial examples. Adversarial examples have recently been considered for neural networks [12]–[15], where the input is typically perturbed locally in order to find counterexamples. In [16], the authors compute adversarial perturbations for a trained neural network policy for a subset of white box and black-box systems. However, these local perturbations are often not meaningful for dynamic systems. Recently, [17], [18] have focused on testing of closed-loop safety critical systems with neural networks by finding “meaningful” perturbations.

Testing black-box systems in simulators is a well studied problem in the formal methods community [19]–[21]. The heart of research in black-box testing focuses on developing smarter search techniques which efficiently samples the uncertainty space. Indeed, in recent years, several sequential search algorithms based on heuristics such as Simulated Annealing [21], Tabu search [22], and CMA-ES [23] have been suggested. Although these algorithms sample the uncertainty space efficiently, they do not utilize any of the information gathered during previous simulations.

One active method that has been used recently for testing black-box systems is Bayesian Optimization (BO) [24], an optimization method that aims to find the global optimum of an *a priori* unknown function based on noisy evaluations. Typically, BO algorithms are based on Gaussian Process (GP [25]) models of the underlying function and certain algorithms provably converge close to the global optimum [26]. It has been used in robotics to, for example, safely optimize controller parameters of a quadrotor [27]. In the testing setting, BO has been used to actively find counter examples by treating the search problem as a minimization problem in [28] over adversarial control signals. However, the authors do not consider the structure of the problem and thereby violate the smoothness assumptions made by the GP model. As a result, their methods are slow to converge or may fail to find counterexamples.

In this paper, we provide a formal framework that uses BO to actively test and verify closed-loop black-box systems in simulation. We model the relation between environments and safety specification using GPs and use BO to predict the environment scenarios most likely to cause failures in our controllers. Unlike previous approaches, we exploit structure in the problem in order to provide a formal way to reason across multiple safety constraints in order to find counterexample. Hence, our approach is able to find counterexamples more quickly than previous approaches. Our main contributions are:

- An active learning framework for testing and verifying robotic controllers in simulation. Our framework can find adversarial examples for a synthesized controller independent of its structure or how it was synthesized.
- A common GP framework to model logical safety specifications along with theoretical analysis on when

a system is verified.

II. PROBLEM STATEMENT

We address the problem of testing complex black-box closed-loop robotic systems in simulation. We assume that we have access to a simulation of the robot that includes the control strategy, i.e., the closed-loop system. The simulator is parameterized by a set of parameters $\mathbf{w} \in \mathbb{W}$, which model all sources of uncertainty. For example, they can represent environment effects such as weather, non-deterministic components such as other agents interacting with the simulator, or uncertain parameters of the physical system, e.g., friction.

The goal is to test whether the system remains safe for all possible sources of uncertainty in \mathbb{W} . We specify these safety constraints on finite-length trajectories of the system that can be obtained by simulating the robot for a given set of environment parameters $\mathbf{w} \in \mathbb{W}$. Safety constraints on these trajectories are specified using logic. We explain this in detail in Sec. III-A, but the result is a specification φ that can, in general, be written as a requirement $\varphi(\mathbf{w}) > 0, \forall \mathbf{w} \in \mathbb{W}$. For example, φ can encode state or input constraints that have to be satisfied over time.

We want to test whether there exists an adversarial example $\mathbf{w} \in \mathbb{W}$ for which the specification is violated, i.e., $\varphi(\mathbf{w}) < 0$. Typically, adversarial examples are found by randomly sampling the environment and simulating the behaviors. However, this approach does not provide any guarantees and does not allow us to conclude that no adversarial example exist if none are found in our samples. Moreover, since high-fidelity simulations can often be very expensive, we want to minimize the number of simulations that we have to carry out in order to find a counterexample.

We propose an active learning framework for testing, where we utilize the results from previous simulation runs to make more informed decisions about which environment to simulate next. In particular, we pose the search problem for a counterexample as an optimization problem,

$$\underset{\mathbf{w} \in \mathbb{W}}{\operatorname{argmin}} \varphi(\mathbf{w}), \quad (1)$$

where we want to minimize the number of queries \mathbf{w} until a counterexample is found or we can verify that no counterexample exists. The main challenge is that the functional dependence $\varphi(\cdot)$ between parameters in \mathbb{W} and the specification is unknown *a priori*, since we treat the simulator as a black-box. Solving this problem is difficult in general, but we can exploit regularity properties of $\varphi(\mathbf{w})$. In particular, in the following we use GP to model the specification and use the model to pick parameters that are likely to be counterexamples.

III. BACKGROUND

In this section, we introduce an overview of formal safety specifications and Gaussian processes, which we use in Sec. IV to verify the closed-loop black-box system.

A. Safety Specification

In the formal methods community, complex safety requirements are expressed using automata [29] and temporal logic [30], [31]. These allow us to specify complex constraints, which can also have temporal dependence.

Example 1. A safety constraint for a quadcopter might be that the quadcopter cannot fly at an altitude h greater than 3m when the battery level b is below 30%.”

In logic, we can express this as “ $b < 0.3$ *implies*(\rightarrow) $h < 3$ ”, which in words says if the battery level is less than 30% the quadcopter is flying at a height less than 3m.

Importantly, these kind specifications make no assumptions about the underlying system themselves. They just state requirements that must hold for all simulations in \mathbb{W} . Formally, a logic specification is a function that tests properties of a particular trajectory. However, we will continue to write $\varphi(\mathbf{w})$ to denote the specification that tests trajectories generated by the simulator with parameters \mathbf{w} .

A specification φ consists of multiple individual constraints, called predicates, which form the basic building blocks of the logic. These predicates can be combined using a syntax or grammar of logical operations:

$$\varphi := \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \varphi \vee \psi. \quad (2)$$

where $\mu : \Xi \rightarrow \mathbb{R}$ is a predicate, and is assumed to be a smooth and continuous function of a trajectory $\xi \in \Xi$. The constraint $\mu > 0$ forms the basic building block of the overall system specification φ . We say a predicate is satisfied if $\mu(\xi)$ is greater than 0 or falsified otherwise. The operations \neg, \wedge, \vee represent *negation, conjunction(and) and disjunction(or)*, respectively. These basic operations can be combined to define complex boolean formula such as implication, \rightarrow , and if-and-only-if, \leftrightarrow using the rules

$$\varphi \rightarrow \psi := \neg\varphi \vee \psi, \text{ and } \varphi \leftrightarrow \psi := (\neg\varphi \wedge \neg\psi) \vee (\varphi \wedge \psi). \quad (3)$$

Since μ is a real valued function, we can convert these boolean logic statements into an equivalent equation with continuous output, which defines the *quantitative semantics*,

$$\begin{aligned} \mu(\xi) &:= \mu(\xi), & (\varphi \wedge \psi)(\xi) &:= \min(\varphi(\xi), \psi(\xi)), \\ \neg\mu(\xi) &:= -\mu(\xi), & (\varphi \vee \psi)(\xi) &:= \max(\varphi(\xi), \psi(\xi)). \end{aligned} \quad (4)$$

This allows us to confirm that a logic statement φ holds true for all trajectories generated by simulators \mathbb{W} , by confirming that the function $\varphi(\mathbf{w})$ takes positive values for all $\mathbf{w} \in \mathbb{W}$.

In the quantitative semantics (4), the satisfaction of a requirement is no longer a yes or no answer, but can be quantified by a real number. The nature of this quantification is similar to that of a reward function, where lower values indicate a larger safety violation. This allows us to introduce a ranking among failures: $\varphi(\mathbf{w}_1) < \varphi(\mathbf{w}_2)$ implies \mathbf{w}_1 is a more “dangerous” failure case than \mathbf{w}_2 . To guarantee safety, we have to take a pessimistic outlook, and denote $\varphi(\mathbf{w}) \leq 0$ as a violation and $\varphi(\mathbf{w}) > 0$ as satisfaction of the specification φ .

Example 2. Let us look at the specification in Example 1, $\varphi := (b < 0.3) \rightarrow (h < 3)$. Applying the re-write rule (3), this can be written as $\neg(b < 0.3) \vee (h < 3)$. Applying the quantitative semantics (4), we get $\varphi = \max(b > 0.3, h < 3)$, which consists of two predicates, $\mu_1 = b - 0.3$ and $\mu_2 = 3 - h$. Intuitively, this means $\varphi > 0$, i.e., the specification is satisfied, if the battery is greater than 30% or if the quadcopter flies at an altitude less than 3m.

B. Gaussian Process

For general black-box systems, the dependence of the specification $\varphi(\cdot)$ on the parameters $\mathbf{w} \in \mathbb{W}$ is unknown *a priori*. We use a GP to approximate each predicate $\mu(\cdot)$ in the domain \mathbb{W} . We detail the modeling of $\varphi(\cdot)$ in Sec. IV. The following introduction about GPs is based on [25].

GPs are non-parametric regression method from machine learning, where the goal is to find an approximation of the nonlinear function $\mu : \mathbb{W} \rightarrow \mathbb{R}$ from an environment $\mathbf{w} \in \mathbb{W}$ to the function value μ . This is done by considering the function values $\mu(\mathbf{w})$ to be random variables, such that any finite number of them have a joint Gaussian distribution.

The Bayesian, non-parametric regression is based on a prior mean function and the kernel function $k(\mathbf{w}, \mathbf{w}')$, which defines the covariance between the function values $\mu(\mathbf{w}), \mu(\mathbf{w}')$ at two points $\mathbf{w}, \mathbf{w}' \in \mathbb{W}$. We set the prior mean to zero, since we do not have any knowledge about the system. The choice of kernel function is problem-dependent and encodes assumptions about the unknown function.

We can obtain the posterior distribution of a function value $\mu(\mathbf{w})$ at an arbitrary state $\mathbf{w} \in \mathbb{W}$ by conditioning the GP distribution of μ on a set of n past measurements, $\mathbf{y}_n = (\hat{\mu}(\mathbf{w}_1), \dots, \hat{\mu}(\mathbf{w}_n))$ at environment scenarios $W_n = \{\mathbf{w}_1, \dots, \mathbf{w}_n\}$, where $\hat{\mu}(\mathbf{w}) = \mu(\mathbf{w}) + \omega$ and $\omega \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise. The posterior over $\mu(\mathbf{w})$ is a GP distribution again, with mean $m_n(\mathbf{w})$, covariance $k_n(\mathbf{w}, \mathbf{w})$, and variance $\sigma_n(\mathbf{w})$:

$$\begin{aligned} m_n(\mathbf{w}) &= \mathbf{k}_n(\mathbf{w})(\mathbf{K}_n + \mathbf{I}_n\sigma^2)^{-1}\mathbf{y}_n, \\ k_n(\mathbf{w}, \mathbf{w}') &= k(\mathbf{w}, \mathbf{w}') - \mathbf{k}_n(\mathbf{w})(\mathbf{K}_n + \mathbf{I}_n\sigma^2)^{-1}\mathbf{k}_n^T(\mathbf{w}'), \\ \sigma_n^2(\mathbf{w}) &= k_n(\mathbf{w}, \mathbf{w}'), \end{aligned} \quad (5)$$

where the vector $\mathbf{k}_n(\mathbf{w}) = [k(\mathbf{w}, \mathbf{w}_1), \dots, k(\mathbf{w}, \mathbf{w}_n)]$ contains the covariances between the new environment, \mathbf{w} , and the environment scenarios in W_n , the kernel matrix $\mathbf{K}_n \in \mathbb{R}^{n \times n}$ has entries $[\mathbf{K}_n](i, j) = k(\mathbf{w}_i, \mathbf{w}_j)$, with $i, j \in \{1, \dots, n\}$, and $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ is the identity matrix.

C. Bayesian Optimization (BO)

In the following we use BO in order to find the minimum of the unknown function φ , which we construct using the GP models on μ in Sec. IV. BO uses a GP model to query parameters that are informative about the minimum of the function. In particular, the GP-LCB algorithm from [26] uses the GP prediction and associated uncertainty in (5) to trade off exploration and exploitation by, at iteration n ,

selecting an environment according to

$$\mathbf{w}_n = \underset{\mathbf{w} \in \mathbb{W}}{\operatorname{argmin}} m_{n-1}(\mathbf{w}) - \beta_n^{1/2} \sigma_{n-1}(\mathbf{w}), \quad (6)$$

where β_n determines the confidence interval. We provide an appropriate choice for β_n in Theorem 1.

At each iteration, (6) selects parameters for which the lower confidence bound of the GP is minimal. Repeatedly evaluating the true function φ at samples given by (6) improves the GP model and decreases uncertainty at candidate locations for the minimum, such that the global minimum is found eventually [26].

IV. ACTIVE TESTING FOR COUNTEREXAMPLES

In this section, we show how to model specifications φ in (1) using GPs without violating smoothness assumptions and use this to find adversarial counterexamples.

In order to use BO to optimize (1), we need to construct reliable confidence intervals on φ . However, if we were to model φ as a GP with commonly-used kernels, it would need it to be a smooth function of \mathbf{w} . Even though the predicates, μ , are typically smooth functions of the trajectories, and hence smooth in \mathbf{w} , conjunction and disjunction (min and max) in (4) are non-smooth operators that render φ to become non-smooth as well. Instead, we exploit the structure of the specification φ and decompose φ into a parse tree, where the leaf nodes are the predicates.

Definition 1 (Parse Tree \mathcal{T}). *Given a specification formula φ , the corresponding parse tree, \mathcal{T} , has leaf nodes that correspond to function predicates, while other nodes are max (disjunctions) and min (conjunctions).*

A parse tree is an equivalent graphical representation of φ . For example, consider the specification

$$\varphi := (\mu_1 \vee \mu_2) \rightarrow (\mu_3 \vee \mu_4) = (\neg\mu_1 \wedge \neg\mu_2) \vee (\mu_3 \vee \mu_4), \quad (7)$$

where the second equality follows from De-Morgan's law. We can obtain an equivalent function $\varphi(\mathbf{w})$ with (4),

$$\varphi(\mathbf{w}) = \max(\min(-\mu_1(\mathbf{w}), -\mu_2(\mathbf{w})), \max(\mu_3(\mathbf{w}), \mu_4(\mathbf{w}))). \quad (8)$$

The parse tree, \mathcal{T} , for φ in (8) is shown in Fig. 2. We can use the parse tree to decompose any complex specification into min and max functions of the individual predicates; that is, $\varphi(\mathbf{w}) = \mathcal{T}(\mu_1(\mathbf{w}), \dots, \mu_q(\mathbf{w}))$.

We now model each predicate $\mu_i(w)$ in the parse tree \mathcal{T} of φ with a GP and combine them with the parse tree to obtain confidence intervals on the overall specification $\varphi(\mathbf{w})$ for BO. GP-LCB as expressed in (6) can be used to search for the minimum for a single GP. A key insight to extending (6) across multiple GPs, is that the minimum of (1) is, with high probability, lower bounded by the lower-confidence interval of one of the GPs used to model the predicates of φ . This is because, the max and min operators do not change the value of the predicates, but only make a choice between them. As a consequence, we can model the smooth parts of φ , i.e., the

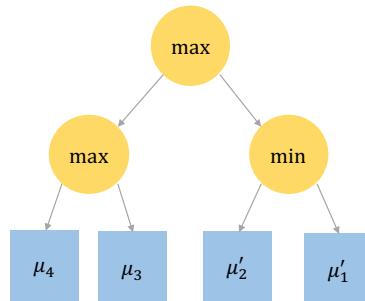


Fig. 2. Equivalent parse tree \mathcal{T} for φ in (7) to the function (8). We replace the predicates μ_i with their corresponding pessimistic GP predictions to obtain a lower bound on $\varphi(\mathbf{w})$.

predicates, using GPs and then consider the non-smoothness through the parse tree.

For each predicate μ_i in the parse tree \mathcal{T} of φ , we construct a lower confidence bound $l_i = m_{n-1}^i(\mathbf{w}) - \beta_n^{1/2} \sigma_{n-1}^i(\mathbf{w})$, where m^i, σ^i are the mean and standard deviation of the GP corresponding to μ_i . From this, we can construct a lower-confidence interval on φ as $\mathcal{T}(l_1(\mathbf{w}), \dots, l_q(\mathbf{w}))$, where we replace the i th leaf node μ_i of the parse tree with the pessimistic prediction l_i of the corresponding GP. Similar to (6), the corresponding acquisition function for BO uses this lower bound to select the next evaluation point,

$$\mathbf{w}_n = \underset{\mathbf{w} \in \mathbb{W}}{\operatorname{argmin}} \mathcal{T}(l_1(\mathbf{w}), \dots, l_q(\mathbf{w})). \quad (9)$$

Intuitively, the next environment selected to simulate is the one that minimizes the worst-case predictions on φ . Effectively, we propagate the confidence intervals associated with the GP for each predicates through the parse tree \mathcal{T} in order to obtain predictions about φ directly. Note, that (9) does not return an environment sample that minimizes the satisfaction of all the predicates, it only minimizes the lower bound on φ .

Algorithm 1 describes our active testing procedure. The algorithm proceeds by first computing the parse tree \mathcal{T} from the specification, φ . At each iteration n of BO, we select new environment parameters \mathbf{w}_n according to (9). We then simulate the system with parameters \mathbf{w}_n and evaluate each predicate μ_i on the simulated trajectories. Lastly, we update each GP with the corresponding measurement of μ_i . The algorithm either returns a counterexample that minimizes (1); or when $\mathcal{T}(l_1(\mathbf{w}), \dots, l_q(\mathbf{w}))$ is greater than zero, and we can conclude that the system has been verified.

A. Theoretical Results

We can transfer theoretical convergence results for GP-LCB [26] to the setting of Algorithm 1. To do this, we need to make structural assumptions about the predicates. In particular, we assume that they have bounded norm in the Reproducing Kernel Hilbert Space (RKHS, [32]) that corresponds to the GP's kernel. These are well-behaved functions of the form $\mu_i(\mathbf{w}) = \sum_{j=0} \alpha_j k_i(\mathbf{w}, \mathbf{w}_j)$ with representer points \mathbf{w}_j and weights α that decay sufficiently quickly. We leverage theoretical results from [33] and [27]

Algorithm 1 Active Testing with Bayesian Optimization

- 1: **procedure** ACTIVETESTING($\varphi, \mathbb{W}, \beta, GPs$)
 - 2: Build parse tree \mathcal{T} based on specification φ
 - 3: **for** $n = 0, \dots$ **do** ▷ Until budget or convergence
 - 4: $l_i(\mathbf{w}) = \mu_i(\mathbf{w}) - \beta_n^{1/2} \sigma_i(\mathbf{w}), i = 1, \dots, q$
 - 5: $w_n = \operatorname{argmin}_{w \in \mathbb{W}} \mathcal{T}(l_1(\mathbf{w}), \dots, l_q(\mathbf{w}))$
 - 6: Update each GP model of the predicates with measurements $(\mathbf{w}_n, \mu_i(\mathbf{w}_n))$.
 - 7: return $\min_i \varphi(w_i)$, the worst result.
-

that allow us to build reliable confidence intervals using the GP models from Sec. III-B. We have the following result.

Theorem 1. *Assume that each predicate μ_i has RKHS norm bounded by B_i and that the measurement noise is σ -sub-Gaussian. Select $\delta \in (0, 1)$, \mathbf{w}_n according to (9), and let $\beta_n^{1/2} = \sum_i B_i + 4\sigma\sqrt{1 + \ln(1/\delta) + \sum_i I(\mathbf{y}_{n-1}^i; \mu_i)}$. If $\mathcal{T}(l_1(\mathbf{w}_n), \dots, l_q(\mathbf{w}_n)) > 0$, then with probability at least $1 - \delta$ we have that $\min_{\mathbf{w} \in \mathbb{W}} \varphi(\mathbf{w}) > 0$ and the system has been verified against all environments in \mathbb{W} .*

Here $I(\mathbf{y}_{n-1}^i; \mu_i)$ is the mutual information between \mathbf{y}_{n-1}^i , the $n - 1$ noisy measurements of μ_i , and the GP prior of μ_i . This function was shown to be sublinear in n for many commonly-used kernels in [26], see the appendix for more details. Theorem 1 states that we can verify the system against adversarial examples with high probability, by checking whether the worst-case lower-confidence bound is greater than zero. We provide additional theoretical results about the existence of a finite n such that the system can be verified up to ϵ accuracy in the appendix.

V. EVALUATION

In this section, we evaluate our method on several challenging test cases. A Python implementation of our framework and the following experiments can be found at https://github.com/shromonag/adversarial_testing.git

In order to use Algorithm 1, we have to solve the optimization problem (9). In practice, different optimization techniques have been proposed to find the global minimum of the function. One popular algorithm is DIRECT [34], a gradient-free optimization method. An alternative is to use gradient-based methods together with random-restarts. Particularly, we sample a large number of potential environment scenarios at random from \mathbb{W} , and run separate optimization routines to minimize (9) from these.

Another challenge is that the dimensionality of the optimization problem can often be very large. However, methods that allow for more efficient computation do exist. These methods reduce the effective size of the input space and thereby make the optimization problem more tractable. One possibility is to use random embedding to reduce the input dimension as done in Random Embedding Bayesian Optimization (REMBO [35]). We can then model the GP in this smaller input dimension and carry out BO in the lower dimension input space.

A. Modeling smooth functions vs non-smooth function

In the following, we show the effectiveness of modeling smooth functions by GPs and considering the non-smooth operations in the BO search as opposed to modeling the non-smooth function by a single GP.

Consider the following, illustrative optimization problem,

$$w^* = \operatorname{argmin}_{w \in (0, 10)} \max(\sin(w) + 0.65, \cos(w) + 0.65) \quad (10)$$

We consider two modeling scenarios, one where we model $\max(\sin(w), \cos(w))$ as a single GP, and another where we model $\sin(w)$ by one GP and $\cos(w)$ by another. We initialize the GP models for $\sin(w)$, $\cos(w)$ and $\max(\sin(w), \cos(w))$ with 5 samples chosen in random. We then use BO to find w^* . We were able to model smooth functions like $\sin(w)$ and $\cos(w)$ with GPs, even with fewer samples. At each iteration of BO, we computed the next sample by solving for the $w \in (0, 10)$ which minimized the maximum across the two GPs. This quickly stabilizes to the true w^* (Fig. 3c). When we model $\max(\sin(w), \cos(w))$ using a GP, in Fig. 3b, the initial 5 samples were not able to model it well. In fact, the original function in orange is not contained within the uncertainty bounds of the GP. Hence, in each iteration of BO, where we chose $w \in (0, 10)$ which minimized this function, we were never able to converge w^* . It is not surprising to see that, given these models, BO does not always converge when we model non-smooth functions such as in (10).

To support our claim, we repeat this experiment 15 times with different initial samples. In each experiment we run BO for 50 iterations. When modeling $\sin(w)$ and $\cos(w)$ as separate GPs, BO stabilized to w^* in about 5 iterations in all 15 experiments. However, when modeling $\max(\sin(w), \cos(w))$ as a single GP, it takes over 35 iterations to converge and in 5 out of the 15 cases, it did not converge to w^* . We show these two different behaviors in Fig. 4.

B. Collision Avoidance with High Dimensional Uncertainty

Consider an autonomous car that travels on a straight road with an obstacle at x_{obs} . We require that the car can come to a stop before colliding with an obstacle. The car has two states; location, x , and velocity, v ; and one control input acceleration; a . The dynamics of the car is given by,

$$\dot{x} = v, \quad \dot{v} = a. \quad (11)$$

Our safety specification for collision avoidance is given by, $\varphi = \min(x_{obs} - x(t))$, i.e., the minimum distance between the position of the car and the obstacle over a horizon of length 100. We assume that the car does not know where the obstacle is *a priori*, but receives locations of the obstacle through a sensor at each time instant, $x_s(t)$. The controller is a simple linear state feedback control, K , such that at time t , $a(t) = K \cdot [x(t) - x_s(t), v(t)]^T$.

We assume that the car initially starts at location $x_{init} = 0$, with a velocity $v_{init} = 3$ m/s. Let the obstacle be at $x_{obs} = 5$, which is not known by the car. Instead, it receives sensor readings for the location of the obstacle such that

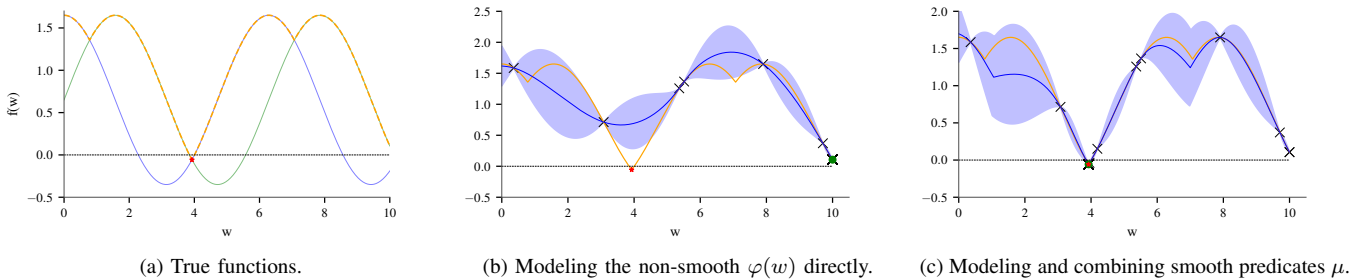
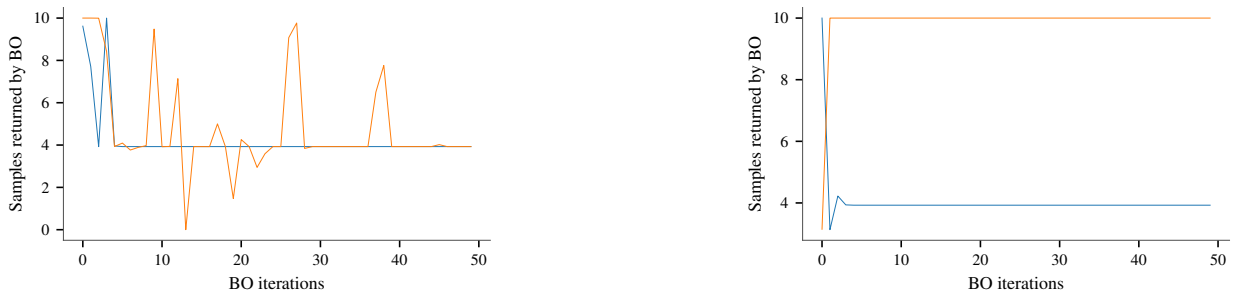


Fig. 3. The dashed orange line in Fig. 3a represents the true, non-smooth optimization function in (10) while the green and blue line represent $\sin(w)$ and $\cos(w)$ respectively. Modeling this function directly as a GP leads to model errors Fig. 3b, where the 95% confidence interval of the GP (blue shaded) with mean estimate (in blue line) does not capture the true function $\varphi(w)$ in orange. In fact, the minimum (red star) is not contained within the shaded region, causing the optimization to diverge. BO converges to the green dot, where $\varphi(w) > 0$ which is not a counterexample. Instead, modeling the two predicates individually and combining them with the parse tree, leads to the model in Fig. 3c. Here, the true function is completely captured in the confidence interval. As a consequence, BO converges to the global minimum (the red star and green dot converge).



(a) Modeling as separate GPs take around 5 iterations to stabilize to w^* (in blue), while modeling as a single GP takes around 45 iteration to stabilize to w^* (in orange)

(b) Modeling as separate GPs take around 5 iterations to stabilize to w^* (in blue), while modeling as a single GP does not stabilize to w^* (in orange)

Fig. 4. The orange and blue lines in Fig. 4a and Fig. 4b show the evolution of samples returned over the BO iterations when (10) is modeled as a single GP and multiple GPs respectively for two different initialization. We see that when modeling as a single GP, it takes longer to stabilize to w^* and in some cases (Fig. 4b) does not stabilize to w^* .

$x_s = [4.5, 5.5]$. If φ is negative, then $x(t) > x_{obs}$ for some t which signifies collision. Moreover, we constrain the acceleration to lie in $a \in [-3, 3]$.

The domain of our uncertainty is $\mathbb{W} = [4.5, 5.5]^{100}$, i.e., the sensor readings x_s over the horizon $H = 100$. We compare across three experimental setups, first, we model the GP in the original space of \mathbb{W} i.e., with 100 inputs; second, we model the GP in a lower dimension input space as described in the preamble of this section; and third, we randomly sample inputs and test them. We run BO for 250 iterations on the GPs, and consider 250 random samples for the random testing. We repeat this experiment 10 times and show our results in Fig. 5. The green and blue bar in Fig. 5 show the average number of counterexamples returned running BO on the GP defined over the original input space and in the low dimension input space. In general, active testing in the high-dimensional input space gives the best results, which deteriorates with an increase in compression of the input space. Random testing, shown in red performs the worst. This is not surprising as, (1) 250 samples is not sufficient to cover an input space of 100 dimensions uniformly; and (2) the samples are all independent of each other. Moreover, in the uncompressed input case, the specification evaluated at the worst counterexample, $\varphi(w^*)$, has a mean and standard deviation of -0.0138 and 0.004 as compared to -0.0067 and 0.0011 for random sampling.

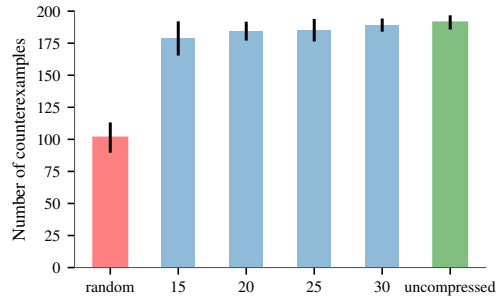


Fig. 5. The red, blue and green bars shows the average number of counterexamples found using random sampling; applying BO on the reduced input space and original input space respectively for the example in Sec. V-B. The black lines show the standard deviation across the experiments.

C. OpenAI Gym Environments

We interfaced our tool with environments from OpenAI gym [36] to test controllers from Open AI baselines [37]. For brevity, we refer the details of the environments to [38]. In both case studies, we introduce uncertainty around the parameters the controller has been trained for. The rationale behind this is that the parameters in a simulator are an estimate of the true values. This ensures that counterexamples found, can indeed occur in the real system.

1) *Reacher*: In the reacher environment, we have a 2D robot trying to reach a target. For this environment we

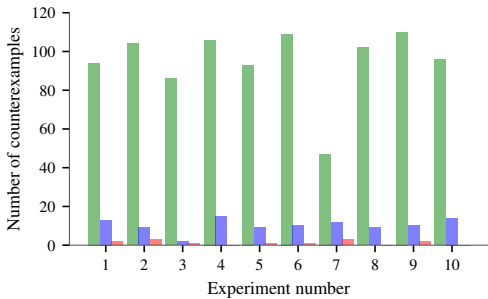


Fig. 6. The green, blue and red bars show the number of counterexamples generated when modeling μ_1, μ_2 as separate GPs; modeling φ as a single GP and random testing respectively for the reacher example (Sec. V-C.1). Our modeling paradigm, finds more counterexamples compared to the other two methods.

have six sources of uncertainty: two for the goal position, $(x_{goal}, y_{goal}) \in [-0.2, 0.2]^2$, two for state perturbations $(\delta_x, \delta_y) \in [-0.1, 0.1]^2$ and two for velocity perturbations $(\delta_{v_x}, \delta_{v_y}) \in [-0.005, 0.005]^2$. The state of the reacher is tuple with the current location, $\mathbf{x} = (x, y)$, velocity $\mathbf{v} = (v_x, v_y)$, and rotation, θ . A trajectory of the system, ξ , is a sequence of states over time, i.e., $\xi = (\mathbf{x}(t), \mathbf{v}(t), \theta(t)), t = 0, 1, 2, \dots$. Our uncertainty space is, $\mathbb{W} = [-0.2, 0.2]^2 \times [-0.1, 0.1]^2 \times [-0.005, 0.005]^2$. Given an instance of $w \in \mathbb{W}$, the trajectory, ξ , of the system is uniquely defined.

We trained a controller using the Proximal Policy Optimization (PPO) [39] implementation available at Open AI baselines. We determine a trajectory to be safe if either the reacher reaches the goal, or if it does not rotate unnecessarily. This can be captured as $\varphi = \mu_1 \vee \mu_2$, where, $\mu_1(w)$ is the minimum distance between the trajectory and the goal position, and μ_2 is total rotation accumulated over the trajectory; and its continuous variant, $\varphi = \max(\mu_1, \mu_2)$.

Using our modeling approach, we model this using two GPs, one for μ_1 and another for μ_2 . We compare this to modeling φ as a single GP and random sampling. We run 200 BO iterations and consider 200 random samples for random testing. We repeat this experiment 10 times. In Fig. 6, we plot the number of counterexamples found by each of the three methods over 10 runs of the experiment. Modeling the predicates by separate GPs and applying BO across them (shown in green) consistently performs better than applying BO on a single GP modeling φ (shown in blue) and random testing (shown in red). We see that random testing performs very poorly, in some cases (experiment runs 4, 8, 10) finds no counterexamples.

By modeling the predicates separately, the specification evaluated at the worst counterexample, $\varphi(w^*)$, has a mean and standard deviation of -0.1283 and 0.0006 as compared to -0.1212 and 0.0042 when considering a single GP. This suggests, that using our modeling paradigm BO converges (since the standard deviation is small) to a more falsifying counterexample (since the mean is smaller).

2) *Mountain Car Environment*: The mountain car environment in OpenAI gym, is a car on a one-dimensional track, positioned between two mountains. The goal is to drive the car up the mountain on the right. The envi-

ronment comes with one source of uncertainty, the initial state $x_{init} \in [-0.6, -0.4]$. We introduced four other sources of uncertainty, for the initial velocity, $v_{init} \in [-0.025, 0.025]$; goal location, $x_{goal} \in [0.4, 0.6]$; maximum speed, $v_{max} \in [0.55, 0.75]$ and maximum power magnitude, $p_{max} \in [0.0005, 0.0025]$. The state of the mountain car is a tuple with the current location, x , and velocity, v . A trajectory of the system, ξ , is a sequence of states over time, i.e., $\xi = (x(t), v(t)), t = 0, 1, 2, \dots$. Our uncertainty space is given by, $\mathbb{W} = [-0.6, -0.4] \times [-0.025, 0.025] \times [0.4, 0.6] \times [0.55, 0.75] \times [0.0005, 0.0025]$. Given an instance of $w \in \mathbb{W}$, the trajectory, ξ , of the system is uniquely defined.

We trained two controllers one using PPO and another using an actor critic method (DDPG) for continuous Deep Q-learning [40]. We determine a trajectory to be safe, if it reaches the goal quickly or if does not deviate too much from its initial location and always maintains its velocity in some bound. Our safety specification can be written as $\varphi = \mu_1 \vee (\mu_2 \wedge \mu_3)$, where, $\mu_1(w)$ is time taken to reach the goal, μ_2 is the deviation from the initial location and μ_3 is the deviation from the velocity bound; and its continuous variant of $\varphi = \max(\mu_1, \min(\mu_2, \mu_3))$. We model φ , by modeling each predicate, μ , by a GP. We compare this to modeling φ with a single GP and random sampling. We run 200 BO iterations for the GPs and consider 200 random samples for random testing. We repeat this experiment 10 times. We show our results in Fig. 7, where we plot the number of counterexamples found by each of the three methods over 10 runs of the experiment for each controller. Fig. 7 demonstrates the strength of our approach. The number of counterexamples found by our method (in green bar) is much higher compared to random sampling (in red) and modeling φ as a single GP (in blue). In Fig. 7a the blue bars are smaller than even the ones in red, suggesting random sampling performs better than applying BO on the GP modeling φ . This is because the GP is not able to model φ , and is so far away from the true model, that the sample returned by the BO is worse than if were to sample randomly.

This is further highlighted by the value of the specification at worst counterexample, $\varphi(w^*)$. The mean and standard deviation for $\varphi(w^*)$ over the 10 experiment runs is -0.5435 and 0.028 for our method, -0.3902 and 0.0621 when φ is modeled as a single GP; and -0.04379 and 0.0596 for random sampling. A similar but less drastic result holds in the case of the controller trained with DDPG.

VI. CONCLUSION

We presented an *active testing* framework that uses Bayesian Optimization to test and verify closed-loop robotic systems in simulation. Our framework handles complex logic specifications and models them efficiently using Gaussian Processes in order to find adversarial examples faster. We showed the effectiveness of our framework on controllers designed on OpenAI gym environments. As future work, we would like to extend this framework to test more complex robotic systems and find regions in the environment parameter space where the closed-loop control is expected to fail.

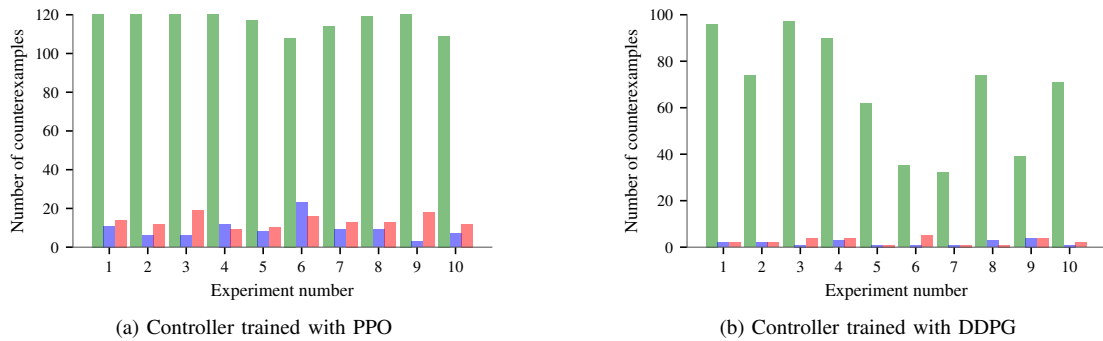


Fig. 7. The green, blue and red bars show the number of counterexamples generated when modeling μ_1, μ_2 as separate GPs, modeling φ as a GP and random testing respectively for the mountain car example (Sec. V-C.2). While our modeling paradigm, finds orders of magnitude more counterexample compared to the other two methods, we notice that modeling φ as a single GP performs much worse than random sampling for the controller trained with PPO Fig. 7a and comparable for the controller trained with DDPG Fig. 7b.

ACKNOWLEDGMENTS

Research reported in this paper was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-17-2-0196¹ and was accomplished under Cooperative Agreement Number W911NF-17-2-0196; and in part by Toyota under the iCyPhy center.

REFERENCES

- [1] S. Sastry and M. Bodson, *Adaptive Control: Stability, Convergence, and Robustness*. Prentice-Hall, Inc., 1989.
- [2] R. F. Stengel, *Stochastic Optimal Control: Theory and Application*. John Wiley & Sons, Inc., 1986.
- [3] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, “Uncertainty-aware reinforcement learning for collision avoidance,” *CoRR*, vol. abs/1702.01182, 2017.
- [4] Y. Niv, D. Joel, I. Meilijson, and E. Ruppín, “Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors,” 2002.
- [5] P. Poupart and N. Vlassis, “Model-based bayesian reinforcement learning in partially observable domains,” in *Proc Int. Symp. on Artificial Intelligence and Mathematics*, 2008, pp. 1–2.
- [6] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.
- [7] I. Mitchell and C. J. Tomlin, “Level set methods for computation in hybrid systems,” in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2000, pp. 310–323.
- [8] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, “Sampling-based motion planning with temporal goals,” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2689–2696.
- [9] V. Raman, A. Donz , M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, “Model predictive control with signal temporal logic specifications,” in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 81–87.
- [10] I. Mitchell, A. Bayen, and C. J. Tomlin, “Computing reachable sets for continuous dynamic games using level set methods,” *Submitted January*, 2004.
- [11] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” in *Proc. of Neural Information Processing Systems (NIPS)*, 2017.
- [12] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [13] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against deep learning systems using adversarial examples,” *arXiv preprint arXiv:1602.02697*, 2016.
- [14] V. Behzadan and A. Munir, “Vulnerability of deep reinforcement learning to policy induction attacks,” *CoRR*, vol. abs/1701.04143, 2017.
- [15] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.
- [16] S. H. Huang, N. Papernot, I. J. Goodfellow, Y. Duan, and P. Abbeel, “Adversarial attacks on neural network policies,” *CoRR*, vol. abs/1702.02284, 2017.
- [17] T. Dreossi, A. Donz , and S. A. Seshia, “Compositional falsification of cyber-physical systems with machine learning components,” in *NASA Formal Methods Symposium*. Springer, 2017, pp. 357–372.
- [18] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP ’17, 2017.
- [19] A. Donz , “Breach, a toolbox for verification and parameter synthesis of hybrid systems,” in *CAV*. Springer, 2010, pp. 167–170.
- [20] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, “C2E2: A verification tool for stateflow models,” in *Tools and Algorithms for the Construction and Analysis of Systems*, C. Baier and C. Tinelli, Eds., 2015.
- [21] Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, “S-taliro: A tool for temporal logic falsification for hybrid systems,” in *TACAS*, vol. 6605. Springer, 2011, pp. 254–257.
- [22] J. Deshmukh, X. Jin, J. Kapinski, and O. Maler, “Stochastic local search for falsification of hybrid systems,” in *ATVA*. Springer, 2015.
- [23] N. Hansen, “The CMA evolution strategy: A tutorial,” *arXiv preprint arXiv:1604.00772*, 2016.
- [24] J. Mockus, *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012, vol. 37.
- [25] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press.
- [26] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: no regret and experimental design,” *IEEE Transactions on Information Theory*, vol. 58, 2012.
- [27] F. Berkenkamp, A. Krause, and Angela P. Schoellig, “Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics,” *arXiv*, 2016.
- [28] J. V. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu, “Testing cyber-physical systems through bayesian optimization,” *Transactions on Embedded Computing Systems*, 2017.
- [29] R. Alur and D. L. Dill, “A theory of timed automata,” *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.
- [30] A. Pnueli, “The temporal logic of programs,” in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, ser. SFCS ’77, 1977, pp. 46–57.
- [31] O. Maler and D. Nickovic, “Monitoring temporal properties of continuous signals,” in *FORMATS/FTRTFT*. Springer, 2004.
- [32] I. Steinwart and A. Christmann, *Support vector machines*. Springer Science & Business Media, 2008.

¹The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

- [33] S. R. Chowdhury and A. Gopalan, "On kernelized multi-armed bandits," in *ICML*, 2017, pp. 844–853.
- [34] D. E. Finkel, "Direct optimization algorithm user guide," 2003.
- [35] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, N. De Freitas *et al.*, "Bayesian optimization in high dimensions via random embeddings," in *IJCAI*, 2013, pp. 1778–1784.
- [36] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.
- [37] "Open AI Baselines," <https://github.com/openai/baselines>.
- [38] "Open AI Gym Environments," <https://gym.openai.com/envs>.
- [39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.
- [40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

APPENDIX

In this section, we prove the convergence of our algorithm under specified regularity assumptions on the underlying predicates. Consider the specification

$$\varphi(\mathbf{w}) = \mathcal{T}(\mu_1(\mathbf{w}), \dots, \mu_q(\mathbf{w})), \quad (12)$$

where q represents the number of predicates. Let the domain of the predicate indices be represented by, $\mathcal{I} = \{1, \dots, q\}$. The convergence proofs for classical Bayesian optimization in [26], [33] proceed by building reliable confidence intervals for the underlying function and then showing, that these confidence intervals concentrate quickly enough at the location of the optimum under the proposed evaluation strategy. For ease of exposition, we assume that measurements of each predicate μ_i are corrupted by the same measurement noise.

To leverage these proofs, we need to account for the fact that our GP model is composed of several individual predicates and that we obtain one measurement for each predicates at every iteration of the algorithm.

We start by defining a composite function $f: \mathbb{W} \times \mathcal{I} \rightarrow \mathbb{R}$, which returns the function values for the individual predicates indexed by i .

$$f(\mathbf{w}, i) = \mu_i(\mathbf{w}) \quad (13)$$

The function $f(\cdot, \cdot)$ is a single output function, which can be modeled with a single GP with a scalar output over the extended input space, $\mathbb{W} \times \mathcal{I}$. For example, if we assume that the predicates are independent of each other, the kernel function for f would look like,

$$k((\mathbf{w}, i), (\mathbf{w}', i')) = \begin{cases} k_i(\mathbf{w}, \mathbf{w}') & \text{if } i = i' \\ 0 & \text{otherwise} \end{cases}, \quad (14)$$

where k_i is the kernel function corresponding to the GP for the i -th predicate, μ_i . It is straightforward to include correlations between functions in this formulation too.

This reformulation allows us to build reliable confidence intervals on the underlying predicates, given regularity assumptions. In particular, we make the assumption that the function f has bounded norm in the Reproducing Kernel Hilbert Space (RKHS, [32]) corresponding to the same kernel $k(\cdot, \cdot)$ that is used for the GP on f .

Remark 1. Note, that this model is more general than the case where we assume that each predicate, μ_i , individually

has bounded RKHS norm B_i . In this case, the function, $f(\mathbf{w}, i)$ has RKHS norm with respect to the kernel in (14) bounded by $B = \sum_i^q B_i$.

Lemma 1. Assume that f has RKHS norm bounded by B and the measurements are corrupted by σ -sub-Gaussian noise. If $\beta_{n,q}^{1/2} = B + 4\sigma\sqrt{I(y_{q,(n-1)}; f) + 1 + \ln 1/\delta}$, then the following holds for all environment scenarios, $\mathbf{w} \in \mathbb{W}$, predicate indices, $i \in \mathcal{I}$, and iterations $n \geq 1$ jointly with probability at least $1 - \delta$,

$$|f(\mathbf{w}, i) - m_{q,(n-1)}^i(\mathbf{w}, i)| \leq \beta_{q,n}^{1/2} \sigma_{q,(n-1)}^i(\mathbf{w}, i) \quad (15)$$

Proof. This follows directly from [27], which extends the results from [33] and Lemma 5.1 from [26] to the case of multiple measurements. \square

The scaling factor for the confidence intervals, $\beta_{n,q}$, depends on the mutual information $I(\mathbf{y}_{q,(n-1)}; f)$ between the GP model of f and the q measurements of the individual predicates that we have obtained for each time step so far. It can easily be computed as

$$\begin{aligned} I(\mathbf{y}_{q,(n-1)}; f) &= \log\left(1 + \frac{1}{\sigma^2} \mathbf{K}_{q,(n-1)}\right), \\ &= \sum_{j=1}^{n-1} \sum_{i=1}^q \log\left(1 + \sigma_{j,q}^2(\mathbf{w}_j, i)/\sigma^2\right), \end{aligned} \quad (16)$$

where $\mathbf{K}_{q,(n-1)}$ is the kernel matrix of the single GP over the extended parameter space and the inner sum in the second equation indicates the fact that we obtain q measurements at every iteration.

Based on these individual confidence intervals on μ , we can construct confidence intervals on φ . In particular, let

$$\begin{aligned} l_i(\mathbf{w}) &= m_{q,(n-1)}(\mathbf{w}, i) - \beta_{q,n}^{1/2} \sigma_{q,(n-1)}(\mathbf{w}, i) \\ u_i(\mathbf{w}) &= m_{q,(n-1)}(\mathbf{w}, i) + \beta_{q,n}^{1/2} \sigma_{q,(n-1)}(\mathbf{w}, i) \end{aligned} \quad (17)$$

be the lower and upper confidence intervals on each predicate. From this, we construct reliable confidence intervals on $\varphi(\mathbf{w})$ as follows:

Lemma 2. Under the assumptions of Lemma 1. Let \mathcal{T} be the parse tree corresponding to φ . Then the following holds for all environment scenarios, $\mathbf{w} \in \mathbb{W}$ and iterations $n \geq 1$ jointly with probability at least $1 - \delta$,

$$\mathcal{T}(l_1(\mathbf{w}), \dots, l_q(\mathbf{w})) \leq \varphi(\mathbf{w}) \leq \mathcal{T}(u_1(\mathbf{w}), \dots, u_q(\mathbf{w})) \quad (18)$$

Proof. This is a direct consequence of Lemma 1 and the properties of the min and max operators. \square

We are now able to prove the main theorem as a direct consequence of Lemma 2.

Theorem 1. Assume that each predicate μ_i has RKHS norm bounded by B_i and that the measurement noise is σ -sub-Gaussian. Select $\delta \in (0, 1)$, \mathbf{w}_n according to (9), and

let $\beta_n^{1/2} = \sum_i B_i + 4\sigma\sqrt{1 + \ln(1/\delta) + \sum_i I(\mathbf{y}_{n-1}^i; \mu_i)}$. If $\mathcal{T}(l_1(\mathbf{w}_n), \dots, l_q(\mathbf{w}_n)) > 0$, then with probability at least $1 - \delta$ we have that $\min_{\mathbf{w} \in \mathbb{W}} \varphi(\mathbf{w}) > 0$ and the system has been verified against all environments in \mathbb{W} .

Proof. For independent variables the mutual information decomposes additively and following Remark 1 this is a direct consequence of Lemma 2, since $\mathcal{T}(l_1(\mathbf{w}), \dots, l_q(\mathbf{w})) \leq \varphi(\mathbf{w})$ holds for all $\mathbf{w} \in \mathbb{W}$ with probability at least $1 - \delta$. \square

A. Convergence proof

In the following, we prove a stronger result about convergence of our algorithm.

The key quantity in the behavior of the algorithm is the mutual information (16). Importantly, it was shown in [27] that it can be upper bounded by the worst-case mutual information, the information capacity, which in turn was shown to be sublinear by [26]. In particular, let $\mathbf{f}_{\mathbb{W}}$ denote the noisy measurements obtained when evaluating the function f at points in \mathbb{W} . The mutual information obtained by the algorithm can be bounded according to

$$\begin{aligned} I(\mathbf{f}_{\mathbb{W} \times \mathcal{I}}; f) &\leq \max_{\substack{\mathbb{W} \subset \mathbb{W}, |\mathbb{W}| \leq n}} I(\mathbf{f}_{\mathbb{W} \times \mathcal{I}}; f); \\ &\leq \max_{\substack{\mathcal{D} \subset \mathbb{W} \times \mathcal{I}, |\mathcal{D}| \leq n \cdot q}} I(\mathbf{f}_{\mathcal{D}}; f); \quad (19) \\ &= \gamma_{q,n}, \end{aligned}$$

where γ_n is the worst-case mutual information that we can obtain from n measurements,

$$\gamma_n = \max_{\substack{\mathcal{D} \subset \mathbb{W} \times \mathcal{I}, |\mathcal{D}| = n}} I(\mathbf{f}_{\mathcal{D}}; f). \quad (20)$$

This quantity was shown to be sublinear in n for many commonly-used kernels in [26].

A key quantity to show convergence of the algorithm is the instantaneous regret,

$$r_n = \min_{\mathbf{w} \in \mathbb{W}} \varphi(\mathbf{w}) - \varphi(\mathbf{w}_n), \quad (21)$$

the difference between the unknown true minimizer of φ and the environment parameters \mathbf{w}_n that Algorithm 1 selects at iteration n . If the instantaneous regret is equal to zero, the algorithm has converged.

In the following, we will show that the cumulative regret, $R_n = \sum_{i=1}^n r_i$ is sublinear in n , which implies convergence of Algorithm 1.

We start by bounding the regret in terms of the confidence intervals on μ_i .

Lemma 3. Fix $n \geq 1$, if $|f(\mathbf{w}, i) - m_{q \cdot (n-1)}(\mathbf{w}, i)| \leq \beta_{q \cdot n}^{1/2} \sigma_{q \cdot (n-1)}(\mathbf{w}, i)$ for all $\mathbf{w}, i \in \mathbb{W} \times \mathcal{I}$, then the regret is bounded by $r_n \leq 2\beta_{q \cdot n}^{1/2} \max_i \sigma_{q \cdot (n-1)}(\mathbf{w}, i)$.

Proof. The proof is analogous to [26, Lemma 5.2]. The maximum standard deviation follows from the properties of the max and min operators in the parse tree \mathcal{T} . In particular, let $a_1, b_1, a_2, b_2 \in \mathbb{R}$ with $a_1 - b_1 < a_2 - b_2$. Then for all $c_1 \in [-b_1, b_1]$ and $c_2 \in [-b_2, b_2]$ we have that

$$a_1 - b_1 \leq \min(a_1 + c_1, a_2 + c_2) \leq a_1 + b_1. \quad (22)$$

The max operator is analogous. Thus, since the parse tree \mathcal{T} is composed only of min and max nodes, the regret is bounded by the maximum error over all predicates. The result follows. \square

Lemma 4. Pick $\delta \in (0, 1)$ and $\beta_{q \cdot n}$ as shown in Lemma 1, then the following holds with probability at least $1 - \delta$,

$$\sum_{i=1}^n r_n^2 \leq \beta_{q \cdot n} C_1 q \mathbf{I}(\mathbf{f}_{\mathbb{W} \times \mathcal{I}}; f) \leq \beta_{q \cdot n} C_1 \gamma_{q \cdot n} \quad (23)$$

where r_n is the regret between the true minimizing environment scenario, \mathbf{w}^* and the current sample, \mathbf{w}_n ; and $C_1 = 8/\log 1 + \sigma^{-2}$

Proof. The first inequality follows similar to [26, Lemma 5.4] and the proofs in [27]. In particular, as in [27],

$$r_n^2 \leq 4\beta_{q \cdot n}^2 \max_{i \in \mathcal{I}} \sigma_{q \cdot (n-1)}^2(\mathbf{w}_n, i)$$

The second inequality follows from (19). \square

Lemma 5. Under the assumptions of Lemma 2, let $\delta \in (0, 1)$ and choose \mathbf{w}_n according to (9). Then, the cumulative regret R_N over N iterations of Algorithm 1 is bounded with high probability,

$$\Pr \left\{ R_N \leq \sqrt{C_1 \beta_N N \gamma_{q \cdot N}} \quad \forall N \geq 1 \right\} \geq 1 - \delta \quad (24)$$

where $C_1 = \frac{8}{\log 1 + \sigma^{-2}}$.

Proof. Since, $R_N = \sum_{i=1}^N r_i$, from Cauchy-Schwartz inequality we have, $R_N^2 \leq N \sum_{i=1}^N r_i^2$. The rest follows from Lemma 4. \square

We introduce some notation, let

$$\hat{\mathbf{w}}_n = \operatorname{argmin}_{\mathbf{w} \in \{\mathbf{w}_1, \dots, \mathbf{w}_n\}} \varphi(\mathbf{w}) \quad (25)$$

be the minimizing environment scenario sampled by BO in n iterations and let

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathbb{W}} \varphi(\mathbf{w}) \quad (26)$$

be the unknown, optimal parameter.

Corollary 1. For any $\delta \in (0, 1)$ and $\epsilon \in \mathbb{R}^+$, there exists a n^* ,

$$\frac{n^*}{\beta_{q \cdot n^*} \gamma_{q \cdot n^*}} = \frac{C_1}{\epsilon^2} \quad (27)$$

such that $\forall n \geq n^*$, $\varphi(\mathbf{w}^*) - \varphi(\hat{\mathbf{w}}_n) \leq \epsilon$ holds with probability at least $1 - \delta$.

Proof. The cumulative reward over n iterations, $R_n = \sum_{i=1}^n \varphi(\mathbf{w}^*) - \varphi(\mathbf{w}_i)$ where \mathbf{w}_i is the i -th BO sample. Defining $\hat{\mathbf{w}}_n$ as in (25) we have,

$$\begin{aligned} R_n &= \sum_{i=1}^n \varphi(\mathbf{w}^*) - \varphi(\mathbf{w}_i) \\ &\geq \sum_{i=1}^n \varphi(\mathbf{w}^*) - \varphi(\hat{\mathbf{w}}_n) \\ &= n(\varphi(\hat{\mathbf{w}}_n) - \varphi(\mathbf{w}^*)) \end{aligned} \quad (28)$$

Combining this result with Lemma 5, we have with probability greater than $1 - \delta$ that

$$\begin{aligned} \varphi(\mathbf{w}^*) - \varphi(\hat{\mathbf{w}}_n) &\leq \frac{R_n}{n} \\ &\leq \sqrt{\frac{C_1 \beta_{q \cdot n} \gamma_{q \cdot n}}{n}} \end{aligned} \quad (29)$$

To find, n^* , we bound the RHS by ϵ ,

$$\sqrt{\frac{C_1 \beta_{q \cdot n^*} \gamma_{q \cdot n^*}}{n^*}} \leq \epsilon \Rightarrow \frac{n^*}{\beta_{q \cdot n^*} \gamma_{q \cdot n^*}} \geq \frac{C_1}{\epsilon^2} \quad (30)$$

For $n > n^*$, the minimum $\varphi(\hat{\mathbf{w}}_n) \leq \varphi(\hat{\mathbf{w}}_{n^*}) \implies \varphi(\hat{\mathbf{w}}_n) - \varphi(\mathbf{w}^*) \leq \epsilon$. \square

We are now ready to prove our main convergence theorem.

Theorem 2. *Under the assumptions of Lemma 2, choose $\delta \in (0, 1)$, $\epsilon \in \mathbb{R}^+$ and define n^* using Corollary 1. If $n \geq n^*$ and $\varphi(\hat{\mathbf{w}}_n) > \epsilon$, then, with probability greater than $1 - \delta$, the following statements hold jointly*

- $\varphi(\mathbf{w}^*) > 0$
- The closed loop satisfies φ , i.e., the control can safely control the system in all environment scenarios, \mathbb{W}
- The system has been verified against all environments, \mathbb{W}

Proof. This holds from Lemma 5 and Corollary 1. From Corollary 1, we have $\forall n \geq n^*$, $\Pr(\varphi(\mathbf{w}^*) - \varphi(\hat{\mathbf{w}}_n) \leq \epsilon) > 1 - \delta$. If $\exists n \geq n^*$, such that $\varphi(\hat{\mathbf{w}}_n) > \epsilon$, then we have $\Pr(\varphi(\mathbf{w}^*) > 0) > 1 - \delta$, i.e., the minimum value φ can achieve on the closed loop system is greater than 0. φ is hence, satisfied by our system in all $\mathbf{w} \in \mathbb{W}$. \square