Received 28 January 2025; revised 18 April 2025; accepted 13 May 2025. Date of publication 22 May 2025; date of current version 24 June 2025. Recommended by Senior Editor S. Martinez.

Digital Object Identifier 10.1109/OJCSYS.2025.3572375

Safe and Robust Binary Classification and Fault Detection Using Reinforcement Learning

JOSH NETTER [1] **(Graduate Student Member, IEEE), KYRIAKOS G. VAMVOUDAKIS** [1] **(Senior Member, IEEE), TIMOTHY F. WALSH** [2]**, AND JAIDEEP RAY** [2]

(Intersection of Machine Learning with Control)

[1]Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30334 USA
[2]Sandia National Laboratories, Albuquerque, NM 87123 USA

CORRESPONDING AUTHOR: JOSH NETTER (e-mail: jnetter6@gatech.edu).

This work was supported in part by NSF under Grant CAREER CPS-1851588, Grant CPS-2038589, and Grant CPS-2227185, in part by NASA ULI under Grant 80NSSC20M0161, and in part by Sandia. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under Contract DE-NA0003525.

ABSTRACT In this paper, we propose a learning-based method utilizing the Soft Actor-Critic (SAC) algorithm to train a binary Support Vector Machine (SVM) classifier. This classifier is designed to identify valid input spaces in high-dimensional, highly constrained systems while minimizing the total runtime of offline simulations. The simulations adapt their runtime based on the likelihood that a given training input will be informative to the classifier. Furthermore, we introduce a method for using the trained SAC model to predict whether a desired system input is likely to violate constraints, along with a technique to adjust the input as necessary. Additionally, we explore the potential of this model to detect faults or adversarial attacks within the system. The effectiveness of our approach is demonstrated through various simulations of challenging classification problems and a constrained quadrotor model.

INDEX TERMS Machine learning, reinforcement learning, robust control.

I. INTRODUCTION

Currently, much of the development of autonomous vehicles has been driven by the potential of automating certain aspects of flight, such as planning in urban air mobility (UAM) scenarios [1], aerodynamic control [2], and optimizing the design process [3]. In each of these use cases, designers must ensure that these autonomous aircraft are safe in a variety of rigorous operating conditions [4]. However, finding the full extent of safe operating conditions is often a significant challenge in the design of these vehicles. When designing an aircraft, one must first confirm its valid operating conditions through extensive simulations dependent on a large variety of environmental parameters, such as velocity, atmospheric pressure, angle of attack, etc. Vehicles also often have many potential constraints that must never be violated, such as exceeding certain loads or temperatures on various components of the aircraft. This may result in very complex sets of valid flight conditions that may be non-convex or split into disjoint subsets, and dependent on a high-dimensional input vector. Fortunately, many challenges of engineering design can be simplified by constraining the values of some *quantities of interest* (QoI), such as temperature or stress in a given component, below certain thresholds. These thresholds form a boundary between the valid and invalid input space, known as *decision boundary*, and allow us to formulate these design challenges as a binary "go/no-go" classification problem. Even after simplifying the problem to a binary classification task, training a classifier for a non-convex or disconnected decision boundary may be difficult. Many methods of learning classifiers that follow the current decision boundary estimate [5], [6], [7] have been developed to minimize the number of training points required to resolve the decision boundary. However, if only high-fidelity models are used to generate training points, each additional input for a simulation used to train the model consumes time, and more complex simulations may take days or weeks to generate sufficient training data. This forces designers to choose between prohibitively long simulation runtimes or potentially resorting to low-fidelity models with resulting inaccurate simulations that increase the misclassification rate. Moreover, even if a classifier is trained offline sufficiently to capture the entire decision boundary, real-life conditions such as system noise, hardware failures, or adversarial attacks can compromise its accuracy.

In this work, we propose a method to intelligently identify informative training points using reinforcement learning to select inputs, thereby quickly training a classifier to recognize a complex valid input space with minimal prior information. In addition, we use the learned actor-critic model together with the trained classifier to assess the probability of constraint violations, adjust inputs to ensure safety, and measure the accuracy of the model in real time to determine if it has been trained accurately or if the system has malfunctioned.

Related Work: First, we examine previous methods of modeling physics-based systems. In [8], the authors propose physics-informed neural networks (PINNs), which have proven effective at modeling problems constrained by laws of physics defined with partial differential equations (PDEs), or in discovering these PDEs. This approach was also successfully applied in [9] and [10]. For our work, we do not necessarily require precise QoI values but instead seek a boundary where these QoIs are likely to violate constraints. PINNs may require too much data to effectively narrow down the valid input space.

Spiking neural networks, as discussed in [11], [12], work well with sparse data by transmitting information through the “firing” of a subset of neurons at a time. However, this approach is predominantly suitable for real-time input processing and not for the offline training simulations considered in significant portions of this work. Similar concerns apply to neural networks governed by event-triggered synapses [13].

In [14], neural networks are not used to directly learn the behavior of a system but to learn the kernel that determines the structure of a Gaussian process. This somewhat mitigates the shortcomings of Gaussian processes, which struggle with modeling particularly erratic or less smooth functions.

We must also discuss methods of binary classification and their effectiveness in dealing with irregular design spaces. For example, Gaussian process classification and Bayesian classification [15], [16] are effective not only in generating a binary classifier but also in providing a probability that the classification is accurate. Support vector machines (SVMs) [17] were also found to be very effective at classifying these systems when provided with sufficient training data in our initial tests. We also examine previous efforts on using learning to train classifiers on the conditions of real systems. For example, in [18], the authors successfully use finely tuned deep neural networks to predict defects in materials with a small training set. Semi-supervised learning has long been established for classifying images [19], but more recent work shown in [20] has combined this approach with physics-informed models to classify the quality of center-less grinders. In [21], the authors demonstrate that physics-informed neural networks combined with a visual classifier can identify bearing fatigue. The authors of [22] alternatively use a learning-based approach for the classification with costly features (CwCF) problem. In this work, classification is structured as a Markov decision process (MDP) where acquiring each feature of an input has an associated cost, and a Q-learning approach is used to decide which features to acquire and when to classify the input. In each of these works, neural networks are successfully used as classifiers or as supplement classifiers of complex systems. We seek to expand upon this work by intelligently selecting informative points to train these networks and an SVM classifier with as small a training set as possible. Previous methods of intelligently sampling training points, or adaptive sampling, are seen in [7], [23], [24].

We also refer to prior work on classification and reinforcement learning of noisy systems, as even small amounts of noise can significantly affect learned models [25]. In [26], the authors address the issue of classification with noisy samples by selectively removing outlying data points. In this work, we do not remove outliers, but instead attempt to weigh the value of training data relative to the likelihood that the classification of the data point is incorrect. The authors of [27] utilize context from neighboring data (in this case, pixels) to mitigate the effects of misclassified pixels in photographs. However, this is not suitable for our work as our training data is sufficiently sparse that we cannot refer to the surrounding data points. The paper [28] formulates a method of learning in noisy real systems by using a surrogate reward function. In this work, we also deal with noisy systems, particularly the noise that is present in simulations of a system. However, our learning approach instead reduces noise present in these simulations in later stages of the learning at the cost of increasing simulation runtime. We also learn with a soft learning approach, similar to those seen in [29], [30], [31].

Contributions: The contribution of the present paper is fivefold. First, we formulate the problem of identifying ideal training inputs as an optimization problem with a cost function to minimize. Second, we develop a reinforcement learning method to find an optimal stochastic policy for a multi-armed bandit problem. Third, we apply this approach to determine the decision boundary of a binary classifier across the entire design space without prior random sampling of the input space to learn an initial estimate of the decision boundary, which may increase total runtime. Fourth, we use these selected inputs to train a binary classifier and compare the results with other intelligent sampling methods. Finally, we extend this approach by developing a method to use reinforcement learning for selecting safe control inputs online and a method to use our learned model to predict system malfunctions by observing quantities of interest.

This paper expands upon our previously presented work [32] in several aspects. First, we adapt our presented soft actor/critic adaptive sampling algorithm to account for potential noise in classifier training data. Second, we compare this approach to additional methods of adaptive sampling such as the Basudhar-Missoum algorithm [7]. Third, this work expands on potential uses of the trained actor/critic model by examining the ability of the trained model to ensure the safety of given control inputs online, as well as the ability of the model to predict faults or adversarial attacks in the system.

Structure: The remainder of the paper is structured as follows. In Section II, we will formulate the problem of selecting effective inputs for training a go/no-go classifier. In Section III we will formulate the learning-based approach to selecting these inputs quickly and prove its effectiveness and elaborate on our method for training a classifier with noisy data in Section IV. In Section V we consider the uses of our learned model in online control. Our complete algorithms are detailed in Section VI, and in Section VII we test these algorithms in a variety of simulations. In Section VIII we conclude the paper and discuss possible future work.

II. PROBLEM FORMULATION

Consider a system with dynamics $\forall t \geq 0$,

$$\dot{x}(t) = g(x, u, t)$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ are the state and control input vectors of the system, respectively. For convenience, define a combined state and control vector $a = [x^\intercal \;\; u^\intercal]^\intercal$ that we refer to as the *input*. Consider a function

$$y = f(a),$$

where the output $y \in \mathbb{R}^p$ is some number p quantities of interest (QoIs) we choose to monitor. Each monitored QoI y_i, $i \in 1, \ldots, p$ is constrained by a maximum allowed value $y_{\max,i}$. For an input a to be valid,

$$f_i(a) \leq y_{\max,i}, \; \forall i \in 1, \ldots, p,$$

where f_i refers to the i-th value in the output vector. There may additionally be minimum constraints $y_{\min,i}$ that require $f_i(a) \geq y_{\min,i}$, but these can be trivially transformed into maximum constraints by taking the negative of both $f_i(a)$ and $y_{\min,i}$.

If any constraint is invalid, then the input is marked as invalid. This is referred to as a "go/no-go" classification problem. For convenience, we express the total valid input space as $\mathcal{A}^+$ and the invalid input space as $\mathcal{A}^-$. These two spaces can be combined to form the complete input space $\mathcal{A}$. We assume that $\mathcal{A}^+, \mathcal{A}^- \neq \emptyset$, as otherwise the problem is trivial.

Problem 1: In this work, we seek to train a classifier for the valid input space $\mathcal{A}^+$ to accurately determine the valid inputs of a system, and to ensure that the system remains inside these boundaries in real-world scenarios. While this can often be done by running an offline physics simulation for all values $a \in \mathcal{A}$ and determining which inputs result in violating the constraints on the output QoIs, these simulations are usually very time-consuming to run. As a result, we seek to accurately learn $\mathcal{A}^+$ while also minimizing the amount of time spent running simulations. To do so, we prioritize an approach that both reduces the number of runs, and shortens the runtime of each of these runs.

A. OFFLINE CLASSIFICATION TRAINING

To begin, we initially train the classifier offline to avoid any risk of failure in a real-world system. To find the valid input space, we first try to simplify the problem. Normalize each output by defining

$$\bar{f}_i(a) = \frac{f_i(a)}{y_{\max,i}}, \; \forall i \in 1, \ldots, p.$$

These normalized outputs allow us to succinctly describe the valid input space $\mathcal{A}^+$ as the set of inputs such that

$$\bar{f}_i(a) \leq 1, \; \forall i \in 1, \ldots, m, \; \forall a \in \mathcal{A}^+. \tag{1}$$

The (1) can alternatively be written as,

$$(1 - \bar{f}_i(a)) \geq 0, \; \forall i \in 1, \ldots, p, \; \forall a \in \mathcal{A}^+. \tag{2}$$

Given (2), define a combined constrained output,

$$\hat{f}(a) = \min_{i \in 1, \ldots, p} \{1 - \bar{f}_i(a)\}. \tag{3}$$

Use the combined constraints to finally define the valid input space as the set of inputs such that

$$\hat{f}(a) \geq 0, \; \forall a \in \mathcal{A}^+,$$

and conversely define the invalid input space as $\hat{f}(a) < 0, \; \forall a \in \mathcal{A}^-$. This simplification allows us to reduce any number of potentially overlapping constraints to a single function.

With our valid and invalid input spaces defined as nonnegative and negative values of $\hat{f}(a)$, it follows from the intermediate value theorem that if $\hat{f}(a)$ is continuous, then $\hat{f}(a) = 0$ for any inputs on the boundary between the valid and invalid input spaces. Thus, we define the decision boundary $\mathcal{A}^{\mathrm{b}}$ such that for an input $a^{\mathrm{b}} \in \mathcal{A}^{\mathrm{b}}$,

$$\hat{f}(a^{\mathrm{b}}) = 0.$$

We additionally make the assumption that $\hat{f}(a)$ is continuous, as without this assumption it is impossible to quantify the decision boundary using $\hat{f}(a)$, and it cannot be intelligently searched for via $\hat{f}(a)$.

Assumption 1: Assume that the outputs $f_i(a), \forall i \in 1, \ldots, p$ are continuous even if the slope may be very steep or not smooth.

With Assumption 1, we can now simplify the problem of finding the valid input space $\mathcal{A}^+$ to finding the decision boundary $\mathcal{A}^{\mathrm{b}}$.

B. MITIGATING NOISE IN TRAINING DATA

Collecting data to train a classifier offline may also raise additional challenges. Scenario-based learning using simulations or experiments to gather system data is generally imprecise, and will often not arrive at the true output of the system $f_{\text{true}}(a)$ for a given input a, but instead will return a value $f(a)$ that is close to the true value. To express this error, we assume

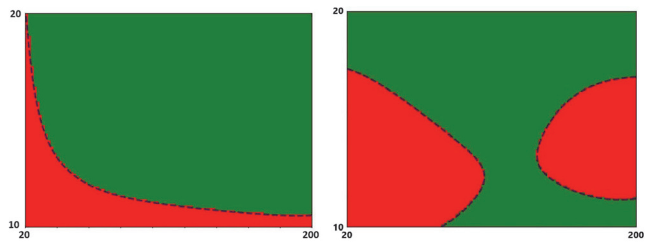


FIGURE 1. A comparison of a real decision boundary of a simple problem (left), and the learned classifier when training data is subjected to noise (right). The green area indicates inputs marked by the classifier as valid, whereas the red area indicates invalid inputs. The dashed line represents the decision boundary.

the value of the quantities of interest for a given input as a function,

$$f(a) = f_{\text{true}}(a) + w(\epsilon),$$

where $f_{\text{true}}(a) \in \mathbb{R}^p$ is the values of the quantities of interest that would be found if the scenario-based learning concluded with no measured error with given the chosen input parameters a, and $w(\epsilon) \in \mathbb{R}^p$ is a noise vector representing errors in the calculated values as a function of the tolerance ϵ. This noise vector can result in extremely inaccurate classifiers if it is allowed to become too large, as shown in Fig. 1, and must be kept as small as possible. Generally, the noise present in scenario-based learning results is inversely proportional to tolerance (as in, a high tolerance value results in more noisy data, and vice versa), and therefore intuitively the tolerance should be set at a low value to ensure accurate results. However, lower tolerances also significantly increase runtime, and sometimes for very little gain. As a result, picking a tolerance requires a difficult balance between accurate results and shorter runtimes.

To achieve this balance, we must evaluate the relative value of each individual iteration of our scenario-based learning. For this work, we seek the precise location of the decision boundary, but we are less concerned with the exact reward of the scenario at other inputs. With this in mind, we can prioritize the the scenario-based learning accuracy with inputs near the expected location of the decision boundary, and prioritize a shorter runtime with inputs that are believed to be far from the decision boundary. As a result, we seek a method to choose the tolerance value of the scenario for each iteration individually, with lower values being chosen for inputs close to the decision boundary.

C. SAFETY AND ROBUSTNESS

Once the valid input space is learned offline, the go/no-go classifier can be used to inform the state and control inputs of the real-world system to ensure no constraints are violated. For example, an online system may be driven by an autonomous controller that supplies the system with a candidate input vector $a_{\text{t}} \in \mathcal{A}$. However, the controller cannot guarantee that the desired input is safe. To address the risk of using an unsafe input, we must confirm that the candidate input is in the valid input space, and if not, we must find an alternative input $a_{\text{t}}^{\star} \in \mathcal{A}$ that is safe and close to the original candidate input.

A real-world system may also come with unpredictable safety risks. Potential failures in sensors or control input can create difficulties in keeping the real system within the valid input space that were not accounted for in scenario-based learning. First, assume that the system includes a series of sensors that return the value of each QoI $y_i(a)$, along with a potential amount of noise w_i, for an input a. We define the observed vector of all QoIs as $\hat{y}(a)$, where $\hat{y}_i(a) = y_i(a) + w_i$, $\forall i \in 1, \ldots, p$. Although some noise is expected, in a real system there is the potential for a faulty sensor or an adversarial attack to significantly increase the amount of noise, or to make a certain QoI y_i undetectable entirely. In these situations, we must instead rely on information gathered a priori to estimate the value of $y(a)$ and determine whether the difference between $\hat{y}(a)$ and $y(a)$ is potentially large enough so that the system cannot operate safely.

III. INPUT SELECTION WITH REINFORCEMENT LEARNING

In this section, we present a formulation for an optimal stochastic policy for selecting actions in a multi-armed bandit problem, as well as an actor-critic method for finding this optimal policy.

A. STATELESS SOFT POLICY

First, we require a method to quickly and intelligently find these optimal inputs. For this, we require a learning approach that can learn an optimal policy efficiently with a relatively small number of samples, and consistently even with little to no prior knowledge of the problem. Our approach also must be effective at finding the decision boundary even in a continuous domain and with a nonconvex decision boundary or multiple disjoint decision boundaries. With these priorities in mind, we formulate a soft actor-critic (SAC) approach to learn an optimal policy [30]. This approach utilizes a stochastic actor-critic approach and has been shown to quickly and reliably find an optimal policy even for complex functions across continuous domains. Additionally, the stochastic approach lends itself naturally to exploration and allows the optimal policy to avoid being trapped in local minima. While the SAC method is well-established in previous literature, we use this section to slightly augment it for the input selection problem and prove that it will still find an optimal policy despite these changes.

The SAC algorithm seeks to both maximize a reward function as well as favor stochastic policies by adding a term for the expected entropy of a policy π,

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{s_t, a_t}[r(s_t, a_t) - \log \pi(\cdot|s_t)], \tag{4}$$

where $s_t \in \mathcal{S}$ represents a state in the state space $\mathcal{S}$, and $a_t \in \mathcal{A}$ represents a potential action in the action space $\mathcal{A}$. For this work, we define the action space as the input space of the system previously defined as $\mathcal{A}$. However, the "state" of the system is not well defined, as our defined input already includes both the state and control of the system. It can be thought of as the constant parameter of every test (such as, for example, the shape of the body to analyze or the medium it is travelling through). Instead, it is better framed as a *multi-armed bandit* problem [33], or more precisely an *infinite-armed bandit* problem [34], where a state and state transitions are not considered. As a result, the effect of the state on the reward and entropy of the policy $r(s,a), -\log\pi(\cdot|s_t)$, respectively, can be ignored. We then augment (4) to a simplified form,

$$J(\pi) = \mathbb{E}_{a\sim\pi}[r(a) - \log\pi(\cdot)]. \tag{5}$$

We next discuss how to find the optimal policy $\pi^\star$ such that $Q^{\pi^\star}(a) \geq Q^{\pi}(a)\,\forall\pi\in\Pi,\ a\in\mathcal{A}$. We begin by defining the soft Q-value of the policy as shown in [29] to reward a higher entropy policy,

$$Q_{\text{soft}}(a_t) = r(a_t) + \mathbb{E}\left[\sum_{l=1}^{\infty}\gamma^l(r(a_{t+l}) - \log\pi(\cdot))\right].$$

In this policy, the term γ represents a user-chosen discount factor meant to augment the reward by including the value of future states. However, our formulation does not include future states or state transitions, and we do not wish to value the reward of future inputs in the reward of a given input. Thus, we find the discount factor to be unnecessary in our formulation and choose $\gamma = 0$. The formulation can then be simplified to resemble the cost function (5),

$$Q_{\text{soft}}(a_t) = \mathbb{E}_{a_t\sim\pi}[r(a_t) - \log\pi(\cdot)].$$

To choose our update rule, we define an *entropy-augmented reward* as $r_\pi(a) = r(a) + \mathbb{E}(\pi(\cdot))$. We then define the update rule as,

$$Q(a_t) \leftarrow r_\pi(a_t). \tag{6}$$

By using the policy evaluation convergence method described in [35] provided $\mathcal{A}<\infty$ and update rule (6) show that Q^k approaches the soft Q-value of the policy π as $k\to\infty$. The next step for finding the optimal policy is to iteratively improve the policy, and to project the improved policy onto the set of possible policies Π. In the original proposed SAC algorithm in [30], this projection is done by minimizing the Kullback-Leibler divergence in each step. We adopt a similar approach, updating the policy in each step and projecting it onto π' with,

$$\pi_{\text{new}} = \min_{\pi'\in\Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'(\cdot)||\exp(Q^{\pi_{\text{old}}}(\cdot))\right). \tag{7}$$

Lemma 1: Consider a policy $\pi_{\text{old}}\in\Pi$ and a policy π_{new} optimized in accordance with (7). Then, $Q^{\pi}_{\text{new}}(a) \geq Q^{\pi}_{\text{old}}(a)\,\forall a\in\mathcal{A}, \mathcal{A}<\infty$ where $Q^{\pi}_{\text{new}}(a), Q^{\pi}_{\text{old}}(a)$ are the action value of the new and old policies, respectively.

Proof: Define $\pi_{\text{new}}(\cdot)$ as

$$\pi_{\text{new}}(\cdot) = \min_{\pi'\in\Pi}\mathrm{D}_{\mathrm{KL}}(\pi'(\cdot)||\exp(Q^{\pi}_{\text{old}}(\cdot))),$$

or $\pi_{\text{new}}(\cdot) = \min_{\pi'\in\Pi} J_{\pi_{\text{old}}}(\pi'(\cdot))$. Since we can choose $\pi_{\text{new}} = \pi_{\text{old}}$ if the newer policy ever has a higher cost, it follows that $J_{\pi_{\text{old}}}(\pi_{\text{new}}(\cdot)) \leq J_{\pi_{\text{old}}}(\pi_{\text{old}}(\cdot))$, or using (5),

$$\begin{aligned}&\mathbb{E}_{a\sim\pi_{\text{new}}}[Q^{\pi_{\text{old}}}(a) - \log\pi_{\text{new}}(a)]\\ &\quad\geq \mathbb{E}_{a\sim\pi_{\text{old}}}[Q^{\pi_{\text{old}}}(a) - \log\pi_{\text{old}}(a)].\end{aligned} \tag{8}$$

By the inequality (8), we see,

$$Q^{\pi_{\text{old}}}(a_t) \leq r(a_t) - \mathbb{E}_{a_t\sim\pi_{\text{new}}}[\log\pi_{\text{new}}(a_t)],$$

which is alternatively expressed as,

$$Q^{\pi_{\text{old}}}(a_t) \leq Q^{\pi_{\text{new}}}(a_t).$$

Thus, as $k\to\infty$ the sequence Q^k approaches the true soft Q-value of the policy π_{new}, Q^k approaches $Q^{\pi_{\text{new}}}$.■

Theorem 1: As the soft policy evaluation and improvement are repeatedly applied, any policy $\pi\in\Pi$ will converge to a policy $\pi^\star$ such that $Q^{\pi^\star}(a) \geq Q^{\pi}(a)$ for all $a\in\mathcal{A}$ assuming $|\mathcal{A}|<\infty$.

Proof: First, consider a sequence of policies π_i, $i = 1, 2, \ldots$, where i indicates the current iteration of policy evaluation and improvement. In Lemma 1 it is shown that Q^{π_i} is monotonically increasing, and because the action space is bounded this implies the policy converges to some policy $\pi^\star$. At that time, $J_{\pi^\star}(\pi^\star(\cdot)) < J_{\pi^\star}(\pi(\cdot))$, $\forall\pi\in\Pi, \pi\neq\pi^\star$, and as a result $Q^{\pi^\star}(a) > Q^{\pi}(a)$, $\forall a\in\mathcal{A}$. Thus, $\pi^\star$ is the optimal policy.■

B. ACTOR-CRITIC IMPLEMENTATION

Now that we have proven that our stateless soft policy approach converges to an optimal policy, we then formulate how to approximate the Q-function and policy. We begin with a Q-function $Q_\theta(a_t)$ and policy $\pi_\phi(a_t)$ parameterized by θ and ϕ, respectively. In this case, we model the Q-function parameters using a neural network, and the policy as a Gaussian random vector with means and covariances given by a neural network. The parameters θ and ϕ are thus the weights of these networks. We train both of these approximations to estimate the Q-value and optimal policy using gradient descent. We train the Q-function approximation to minimize the residual error,

$$J_Q(\theta) = \mathbb{E}_{a_t}\left[\frac{1}{2}(Q_\theta(a_t) - (r(a_t) - \log_{\pi_\phi}(a_t)))^2\right],$$

with the gradient found to be,

$$\begin{aligned}\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t)(Q_\theta(a_t)\\ - r(a_t) + \mathbb{E}_{a_t}[\log_{\pi_\phi}(a_t)]).\end{aligned} \tag{9}$$

To approximate the policy, we define the cost to minimize as (7), which can be simplified as,

$$J_\pi(\phi) = \mathbb{E}_{\epsilon_t}\left[\log\pi_\phi(f_\phi(\epsilon_t)) - Q_\theta(f_\phi(\epsilon_t))\right],$$

where $f_\phi(\epsilon_t)$ is a transformation of the policy including an input noise vector ϵ_t to allow for the differentiation of the cost. This allows us to define the gradient as,

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a_t) + (\nabla_{a_t} \log \pi_\phi(a_t) - \nabla_{a_t} Q(a_t))\nabla_\phi f_\phi(\epsilon_t). \quad (10)$$

We use the gradients (9) and (10) to adjust the values of θ and ϕ to find the Q-function and the optimal policy.

We now define a reward function $r(a)$ to effectively search for the boundary inputs using the proposed actor-critic method. For training the classifier, we want to prioritize inputs close to the decision boundary regardless of if they are valid or invalid. As a result, we base our reward on the absolute value of the combined constrained output defined in (3),

$$r(a) = -|\hat{f}(a)|. \quad (11)$$

It can be trivially seen that given the reward function (11), the highest possible is $r(a) = 0$, and an input $a^\star$ is optimal if $\hat{f}(a^\star) = 0$. In other words, the reward increases as an input a approaches the decision boundary, and is maximized when $a^\star$ represents a input on the decision boundary. In turn, the reward of the policy as defined by (5) is maximized when the policy both approaches the decision boundary while also maximizing expected entropy. With this optimal policy, we prioritize sampling a large input space where the inputs are relatively close to the decision boundary, allowing us to intelligently sample the input regions of most interest for efficiently training a binary classifier.

IV. EFFICIENT TRAINING FOR CLASSIFICATION

In this section we discuss our methods of improving the total runtime of training a classifier with the SAC approach, as well as ensuring the approach accurately models the complete input space. We first discuss our method of using the estimated reward of the critic to adaptively select our scenario tolerance, and continue with our method of restricting the input space to certain subspaces to ensure a complete mapping of the decision boundary.

A. ADAPTIVE RUNTIME TOLERANCE

We have proven that the proposed approach will eventually converge to the decision boundary during runtime, but there are no guarantees on the speed of convergence. Instead, it is unavoidable, particularly when the training set is small, that many sampled inputs will not be near the decision boundary as the critic and actor learn the reward function and optimal policy and take up valuable time in scenario-based learning. To mitigate the runtime while still obtaining accurate relevant training data, we formulate a method to use estimates made by the critic to shorten the time spent on scenario inputs believed to be far from the decision boundary.

Generally, computationally-intensive simulations for training the scenario-based learning calculate a quantity of interest y_i first by creating an estimate $y_{i,0}$, and then repeatedly updating this estimate to bring it closer to the true value. The estimated QoI of an iteration k is defined as $y_{i,k}$. The run has converged at an iteration of k when $|y_{i,k} - y_{i,(k-1)}| \leq \epsilon$, where ϵ is a user-defined tolerance. As a result, higher values of ϵ result in faster convergence times, whereas lower values return more accurate estimates of a quantity of interest. With this in mind, we formulate a method for choosing a value ϵ for training our actor-critic structure that prioritizes accuracy near the likely decision boundary and speed when the decision boundary is unknown.

We define the critic's estimate of the reward function for an input a as $\bar{r}(a)$. As the highest value of the reward function is $r(a) = 0$, it follows that as $|\bar{r}(a)| \to 0$, the input a is approaching an input the critic believes to be near the decision boundary. As a result, we seek to decrease the tolerance $\epsilon(a)$ as the estimated reward $\bar{r}(a)$ decreases, but we also must ensure that the tolerance is never so high that scenario-based learning is useless, and never so low that the scenario-based learning takes prohibitively long for little gain. We first define a bounding function,

$$\mathrm{bound}(\mathrm{x}, \mathrm{x}_{\min}, \mathrm{x}_{\max}) = \begin{cases} x_{\min} & x \leq x_{\min} \\ x & x_{\min} \leq x \leq x_{\max} \\ x_{\max} & x \geq x_{\max} \end{cases} \quad \text{and}$$

then define the tolerance for a training input a^{t} as,

$$\epsilon(a^{\mathrm{t}}) = \mathrm{bound}(|d_{\mathrm{t}}\bar{r}(a^{\mathrm{t}})|, \epsilon_{\min}, \epsilon_{\max}), \quad (12)$$

where $d_{\mathrm{t}} > 0$ is a user-chosen constant to tune the relationship between the estimated reward of the training input and the tolerance, and $\epsilon_{\max} > \epsilon_{\min} > 0$ are a maximum and minimum tolerance, respectively, to ensure a minimum degree of accuracy in early stages of training with a shorter runtime in later stages of training. This adaptive tolerance adjustment allows us to spend as much total time as possible only on inputs near the decision boundary and reduce the time spent on inputs far away as the critic learns.

B. SAMPLING POTENTIAL INPUTS

Although we have shown that our reinforcement learning approach will eventually accurately evaluate points on or near the decision boundary, we must also be cautious of testing inputs that we expect to only be marginally beneficial. Any one run of the scenario-based learning used to generate training data is expected to be very time-consuming, and as a result it is crucial to ensure that we maximize the information gained by these runs. However, it is very likely that the actor will eventually return an input almost identical to a previously tested point as it begins to converge. This possibility is made even more likely if the actor and critic learning are not well-tuned at first, which may make the policy slow to converge. Simulating these repeated inputs can be a liability, as we can reasonably expect these results to be very similar to results, which we already know and have included in our training data and could amount to minutes or hours "repeating work" when we want to spread out the sampled points. To address this, we randomly sample a large set of potential training points $\mathcal{A}^{\mathrm{sampled}}$ before we begin to train the actor and critic. Then, for each new input

a^{new} returned by the actor, we locate an input a^{s} that is close to a^{new} and is defined as,

$$a^{\text{s}} = \min_{a \in \mathcal{A}^{\text{sampled}}} \sqrt{\sum_{i=1}^{(m+n)} ((a_i^{\text{new}} - a_i))^2}\,. \tag{13}$$

The input a^{s} is then used as an input and added to the set of training data $\mathcal{A}^{\text{train}}$ used both for the SVM classifier and the actor-critic framework rather than the originally returned input a^{new}. Additionally, after the input a^{s} is used for learning, it is removed from the set $\mathcal{A}^{\text{sampled}}$ to ensure that we do not repeatedly calculate the reward for the same input, and that we progressively explore the surrounding input space if the actor repeatedly returns the same or nearly the same input value a^{new}. If $\mathcal{A}^{\text{sampled}} = \emptyset$ after a certain number of learning iterations, additional points can be randomly sampled and added to the set of potential training inputs, thereby ensuring a guarantee of convergence to an optimal policy.

The proposed method provides a guarantee of finding at least a single input $a^\star$ along the decision boundary, but a single training input is still clearly insufficient for training a classifier capable of identifying the whole valid input space. Accurately modeling a complete decision boundary instead requires numerous training inputs along different points of the decision boundary to ensure accuracy across the input space. To find additional inputs and regions close to the decision boundary, we require a way to drive the actor-critic method to continue exploring despite already converging to an optimal or near-optimal policy, and to encourage this exploration without simulating many unnecessary inputs in $\mathcal{A}^{\text{sampled}}$ close to the discovered optimal policy. To accomplish this, we augment the set $\mathcal{A}^{\text{sampled}}$ to remove inputs nearby any previously found input on the decision boundary.

First, we define the set of boundary inputs $\mathcal{A}^{\text{b}} \subset \mathcal{A}^{\text{train}}$, where $\mathcal{A}^{\text{b}}$ contains each input $a^{\text{b}} \in \mathcal{A}^{\text{train}}$ such that $r(a^{\text{b}}) \geq r_{\text{c}}$, where $r_{\text{c}} \leq 0$ is a cut-off selected by the user that represents a reward considered sufficiently close to the decision boundary. In simpler terms, the set $\mathcal{A}^{\text{b}}$ consists of all tested inputs that are close enough to the decision boundary that we assume their correlating region of the input space is well-explored. Once these inputs are found, we then search the set of potential training points $\mathcal{A}^{\text{sampled}}$, and define the set of inputs $\mathcal{A}^{\text{bn}} \in \mathcal{A}^{\text{sampled}}$ where for each input $a^{\text{bn}} \in \mathcal{A}^{\text{bn}}$,

$$a^{\text{bn}} = \min_{a^{\text{b}} \in \mathcal{A}^{\text{b}}} \left\{ \sqrt{\sum_{i=1}^{(m+n)} \left((a_i^{\text{b}} - a_i^{\text{bn}}) \right)^2} \right\} \leq d_{\text{b}}, \tag{14}$$

where $d_{\text{b}} \geq 0$ is a user-chosen minimum required distance from the known boundary points. The set $\mathcal{A}^{\text{bn}}$, referred to as the *boundary neighborhood*, is then removed from $\mathcal{A}^{\text{sampled}}$ to mark portions of the action space as sufficiently explored. This process can be repeated until a large portion of the total input space has been examined, or until the classifier has met any given convergence criteria.

V. ONLINE ACTOR-CRITIC CONTROL

Another significant benefit of formulating the initial "go/no-go" classifier using training inputs supplied through the actor-critic framework is its adaptability for safe online control. In this section, we detail a method for ensuring robust control using a minimax approach to evaluate the risk of a given candidate input. Additionally, we define a "buffer" region around the decision boundary to account for any potential errors in the classifier near the boundary.

A. ROBUST ONLINE CONTROL

Controlling a real-world system requires robustness against numerous irregularities, such as errors in the model, perturbations in flight conditions, and noise or potential adversarial attacks in control inputs or sensor outputs. To mitigate the dangers of these discrepancies, we use a minimax approach to identify inputs that are the least likely to violate the constraints on QoIs even if subjected to noise.

To begin, the controller supplies a candidate input value $a^{\text{t}} \in \mathcal{A}$ that the controller seeks to reach. To verify that this candidate input is in the valid input space, we generate a set of D perturbation vectors,

$$A_D = \{\alpha_1, \alpha_2, \ldots, \alpha_D\},$$

where each value α_i, $i = 1, 2, \ldots, D$ is a vector of length $(m + n)$ consisting of a series of random normal variables of mean 0 and user-defined variance σ_α. We then define D perturbed test candidate inputs $\mathcal{A}^{\text{t}} = \{a_1^{\text{t}}, a_2^{\text{t}}, \ldots, a_D^{\text{t}}\}$ where $a_i^{\text{t}} = a^{\text{t}} + \alpha_i$, $i = 1, \ldots, D$. To find the safest input among these test candidate inputs, we naturally exclude any initial test inputs that are not possible in our valid input space $\mathcal{A}^+$ as defined by the classifier trained in the offline portion.

After finding each test input believed to be safe, we then use the critic to estimate their reward. In the offline training used to find inputs to train the classifier, the critic learns an approximation $\bar{r}(a)$ of the reward function $r(a)$ (11). As the decision boundary of the classifier is defined as inputs where $r(a) = 0$, or where one or more quantities of interest is equal to its maximum allowed value, it follows that as $r(a)$ decreases, the QoIs of the input a are an increasing distance from the initial constraints. As a result, lower values of the reward function in the valid input space correlate with more safe inputs farther away from the decision boundary. With this in mind, we can restructure this reward function to instead be a *cost* function $J(a)$ to be minimized to find the safest test input. To select these candidate inputs, we utilize a minimax approach where we seek the minimum cost test input in $\mathcal{A}^{\text{t}}$, where the cost $J(a_i^{\text{t}})$ is defined as

$$J(a_i^{\text{t}}) = \max_{j=1,\ldots,D} (|\bar{r}(a_i^{\text{t}})|),\ i = 1, 2, \ldots, D. \tag{15}$$

In other words, we seek to find the point that the learned critic predicts is the farthest away from violating any constraints in the worst case of potential disturbances. We define this input as $a^{\text{t},\star}$. It should be noted that this method of determining the safety of various inputs involves numerous calculations of the critic's cost function. Fortunately, evaluating the cost of an input using a trained critic is not very computationally taxing compared to full scenario-based learning runs, and can easily be done many times in a short time-span.

To confirm the safety of an input $a^{t,\star}$, we must also account for errors in learned model of the reward function $\bar{r}(a)$. As the system operates, the actor and the critic are provided with the current combined state and control of the system a, as well as the quantities of the vector of interest y. The detected vector is used to calculate the true reward of the system $r(a)$, and compared to the reward estimated by the critic $\bar{r}(a)$. We define the critic reward error as $\bar{r}_{\text{err}}(a) = |r(a) - \bar{r}(a)|$, and define the *maximum reward error* $\bar{r}_{\text{err,max}}$ as the maximum detected value of $\bar{r}_{\text{err}}(a)$ in the operation of the system.

Theorem 2: If the maximum reward error $\bar{r}_{\text{err,max}}$ is known, then the critic can guarantee that a given candidate input a^{t} does not cross the decision boundary provided that the predicted reward $\bar{r}(a^{\text{t}})$ is greater than $\bar{r}_{\text{err,max}}$.

Proof: We prove this theorem by contradiction. Consider a candidate input a^{t} that violates some number of constraints in the real system but is believed by the critic to be a valid input with a reward $\bar{r}(a^{\text{t}})$. When this occurs, it indicates the reward error is sufficiently large for the difference between the expected and actual reward to cross over the decision boundary, $\bar{r}_{\text{err}}(a^{\text{t}}) > \bar{r}(a^{\text{t}})$. Assume that this input's calculated reward $\bar{r}(a^{\text{t}}) > \bar{r}_{\text{err,max}}$, then it is implied that $\bar{r}_{\text{err}}(a^{\text{t}}) > \bar{r}(a^{\text{t}}) > \bar{r}_{\text{err,max}}$. This is simplified to $\bar{r}_{\text{err}}(a^{\text{t}}) > \bar{r}_{\text{err,max}}$. We assumed that $\bar{r}_{\text{err}}(a^{\text{t}}) > \bar{r}_{\text{err,max}}$, but by definition, any reward error $\bar{r}_{\text{err}}(a^{\text{t}}) \leq \bar{r}_{\text{err,max}}$. Therefore, for an invalid input to be considered valid, it must be true that $\bar{r}(a^{\text{t}}) \leq \bar{r}_{\text{err,max}}$, as in this case $\bar{r}_{\text{err}}(a^{\text{t}}) > \bar{r}(a^{\text{t}}) \leq \bar{r}_{\text{err,max}}$ and the relation $\bar{r}_{\text{err}}(a^{\text{t}}) \leq \bar{r}_{\text{err,max}}$ may be true. Thus, the theorem is true by contradiction.■

Using the critic reward function and Theorem 2 we can ensure that the test input $a^{t,\star}$ is safe as the critic reward function is improved, and the system can use this as its target input. This approach also presents another advantage of using an actor-critic approach to find training inputs, as the system requires a quantitative estimate of how close a candidate input is to the decision boundary to ensure safety, which cannot be effectively made without numerical estimates of the quantities of interest. It should additionally be noted that while this approach to robust control ensures safety, it becomes more restrictive as the reward function improves over time and the average reward error decreases. To mitigate this, the average reward error and error variance may be used to instead select safety margins for appropriate target inputs.

B. FAULT RECOGNITION

We now formulate how to use the learned critic as well as real-time data to predict whether the system has suffered any hardware failure, or if the model is dangerously inaccurate by comparing the model's predicted quantities of interest with the true values of the system. We consider a constrained system modeled with an actor-critic structure initially learned offline and outfitted with sensors that detect any quantity of interest. We consider the system unsafe if,

$$\bar{r}_{\text{err}}(a) > \bar{r}_{\text{err,cutoff}}(a), \tag{16}$$

where $\bar{r}_{\text{err,cutoff}}(a)$ is a user-chosen maximum permissible critic error. If the critic error exceeds this threshold, the model is flagged as too inaccurate to model the system, and the system should be shut down to ensure safety. This error may be due to hardware malfunctions, adversarial attacks, or poor training of the actor and the critic. The proposed approach to recognizing faults in the system or model allows us to detect indicators of malfunctions even when the system is currently safe by measuring errors in the model, whereas using only a binary classifier cannot detect any errors unless the system has already violated some number of constraints.

Choosing a reasonable maximum critic error beforehand may prove to be impossible in systems where we lack prior information about the expected level of noise in the system. To compensate, we define $\bar{r}_{\text{err,cutoff}}$ as a function of the error of some number N prior training inputs,

$$\bar{r}_{\text{err,cutoff}} = \frac{d_e \sum_{i=1}^{N} \bar{r}(a_i)_{\text{err}}/\epsilon(a_i)}{N}, \tag{17}$$

where d_e is a user-defined coefficient. In other words, $\bar{r}_{\text{err,cutoff}}$ is determined as a function of the average reward error of the training data, where each input of the training data is weighed in inverse proportion to its respective tolerance. This approach allows us to set the maximum critic error simultaneously with training the model, ensuring a very high tolerance at first and progressively decreasing as the model becomes more certain and approaches the minimum tolerance ϵ_{min} as defined in 12.

We similarly adjust the value of d_e as more data become available and we acquire more information on the accuracy and precision of our critic's reward function. Initially, the critic is prone to high variance in its errors as it samples points in less-sampled areas of the input space. However, as it learns a better reward function, these inputs with extremely high reward errors become less common. We use this behavior by modeling the adjusted reward error as a normal random variable $\mathcal{N}((\sum_{i=1}^{N} \bar{r}(a_i)_{\text{err}}), \sigma_{\text{err}})$, with a mean of the average adjusted reward error and a standard deviation σ_{err}. We can then define d_e as a multiple of σ_{err} to adjust d_e as necessary to ensure the desired degree of precision for our fault recognition approach. For example, selecting $d_e = 2\sigma_{\text{err}}$ indicates that 95% of inputs will not falsely generate significant enough errors to potentially indicate a fault or adversarial attack in the system, and $d_e = 2\sigma_{\text{err}}$ indicates that 99.7% of inputs would not indicate a fault or an attack. It should be noted that the scope of this paper only includes sensor error in the form of bounded noise, and not other potential failures such as a loss of sensor data.

VI. ALGORITHMIC FRAMEWORK

We now formulate the complete framework for creating the offline go/no-go classifier, as well as expand upon this framework for online control. We first initialize the actor and critic and take repeated training steps where we use the actor to select inputs for the problem, and use the results to update the model weights of both the actor and critic to improve the policy. With each input returned by the modeled policy, we calculate the reward with a run of the simulation of the system with the returned input and a tolerance value selected as a function of the expected reward as described in (12) and use it to train the actor-critic framework. We also take note of each input a^{b} where $r(a^{\mathrm{b}}) \geq r_{\mathrm{c}}$ and use this to form the set of boundary inputs $\mathcal{A}^{\mathrm{b}}$. After the learned policy meets the convergence criteria (such as a pre-determined number of simulation runs), each saved input is added to the training set of a soft-margin SVM classifier. We then remove the boundary neighborhood $\mathcal{A}^{\mathrm{bn}}$ from the action space using $\mathcal{A}^{\mathrm{b}}$ as defined by (14). The process is repeated until the classifier reaches the determined convergence criteria. This complete approach is shown in Algorithm 1.

In the online portion, the learned classifier and actor-critic structure are continually updated with new, real-time information rather than solely simulation data. This data is also used by the learned critic model to verify if the system output is still similar to the output predicted by the critic. If the difference between the predicted critic reward and the actual reward of the system exceeds a bound as defined in 17, then the system is flagged as potentially at risk of failure. In addition, the critic and the classifier are used to ensure safety and potentially adjust the values of a desired input $a^{\star}$. This process is described in Algorithm 2.

Algorithm 1: Soft Actor-Critic (SAC) Input Selection.

```
Input: $\mathcal{A}$ - input space; $d_t$ - Tolerance constant; $\epsilon_{\max}$ -
  Maximum tolerance; $\epsilon_{\min}$ - Minimum tolerance; $d_b$ -
  boundary neighborhood size; $r_c$ - boundary input reward
  cutoff; $\phi, \theta$ - SAC parameters; $y_{\max,1}, \ldots, y_{\max,n}$ -
  Output constraints
Output: SVM - trained support vector machine classifier
 1: $\mathcal{A}^{\mathrm{train}} \leftarrow \{\}$
 2: $\mathcal{A}^{\mathrm{b}} \leftarrow \{\}$
 3: $\mathcal{A}^{\mathrm{sampled}} \leftarrow \texttt{GenerateSamples}(\mathcal{A})$      ▷
    Initialization
 4: while SVM not converged do
 5:   while $\pi$ not converged do
 6:     $a^{\mathrm{new}} \leftarrow \texttt{Actor}(\pi_\phi)$
 7:     $a^{\mathrm{s}} \leftarrow \texttt{ClosestValue}(\mathcal{A}^{\mathrm{sampled}}, a^{\mathrm{new}})$ (13)
 8:     $\bar{r} \leftarrow \texttt{Critic}(a^{\mathrm{s}}, Q_\theta)$
 9:     $\epsilon(a^{\mathrm{s}}) \leftarrow \texttt{Bound}(d_t \bar{r}(a^{\mathrm{s}}), \epsilon_{\max}, \epsilon_{\min})$ (12)
10:     $y \leftarrow \texttt{SimulationRun}(a^{\mathrm{s}}, \epsilon(a^{\mathrm{s}}))$
11:     $r(a^{\mathrm{s}}) \leftarrow \texttt{Reward}(y, y_{\max,1}, \ldots, y_{\max,n})$
12:     if $r(a^{\mathrm{s}}) \geq r_{\mathrm{c}}$ then
13:       $\mathcal{A}^{\mathrm{b}} \leftarrow \mathcal{A}^{\mathrm{b}} \bigcup a^{\mathrm{s}}$
14:     end if
15:     $\mathcal{A}^{\mathrm{train}} \leftarrow \mathcal{A}^{\mathrm{train}} \bigcup a^{\mathrm{s}}$
16:     $\theta \leftarrow \theta - \nabla_\theta J_Q(\theta)$ (9)
17:     $\phi \leftarrow \phi - \nabla_\phi J_\pi(\phi)$ (10)
18:   end while
19:   SVM $\leftarrow \texttt{TrainClassifier}(\mathcal{A}^{\mathrm{train}}, \epsilon(a^{\mathrm{s}}))$
20:   $\mathcal{A}^{\mathrm{bn}} \leftarrow$
      $\texttt{BoundaryNeighborhood}(\mathcal{A}^{\mathrm{train}}, \mathcal{A}^{\mathrm{b}}, d_{\mathrm{b}})$ (14)
21:   $\mathcal{A}^{\mathrm{sampled}} \leftarrow \mathcal{A}^{\mathrm{sampled}} \setminus \mathcal{A}^{\mathrm{bn}}$
22: end while
23: return SVM
```

VII. SIMULATIONS

The proposed method of training an actor-critic method for binary classification was tested on several problems. The selected problems are of both varying complexity and dimensionality to measure the effectiveness of the approach with a variety of valid input spaces, such as both convex and non-convex input spaces, low- and high-dimensional inputs, and continuous and disjoint sets of valid input spaces.

For the first problem, we model a bar with a uniform area cross-section made of N materials, each with a fixed length $l_1, l_2, \ldots, l_N$ and a Young's modulus $E_1, E_2, \ldots, E_N$. We then apply a point force F at one end along the length of the bar, displacing the bar by a distance $u = F \sum_{i=1}^{N} \frac{l_i}{E_i}$. We seek Young's moduli values of each material where the bar displacement does not exceed a given threshold u_0. This example produces a relatively simple convex valid input space, and additionally the input vector can be easily scaled to any desired size to determine the dimensionality of the problem. In this work, we tested our approach in two cases on a bar containing 2 and 10 unique segments. The valid input space for the 2-dimensional case is shown in Fig. 2.

The second problem models a function with an input of size N, $f(a) = \sum_{n=1}^{N}, a_{(2n-1)} - \left|\tan\left(\frac{a_{2n}}{2} + 2\right)\right|$, with a constraint that $f(a) < c$, where c is a user-chosen constant. This results in an irregular, non-convex valid input space. To visualize the non-convex valid input space, the 2-dimensional case is shown in Fig. 3.

Our next example again models a function with a 2-dimensional input $f(a) = \frac{(a_1+4)(a_2-1)}{20} - \sin\frac{5a_1}{2}$ with constraint $f(a) > 1$. This results in a heavily multimodal function that presents significant challenges for traditional sampling approaches. A picture of the multimodal function valid input space is provided in Fig. 4.

The fourth problem uses an input of arbitrary length $N \geq 2$. For this problem, each input value a_i, $i \in 1, \ldots, N$ falls into a range $a_{\min} < a_i < a_{\max}$, $i \in N$ where $a_{\min} < 0$, $a_{\max} > 0$. For this problem, we define a constraint $\prod_{i=1}^{N} a_i > c$, where $c \geq 0$ is a user-chosen constant, resulting in numerous disjoint sets of valid inputs. A visualization of the disjoint problem valid input space is shown in Fig. 5.

For the final problem, we simulate an aircraft with an input vector $a = [\phi \;\; \theta \;\; \psi \;\; \dot{\phi} \;\; \dot{\theta} \;\; \dot{\psi}]$ and outputs

$$p = \dot{\phi} - \dot{\psi} \sin\theta$$
$$q = \dot{\theta} \cos\phi + \dot{\psi} \cos\theta \sin\phi$$
$$r = -\dot{\theta} \sin\theta + \dot{\psi} \cos\theta \cos\phi.$$

Algorithm 2: Safe Input Selection.

```
Input: $a^{t}$ - Desired input; $\sigma_\alpha$ - Variance of perturbations;
  SVM - Trained binary classifier
Output: $a^{t,\star}$ - Predicted safe input
1: $a^{t,\star} \leftarrow a^{t}$
2: for $i = 1, 2, \ldots, D$ do
3:     $\alpha_{i,j} \leftarrow N(0, \sigma_\alpha),\ j = 1, \ldots, (m+n)$
4:     $a^{t}_{i} \leftarrow a^{t}_{i} + \alpha_d$
5:     if $|r(a^{t}_{i})| > |r(a^{t,\star})|$&IsValid($a^{t}_{i}$, SVM) then (15)
6:         $a^{t,\star} \leftarrow a^{t}_{i}$
7:     end if
8: end for
9: if IsValid($a^{t,\star}$, SVM) then
10:     return $a^{t,\star}$
11: else
12:     return $\emptyset$
13: end if
```

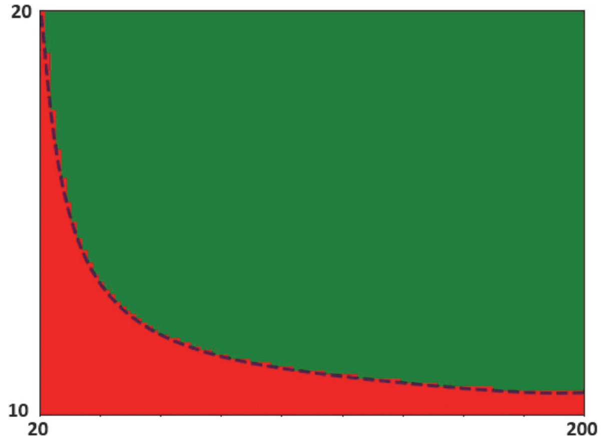


FIGURE 2. An illustration of the valid input space of the 2-dimensional uniaxial bar problem.

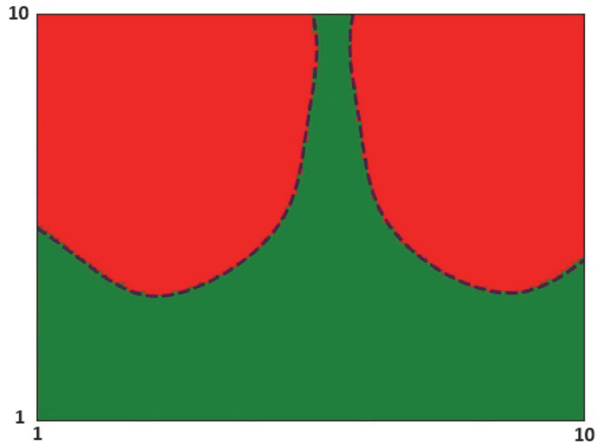


FIGURE 3. An illustration of the valid input space of the 2-dimensional non-convex problem when $c = 3$.

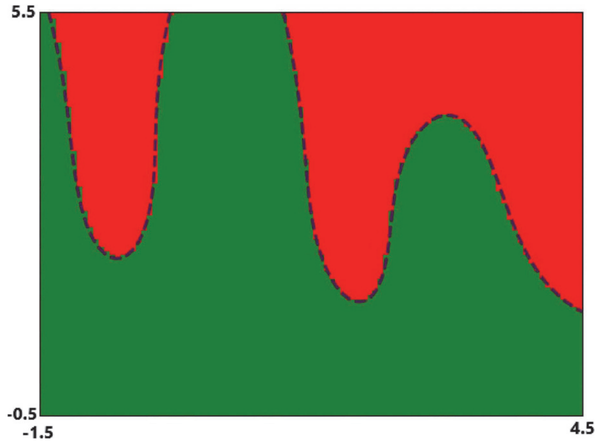


FIGURE 4. An illustration of the valid input space of the 2-dimensional multimodal problem.

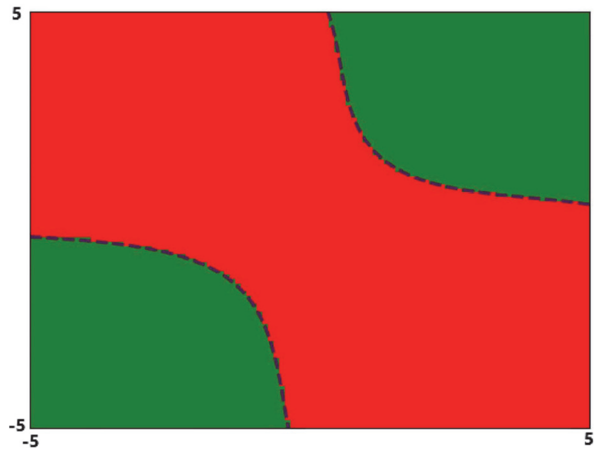


FIGURE 5. An illustration of the valid input space of the 2-dimensional disjoint problem when $c = 2$.

This problem has multiple constraints, and an input a is valid if $|p(a)| < 3.5$, $|\dot{\phi}(a)| < 2.5$, and $|\dot{\theta}(a)| < 2.5$. This serves as both a practical application of the soft actor-critic approach to classification, as well as a more complex nonlinear system containing several overlapping constraints.

A. OFFLINE TRAINING WITH NOISY DATA

We first examine the effectiveness of using the proposed soft actor-critic approach to find boundary inputs to train a binary classifier offline with potentially noisy data. We compare our method with several existing algorithms. First, we test a greedy sampling approach on each experiment as a baseline that adds two inputs a_k^+, a_k^- on each iteration k to the training data used by the classifier. These inputs are defined as

$$a_k^+ = \min_{a \in \mathcal{A}_k^+} |y_k(x)|,\ a_k^- = \min_{a \in \mathcal{A}_k^-} |y_k(a)|,$$

where y_k represents the SVM decision function at iteration k, and $\mathcal{A}_k^+, \mathcal{A}_k^-$ represent the believed valid and invalid input

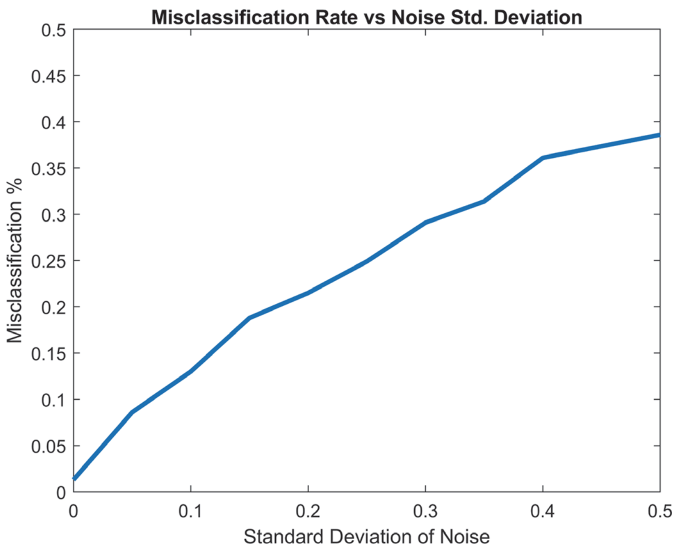


FIGURE 6. **The average misclassification rate of the greedy input selection methods on the 2-dimensional uniaxial bar problem versus the baseline standard deviation σ_0 of the noise vector over 100 training inputs. The misclassification rate steadily increases with σ_0 until it reaches a point where the classifier performs only slightly better than a random guess (50%).**

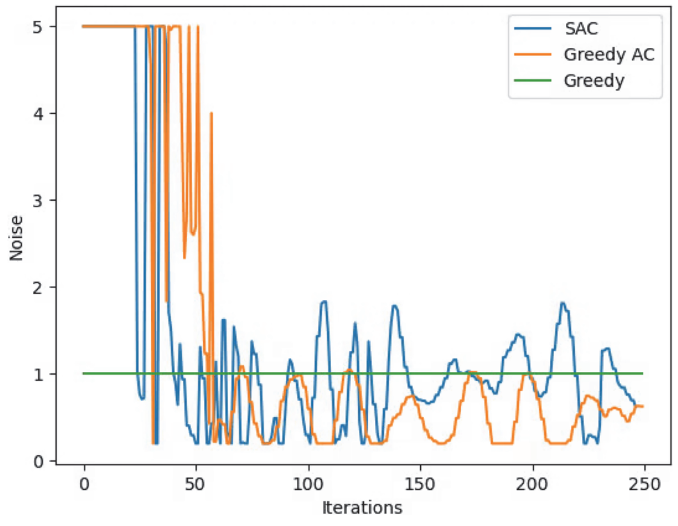


FIGURE 7. **The standard deviation of noise for each training input on the 10-dimensional uniaxial bar problem for the SAC, greedy AC, and greedy approaches.**

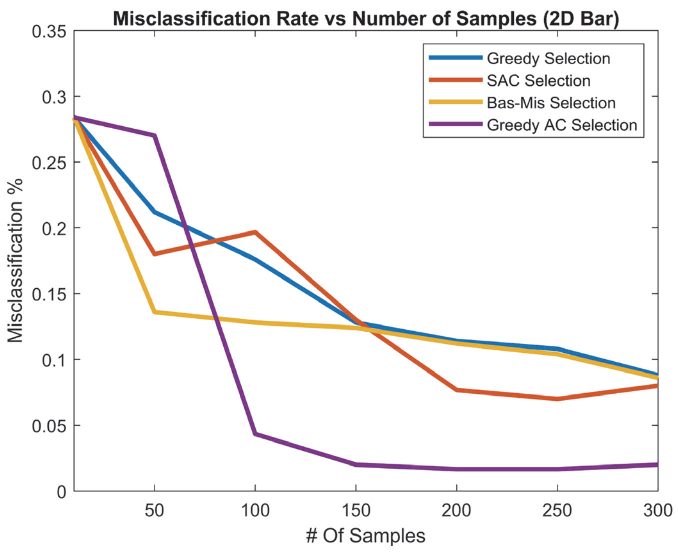


FIGURE 8. **Comparison of the average misclassification rate of several input methods on the 2-dimensional uniaxial bar problem.**

spaces at iteration k, respectively. This approach attempts to choose two inputs that are close to the decision boundary and lie on opposite sides. This approach is further detailed in [5], and was found to be effective in selecting an informative training set for several problems, including the uniaxial bar problem. We also compare the SAC and greedy approaches with the Basudhar-Missoum point selection algorithm, detailed and shown to be effective in [7].

To determine the efficacy of both approaches when using noisy data created by imprecise simulations or tests, an input a is valid if $y_i(a) + w_i(a) \leq y_{\max,i},\ \forall i \in 1, \ldots, p$, where w is a random noise vector where for each value $i \in 1, \ldots, p$, w_i is a random normal variable with mean 0 and standard deviation $(\sigma_0 y_i)$, where σ_0 represents a baseline standard deviation chosen by the user. In other words, the noise present for a given output y_i is dependent on the output value itself. The value of σ_0 for the SAC algorithm is defined as

$$\sigma_0 = \text{bound}(|d_t \bar{r}(a^t)|,\ \epsilon_{\min},\ \epsilon_{\max}) \tag{18}$$

to simulate adaptive tolerances. For the greedy input selection algorithm, σ_0 is defined as a static value because the greedy approach does not know, quantitatively, how close constraints are to being violated and, as a result, does not judge how close it is to the true decision boundary. For the SAC approach, $\epsilon_{\min}$ and $\epsilon_{\max}$ are defined to allow a maximum and minimum standard deviation value of $5\sigma_{0,\text{greedy}}$ and $0.2\sigma_{0,\text{greedy}}$, respectively, where $\sigma_{0,\text{greedy}}$ is the value of σ_0 for simulations using greedy input selection. These limits are chosen to ensure that the SAC input selection initially uses noisier data than the greedy input selection, but will allow the SAC algorithm to become more precise than $\sigma_{0,\text{greedy}}$ as more data are added. A visualization of the greedy approach's misclassification rate as a function of the value of σ_0 is shown in Fig. 6 to demonstrate the effect of this noise on an SVM classifier. To ensure a reasonable comparison between the algorithms presented in this work, we also examine the efficacy of a combination of the greedy and SAC algorithms we refer to as Greedy AC. In this approach, greedy sampling is used to select the training points for the classifier and a trained critic reward function determines the value of σ_0 via (18). A comparison of the noise levels of the SAC, greedy AC, and greedy approaches is shown in Fig. 7, and illustrates how our adaptive tolerance approach begins with higher levels of noise in its training samples, but decreases this noise as the critic learns what points are close to the decision boundary.

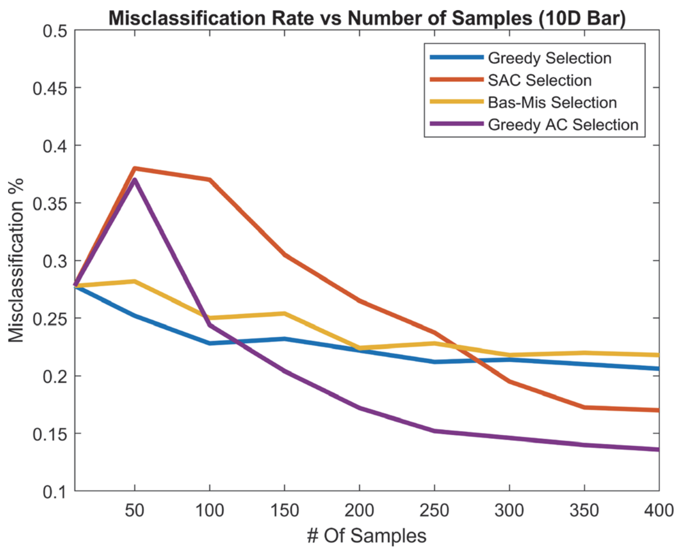


FIGURE 9. **Comparison of the average misclassification rate of several input selection methods on the 10-dimensional uniaxial bar problem.**

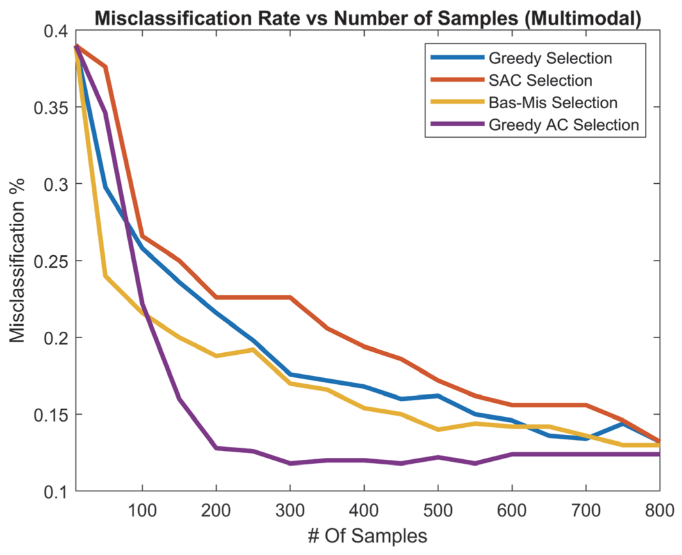


FIGURE 11. **Comparison of the average misclassification rate of several input selection methods on a 2-dimensional multimodal classification problem.**

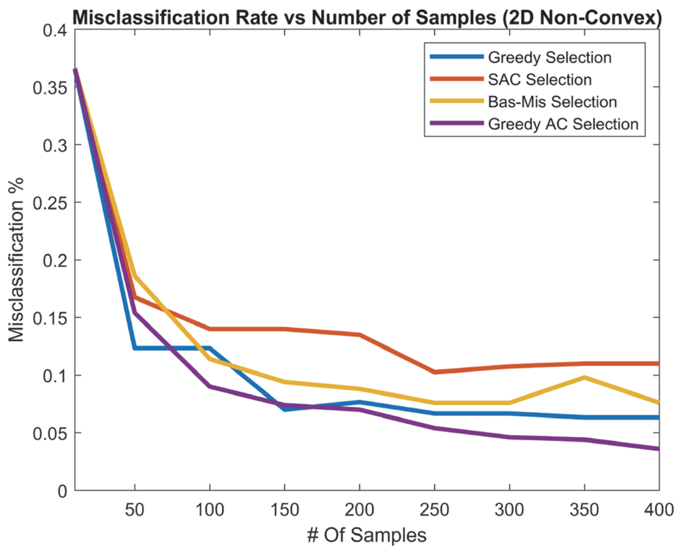


FIGURE 10. **Comparison of the average misclassification rate of several input selection methods on a 2-dimensional non-convex classification problem.**

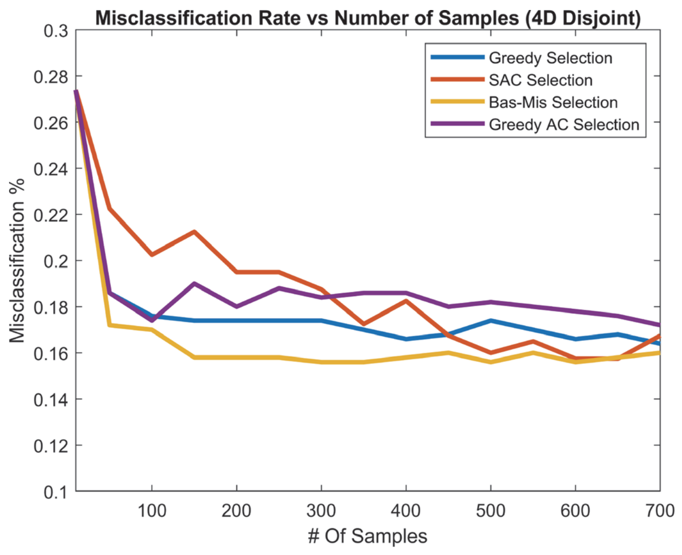


FIGURE 12. **Comparison of the average misclassification rate of several input selection methods on the 4-dimensional disjoint set problem.**

The resulting average misclassification rates for the uniaxial bar experiments as a function of the number of samples are shown in Figs. 8 and 9. In both cases, we see both the SAC and greedy AC input selection approaches initially under-perform both the greedy and Basudhar-Missoum approaches due to the additional noise present in their training data. However, as the critic learns to identify points along the decision boundary and adapt the simulation tolerances accordingly, both the SAC and greedy AC approaches overtake the greedy and Basudhar-Missoum approaches. In particular, the greedy AC approach performs the best by a significant margin in both cases. This shows that the adaptive tolerance approach used by the actor-critic is able to mitigate the effects of the noise by quickly learning an optimal policy and running more precise simulations. This trend is shown to continue for the greedy AC approach in noisy classifiers trained on the 2-dimensional non-convex problem as well as the multimodal problem, shown in Figs. 10 and 11, respectively. However, the SAC approach seems to slightly fall behind in these cases.

We continue by testing all approaches on the 4-dimensional disjoint sets problem, and show the resulting misclassification rates in Fig. 12. In this case, all algorithms seem to converge at approximately the same misclassification rate. This is likely indicative of a shortcoming of the SVM classifier's ability to properly classify valid inputs across numerous disjointed valid input spaces in higher-dimensional problems. Notably,

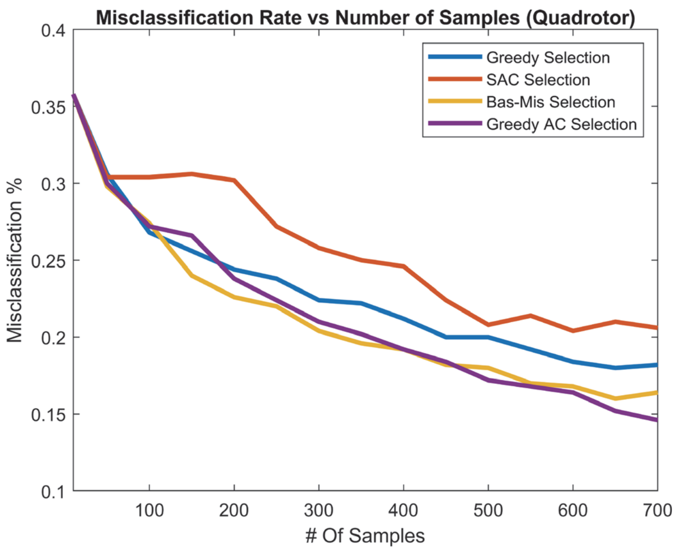


FIGURE 13. Comparison of the average misclassification rate of several input selection methods on a simulated quadrotor with multiple constraints.

this is not an issue in the lower-dimensional example shown in Fig. 5.

Finally, the misclassification rates of each algorithm for the quadrotor problem is shown in Fig. 13. In this scenario we once again see the greedy AC approach outperform the other approaches. Interestingly, the SAC input selection method initially takes significantly longer to see improvements in the classification rate. This potentially indicates that using the actor to select points is not as effective in problems with numerous constraints, as the reward function guiding the actor to the decision boundary contains numerous local minima. In total, across most problems SAC input selection remains competitive with other input selection algorithms, and outperforms other approaches in several cases. In addition, the greedy AC input selection performs as well or better than other tested approaches at all times, showing the efficacy of our adaptive tolerance method, even when beginning with significantly noisier training data.

B. ONLINE CONTROL

After conducting offline training for each of the previously mentioned experiments, we also test our robust online control methodology for the actor-critic method. The classifier trained by the greedy approach has no method of predicting the average noise present in the data, so this approach can only be applied with the additional information supplied by the trained critic. After training the initial classifier using the actor-critic framework, we then test a collection of C randomly generated candidate inputs $A = \{a_1, a_2, \ldots, a_C\}$. For each candidate input a_i, $i = 1, 2, \ldots, C$, an additional set of D inputs $A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,D}\}$ is generated, where each input is defined as

$$a_{i,j} = a_i + \alpha_{i,j}, \; j = 1, 2, \ldots, D,$$

where $\alpha_{i,j}$ is a randomly generated perturbation vector. For each set of perturbed inputs, one is selected as the target input $a_i^\star$ using the methodology outlined in Section V with the goal of selecting a target away from the decision boundary to protect from noise or variance in real-world systems and to ensure a higher likelihood that the target input is correctly classified. Results comparing the misclassification rate of the classifier on randomly selected candidate inputs, as well as the misclassification rate of these target inputs, is shown in Table 1. In these simulations, the classifier finds that these target inputs are significantly more likely to be properly identified by the classifier as safe, showing the effectiveness of using a binary classifier trained offline in conjunction with an actor-critic structure to select safe inputs for robust online control.

TABLE 1. Comparison of the misclassification rates of a classifier trained using soft actor-critic input selection on randomly selected inputs, and inputs believed to be farther from the decision boundary according to the trained critic for robust control. the inputs chosen for robust control are found to be significantly more likely to be safe and properly classified.

Misclass. Rate	Random Selection	Target Inputs
Uni. Bar (2 Dim)	.24	.07
Uni. Bar (10 Dim)	.27	.18
Non-Convex (2 Dim)	.31	.27
Quadrotor	.32	.22

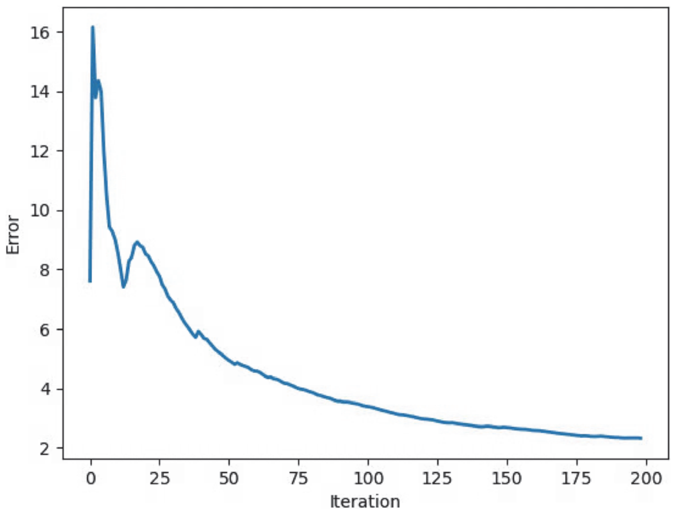


FIGURE 14. Change in average error of a critic estimating the reward of the quadrotor problem as a function of number of test samples.

C. ONLINE FAULT RECOGNITION

For the final set of simulations, we evaluate the ability of the SAC algorithm to recognize faults in its estimates of the reward function. To begin with, the SAC algorithm is first trained with noisy offline data with a value $\sigma_0 = \text{bound}(|d_t \bar{r}(a^t)|, \epsilon_{\min}, \epsilon_{\max})$. The resulting average error is shown in Fig. 14. After some time, the values d_t, $\epsilon_{\min}$, $\epsilon_{\max}$, and by extension the noise present in the system, are increased to simulate either a system fault or adversarial attack. We then

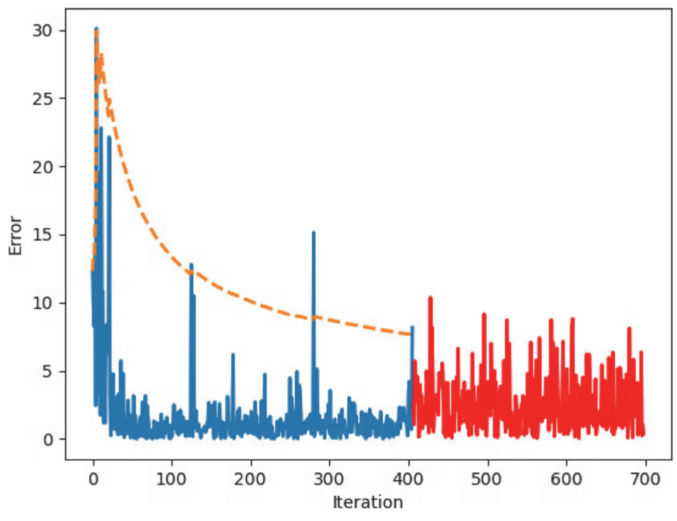


FIGURE 15. The error of the critic in the quadrotor problem, where the blue line indicates before any significant error is detected in the system, and red indicates after a sufficient error is detected. The orange dotted line indicates the threshold of maximum critic error, defined as $2\sigma_{\text{err}}$ over the mean error. The first 300 iterations are used to learn the reward function and no attempts are made to identify significant errors. At iteration 300, we begin to attempt to detect a significant error in the critic reward function. At time-step 400, noise is introduced into the system and is almost immediately flagged by our fault recognition.

monitor the time it takes for the critic to mark the system unsafe as defined by (16) and (17), using a value $d_e = 2\sigma_{\text{err}}$. The real-time error of the simulation over time, both before and after noise addition, as well as the value d_e, is shown in Fig. 15. This first figure shows the effectiveness of our critic's model of the reward function. The second figure shows that our method of detecting errors in the system both quickly recognizes additional noise in the system once the critic is sufficiently trained in real-time and does not falsely report this noise too early.

VIII. CONCLUSION

In this work, we introduced an SAC method for creating a binary classifier for complex dynamic systems, such as flight vehicles, and explore its applications in both off-line classification and on-line robust control. The goals of this approach were to reduce simulation runtime during offline training while ensuring the classifier remained robust to noise and system faults when online. First, we compared this approach to training a classifier offline with noisy data with several other algorithms for selecting training inputs. We found that the learning-based approach was competitive with other algorithms in various example problems, despite the presence of additional initial noise, and that our proposed adaptive tolerance algorithm was very effective in mitigating this noise. We also found that the learned critic could be used to ensure the safety of selected system inputs, guide desired inputs toward safer alternatives, and detect faults in either the system or the critic itself.

In future work, we will explore potential applications of combining the adaptive tolerance of our SAC approach with other input selection strategies to reduce simulation runtime without compromising classifier accuracy. Furthermore, future work will include integrating the SAC approach directly with a flight controller to enhance safe autonomous control. We also consider potential improvements, such as experimenting with normalizing both input parameters and the expected reward of a problem to achieve more consistent results in cases with rapidly changing reward functions. We will also consider methods of detecting more intricate sensor failures such as loss of data.

ACKNOWLEDGMENT

Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents.

REFERENCES

[1] A. Straubinger, R. Rothfeld, M. Shamiyeh, K. -D. Büchter, J. Kaiser, and K. O. Plötner, "An overview of current research and developments in urban air mobility–setting the scene for UAM introduction," *J. Air Transport Manage.*, vol. 87, 2020, Art. no. 101852.
[2] W. R. Johnson, P. Lu, and S. Stachowiak, "Automated re-entry system using FNPEG," in *Proc. AIAA Guid., Navigation, Control Conf.*, 2017, p. 1899.
[3] V. Shukla, A. Gelsey, M. Schwabacher, D. Smith, and D. D. Knight, "Automated design optimization for hypersonic inlets," *AIAA J. Aircr.*, 1996.
[4] M. Vrdoljak, O. Halbe, T. Mehling, and M. Hajek, "Flight guidance concepts to mitigate flight control system degradation in urban air mobility scenarios," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 38, no. 5, pp. 18–33, May 2023.
[5] T. Walsh et al., "Support vector machines for estimating decision boundaries with numerical simulations," Sandia Nat. Lab., Albuquerque, NM, USA, Tech. Rep. SAND2022-11061, 2022.
[6] W. Aquino et al., "Assessing decision boundaries under uncertainty," *Struct. Multidisciplinary Optim.*, vol. 67, 2024, Art. no. 113.
[7] A. Basudhar and S. Missoum, "An improved adaptive sampling scheme for the construction of explicit boundaries," *Struct. Multidisciplinary Optim.*, vol. 42, pp. 517–529, 2010.
[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, 2019.
[9] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (PINNs) for fluid mechanics: A review," *Acta Mechanica Sinica*, vol. 37, no. 12, pp. 1727–1738, 2021.
[10] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks for heat transfer problems," *J. Heat Transfer*, vol. 143, no. 6, 2021, Art. no. 060801.
[11] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *Int. J. Neural Syst.*, vol. 19, no. 04, pp. 295–308, 2009.
[12] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[13] R. Zheng, X. Yi, W. Lu, and T. Chen, "Stability of analytic neural networks with event-triggered synaptic feedbacks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 483–494, Feb. 2016.

[14] I. R. Goumiri, B. W. Priest, and M. D. Schneider, "Reinforcement learning via Gaussian processes with neural network dual kernels," in *Proc. 2020 IEEE Conf. Games*, 2020, pp. 1–8.

[15] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*, vol. 2. Cambridge, MA, USA: MIT Press, 2006.

[16] C. K. Williams and D. Barber, "Bayesian classification with Gaussian processes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 12, pp. 1342–1351, Dec. 1998.

[17] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, 1995.

[18] S. Feng, H. Zhou, and H. Dong, "Using deep neural network with small dataset to predict material defects," *Materials Des.*, vol. 162, pp. 300–310, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0264127518308682

[19] M. Guillaumin, J. Verbeek, and C. Schmid, "Multimodal semi-supervised learning for image classification," in *Proc. 2010 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 902–909.

[20] M. Leonesio and L. Fagiano, "A semi-supervised physics-informed classifier for centerless grinding operations," in *Proc. 2022 IEEE Conf. Control Technol. Appl.*, 2022, pp. 977–982.

[21] Y. A. Yucesan and F. A. Viana, "Hybrid physics-informed neural networks for main bearing fatigue prognosis with visual grease inspection," *Comput. Industry*, vol. 125, 2021, Art. no. 103386.

[22] J. Janisch, T. Pevny, and V. Lisy, "Classification with costly features using deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 01, pp. 3959–3966.

[23] G. Kampmann, N. Kieft, and O. Nelles, "Support vector machines for design space exploration," in *Proc. World Congr. Eng. Comput. Sci.*, 2012, vol. 2, pp. 1116–1121.

[24] M. E. Cholette, P. Borghesani, E. Di Gialleonardo, and F. Braghin, "Using support vector machines for the computationally efficient identification of acceptable design parameters in computer-aided engineering applications," *Expert Syst. Appl.*, vol. 81, pp. 39–52, 2017.

[25] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," in *Comput. Res. Repository*, 2017.

[26] K. Lee, S. Yun, K. Lee, H. Lee, B. Li, and J. Shin, "Robust inference via generative classifiers for handling noisy labels," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3763–3772.

[27] L. Bruzzone and C. Persello, "A novel context-sensitive semisupervised SVM classifier robust to mislabeled training samples," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 7, pp. 2142–2154, Jul. 2009.

[28] J. Wang, Y. Liu, and B. Li, "Reinforcement learning with perturbed rewards," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 04, pp. 6202–6209.

[29] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1352–1361.

[30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

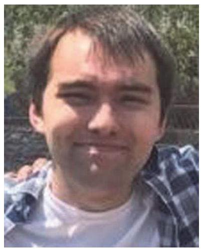
[31] R. Fox, A. Pakman, and N. Tishby, "G-Learning: Taming the noise in reinforcement learning via soft updates," *Comput. Res. Repository*, 2015.

[32] J. Netter, K. G. Vamvoudakis, T. F. Walsh, and J. Ray, "Decision boundary estimation using reinforcement learning for complex classification problems," in *Proc. 63rd IEEE Conf. Decis. Control*, 2024, pp. 3396–3401.

[33] V. Kuleshov and D. Precup, "Algorithms for multi-armed bandit problems," *Comput. Res. Repository*, 2014.

[34] Y. Wang, J. -Y. Audibert, and R. Munos, "Algorithms for infinitely many-armed bandits," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, vol. 21, pp. 1729–1736.

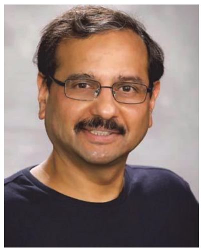
[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.



JOSH NETTER (Graduate Student Member, IEEE) is from Athens, Georgia, USA. He received the bachelor's degree in aerospace engineering and computer science from the Georgia Institute of Technology, Atlanta, GA, USA, in 2018, and the Master of Science degree in aerospace engineering in 2020 from Georgia Tech, Atlanta, GA, where he is currently working toward the Ph.D. degree under Dr. Kyriakos G. Vamvoudakis with the Guggenheim School of Aerospace Engineering. He is a Graduate Research Assistant with the Guggenheim School of Aerospace Engineering, Georgia Tech. His expertise is primarily in control theory, safe autonomy, game theory, bounded rationality, urban air mobility, and reinforcement learning.

KYRIAKOS G. VAMVOUDAKIS (Senior Member, IEEE) was born in Athens, Greece. He received the Diploma (graduating with highest Hons.) in electronic and computer engineering (equivalent to a Master of Science) from the Technical University of Crete, Kounoupidiana, Greece, in 2006, and the M.S. and Ph.D. degrees in electrical engineering from The University of Texas at Arlington, Arlington, TX, USA, under the guidance of Frank L. Lewis, in 2008 and 2011, respectively. From 2011 to 2012, he was an Adjunct Professor and Faculty Research Associate with The University of Texas at Arlington and the Automation and Robotics Research Institute. Between 2012 and 2016, he was a Project Research Scientist with the Center for Control, Dynamical Systems, and Computation, University of California, Santa Barbara, Santa Barbara, CA, USA. He joined the Kevin T. Crofton Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, VA, USA, as an Assistant Professor, till 2018. He is currently the Dutton-Ducoffe Endowed Professor with the Daniel Guggenheim School of Aerospace Engineering, Georgia Tech, Atlanta, GA, USA. He holds a secondary appointment with the School of Electrical and Computer Engineering. His expertise is in reinforcement learning, control theory, game theory, cyber-physical security, bounded rationality, and safe/assured autonomy.

TIMOTHY F. WALSH received the Ph.D. degree in computational and applied mathematics from The University of Texas at Austin, Austin, TX, USA, in 2001. He is a Distinguished Member of Technical Staff with Sandia National Laboratories, NM. His research interests include optimization, inverse methods, and machine learning in computational mechanics.



JAIDEEP RAY received the Ph.D. degree in mechanical and aerospace engineering from Rutgers, The State University of New Jersey, New Brunswick, NJ, USA, in 1999. He is a Distinguished Member of the Technical Staff with Sandia National Laboratories, CA. His research interests include machine learning, bayesian inference using scientific and engineering models, and high performance computing. He has contributed to compressible fluid dynamics and turbulence modeling. He has developed high-order methods for simulating reactive flows on block-structured adaptive meshes. He maintains and distributes open-source software. Details can be found at http://www.sandia.gov/-jairay