AIAA SciTech Forum
23-27 January 2023, National Harbor, MD & Online
AIAA SCITECH 2023 Forum

10.2514/6.2023-1070

Verification of Adversarially Robust Reinforcement Learning Mechanisms in Aerospace Systems

Taehwan Seo*, Prachi P. Sahoo,† and Kyriakos G. Vamvoudakis‡
Georgia Institute of Technology, Atlanta, Georgia, 30313

In this paper, we present a detailed framework for the verification and validation of learning-based reinforcement learning (RL) mechanisms in aerospace control software. First, we integrate an adversarial input mitigation and moving target defense framework and verify its efficacy in real-time. Then, we provide a testing framework to verify the robustness of closed-loop RL mechanisms. The reliability of the adversarially robust RL mechanism is tested using the VerifAI toolkit and an X-plane 11 **Cessna** 172**.**

I. Nomenclature

$t \in [t_0, \infty)$	=	Time; independent unless stated
$t_0 \in \mathbb{R} \geq 0$	=	Initial time
$t_s \in \mathbb{R} > 0$	=	Switching time
$x(t) : [t_0, \infty) \to \mathbb{R}^n$	=	State variable; $x = [x_1, ..., x_n]^{\mathrm{T}}$
$u(t) : [t_0, \infty) \to \mathbb{R}^m$	=	Control input
$u_b(t) : [t_0, \infty) \to \mathbb{R}^m$	=	Behavioral policy
$c \in \mathbb{R}^n$	=	Desired state (time invariant or time varying)
$v(t) : [t_0, \infty) \to \mathbb{R}^m$	=	Input deviation $v := u_b - u$
$f(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$	=	Unknown nonlinear state function
$g(\cdot) : \mathbb{R}^n \to \mathbb{R}^{n \times m}$	=	Unknown nonlinear input function
$\epsilon \in \mathbb{R} > 0$	=	Convergence threshold
$\lambda \in (-0.2, 0.2)$	=	Adversarial gain
$\psi^u : \mathbb{R}^n \times [t_k, t_l] \to \mathbb{R}^{N_u}$	=	Actor basis function
$\psi^v : \mathbb{R}^n \times [t_k, t_l] \to \mathbb{R}^{N_v}$	=	Critic basis function
$\hat{W}^u \in \mathbb{R}^{N_u \times m}$	=	Actor weight matrix
$\hat{W}^v \in \mathbb{R}^{N_v}$	=	Critic weight matrix
$\mathcal{F} \subseteq \mathbb{R}^n$	=	Free space
$\otimes$	=	Kronecker product
$\oslash$	=	Hadamard division
$\mathrm{vec}(\cdot)$	=	Vectorization of a matrix
$\mathrm{diag}([\cdot])$	=	Diagonal matrix
$\neg$	=	Negation
$\wedge$	=	Conjunction

II. Introduction

Artificial Intelligence (AI), widely implemented in autonomy [1–4] is the perceiving, synthesizing, and inferring intelligence demonstrated by machines, as opposed to that displayed by animals and humans. Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents make decisions and take actions in an environment to maximize a user-defined cumulative reward. RL does not require labeled data or the need to correct sub-optimal actions explicitly. Instead, the focus is on finding a balance between exploration of uncharted territory and exploitation of current knowledge. RL, in the context of nonlinear control problems, generates optimal control policies

*Graduate Student, The Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology.
†Graduate Student, The Woodruff School of Mechanical Engineering, Georgia Institute of Technology.
‡Dutton-Ducoffe Endowed Professor, The Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology.

for known and unknown systems in the presence of several perturbations including observation noise, adversarial attacks, and response errors. RL is one of the key methods to solve optimal control problems [5, 6], as it implements learning methods such as Q-learning [7], off-policy learning, policy iteration, and actor-critic [6] structures. These methods may also find applications in non-linear systems with unknown dynamical model parameters [8]. This makes RL a powerful and critical tool for the control of Cyber-Physical Systems (CPS) and necessitates the importance of verification and robustness checks of RL mechanisms using specific performance measures.

Specifically, the purpose of verification is to check if the performance of the system is within the desired design and operation limits. Such performance evaluation is mandated by the black/grey-box nature of RL [9]. Since the cost of fixing errors and solving problems to align with requirements exponentially grows with time [10], it is crucial to perform diagnostics in the early engineering stages. Verification in RL-based control software can largely be divided into performance and stability verification methods [11].

Performance verification [12, 13] measures the effect on performance of chosen algorithm after training with different methods such as, neural network controllers [14, 15], optimal policy learning [16], and actor/critic structures [17] via a series of testing protocols. Sensitivity analysis measures maximum deviation of the algorithm's output from desired output [18]. Falsification analysis tests the model with different environmental parameters and conditions to verify robustness [19]. Another form of performance verification is adversarial robustness which checks the response of the model in the presence of adversarial inputs [20].

Stability verification monitors unstable states of a system during policy learning. Using Lyapunov stability theory [14, 21, 22], it limits unstable learning and flags unstable states via mathematical monitoring, i.e., reachability analysis. Reachability analysis estimates the possible sets of network outputs over some input distribution using optimality measures derived from Hamilton-Jacobi theory [23–25].

Contributions: The main contributions of this work are twofold. First, this work presents the integration of attack mitigation RL control with a verification toolkit for an explicit model verification process. Secondly, this work presents a framework for the verification testbed to evaluate RL algorithm effectiveness through simulation. When used in tandem, the model verification contributions in this work will enable an algorithm engineer to evaluate the reliability of any given attack mitigation algorithm and assess the performance of nonlinear control algorithms over their objectives.

Outline: Section III describes the problem formulation based on an off-policy RL and an on-off adversarially robust learning mechanism. A verification framework named "VerifAI" [26] is used to provide a test-bed for simulating the integrated algorithm and evaluating the performance of the model in Section IV. Section V provides an experimental setup for an X-plane 11 Cessna 172 to carry out proposed verification process in the presence of adversarial inputs. Finally, Section VI presents conclusions and briefly discusses future work.

III. Problem Formulation

A. Preliminaries

Definition 1. A time sequence $t_1, t_2, \ldots$ is a (infinite unless otherwise stated) sequence of time values $t_j \in \mathbb{R}_{\geq 0}$, for all $j \in \mathbb{N}$, satisfying (i) $t_j < t_{j+1}$, for all $j \in \mathbb{N}$ and (ii) for all $t' \in \mathbb{R}_{\geq 0}$ there exists $j \geq 1$ such that $t_j \geq t'$. □

An *atomic proposition* is a statement over the variables or parameters of a problem that is either True ($\top$) or False ($\bot$). Assume now that $\mathcal{AP}$ is a finite set of such propositions.

Definition 2. Let $\mathcal{AP}$ be a finite set of atomic propositions. A *timed word* w over $\mathcal{AP}$ is an infinite sequence $w \coloneqq (w_1, t_1)(w_2, t_2)\ldots$ where $w_1, w_2, \ldots$ is an infinite word over $2^{\mathcal{AP}}$ and $t_1, t_2, \ldots$ is a time sequence. □

Definition 3. A *Weighted Transition System (WTS)* is a tuple $(\Pi, \Pi_0, \longrightarrow, \mathcal{AP}, \mathcal{L}, \gamma)$, where Π is a finite set of states, $\Pi_0 \subseteq \Pi$ is a set of initial states, $\longrightarrow \subseteq \Pi \times \Pi$ is a transition relation, $\mathcal{AP}$ is a finite set of atomic propositions, $\mathcal{L} : \Pi \to 2^{\mathcal{AP}}$ is a labeling function, and $\gamma : \longrightarrow \to \mathbb{R}_{>0}$ is a map that assigns a positive weight to each transition. □

Definition 4. A *timed run* of a WTS is an infinite sequence $r = (r_1, t_1)(r_2, t_2)\ldots$, such that $r_1 \in \Pi_0$ and $r_j \in \Pi$, $(r_j, r_{j+1}) \in \longrightarrow$, for all $j \in \mathbb{N}$. The time stamps t_j are inductively defined with $t_1 = 0$ and $t_{j+1} = t_j + \gamma(r_j, r_{j+1})$, for all $j \in \mathbb{N}$. The timed run r generates a timed word $w(r) = w_1(r_1), w_2(r_2), \ldots = (\mathcal{L}(r_1), t_1), (\mathcal{L}(r_2), t_2), \ldots$ over the set $2^{\mathcal{AP}}$, where $\mathcal{L}(r_j)$ is the subset of atomic propositions $\mathcal{AP}$ that are true at state r_j at time t_j, $j \in \mathbb{N}$. □

The syntax of a timed temporal logic formula The timed temporal logic formula φ over $\mathcal{AP}$ is defined by a grammar that has the form

$$\varphi \coloneqq p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc_I \varphi \mid \Diamond_I \varphi \mid \Box_I \varphi \mid \varphi_1 \mathcal{U}_I \varphi_2, \tag{1}$$

where $\varphi \in \mathcal{AP}$, and $\bigcirc_{(.)}$, $\Diamond_{(.)}$, $\Box_{(.)}$, and $\mathcal{U}_{(.)}$ are the next, future, always, and until operators, respectively; the subscript I is a nonempty time interval in one of the followings forms: $[i_1, i_2]$, $[i_1, i_2)$, $(i_1, i_2]$, (i_1, i_2), $[i_1, \infty)$, (i_1, ∞), with $i_1, i_2 \in \mathbb{Q}$. φ_1 and φ_2 are two different timed temporal logic formulas, and p is one of *atomic proposition*. Several languages are subsets of the form (1), such as Metric Temporal Logic (MTL), Metric Interval Temporal Logic (MITL), Bounded MTL, coFlat MTL, or Time Window Temporal Logic (TWTL) [27, 28]. Here we define the generalized semantics of (1) over discrete observations (point-wise semantics) [29]. The next definition considers the satisfaction of a formula by a timed run.

Definition 5. [29, 30] Given a sequence $\mathsf{R} = (\pi_0, t_0)(\pi_1, t_1)\ldots$ and a timed formula φ, we define $(\mathsf{R}, i) \models \varphi$, $i \in \mathbb{N}_0$ (R satisfies φ at i) as follows:

$$
\begin{aligned}
&(\mathsf{R}, i) \models p \Leftrightarrow p \in \mathcal{L}(\pi_i), \qquad (\mathsf{R}, i) \models \neg\varphi \Leftrightarrow (\mathsf{R}, i) \not\models \varphi,\\
&(\mathsf{R}, i) \models \varphi_1 \wedge \varphi_2 \Leftrightarrow (\mathsf{R}, i) \models \varphi_1 \text{ and } (\mathsf{R}, i) \models \varphi_2,\\
&(\mathsf{R}, i) \models \bigcirc_I \varphi \Leftrightarrow (\mathsf{R}, i+1) \models \varphi \text{ and } t_{i+1} - t_i \in I,\\
&(\mathsf{R}, i) \models \varphi_1 \mathcal{U}_I \varphi_2 \Leftrightarrow \exists k \geq i \text{ such that } (\mathsf{R}, k) \models \varphi_2,\\
&\qquad\qquad t_k - t_i \in I \text{ and } (\mathsf{R}, m) \models \varphi_1, \forall m \in \{i, \ldots, k\}.
\end{aligned}
$$

Also, $\Diamond_I \varphi = \top \mathcal{U}_I \varphi$ and $\Box_I \varphi = \neg\Diamond_I \neg\varphi$. Finally, R satisfies φ, denoted by $\mathsf{R} \models \varphi$, if and only if $(\mathsf{R}, 0) \models \varphi$. □

B. Off-Policy RL

Consider a system with the following nonlinear dynamics,

$$\dot{x}(t) = f(x(t)) + g(x(t))u(t), \quad x(t_0) = x_0, \ t \geq 0 \tag{2}$$

$$\dot{e}(t) = F(e(t)) + G(e(t))u(t) := f(e(t) + c(t)) + g(e(t) + c(t))u(t) \tag{3}$$

where $e(t) := x(t) - c_(t) \in \mathbb{R}^n$ is the error between the state $x \in \mathbb{R}^n$ and the state of interest $c \in \mathbb{R}^n$. x_0 is the initial state of the system and $u \in \mathbb{R}^m$ is the control input. Let the time window $[t_k, t_l]$, $k, l \in \mathbb{N}$ define a duration of T seconds. Finally $f : \mathbb{R}^n \to \mathbb{R}^n$, $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are unknown, locally Lipschitz functions that define the plant and the input functions for the dynamical system in (2) and (3). $F : \mathbb{R}^n \to \mathbb{R}^n$ and $G : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are state and input function modified from f and g with the changed in states $e(t)$ from $x(t)$. Consider $K \in \mathbb{N}$ points of interest in the state space, denoted by $c_k \in \mathbb{R}^n$, for $k \in \mathcal{K} := \{1, \ldots, K\}$. Each point c_k, $k \in \mathcal{K}$, corresponds to certain properties of interest, which are expressed as Boolean variables via the finite set of atomic propositions $\mathcal{AP}$.

The background in transitions Π and timed formula φ that are from [31] are preliminaries for safe transition execution between different states of interest and corresponding phases of control in learning-based control mechanisms. In this learning-based control framework, the development of control strategies with optimality-based transitions Π, and the timed-behavior, given in the definition below, is defined with the eventual satisfaction of timed formula φ.

Definition 6. Consider an agent trajectory $x : [t_0, \infty) \to \mathbb{R}^n$ of (2). Then, a *timed behavior* of x is the infinite sequence $\mathsf{b}(t_0) := (x(t_0), \sigma_0, t_0)(x(t_1), \sigma_1, t_1)\ldots$, where $t_0, t_1, \ldots$ is a time sequence according to Definition 1, $x(t_i) \in \pi_{j_i}$, $j_i \in \mathcal{K}$ for all $i \in \mathbb{N}_0$, and $\sigma_i = \mathcal{L}(\pi_{j_i}) \subseteq 2^{\mathcal{AP}}$, i.e., the subset of atomic propositions that are true when $x(t_j) \in \pi_{j_i}$, for $i \in \mathbb{N}_0$. The timed behavior b satisfies a timed formula φ *safely* if $\mathsf{b}_\sigma(t_0) := (\sigma_0, t_0)(\sigma_1, t_1)\ldots \models \varphi$ and $x(t) \in \mathcal{F}$, for all $t \geq t_0$. It *eventually* satisfies φ safely if there exists $j \in \mathbb{N}$ such that $\mathsf{suff}(b_\sigma(t_0), j) = \mathsf{suff}(a_\sigma(t_0), j)$, $\big(\mathsf{suff}(\mathsf{s}, j) = s_j s_{j+1}\ldots\big)$ for some $a_\sigma(t_0) \models \varphi$ and $x(t) \in \mathcal{F}$, for all $t \geq t_j$. □

The performance criteria for all $i \in \mathcal{K}$ is given by:

$$J(e_i(t_0), t_0, t_f, u) := \int_{t_0}^{t_f} r(e_i(\tau), u(e_i(\tau), \tau))d\tau \tag{4}$$

where $t_0 \geq 0$, $t_f > t_0$. The $r(e, u) := Q(e) + S(u)$ is metric of performance with $S(u) := u^{\mathrm{T}} R u, R > 0$ and $Q : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ being positive-semi-definite function. This gives rise to the *timed behavior cost* of a timed behavior b.

In the time window (t_k, t_l) the state of the system $x(t)$ will transition from region of interest π_k to region of interest π_l in an optimal manner using the finite weighted transition system,

$$\mathcal{T} := (\Pi, \Pi_0, \longrightarrow, \mathcal{AP}, \mathcal{L}, \gamma). \tag{5}$$

Such a state transition process is interpreted within the context of optimal timed transition $\pi_k \longrightarrow \pi_l$ with the following assumptions:

1) $x(t_k) \in \pi_k$ and $x(t) \in \pi_l$ for all $t \in (t_k, t_l)$
2) $x(t) \in \mathcal{F} \setminus \bigcup_{m \in \mathcal{K}} \pi_m$ for all $t \in [t_k, t_l]$
3) $u(x(t), t) = \arg J_i^\star$, where $J_i^\star = \min_{\alpha \in \mathcal{A}(t_k, t_l)} J(e_l(t_k), t_k, t_l, \alpha)$

where $\mathcal{A}$ is set of all functions $\mathbb{R}^n \times [t_a, t_b] \to \mathbb{R}^m, (t_b > t_a \geq 0)$, $\mathcal{F}$ is free space $\mathbb{R}^n$. Append the existing cost functional to include the above constraints, we obtain

$$J_l(e_l(t_k), t_k, t_l, u) := \int_{t_k}^{t_l} \big(\gamma r(e_l(s), u(e_l(s), s)) + L_{k,l}(e_l(s))\big) \mathrm{d}s + \phi(e_l(t_l)). \tag{6}$$

In this research of performance verification for learning-based control in the presence of adversarial attacks, the multiple states of interest are considered as isolated missions for the verification analysis. We quantify the algorithm performance of each missions with a proposed verification toolkit that is introduced in Section III.D. Each of the separate missions have $K = 1$, $\varphi = \square\{1\}$ with consideration of a single state of interest. $L_{k,l}(\cdot) > 0$ is the penalty for the unsafe zones, and $\gamma > 0$ is the trade-off factor between $L_{k,l}(\cdot)$ and $r(\cdot, \cdot)$. $r := Q(e) + S(u)$ is the performance measure, with $Q(e) := e^{\mathrm{T}} Q_m e$, $Q_m \geq 0$, $S(u) := u^{\mathrm{T}} R u$, $R > 0$, and $\phi : \mathbb{R}^n \to \mathbb{R} \geq 0$ is a positive definite term that penalizes the deviation of the terminal state at t_l from the point or trajectory of interest c.

The optimal value function associated with (3), a continuously differentiable system-dynamics, is given by $V^\star(e, t) = \min_u J(e, t, u)$. The corresponding minimizing control policy $u^\star$ can be written as,

$$u^\star(e, t) = -\frac{1}{2\gamma} R^{-1} G(e)^{\mathrm{T}} \nabla_e V^\star(e, t), \forall e,\ t \geq 0 \tag{7}$$

and the solution for the optimal value function $V^\star$ can be found by solving the following partial differential equation $\forall e,\ t \geq 0$,

$$\begin{gathered} \nabla_t V^\star(e, t) + \nabla_e V^\star(e, t)^{\mathrm{T}} (F(e) + G(e) u^\star) + \gamma r(e, u^\star) + L(e) = 0, \\ \forall t \in [t_k, t_l),\ V^\star\left(e\left(t_l\right), t_l\right) = \phi\left(e\left(t_l\right)\right). \end{gathered} \tag{8}$$

It is generally hard to solve (8) directly. Thus, we will approximate the value function $V^\star$ using a critic approximator and the optimal control $u^\star$ using an actor approximator. Given iterations $i \in \mathbb{N}$, we denote the value function approximation as $\hat{V}^{u_i}$ and the control input as $\hat{u}_{i+1}$, where $\lim_{i \to \infty} \hat{V}^{u_i} = V^\star$ and $\lim_{i \to \infty} \hat{u}_i = u^\star$. The current critic and actor approximators are given $\forall e,\ t \geq 0$ as follows,

$$\hat{V}^{u_i}(e, t) := (\hat{W}_i^v)^{\mathrm{T}} \psi^v(e, t) + \phi(e) \tag{9}$$

$$\hat{u}_{i+1}(e, t) := (\hat{W}_i^u)^{\mathrm{T}} \psi^u(e, t). \tag{10}$$

By combining (7), (8) with (9) and (10) the update equation for the combined actor and critic weight matrices, $\hat{W}_{i+1}$, is expressed as,

$$\hat{W}_{i+1} = -\left(\sum_{j=0}^{l} \Theta_{j,i}^{\mathrm{T}} \Theta_{j,i}\right)^{-1} \left(\sum_{j=0}^{l} \Theta_{j,i}^{\mathrm{T}} \Psi_{j,i}\right) \tag{11}$$

where $\Theta_{j,i} = [\Theta_{j,i}^v \Theta_{j,i}^u]$. The relationship for $\Theta_{j,i}^v$ is given by $\Theta_{j,i}^v = \left(\psi^v(e(\tau_{j+1}), \tau_{j+1}) - \psi^v(e(\tau_j), \tau_j)\right)^{\mathrm{T}}$ and for $\Theta_{j,i}^u$ is given by $\Theta_{j,i}^u = \int_{\tau_j}^{\tau_{j+1}} 2\gamma \left((\hat{v}_i(e, \tau)^{\mathrm{T}} R) \otimes \psi^u(e, \tau)^{\mathrm{T}}\right) \mathrm{d}\tau$. Moreover, the actor-critic combined weights are given by,

$$\hat{W}_i = [(\hat{W}_i^v)^{\mathrm{T}} \mathrm{vec}(\hat{W}_i^u)^{\mathrm{T}}]^{\mathrm{T}} \tag{12}$$

and Ψ, the cost to go over a small duration of time τ_j to τ_{j+1} is given by $\Psi = V^{u_i}(e(\tau_{i+1}), \tau_{i+1}) - V^{u_i}(e(\tau_i), \tau_i) + \int_{\tau_j}^{\tau_{j+1}} (\gamma Q(e) + L(e) + \gamma S(\hat{u}_i(e, \tau)))\, \mathrm{d}\tau$.

Note that $\tau_j = t$ and $\tau_{j+1} = t + T$, where $j \in \{0, ..., N\}$ and $t_k = \tau_0 < \tau_1 < ... < \tau_N = t_l$ are the time stamps when (11) updates the actor-critic weights until they converge with respect to $\epsilon > 0$. The Algorithm 1 summarises off-policy RL.

Algorithm 1: Off-Policy RL

Data: Learning parameters, Behavioral policy
Result: Simulation data
begin
  for $t = 0 : t_b$ **do**
    Employ arbitrary behavioral policy u_b.
    Collect input/state/switch data from system (3).

  for $k = 1 : K$ **do**
    Select $\epsilon > 0$, i = 0.
    while $||\hat{W}_{i+1} - \hat{W}_i|| > \epsilon$ **do**
      Solve for $\hat{W}_{i+1}$ from equation (11).
      Extract $\hat{W}^v_{i+1}$ and $\hat{W}^u_{i+1}$ from $\hat{W}_{i+1}$ by (12).
      Update index i by $i = i + 1$.
    $\hat{W}^u_{\text{optimal},k} = \hat{W}^u_{i+1}$.

  for $t = t_b : t_f$ **do**
    Simulate the dynamical system with learned policy with $\hat{W}^u_{\text{optimal},k}$ with (10) by different c_k.
    Collect input-state data from system (3).

C. On-Off Adversarially Robust RL

To minimize the effect of adversarial signals and increase the unpredictability of our RL policy, we will use a moving target defense (MTD) mechanism by incorporating an on-off switching mechanism in the controller gain matrix [32]. Specifically, the initial behavioral policy is designed to incorporate switching. This gain switching is performed as follows,

$$K_i = \begin{cases} k, & (i = \text{switch}) \\ 1, & (i \neq \text{switch}) \end{cases} \tag{13}$$

$$u_j = \sum_{i=0}^{n} -K_i \cdot \tan\left(\frac{h_i(x_i)}{b_i}\right) \tag{14}$$

$$u = [u_1, u_2, ..., u_m]^{\mathrm{T}} \tag{15}$$

$$u_b = \tanh\left(u \oslash D\right) \tag{16}$$

Where control input elements $u_j, \forall j \in \{1, 2, 3, ..., m\}$ are formed with sum of each state feedback that is multiplied with scalar gain $K_i, \forall i \in \{1, 2, 3, ..., n\}$. These elements K_i are either $k \in \mathbb{R} > 1$ or 1 if i = switch or $i \neq$ switch respectively, as shown in (13). The switch is randomly selected from $\{1, 2, 3, ..., n\}$ over different time blocks. $D \in \mathbb{R}^m$ is the input constraint, and $b_i \in \mathbb{R}$ is the state constraint. Also, $h_{(\cdot)}\left(x_{(\cdot)}\right)$ is an arbitrary function that aids exploration and includes an observation noise and preliminary control input for initial data collection of the system.

To apply of switching actuators, the gain K_i is multiplied with the corresponding arbitrary state function, $h_i(x_i)$, for each of the actuator switch modes. Adversarial noise is applied to the behavioral policy as follows,

$$\begin{aligned} u_{\text{attack}} &= \lambda u_b \\ u_{\text{b, attacked}} &= (1 + \lambda) u_b \end{aligned} \tag{17}$$

where $\lambda \in (-0.2, 0.2)$ is user-defined. The behavioral policy u_b is modulated with multiplier ratio λ that is chosen either randomly within the pre-defined range or specifically worst-case choices in the range. In contrast, the behavioral policy that is calculated over the vanilla RL for optimality is not modulated. When exposed to adversarial noise, there is a discrepancy between $u_{b,attacked}$ applied to the system and u_b that is used for policy iteration, and this will hinder the effective learning of the optimal policy. The procedure for adversarial noise addition is summarized in Algorithm 2.

Algorithm 2: On-Off Switching Arbitrary Behavioral Policy u_b

Output : Behavioral policy u_b

begin
- **for** *every* t_s *seconds* **do**
  - Randomize switch from $[1, 2, 3, 4, ..., n] \in \mathbb{N}$.
- Get K_i from (13) with arbitrary gain k.
- Compute $u_j \ \forall j \in \{1, 2, 3, \ldots m\}$ as per (14).
- Compute u_b as given by (16).
- Add adversarial noise u_{attack} to u_b by (17).

A filtered update of weight matrix is applied to off-policy learning. Since the off-policy learning updates the actor weight $\hat{W}^u$ by (11), the gradient of the actor weight is calculated as $\Delta\hat{W}^u_i = \hat{W}^u_{i+1} - \hat{W}^u_i$, then pruned as,

$$\hat{W}^u_{i,\text{grad}} = \begin{cases} \Delta\hat{W}^u_i & \in \text{3s steepest gradients among } \Delta\hat{W}^u_i \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

Where s is number of states, which is same as n. This $\hat{W}^u_{i,\text{grad}}$ is added to the current weight matrix $\hat{W}^u_i$ to get the updated value, $\hat{W}^u_{i+1,\text{prune}}$. This process is summarized in Algorithm 3.

Algorithm 3: Tuning with Switching

Input : $(\hat{W}^u_{i+1}, \hat{W}^u_i)$

Output : $\hat{W}^u_{i+1,\text{prune}}$ which is a pruned version

begin
- **if** $i \geq 2$ **then**
  - Calculate $\Delta\hat{W}^u_i$ from $\hat{W}^u_{i+1} - \hat{W}^u_i$.
  - Sort $\Delta\hat{W}^u_i$ and find 3s steepest gradient index named i_{steep}.
  - Apply (18) to compute $\hat{W}^u_{i,\text{grad}}$ with i_{steep}.
  - Get updated $\hat{W}^u$ as $\hat{W}^u_{i+1,\text{prune}} = \hat{W}^u_{i,\text{grad}} + \hat{W}^u_i$.

After combining the off-policy RL with the on-off switching and tuning actor weights with pruning, we obtain the main RL algorithmic result shown in Algorithm 4.

Algorithm 4: Off-Policy RL with On-Off Switching

Input : Learning parameters, Behavioral policy
Output : Simulation data

begin
  Load global variable parameters.
  for $t = 0 : t_b$ **do**
    Randomize switch.
    Get u_b from **Algorithm 2**; Simulate the dynamical system with u_b.
    Collect input/state/switch data from system (3).

  Select $\epsilon > 0$, i = 0.
  while $||\hat{W}_{i+1} - \hat{W}_i|| > \epsilon$ **do**
    Recall switch information for u_b.
    Solve for $\hat{W}_{i+1}$ from equation (11).
    Extract $\hat{W}^v_{i+1}$ and $\hat{W}^u_{i+1}$ from $\hat{W}_{i+1}$ by (12).
    Get updated $\hat{W}^u_{i+1,\text{prune}}$ from **Algorithm 3** ($\hat{W}^u_{i+1}$,$\hat{W}^u_i$).
    Replace $\hat{W}^u_{i+1}$ with $\hat{W}^u_{i+1,\text{prune}}$.
    Update index i by $i = i + 1$.
  $\hat{W}^u_{\text{optimal}} = \hat{W}^u_{i+1}$.

  for $t = t_b : t_f$ **do**
    Simulate the dynamical system with learned policy with $\hat{W}^u_{\text{optimal}}$ with (10).
    Collect input-state data from system (3).

D. VerifAI

VerifAI [26] software toolkit is a simulation-based AI verification toolkit that verifies AI-based models to guide model-based design improvements. VerifAI proposes a sampling method and analysis table to solve the verification challenges [13]. The verification process can be summarized in the following five steps: (1) environment generation and modeling, (2) falsification, (3) analysis of counterexamples, (4) re-training or finding high-performance parameters, and (5) re-validation of the improved model. First, from the given range of environmental parameters or given set of environments, VerifAI samples combinations of environment parameters from the abstract feature space to test the model via simulation. The abstract feature space, which is a set of test environments in which the model will be tested, dictates what combinations of test environment parameters will be chosen for verification based on probabilistic parameter selection. The selection and sampling is done by SCENIC [33], which samples the parameters and applies selected information to the simulation. This information includes range, available options, and the probability model of each parameter. These are provided by the verification designer based on the objective of the verification. Next, from the set of simulations, VerifAI finds faulty conditions or environments, and diagnoses faulty cases using analysis with error tables and property monitors. Error tables provide fault cases with sampled parameter information that is applied to the simulation environment, while the property monitor prints out quantitative data of the simulation. From the iterative simulation with sampled parameters, the scores of each performance with arbitrary measure of the system are also recorded in the table. Each iteration of the simulation follows the processes above, providing performance score and parameter selection data for analysis to diagnose weakest part of the model. Finally, the analysis information of the model provides methods to improve and re-train the model to enhance its performance in the weak environments.

IV. Verification Framework Test-bed Design

A. Process Design

We built the entire AI framework and process from the ground up as summarized in Figure 1. This includes all components that are addressed above, with a nonlinear cart-pole simulation in OpenAI Gym [34] and X-plane 11 [35]

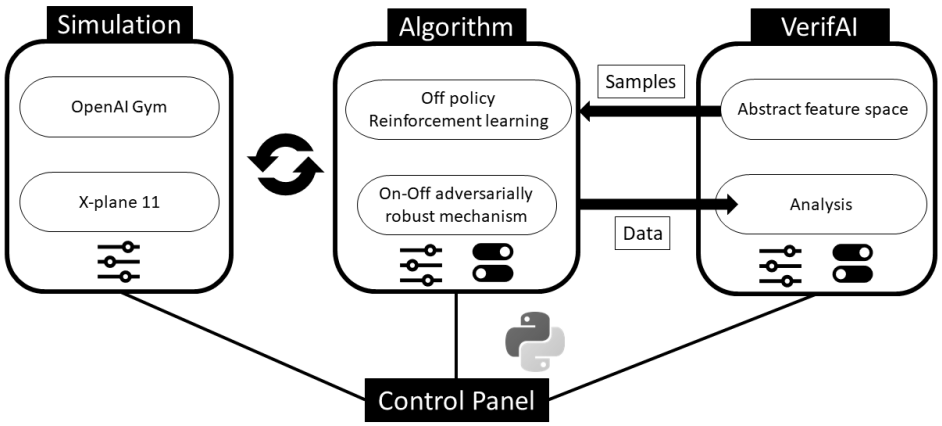


Fig. 1 Diagram description of the test framework.

as a test-bed. The implementation of the proposed structure consists of three parts: (1) algorithmic, (2) simulation, and (3) incorporation into the VerifAI toolkit. The algorithm includes an off-policy learning with an adversarial robustness mechanism incorporating MTD. This is summarized in (16) and Algorithm 2. From its generated abstract space, which are sampled verification parameters including state measures, VerifAI runs each epoch with a different set of sampled parameters. From running VerifAI, it communicates with the algorithm and simulation results to conduct analysis for verification. The entire process can be tuned with a control panel script that has all sets of parameters and switches for effective control of the verification process.

Upon connecting VerifAI to the working model, the framework samples the adversarial perturbation. From the simulation of sampled data, fault cases are tabulated with simulated parameters and performance scores for analysis. The performance score is calculated from the state traces of the system by summing up L_2-norm errors that were captured over the last T seconds window of simulation. This is given by

$$\text{Score} = \sum_{i=T_f-T}^{T_f} ||c - x(i)||_2. \tag{19}$$

The process design is summarized in Algorithm 5, including complete connection of algorithms and verification systems. The N represents the total number of test simulations.

Algorithm 5: Verification Process

Data: Control parameters, Simulation environment
Result: Error table of Falsification

```
begin
    Load global variable parameters and import VerifAI.
    Set parameter to change from environment.

    for 1:N do
        Generate the Sample environment.
        Simulation data = Algorithm 4(sample environment).
        Extract x from simulation data.
        Calculate (Score) as per (19).
        if (Score) ≥ criterion then
            Save environment information with score to error table.
        else
            Save environment information with score to safe table.
```

B. Aerospace Test-Bed

Simulation tool	Supporting languages	Complex state performance
X-plane 11	C, Java, Python, MATLAB	High
Gazebo	C++, ROS	High
Webots	C, C++, Python, Java, MATLAB or ROS	High
Mujoco	XML	Low
PyBullet	Python	Medium
OpenAI Gym	Python	Low

Table 1 Comparison of test framework simulation tools.

The simulation component described in Figure 1 in the verification framework can be replaced with different configurations. Examples for such simulation software are Gazebo, Webots, Pybullet, and Mujoco [36, 37]. The Table 1 shows possible simulation tools with the information on supporting languages and performances. These options are suggested testbeds for the proposed verification framework. Since the verification toolkit, VERIFAI is provided with Python, the simulations need to be compatible with Python to use current setup of the verification. In the options in Table 1, ROS has python client rospy and Mujoco supports python bindings. Therefore, all of the options in the table are available to connect VerifAI for verification. In [38], the authors used VerifAI over the "X-Plane" flight simulator software to verify the neural-net based taxiing system, "TaxiNet." By replacing the existing simulator with a flight simulator, multiple aeronautical vehicles can easily be tested in their use of the on-off adversarially robust algorithm successfully. VerifAI is equipped with the ability to generate an abstract feature space with a set of test conditions in the simulator regardless of simulation tool selection.

V. Experiments

To conduct simulation studies, this work considered a Cessna 172 in X-plane 11 aircraft model [35]. The presented analysis of verification data will show the algorithm's performance with respect to changing adversarial gains.

A. Control Problem

The attitude control problem for the Cessna 172 X-plane 11, shown in Figure 2, will be addressed in this section. The algorithm aims to stabilize the roll and pitch angles of the aircraft with three different control surfaces: aileron,

elevator, and rudder. The states of the system are expressed as $[\theta, \phi, \dot{\theta}, \dot{\phi}]^{\mathrm{T}}$, which are pitch angle, roll angle, pitch angular rate, and roll angular rate respectively. The input u is defined as $[u_a, u_e, u_r]$, which are yoke inputs of the aileron, elevator, and rudder. Each of the input values is limited in a range between $[-1, 1]$, where the 0 value represents an idle control input. The throttle of the aircraft, u_t is fixed to 82% of full throttle. The aircraft is placed in the air at an altitude of 1200 meters above the ground. The reference position of the ground is set as the start line of runway 04 in Grant County International Airport (ICAO: KMHW), facing 4 degrees east from the north, which coincides with the runway heading. The initial speed of the plane in the air is set to 50 m/s, and no initial angular rate is applied.

Fig. 2 X-plane 11 Cessna 172 model.

B. Parameters & Discussion

The cost matrices that define the optimization objective are $Q = \mathrm{diag}([1, 0.3, 0.1, 0.1])$ and $R = \mathrm{diag}([1, 1, 1])$, that prioritize roll then pitch attitudes to be stabilized. The profile of Cessna 172 is simulated with default parameters as described in the operating manual [39]. The X-plane 11 uses blade element coefficient information for the implementation of the aerodynamic calculation. Each of the elements such as the two main wings, fuselage, and the tail wings contain their own dynamical profile with coefficient information, thus simulating the whole body of the aircraft based on rigid body dynamics with attitude and airspeed information of the aircraft. The simulation time step is determined based on the looping time of call iteration of communication. Normally, the simulation step time for algorithm application is 0.1 seconds, however, there exist minor deviations due to communication latency between Python and the X-plane simulation. From the parameter script, the environment parameters such as turbulence, clouds, and weather are selected based on a uniform distribution probabilistic model. Based on the assigned parameter values, the simulation result is converted to a quantitative score given by (19). VerifAI records X-plane simulation for every set of environmental parameters and provides visualizations of each verification test.

C. Algorithm Verification

Figure 3 shows the average penalty scores over the changing adversarial gains with the specific adversarial inputs. The verification is performed with 100 test cases for each sampled value of λ running over 1200 seconds with the behavioral policy and 20 seconds of testing. The score corresponds to the final 20 seconds of the test's output performance calculated using (19). Instances when the algorithm failed to converge were assigned the highest penalty score of 185. Each point on the red and blue curves in the figure represents the average penalty score for each of the 9 chosen values of λ between $[-0.2, 0.2]$.

For comparative studies, the same testing setup was run with the switching mechanism incorporated into the behavioral policy to verify the efficacy of the defense mechanism. The algorithm did not significantly improve performance over the adversary in the presence of negative adversarial gains, but in the presence of positive adversarial gains, the switching mechanism demonstrated significantly improved performance over the RL behavioral policy.

Figure 4 provides sample roll and pitch performance with two different penalty scores, one safe and one unsafe

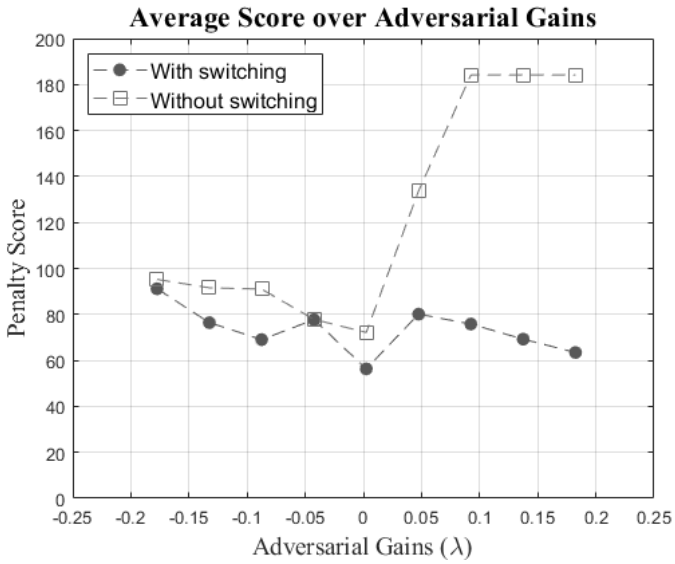
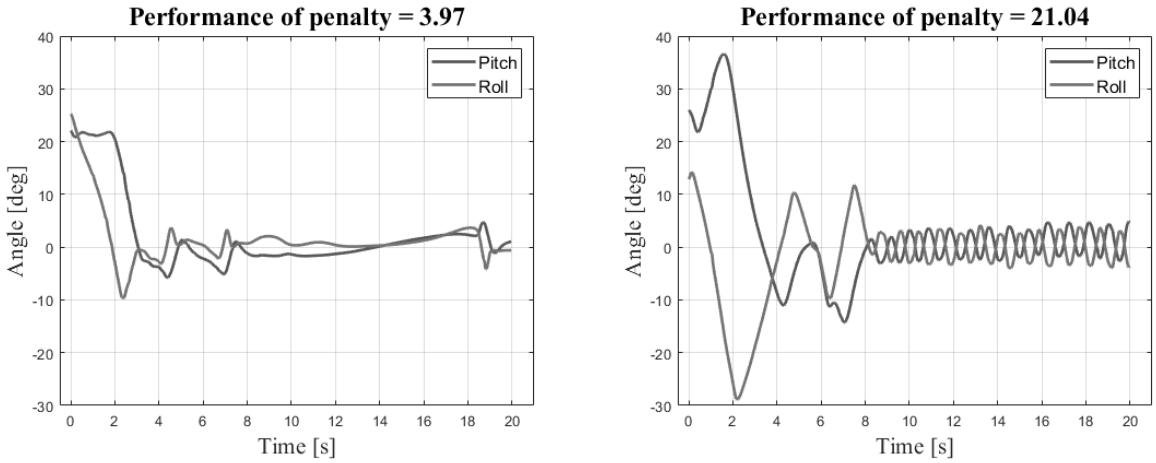


Fig. 3 X-plane algorithm verification test.



(a) Performance of penalty score 3.97.

(b) Performance of penalty score 21.04.

Fig. 4 Performances of X-plane Cessna 172 samples.

performance of the X-plane Cessna 172. Figure 4(a) shows the evolution of the pitch and roll performance with a relatively lower penalty score value of 3.97. The evolution of the pitch and roll angles shown in Figure 4(b) approaches zero average values with high frequency oscillations and stabilization, thus generating a bigger penalty score of 21.04.

D. RL Model Verification

For closed-loop RL model verification with varying environmental conditions on the X-plane, simulation studies are performed with the policy that recorded lowest penalty score of 3.97 from the algorithm verification step.

In this test, as described in Table 2, wind turbulence, cloud level, and rain probability were chosen as changing environment parameters for the verification process. A cumulative number of 500 tests were performed with varying parameter combinations. The resulting correlation coefficients are tabulated in Table 3 and the resulting penalty scores for each combination of environmental parameters are shown in Figure 5.

From Table 3, one can conclude that turbulence is most highly correlated to the penalty score. Rain probability has the least amount of correlation to the performance score, while cloud level strikes halfway between turbulence and rain

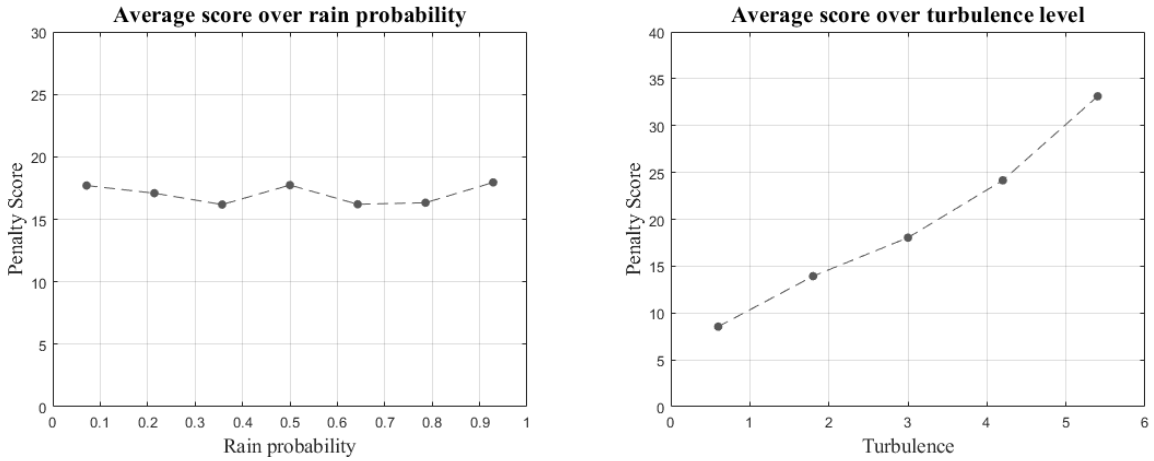
Parameters	Sampling range
Cloud level	0 - 5
Rain percentage	0 - 1
Turbulence	0 - 6

Table 2 X-plane RL model verification parameter information.

Parameters	Correlation coefficient
Cloud level	0.552
Rain probability	0.065
Turbulence	0.924

Table 3 X-plane correlation coefficient with performance penalty score.

probability. Since rain can only occur when cloud levels are 4 or higher, the correlation between cloud level to the model performance score may be higher.



(a) Penalty score with rain probability.

(b) Penalty score with turbulence.

Fig. 5 Score comparison with different environment parameters.

The Figure 5 compares the average penalty scores over rain probability and turbulence level since these features are most and least correlated with the penalty score respectively. Turbulence plot in Figure 5(b) showcases a linear relationship with the penalty score, while rain probability plot in Figure 5(a) presents a flat relationship with the penalty score in the 15 and 20 range. These plots demonstrate how the suggested verification approach aids environmental parameter identification in the context of its impact on model performance and verification of control algorithm of choice.

VI. Conclusion and Future Work

We propose a verification process for attack mitigating control rooted in RL. This work integrates an RL algorithm to solve optimal control problems, an adversarial attack mitigation mechanism, and a VerifAI toolkit. The verification test-bed framework is applied to an X-plane 11 Cessna 172 to successfully evaluate and verify the reliability of off-policy RL. Future work will include working with more complex dynamic systems such as multi-agent networked systems, improvement of the control AI by verification-based re-training, and more nuanced scoring mechanism used for verification.

Acknowledgments

The authors acknowledge partial support for this research provided by NSF (under grant Nos. S&AS-1849198, CPS-2038589, CPS-1851588, and CPS-2227185), by NASA ULI (under grant number 80NSSC20M0161), and by the United States Army (under Contract No. W56HZV-17-C-0095).

References

[1] Gao, W., Jiang, Z. P., and Ozbay, K., "Data-Driven Adaptive Optimal Control of Connected Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 18, No. 5, 2017. https://doi.org/10.1109/TITS.2016.2597279.

[2] Chen, L., Chen, Y., Yao, X., Shan, Y., and Chen, L., "An adaptive path tracking controller based on reinforcement learning with urban driving application," *IEEE Intelligent Vehicles Symposium, Proceedings*, Vol. 2019-June, 2019. https://doi.org/10.1109/IVS.2019.8814130.

[3] Li, C., and Czarnecki, K., "Urban driving with multi-objective deep reinforcement learning," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, Vol. 1, 2019.

[4] Ni, Z., and Paul, S., "A Multistage Game in Smart Grid Security: A Reinforcement Learning Solution," *IEEE transactions on neural networks and learning systems*, Vol. 30, No. 9, 2019. https://doi.org/10.1109/TNNLS.2018.2885530.

[5] Lewis, F. L., Vrabie, D. L., and Syrmos, V. L., *Optimal Control.*, 3rd ed., John Wiley & Sons, 2012.

[6] Lewis, F. L., Vrabie, D., and Vamvoudakis, K. G., "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," , 2012. https://doi.org/10.1109/MCS.2012.2214134.

[7] Vamvoudakis, K. G., "Q-learning for continuous-time linear systems: A model-free infinite horizon optimal control approach," *Systems and Control Letters*, Vol. 100, 2017, pp. 14–20. https://doi.org/10.1016/j.sysconle.2016.12.003.

[8] Song, R., Lewis, F. L., Wei, Q., and Zhang, H., "Off-Policy Actor-Critic Structure for Optimal Control of Unknown Systems with Disturbances," *IEEE Transactions on Cybernetics*, Vol. 46, No. 5, 2016, pp. 1041–1050. https://doi.org/10.1109/TCYB.2015.2421338.

[9] Adadi, A., and Berrada, M., "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," *IEEE Access*, Vol. 6, 2018. https://doi.org/10.1109/ACCESS.2018.2870052.

[10] Haskins, B., Stecklein, J., Dick, B., Moroney, G., Lovell, R., and Dabney, J., "8.4.2 Error Cost Escalation Through the Project Life Cycle," *INCOSE International Symposium*, Vol. 14, No. 1, 2004. https://doi.org/10.1002/j.2334-5837.2004.tb00608.x.

[11] Xiang, W., Musau, P., Wild, A. A., Lopez, D. M., Hamilton, N., Yang, X., Rosenfeld, J., and Johnson, T. T., "Verification for Machine Learning, Autonomy, and Neural Networks Survey," 2018.

[12] Baheti, R., and Gill, H., "Cyber-physical systems," *The impact of control technology*, Vol. 12, No. 1, 2011, pp. 161–166.

[13] Seshia, S. A., and Sadigh, D., "Towards Verified Artificial Intelligence," *CoRR*, Vol. abs/1606.08514, 2016. URL http://arxiv.org/abs/1606.08514.

[14] Zhu, Q., and Guo, L., "Stable adaptive neurocontrol for nonlinear discrete-time systems," *IEEE Transactions on Neural Networks*, Vol. 15, No. 3, 2004, pp. 653–662.

[15] Tran, H.-D., Yang, X., Manzanas, L. D., Musau, P., Nguyen, L. V., Xiang, W., Bak, S., and Johnson, T. T., "NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems," *Computer Aided Verification*, edited by S. K. Lahiri and C. Wang, Springer International Publishing, Cham, 2020, pp. 3–17.

[16] Kamalapurkar, R., Walters, P., Rosenfeld, J., and Dixon, W., *Reinforcement learning for optimal feedback control*, Springer, 2018.

[17] Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A., "Safe model-based reinforcement learning with stability guarantees," *Advances in neural information processing systems*, Vol. 30, 2017.

[18] Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J., "Sensitivity and generalization in neural networks: an empirical study," *arXiv preprint arXiv:1802.08760*, 2018.

[19] Tuncali, C. E., Fainekos, G., Ito, H., and Kapinski, J., "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1555–1562.

[20] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A., "Robust adversarial reinforcement learning," *International Conference on Machine Learning*, 2017, pp. 2817–2826.

[21] Zribi, A., Chtourou, M., and Djemel, M., "A new PID neural network controller design for nonlinear processes," *Journal of Circuits, Systems and Computers*, Vol. 27, No. 04, 2018, p. 1850065.

[22] Tanaka, K., "An approach to stability criteria of neural-network control systems," *IEEE Transactions on Neural Networks*, Vol. 7, No. 3, 1996, pp. 629–642.

[23] Gillula, J. H., and Tomlin, C. J., "Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor," *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 2723–2730.

[24] Bansal, S., Chen, M., Herbert, S., and Tomlin, C. J., "Hamilton-jacobi reachability: A brief overview and recent advances," *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, Vol. 2018-January, 2018. https://doi.org/10.1109/CDC.2017.8263977.

[25] Tran, H.-D., Cai, F., Diego, M. L., Musau, P., Johnson, T. T., and Koutsoukos, X., "Safety verification of cyber-physical systems with reinforcement learning control," *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 18, No. 5s, 2019, pp. 1–22.

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

[27] Bouyer, P., Markey, N., Ouaknine, J., and Worrell, J., "The cost of punctuality," *Proceedings - Symposium on Logic in Computer Science*, 2007. https://doi.org/10.1109/LICS.2007.49.

[28] Vasile, C. I., Aksaray, D., and Belta, C., "Time window temporal logic," *Theoretical Computer Science*, Vol. 691, 2017. https://doi.org/10.1016/j.tcs.2017.07.012.

[29] D'Souza, D., and Prabhakar, P., "On the expressiveness of MTL in the pointwise and continuous semantics," *International Journal on Software Tools for Technology Transfer*, Vol. 9, No. 1, 2007. https://doi.org/10.1007/s10009-005-0214-9.

[30] Ouaknine, J., and Worrell, J., "On the decidability of metric temporal logic," *Proceedings - Symposium on Logic in Computer Science*, 2005. https://doi.org/10.1109/lics.2005.33.

[31] Fotiadis, F., Verginis, C. K., Vamvoudakis, K. G., and Topcu, U., "Assured Learning-Based Optimal Control subject to Timed Temporal Logic Constraints," *IEEE Conference on Decision and Control*, 2021, pp. 750–756.

[32] Sahoo, P. P., and Vamvoudakis, K. G., "On-Off Adversarially Robust Q-Learning," *IEEE Control Systems Letters*, Vol. 4, No. 3, 2020, pp. 749–754. https://doi.org/10.1109/LCSYS.2020.2979572.

[33] Fremont, D. J., Yue, X., Dreossi, T., Sangiovanni-Vincentelli, A. L., Ghosh, S., and Seshia, S. A., "Scenic: A language for scenario specification and scene generation," *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019. https://doi.org/10.1145/3314221.3314633.

[34] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym," 2016. https://doi.org/10.48550/arxiv.1606.01540, URL https://arxiv.org/abs/1606.01540.

[35] Laminar Research, "X-plane 11," , 2022. URL https://www.x-plane.com/.

[36] Körber, M., Lange, J., Rediske, S., Steinmann, S., and Glück, R., "Comparing popular simulation environments in the scope of robotics and reinforcement learning," *arXiv preprint arXiv:2103.04616*, 2021.

[37] Pitonakova, L., Giuliani, M., Pipe, A., and Winfield, A., "Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators," *Towards Autonomous Robotic Systems*, edited by M. Giuliani, T. Assaf, and M. E. Giannaccini, Springer International Publishing, Cham, 2018, pp. 357–368.

[38] Fremont, D. J., Chiu, J., Margineantu, D. D., Osipychev, D., and Seshia, S. A., "Formal Analysis and Redesign of a Neural Network-Based Aircraft Taxiing System with VerifAI." *Computer Aided Verification*, Vol. 12224, 2020, pp. 122–134.

[39] Julian Lockwood, "X-Plane 11 Cessna 172 Pilot's Operating Manual," , 2017. URL https://www.x-plane.com/manuals/C172_Pilot_Operating_Manual.pdf.