

**VERIFICATION OF ADVERSARIALLY ROBUST REINFORCEMENT
LEARNING MECHANISMS IN AUTONOMOUS SYSTEMS**

A Dissertation
Presented to
The Academic Faculty

By

Taehwan Seo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
Guggenheim School of Aerospace Engineering
Department of Engineering

Georgia Institute of Technology

December 2022

© Taehwan Seo 2022

**VERIFICATION OF ADVERSARIALLY ROBUST REINFORCEMENT
LEARNING MECHANISMS IN AUTONOMOUS SYSTEMS**

Thesis committee:

Dr. Kyriakos G. Vamvoudakis (Advisor)
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Yongxin Chen
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Dimitri Mavris
School of Aerospace Engineering
Georgia Institute of Technology

Date approved: December 5, 2022

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Kyriakos Vamvoudakis for his guidance during the past two years of study. I had learned and guided thoroughly over degree works thanks to his support and instructions. I would also like to thank the members of my thesis committee, Prof. Dimitri Mavris and Prof. Yongxin Chen for their service to thesis committee.

Thank you for all lab mates and special thanks to Prachi Sahoo for helping me document revision on related research work.

I would want to thank all of my academic and life coaches who have all contributed to my education and helped me become the person I am today.

Finally, I want to express my gratitude to my parents Kyungkug Seo and Junglim Ko as well as my brother Taehong Seo for their unwavering love and continuous encouragement throughout my years of study and my whole life.

TABLE OF CONTENTS

Acknowledgments	iii
List of Tables	vii
List of Figures	viii
Summary	ix
Chapter 1: Introduction	1
1.1 Verification of Learning Mechanisms	1
1.2 Objectives	2
1.3 Contributions	3
1.4 Outline	3
Chapter 2: Literature Review	4
2.1 Background of Verification	4
2.1.1 Performance Verification	4
2.1.2 Stability Verification	5
2.1.3 Verification Challenges	5
2.2 Adversarially robust mechanisms	6
2.2.1 Moving Target Defense	6

2.3	Control Topics	6
2.3.1	Optimal and Adaptive Control	6
2.3.2	Off-Policy Learning	7
Chapter 3: Off-Policy Reinforcement Learning		8
Chapter 4: On-off Adversarially Robust Learning		12
Chapter 5: VERIFAI: AI verification toolkit		18
Chapter 6: Verification Framework Test-bed Design		20
6.1	Process Design	20
6.2	Simulation Test-bed	22
Chapter 7: Simulation Experiments		25
7.1	Overview	25
7.2	OpenAI Gym: cart-pole	25
7.2.1	Control Problem	25
7.2.2	Parameters & Simulation	27
7.2.3	Algorithm Verification	27
7.2.4	Model Verification	29
7.3	X-plane 11: Cessna 172 attitude control	30
7.3.1	Control Problem	30
7.3.2	Parameters & Simulation	31
7.3.3	Algorithm Verification	32

7.3.4 Model Verification	34
Chapter 8: Conclusion	36
Appendices	38
References	46

LIST OF TABLES

6.1	Comparison of open source simulation tools	23
7.1	Cart-pole model verification parameter information	29
7.2	Cart-pole correlation coefficient with performance penalty score	29
7.3	X-plane model verification parameter information	34
7.4	X-plane correlation coefficient with performance penalty score	34
1	X-plane 11 - Cessna 172 Skyhawk specifications	44
2	X-plane 11 - Cessna 172 Skyhawk speed profile	45

LIST OF FIGURES

6.1	Diagram description of test framework	20
6.2	Inverted pendulum test environment with different simulations	23
7.1	OpenAI Gym cart-pole model	26
7.2	Cart-pole algorithm verification test analysis	28
7.3	Performances of cart-pole test samples.	28
7.4	Histogram comparison with different environment parameters	30
7.5	X-plane 11 Cessna 172 model	31
7.6	X-plane algorithm verification test.	32
7.7	Performances of X-plane Cessna 172 samples.	33
7.8	Score comparison with different environment parameters	35

SUMMARY

Artificial Intelligence (AI) is an effective algorithm for satisfying both optimality and adaptability in autonomous control systems. However, the policy generated from the AI is black-box, and since the algorithm cannot be analyzed in advance, this motivates the performance measurement of the AI model with verification. The performance and safety of the Cyber-Physical System(CPS) are subject to cyberattacks that intend to fail the system in operation or to interrupt the system from learning by modulation of learning data. For the safety and reliability scheme, verifying the impact of attacks on the CPS with the learning system is critical. This thesis proposal focuses on proposing one verification framework of adversarially robust Reinforcement Learning (RL) policy using the software toolkit ‘VER-IFAI’, providing robustness measures over adversarial attack perturbations. This allows an algorithm engineer would be equipped with an RL control model verification toolbox that may be used to evaluate the reliability of any given attack mitigation algorithm and the performance of nonlinear control algorithms over their objectives. For this specified work, we developed the attack mitigating RL on nonlinear dynamics by the interconnection of off-policy RL and on-off adversarially robust mechanisms. After that, we connected with the simulation and verification toolkit for testing both the verification framework and integrated algorithm. The simulation experiment of the whole verification process was performed with two different control problems, one is a cart-pole problem from OpenAI gym, and the other problem is the attitude control of Cessna 172 in X-plane 11. From the experiment, we analyzed how the attack-mitigating RL algorithm performed with gain varying specific adversary attacks, and evaluated the generated model performance over the changing environmental parameters.

CHAPTER 1

INTRODUCTION

1.1 Verification of Learning Mechanisms

Artificial Intelligence (AI), widely implemented in autonomy [1, 2, 3, 4] is the perceiving, synthesizing, and inferring intelligence demonstrated by machines, as opposed to that displayed by animals and humans. It has been widely implemented in autonomous dynamical systems currently, and this necessitates the importance of AI-model verification using specific performance measures. In many cases, AI has complex dimensional space and incomprehensible reasoning in the optimal policy. This makes it hard for engineers to evaluate the performance of the control model and motivates the effective performance monitoring process.

Data-driven optimization is a key idea for AI in control topics. It easily provides approximated optimal policy with small or unknown information of dynamics. Based on the objective of different control problems, it is developed as a form of adaptive control, Reinforcement Learning, and Deep Reinforcement Learning. Reinforcement Learning (RL), in the context of nonlinear control problems, generates optimal control policies for known and unknown systems in the presence of several perturbations including observation noise, adversarial attacks, and response errors. This makes RL an extremely powerful and critical tool for Cyber-Physical Systems (CPS). However, not every RL possesses this capability, nor can it guarantee this resilience under all circumstances, especially there involves deliberate attacks on the learning model. That is where other techniques, namely attack mitigation systems, are applied. Moreover, since the methods may not always work under all circumstances, the performance of the autonomous system must be extensively examined before implementation to prevent serious errors. Such problems emphasize the need

for complete analysis and verification for attack mitigating RL in order to measure the safety of the method and control performance. However, the lack of verification results on attack-mitigating RL in past studies restricts the potential discovery of effectiveness and flaws in strategies that can boost its further study. The main goal of this thesis is to construct an attack-mitigating off-policy RL with Moving Target Defense (MTD) mechanism and verify the algorithm and model with selected parameters to analyze the effectiveness and flaws of the attack-mitigating RL. This will provide control performance analysis over attack-mitigating RL with the proposed verification framework, and provide possibilities for improvement on the current RL attack mitigation strategy on CPS. We hypothesize that the verification result forms a improvement tendency on the control performance over the selected attack and environment parameters, and this will be examined by simulation experiment in chapter 7.

1.2 Objectives

To summarize, in this research of verification of attack mitigating RL, following objectives will be achieved:

1. Formulate the verification problem with off-policy RL and attack mitigation mechanism to fulfill research gap over verification of attack-mitigating RL.
2. Select verification method based on the literature review of current verification methods and challenges of learning-based controls.
3. Propose the verification framework of attack mitigating RL by the integration of selected verification method.
4. Perform analysis of verification result on attack mitigating RL in terms of algorithm and model performance with different control problem simulations.

1.3 Contributions

The main contributions of this work are fourfold. Firstly, this work presents an integration of off-policy RL and attack-mitigation mechanism with an on-off adversarially robust learning mechanism in order to construct attack-mitigating off-policy RL for nonlinear systems. Second, this work implements performance verification of attack-mitigating RL with a verification toolkit by the connection of control task simulations. Third, this work analyzes the possible simulations where the proposed verification method can be applied. Fourth, this work performs a simulation experiment in two different systems with a verification method to analyze the verification result of attack-mitigating off-policy RL. From conducting four studies, an algorithm engineer would be equipped with an RL control model verification toolbox that may be used to evaluate the reliability of a given attack mitigation algorithm and the performance of nonlinear control algorithms over their objectives.

1.4 Outline

Here is the overview of this research work. In this thesis, we present the complete process of real-time verification with three major background components. First, from the off-policy RL [5], we construct the core learning algorithm for the system in the chapter 3. Then, we integrate the on-off adversarially robust learning mechanism [6] in chapter 4 into the aforementioned framework. Finally, in chapter 5, we construct the verification framework by connecting the verification toolkit “VERIFAI” [7] to the algorithm, providing a test-bed for simulating the integrated algorithm and evaluating the performance of the model. In chapter 6, Whole process and scoring mechanism will be provided, and testing protocol will compare simulation software configurations that are selected for the testbed in the process. Finally, in chapter 7, the simulation experiment from OpenAI gym cart-pole and X-plane 11 Cessna 172 were gone through verification process with presence of attacks with analysis of performances.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we will discuss associated background in verification of intelligent control system. This includes the study on different types of the verification, attack on system and control topics, as well as the challenges and common methods.

2.1 Background of Verification

The purpose of verification is to find out if the performance of the system is within the desired design limits. Such performance evaluation is essential due to the nature of AI of black-box [8], which is not explainable. As the cost of fixing errors to requirements exponentially grows over time and steps of the whole project process [9], it is crucial to detect the problem in the early stages to fix the problems with lower costs. This motivates the verification applies to AI, finding methods to detect the problems in early stages to fix the problem at a smaller cost. The subject for verification is varied with different methods and systems, but the verification in intelligent control can be largely divided into performance verification and stability verification [10].

2.1.1 Performance Verification

Performance verification is a verification to measure control performance after training with data-driven learning, such as neural net controllers [11], optimal policy learning [12], and reinforcement learning [13]. This is mainly done in a way to check the achievement and effect through several tests on the learned model. Multiple methods were proposed for the verification of performances. First, Sensitivity analysis measures maximum output deviation from input deviation, and empirical study of sensitivity in neural networks performed in [14]. Moreover, Falsification analysis checks case the model to be failed with

multiple test cases [15]. This tests the model with different environmental parameters and conditions to test out how robust the model is for different conditions and what affects the model to fail. Another form of performance verification is adversarial robustness. This analysis checks the response of the model with the presence of attack signals [16]. From changing gains and forms of adversarial inputs and noises, it tests out the performance with the learned model with the presence of attacks. This allows the user to test the robustness of the model over the attacks.

2.1.2 Stability Verification

Stability verification is a verification to monitor an unstable state that may occur during policy learning. As it is a part that can be verified either by marginal stability [17] or Lyapunov theory [18, 11], it mainly limits the learning that can be unstable or indicates its unstable states through mathematical monitoring. For the method details of verifying performance on learning, one of the example verification techniques for stability verification is reachability analysis. reachability analysis estimates possible sets of network outputs over some input distribution, using optimal measures with hamilton-jacobi to test the system margin from optimality [19, 20].

2.1.3 Verification Challenges

Verification for AI algorithms implemented in CPS tends to be over-designed as per the design limit [21]. Such strict testing-based methods possess multiple challenges including environment formulation, high-dimensional/complex policy generation, and translation errors introduced by virtue of casting high-dimensional knowledge to mathematical information [22]. Most existing attempts to counteract these issues rely on approximations. For example, verification studies of trained Neural Networks (NN) use constructor-evaluator architectures [23], reachability analysis [24], sampling and table analysis, validation through several test case sets [25], etc.

2.2 Adversarially robust mechanisms

The CPS is prone to adversarial attacks in process of building up optimal policy based on learning. Numerous CPS are static, and such flaws provide a determined attacker the time they need to penetrate the system and formulate effective defenses.

2.2.1 Moving Target Defense

A defense paradigm called Moving Target Defense (MTD) [26] tries to reduce the natural advantage that the attacker has over the defender. While the system's defender must constantly keep an eye on all the weaknesses and take precautions against all possible attack vectors, the attacker may only need to get past those barriers once. By creating processes that modify the system's settings continuously and erratically, MTD protocols seek to address this disparity. Three objectives of this unpredictable behavior are to raise the cost of an attack, restrict the disclosure of weak points, and confuse the adversary.

2.3 Control Topics

2.3.1 Optimal and Adaptive Control

In intelligent control systems, there are two fields of control mathematics, optimal control [27] and Adaptive Control [28]. A branch of mathematics called optimal control uses the calculus of variations to optimize control inputs in order to reduce user-designed cost functions. As an alternative, adaptive control makes use of parameter estimates to establish the proper control parameters in order to regulate systems with variable or originally undetermined parameters. The optimal policy can be approximated through adaptive control through reinforcement learning.

2.3.2 Off-Policy Learning

The off-policy learning is one of the reinforcement learning methods that the agent learns the policy differently from the one it is executing [29]. This includes learning optimal policy while executing behavioral policy, learning from behavioral policy, and learning multiple policies with single observance from the behavioral policy. The off-policy learning allows multiple features. First, the model can apply behavioral policy from other agents such as humans. Second, the model could collect enough exploration to make the gradient to optimal. Third, the model can re-evaluate previous empirical actions by comparing rewards from the target policy to behavioral policy. Conventional Q-learning is one typical example of off-policy reinforcement learning.

CHAPTER 3

OFF-POLICY REINFORCEMENT LEARNING

Reinforcement learning is one of the key methods to solve optimal control problems [27, 30], as it implements learning methods such as Q-learning [31], off-policy learning, policy iteration, and actor-critic [30] structures. These methods may also find applications in non-linear systems with unknown dynamical model parameters [32]. Our proposed work focuses on the RL framework introduced in [5]. To formulate the problem as per [5], we begin by defining the state, and error dynamics as follows,

$$\dot{x}(t) = f(x(t)) + g(x(t))u(x(t), t), \quad x(t_0) = x_0, \quad t \geq 0 \quad (3.1)$$

$$\dot{e} = F(e) + G(e)u := f(e + c) + g(e + c)u \quad (3.2)$$

where $e := x - c \in R^n$ is the error between state, x and the state of interest, c and x_0 is the initial state of the system. For the time period $[t_k, t_l]$, which is a small window of time of T seconds, The cost functional is given by,

$$J(e(t_k), t_k, t_l, u) := \int_{t_k}^{t_l} r(e(s), u(e(s), s)) + L_{k,l}(e(s)) ds + \phi(e(t_l)) \quad (3.3)$$

where $L_{k,l} > 0$ is the penalty for unsafe zones, $r := Q(e) + S(u)$ is performance measure, defined with $Q(e) := e^T Q_m e$, $Q_m \succeq 0$, $S(u) := u^T R u$, $R \succ 0$, and $\phi : R^n \rightarrow R \geq 0$ is a positive definite term that penalizes the deviation of terminal state at t_l from point of interest c .

Assuming that the optimal value function $V^* = \min_u J(e, t, u)$ is a continuously differ-

entiable function, the corresponding minimizing control policy u^* can be derived as

$$u^*(e, t) = -\frac{1}{2\gamma}R^{-1}G(e)^T\nabla_e V^*(e, t) \quad (3.4)$$

from the [5], the solution for optimal value function V^* can be obtained from the partial differential equation,

$$\begin{aligned} \nabla_t V^*(e, t) + \nabla_e V^*(e, t)^T(F(e) + G(e)u^*) + \gamma r(e, u^*) + L(e) &= 0, \\ \forall t \in [t_k, t_l), V^*(e(t_l), t_l) &= \phi(e(t_l)) \end{aligned} \quad (3.5)$$

where $\gamma > 0$ controls the trade-off between L and $r(\cdot, \cdot)$. It is generally hard to solve the partial differential equation directly, so we approximate the value function using learning tools. To approximate V^* , and u^* in the equation, the critic and actor weight and their basis functions were defined. With $i \in N$, we denote value function approximation as \hat{V}^{u_i} and next step control input as \hat{u}_{i+1} , where $\lim_{i \rightarrow \infty} \hat{V}^{u_i} = V^*$ and $\lim_{i \rightarrow \infty} \hat{u}_i = u^*$. The approximation neural networks along with basis functions are given as follows,

$$\hat{V}^{u_i}(e, t) := (\hat{W}_i^v)^T \psi^v(e, t) + \phi(e) \quad (3.6)$$

$$\hat{u}_{i+1}(e, t) := (\hat{W}_i^u)^T \psi^u(e, t). \quad (3.7)$$

From the combination of optimal controller (Equation 3.4), value(cost) function gradient (Equation 3.5), and approximation of the optimal value function and optimal input with actor and critic weight (Equation 3.6)-(Equation 3.7), the update equation for the combined actor and critic weight matrices, given by \hat{W}_{i+1} is expressed as,

$$\hat{W}_{i+1} = - \left(\sum_{j=0}^l \Theta_{j,i}^T \Theta_{j,i} \right)^{-1} \left(\sum_{j=0}^l \Theta_{j,i}^T \Psi_{j,i} \right) \quad (3.8)$$

where

$$\begin{aligned}
\Theta_{j,i} &= [\Theta_{j,i}^v, \Theta_{j,i}^u] \\
\hat{W}_i &= [(\hat{W}_i^v)^\top \text{vec}(\hat{W}_i^u)^\top]^\top \\
\Theta_{j,i}^v &= (\psi^v(e(\tau_{j+1}), \tau_{j+1}) - \psi^v(e(\tau_j), \tau_j))^\top \\
\Theta_{j,i}^u &= \int_{\tau_j}^{\tau_{j+1}} 2\gamma ((\hat{v}_i(e, \tau)^\top R) \otimes \psi^u(e, \tau)^\top) \mathbf{d}\tau \\
\Psi &= V^{u_i}(e(\tau_{i+1}), \tau_{i+1}) - V^{u_i}(e(\tau_i), \tau_i) + \int_{\tau_j}^{\tau_{j+1}} (\gamma Q(e) + L(e) + \gamma S(\hat{u}_i(e, \tau))) \mathbf{d}\tau.
\end{aligned} \tag{3.9}$$

The $\tau_j = t$ and $\tau_{j+1} = t + T$, where $j \in \{0, \dots, N\}$ that $t_k = \tau_0 < \tau_1 < \dots < \tau_N = t_l$, and iterate through (Equation 3.8) until the weights converge with criterion $\epsilon > 0$, which will return the approximated policy of optimality. 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 (Equation 3.2)

Select $\epsilon > 0, i = 0$

while $\|\hat{W}_{i+1} - \hat{W}_i\| > \epsilon$ **do**

 Solve for \hat{W}_{i+1} from equation (Equation 3.8)

 Extract \hat{W}_{i+1}^v and \hat{W}_{i+1}^u from \hat{W}_{i+1} by (Equation 3.9)

 Update index i by $i = i + 1$

$\hat{W}_{\text{optimal}}^u = \hat{W}_{i+1}^u$

for $t = t_b : t_f$ **do**

 Simulate the dynamical system with learned policy with $\hat{W}_{\text{optimal}}^u$ with
 (Equation 3.7)

 collect input-state data from system (Equation 3.2)

CHAPTER 4

ON-OFF ADVERSARIALLY ROBUST LEARNING

Real world applications of RL optimal control frameworks include CPS which rely on complex interconnections between physical and computational subsystems. These are prone to several systemic perturbations including environmental noise and adversarial attacks. In [6], on-off switching in the gain matrix, tuned intermittently via a dropout-descent algorithm is devised and applied to Q-learning, minimizing the effect of adversarial signals by increasing the unpredictability of the descent mechanism while guaranteeing convergence.

To briefly describe the on-off policy generation framework we begin with a continuous Linear Time-Invariant(LTI) system given by

$$\dot{x}(t) = Ax(t) + Bu(t), x(0) = x_0 \quad t \geq 0$$

wherein $A \in R^{n \times n}$ is the plant matrix and $B \in R^{n \times m}$ is the input matrix. To introduce unpredictability into the system, the authors of [6] aim to utilize the controllability based redundancy in input matrix B . To that end, in the feedback design $u = Kx$, elements of gain matrix K that are components related to activation of actuators can be removed for each controllable actuation mode i while maintaining controllability of the resulting system dynamics. This extends the CPS' attack-surface thus making it difficult for an attacker to conduct a low-cost attack.

The on-off induced gain matrix is given by

$$K_i[:, :] = \begin{cases} \hat{W}_a[:, i]^T, & \forall i \in \text{allowable actuation modes } S_i \\ 0, & \text{otherwise} \end{cases} \quad (4.1)$$

where $K_i[:, :]$ is the i 'th row of the gain matrix.

Additionally, to minimize actuator related cost, the update of the weight matrix is filtered out by maximum steepness of gradient directions as follows,

$$\dot{W}_{a_{ij}} = \begin{cases} -d_{ij}, & d_{ij} \in 3s \text{ steepest gradients and } K_i[j] \neq 0 \\ 0, & \text{otherwise} \end{cases} \quad (4.2)$$

where the s is defined in [6] as the level of sparsity, and $d = xe_a^T$.

In this work, the on-off switching mechanism is integrated with previously laid out off-policy learning. Both adversarial signals and switch mechanisms are applied to the known behavioral policy, where the data for the training is being collected before the approximation of optimal policy. Specifically in this work, the arbitrary initial behavioral policy is designed to be changed every specific time with switching gains that are assigned to each of the state modes. These state modes are designed to bound connected states of the system with soft constraints, containing observation noise and state feedback signals. This gain switching supplements the existing on-off mechanism for seamless integration to off policy RL as follows,

$$K_i = \begin{cases} k, & (i = \text{switch}) \\ 1, & (i \neq \text{switch}) \end{cases} \quad (4.3)$$

$$u_j = \sum_{i=1}^n -K_i \cdot \tan\left(\frac{h_i(x_i)}{b_i}\right) \quad (4.4)$$

$$u = [u_1, u_2, \dots, u_m]^T \quad (4.5)$$

$$u_b = \tanh\left(\frac{u}{D}\right) \quad (4.6)$$

Where control input elements $u_j, \forall j \in \{1, 2, 3, \dots, m\}$ are formed with sum of each state feedback that is multiplied with scalar gain $K_i, \forall i \in \{1, 2, 3, \dots, n\}$. These elements K_i are either $k \in R > 1$ or 1 if $i = \text{switch}$ or $i \neq \text{switch}$ respectively, as shown in (Equation 4.3).

This is a modified form of (Equation 4.1) for algorithm integration. The switch is randomly selected from $\{1, 2, 3, \dots, n\}$ over different time blocks. $D \in R$ is input boundary, and $b_i \in R$ is each state boundary. Also, $h_{(\cdot)}(x_{(\cdot)})$ is an arbitrary function that can foster system to explore within it's boundary. It includes observation noise and preliminary control input for the initial data collection of the system. For the application of switching actuators, the gain K_i is multiplied to the corresponding arbitrary state function $h_i(x_i)$ for each of the actuator switch modes. This allows the model to collect each state's response data with specific emphasis for input modes. This procedure is summarised in Algorithm 2.

Algorithm 2: On-off switching arbitrary behavioral policy u_b

output: behavioral policy u_b

```

1 begin
2   Initialize  $K = [1,1,1,\dots,1]$ 
3   for every  $t$  seconds do
4     Randomize switch key in  $[1,2,3,4,\dots,n]$ 
5     Replace index corresponding to switch in  $K$  with arbitrary gain  $k$ 
6     Compute  $u_j \forall j \in \{1, 2, 3, \dots, m\}$  as per (Equation 4.4).
7     Compute  $u_b$  as given by (Equation 4.6).
8     Add adversarial noise  $u_{attack}$  to  $u_b$  by (Equation 4.7)

```

For the adversarial noise, it is designed to applied to the behavioral policy since it is where the attack can negatively affect in case of off-policy algorithm,

$$\begin{aligned}
 u_{attack} &= \lambda u_b \\
 u_{b,attacked} &= (1 + \lambda)u_b
 \end{aligned} \tag{4.7}$$

where $\lambda \in (-0.2, 0.2)$, while it can be selected specifically or randomly chosen. Behavioral policy u_b is modulated with either random or specific ratio between the selected range, while the behavioral policy that being calculated over the RL for optimality is not being

modulated.

Filtered update of weight matrix in (Equation 4.2) was applied to off-policy learning while maintaining the mechanism of filtering maximum steepness gradients. Since the off-policy updates the actor weight \hat{W}^u by (Equation 3.8), the gradient of actor weight is calculated as $\Delta\hat{W}_i^u = \hat{W}_{i+1}^u - \hat{W}_i^u$, then pruned as per,

$$\hat{W}_i^u[j, k]_{\text{grad}} = \begin{cases} \Delta\hat{W}_i^u & \in 3\text{s steepest gradients among } \Delta\hat{W}_i^u \\ 0, & \text{otherwise.} \end{cases} \quad (4.8)$$

This $\hat{W}_{i,\text{grad}}^u$ is added to the current weight matrix \hat{W}_i^u to get the updated value, $\hat{W}_{i+1,\text{prune}}^u$.

This process is summarised in Algorithm 3.

Algorithm 3: Tuning with Switching

input : $(\hat{W}_{i+1}^u, \hat{W}_i^u)$

output: $\hat{W}_{i+1,\text{prune}}^u$ which is a pruned version

- 1 **begin**
- 2 **if** $i \geq 2$ **then**
- 3 Calculate $\Delta\hat{W}_i^u$ from $\hat{W}_{i+1}^u - \hat{W}_i^u$
- 4 Sort $\Delta\hat{W}_i^u$ and find 3s steepest gradient index named i_{steep}
- 5 Apply (Equation 4.8) to compute $\hat{W}_{i,\text{grad}}^u$ with i_{steep}
- 6 Get updated \hat{W}^u as $\hat{W}_{i+1,\text{prune}}^u = \hat{W}_{i,\text{grad}}^u + \hat{W}_i^u$

Now, combining the off-policy RL with on-off adversarially robust Q-learning, we obtain the following framework given in Algorithm 4.

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

input : Learning parameters, Behavioral policy

output: Simulation data

```
1 begin
2   Load global variable parameters
3   for  $t = 0 : t_b$  do
4     Randomize switch
5     Get  $u_b$  from Algorithm 2; Simulate the dynamical system with  $u_b$ 
6     collect input/state/switch data from system (Equation 3.2)
7
8   Select  $\epsilon > 0, i = 0$ 
9   while  $\|\hat{W}_{i+1} - \hat{W}_i\| > \epsilon$  do
10    Recall switch information for  $u_b$ 
11    Solve for  $\hat{W}_{i+1}$  from equation (Equation 3.8)
12    Extract  $\hat{W}_{i+1}^v$  and  $\hat{W}_{i+1}^u$  from  $\hat{W}_{i+1}$  by (Equation 3.9)
13    Get updated  $\hat{W}_{i+1,prune}^u$  from Algorithm 3 ( $\hat{W}_{i+1}^u, \hat{W}_i^u$ )
14    Update index  $i$  by  $i = i + 1$ 
15     $\hat{W}_{optimal}^u = \hat{W}_{i+1}^u$ 
16
17  for  $t = t_b : t_f$  do
18    Simulate the dynamical system with learned policy with  $\hat{W}_{optimal}^u$  with
19    (Equation 3.7)
20    collect input-state data from system (Equation 3.2)
```

To remark, from this proposed defense mechanism, this will ensure the system to perform with similar measures to the system without the presence of attacks, while defending the system by increasing attacking surface for proactive defense with dynamical modification of behavioral policy. Although this will slow the weight convergence speed and will

also require more behavioral movement data from the system, it provides a trade-off option between security and training resources. In this research work, the enhanced security option is evaluated by the verification toolkit that will be described in following chapter, the chapter 5.

CHAPTER 5

VERIFAI: AI VERIFICATION TOOLKIT

VERIFAI [7] 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 [33]. 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 [34], 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.

CHAPTER 6

VERIFICATION FRAMEWORK TEST-BED DESIGN

6.1 Process Design

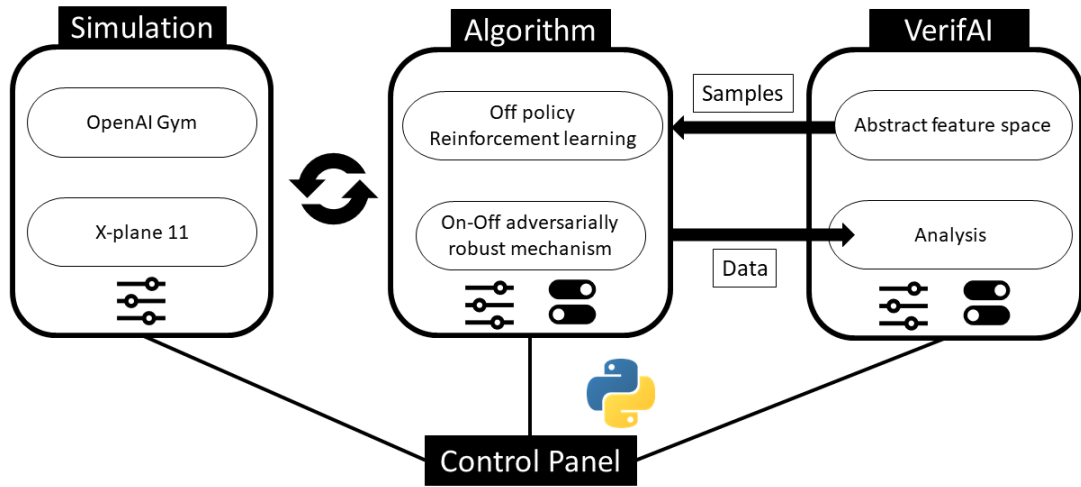


Figure 6.1: Diagram description of test framework

In our proposed work, we built the entire AI framework and process from the ground up: the application of the RL algorithm all the way to verification. This is summarized in Figure 6.1 and includes all components that are addressed above, with a nonlinear cart-pole simulation in OpenAI Gym [35] and X-plane 11 [36] as a test-bed. We built the entire AI framework and process from the ground up as summarized in Figure Figure 6.1. This includes all components that are addressed above, with a nonlinear cart-pole simulation in OpenAI Gym [35] and X-plane 11 [36] 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 (Equation 4.6) 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 was calculated from the state traces of the system, summing up L_2 -norm errors that were captured over the last T second window of simulation, given by

$$\text{Score} = \sum_{i=T_f-T}^{T_f} \|c - x(i)\|_2. \quad (6.1)$$

The process design is summarized in Algorithm 5, referring to Algorithm 4. By the whole connection of algorithms, the whole framework is described with all components connected.

Algorithm 5: Verification Process

Data: control parameters, simulation environment

Result: Error table of Falsification

```
1 begin
2   Load global variable parameters and import VERIFAI
3   Set parameter to change from environment
4
5   for  $1:N$  do
6     Generate the Sample environment
7     Simulation data = Algorithm 4(sample environment)
8      $x$  = simulation data
9     Calculate (Score) as per (Equation 6.1).
10    if ( $Score \geq criterion$ ) then
11      Save environment information to Error table
```

6.2 Simulation Test-bed

The simulation component described in Figure 6.1 in the verification framework can be replaced with different configurations based on the purpose of the control problem and verification process. Examples for such simulation software are Gazebo, Webots, Pybullet, and Mujoco [37, 38].

The table above summarizes the comparison over the typical RL control system simulation. the criterion was set based on the important factor for the current setup of verification. Gazebo, Webots, and V-Rep are basically Robot Operating System(ROS)-based simulation tools, which are aiming for real-world implementation but in a simulation scheme. Due to such objectives, its supporting languages are closer to machine-level to reduce the computing time response, which are C/C++ and ROS packets. Also, since the real robot system

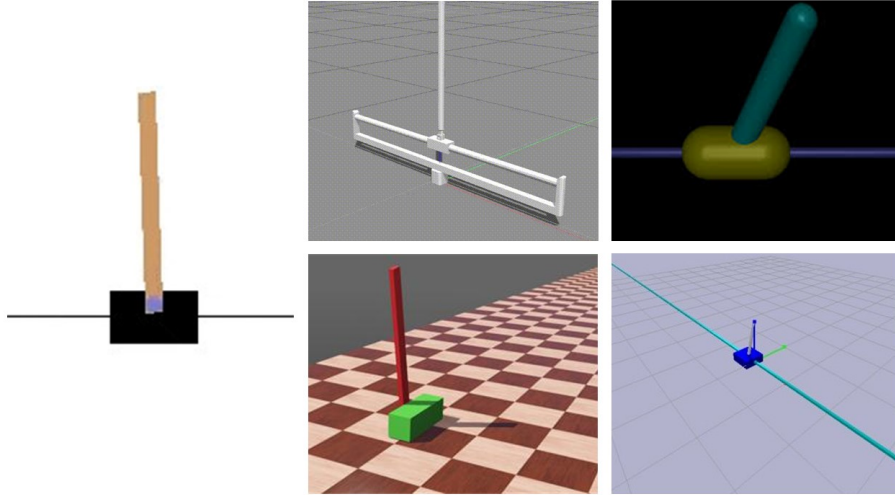


Figure 6.2: Inverted pendulum test environment with different simulations

Table 6.1: Comparison of open source simulation tools

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

requires a complicated and higher degree of freedom in states, these simulation tools allocate the computing power to solve complex state dynamics with high degrees of freedom. However, data communication with Python can be harder, and it is hard to find a built-in example model, rather it is easier to design and use it. 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 6.1, 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.

The Mujoco, PyBullet, and OpenAI Gym, in comparison, are built on higher-level languages such as Python and XML since they are intended for testing and developing RL

algorithms rather than focusing on the system's actual performance in the real world. It offers a number of examples for such testing and has built-in typical modeling that is simple to load and use. The majority of it is based on/supports Python, making data transfer much simpler. Low performance is displayed while solving complex state dynamics because these tools are not utilized to conduct real-world analysis.

CHAPTER 7

SIMULATION EXPERIMENTS

7.1 Overview

This chapter will present the simulation experiment that implementing verification framework presented in chapter 6. Simulations are a cart-pole problem in OpenAI Gym [35] and Cessna 172 in X-plane 11 [36], selected as one simple and one complex simulation option. Over two different simulations, the sample analysis of verification data will show the algorithm performance with changing adversary gains, and show model performance with changing environmental parameters. Visualization examples of performances were provided with simulation plots and videos, the plots described in Figure 7.3 and Figure 7.7, and the according video is provided with the following URLs:

Cart-pole: <https://youtu.be/2HI63rHL4FA>

X-plane 11: <https://youtu.be/SjuNo5DR7qU>

Both simulation video shows the sample results and processes of the verification with parameters and scores, demonstrating how the verification is visually presented.

7.2 OpenAI Gym: cart-pole

7.2.1 Control Problem

In the openAI Gym, which is a toolkit for developing and comparing reinforcement learning algorithms, has a classical nonlinear control problem of inverted pendulum system [39]

with the negligence of the friction is defined with dynamical equation,

$$\ddot{x} = \frac{u + m_p l (\dot{\theta}^2 \sin\theta - \ddot{\theta} \cos\theta)}{m_c + m_p} \quad (7.1)$$

$$\ddot{\theta} = \frac{g \cdot \sin\theta + \cos\theta \left(\frac{-u - m_p l \dot{\theta}^2 \sin\theta}{m_c + m_p} \right)}{l \left(\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p} \right)} \quad (7.2)$$

Where x is cart position (m), θ is pole angle (radians), m_c is cart mass (kg), m_p is pole mass (kg), l is pole length (m), g is gravitational constant (9.81 m/s^2). The state of the system is expressed as $[x, \dot{x}, \theta, \dot{\theta}]^T$, which are cart position, velocity, and pole angle, angular rate. The input u only exerted into the cart, which is the only control that can change the state of the system. The objective of control of this system is transition from stable equilibrium to unstable equilibrium, which is controlling cart movement to swing the pole to raise and stay upright starting from downward pole position which is stable. Specifically, the state starts at $[0, 0, \pi, 0]^T$ and aims to the zero state of $[0, 0, 0, 0]^T$. The pole angle state has range of $(-\pi, \pi]$, where the counterclockwise angle from upright position is negative angle and the opposite has positive angles. The angle value changes its sign based on the position of pole tip, where it stays negative in left half and positive in right half.

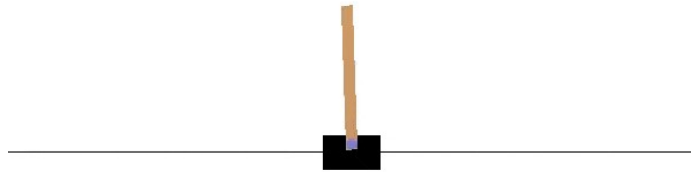


Figure 7.1: OpenAI Gym cart-pole model

7.2.2 Parameters & Simulation

In this problem, the cost gains that are engaging trade-off of optimization objective were set to $Q = \text{diag}([1, 1, 100, 60])$, $R = 0.01$, prioritizing pole angle and pole angular rate to be stabilized as first. The simulation step time was $dt = 0.01$ seconds. The full list of parameters used and the values for the algorithm were attached in Appendix. For the simulation solver, the Runge-Kutta method of 5th(4) order (RK45) [40] was used for the integration of the differential equation given in (Equation 7.1) cart-pole simulation solver. The simulation data were saved with each time step, and the data were visually rendered with built-in OpenAI Gym renderer of cart-pole simulation.

7.2.3 Algorithm Verification

To verify the effectiveness of the defensive mechanism from adversarial attacks to the algorithm and to the system, a falsification test was performed with ranging adversarial gains. From the defined noise profile which is designed as (Equation 4.7), the λ gains were sampled in the falsification test environment and the performance of the system was measured. Selected λ applied to the algorithm as a specific attack profile, which refers to the original behavioral policy for the attack. The performance was calculated based on the state data of the last 5 seconds, which is calculated based on (Equation 6.1), while $T = 5$ seconds. To avoid the excessive calculation time due to blow up of the system when the system fails, the simulations that exceeded 200 of cart position were scored as large penalty values to record the test case to be a concrete failure.

The Figure 7.2 shows the failure rate over varying specific adversary attacks. The verification was performed with 500 tests, and the criterion for the failure of the system is decided with penalty scores that are larger than 30, while smaller penalty scores are labeled as safe runs. The failure rate of each point is formed from the histogram of 7 bins over the different adversary gains, dividing the failures by the trials of the same bin. From the analysis result, the system showed higher failure rates on negative adversarial gains compared

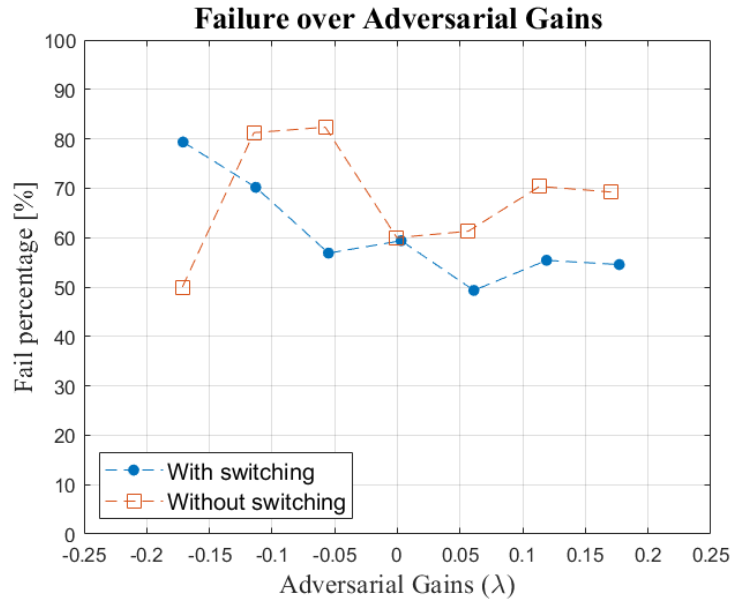


Figure 7.2: Cart-pole algorithm verification test analysis

to positive gains. This means that the exaggerated behavioral policy with positive gains was less harmful than the reduced behavioral policy with negative gains. In positive gain attacks, performance over the adversary is marginally better with switching than without non-switching, but in negative gain attacks, performance did not significantly increase. The behavioral movement of the plant for training was simulated and recorded state and input information over 500 seconds with behavioral policy as described in (Equation 4.6).

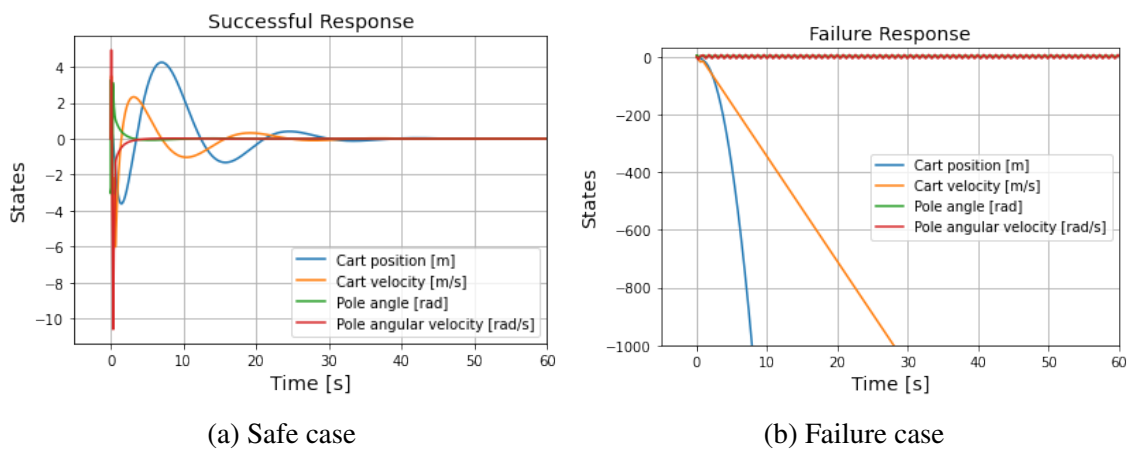


Figure 7.3: Performances of cart-pole test samples.

The Figure 7.3 describes the safe and fail states of the control system that depends on

the algorithm. The system successfully converged to zero states with some oscillations in one example of Figure 7.3a, while the case of Figure 7.3b displayed failure with blown cart position.

7.2.4 Model Verification

For the model verification, one effective model that is generated from the proposed algorithm was selected for the test. To verify the selected model, multiple simulations were performed on the model with generated environments by given parameters.

Table 7.1: Cart-pole model verification parameter information

Parameters	Sampling range
Pole length [m]	0.9 - 1.1
Cart mass [kg]	0.9 - 1.1
Pole mass [kg]	0.09 - 0.11
Cart initial position [m]	-1 - +1

In terms of the cart-pole verification test, the model in Figure 7.3a is selected among the simulation test that is performed in algorithm verification, which is the best performance with the lowest score. In Table 7.1, the sampling range of environmental parameters was described with a possible range of sampling. A total of 1,000 tests were performed with different parameter combinations, recording corresponding penalty scores for each of the simulations.

Table 7.2: Cart-pole correlation coefficient with performance penalty score

Parameters	Correlation coefficient
Pole length	0.204
Cart mass	0.194
Pole mass	0.142
Cart initial position	0.104

The analysis of the results is shown in the Table 7.2, which shows the relationship between each environment parameter and the penalty score and which parameter is more or less responsible for the failures with higher penalty scores. Figure 7.4 shows the histogram

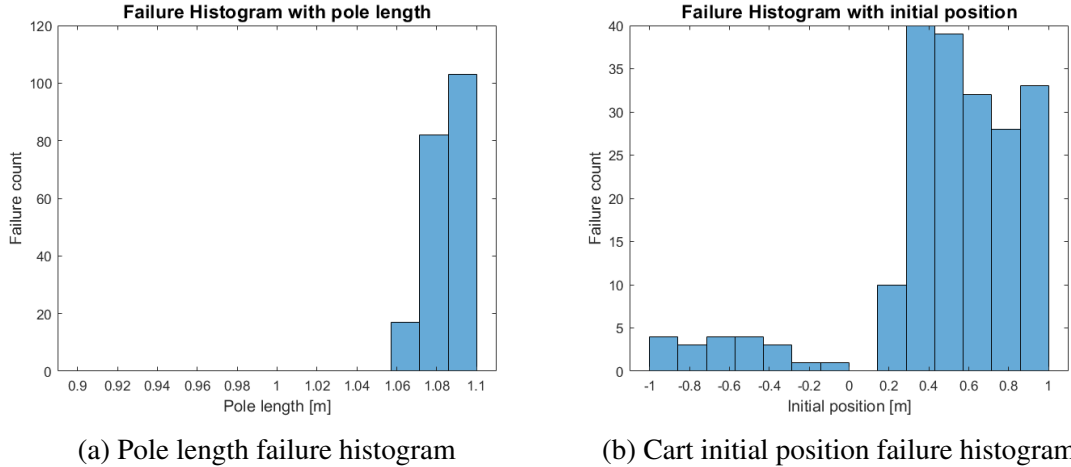


Figure 7.4: Histogram comparison with different environment parameters

of failures with the largest and smallest correlation factors, which are pole length and initial cart position. The pole length histogram in Figure 7.4a showed more skewed failure rates in larger values compared to the cart initial position in Figure 7.4b, and it follows the result from the correlation coefficient.

7.3 X-plane 11: Cessna 172 attitude control

7.3.1 Control Problem

The control problem for X-plane simulation was selected as the attitude control of Cessna 172. The algorithm aims to stabilize the roll and pitch angle of the aircraft to zero angles with three different control surfaces: aileron, elevator, and rudder. The state of the system is expressed as $[\theta, \phi, \dot{\theta}, \dot{\phi}]^T$, which are pitch angle, roll angle, pitch angular rate, and roll angular rate respectively. Input u is defined as $[u_a, u_e, u_r, u_t]$, which are yoke inputs of the aileron, elevator, rudder, and throttle. Each of the input values was limited to the maximum value of -1 to 1 in the simulation, where the 0 value represents the idle of the control input. The generated optimal signal is limited with each magnitude of -0.6 to 0.6 to avoid excessive steering of the aircraft. The throttle of the aircraft, u_t is fixed to 82% of full throttle, which is valued at 0.82. At the start of the simulation, the aircraft is placed in the

air at an altitude of 1200 m above the ground. The reference position of the ground was set as the start line of runway 04 in Grant County International Airport(ICAO: KMHW), facing 4 degrees east from the north, which is the runway heading. The initial speed of the plane in the air was set to 50 m/s, and no initial angular rate was applied.



Figure 7.5: X-plane 11 Cessna 172 model

7.3.2 Parameters & Simulation

In this problem, the cost gains that are engaging trade-off of optimization objective were set as $Q = \text{diag}([1, 0.3, 0.1, 0.1])$ in quadratic cost, $S = \text{diag}([1, 1, 0.1, 0.1])$ in end-state cost for each divided time blocks, $R = \text{diag}([1, 1, 1])$ in input cost, prioritizing roll and then pitch attitude to be stabilized as first and second. The profile of Cessna 172 was simulated with default parameters, which are described in Appendix and operating manual [41]. For the simulation solver of dynamical system, the X-plane 11 uses blade element coefficient information for the implementation of the aerodynamic calculation. Each of the elements such as two main wings, fuselage, and tail wings contains its own dynamical profile with coefficient information, simulating the whole body based on rigid body dynamics with the

attitude and airspeed information of the aircraft. The simulation time step was determined based on the looping time of call iteration of communication. Normally, the simulation step time for algorithm application was near 0.1 seconds, but little deviation was present by the communication of python and x-plane simulation. From the parameter script, the environment parameters such as turbulence, clouds, and weather are selected based on the assigned probability model, which is selected as uniform distribution for this simulation experiment. Based on the assigned parameter values, the simulation result is converted to a quantitative score with (Equation 6.1) calculation. From each of the tests, the VERIFAI records the x-plane simulation and provides the visualization of each verification test.

7.3.3 Algorithm Verification

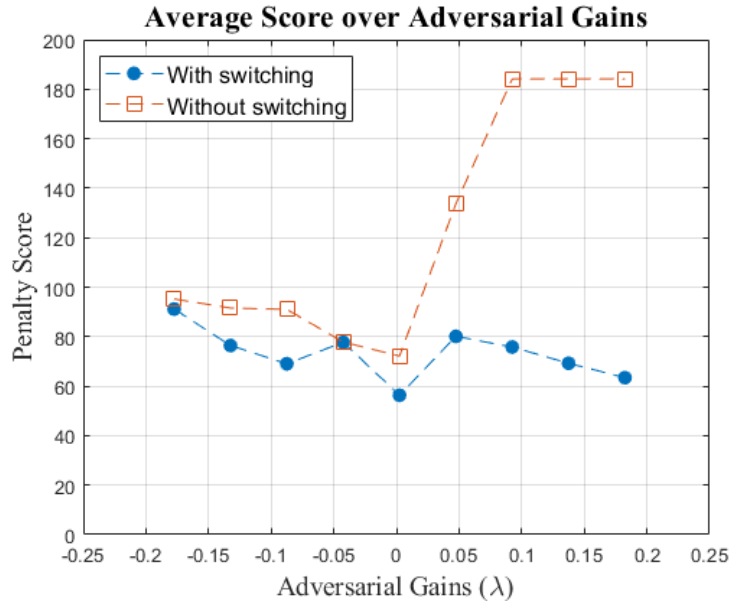


Figure 7.6: X-plane algorithm verification test.

The Figure 7.6 shows the average penalty scores over the changing adversarial gains with the specific adversary attacks. The verification was performed with 100 test cases for each of the tests, 1200 seconds of behavioral policy session, and 20 seconds of a test session for each of the tests. The score corresponds to the final 20 seconds of the test’s output performance calculation with (Equation 6.1), whereas instances in which the algorithm failed

to converge were replaced with the highest penalty score of 185. Each point represents the average penalty score of each histogram bin from that created with 9 bins over -0.2 to 0.2. For the analysis, the same tests were run before and after the implementation of the switching mechanism to verify the efficacy of the defense mechanism incorporated into the algorithm. 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 algorithm demonstrated significantly improved performance over failures that occurred in the absence of a switching mechanism. The trend with switching also demonstrated a reduced penalty for positive adversarial gains when compared to negative adversarial gains, reiterating the algorithm's strength to positive adversarial gains.

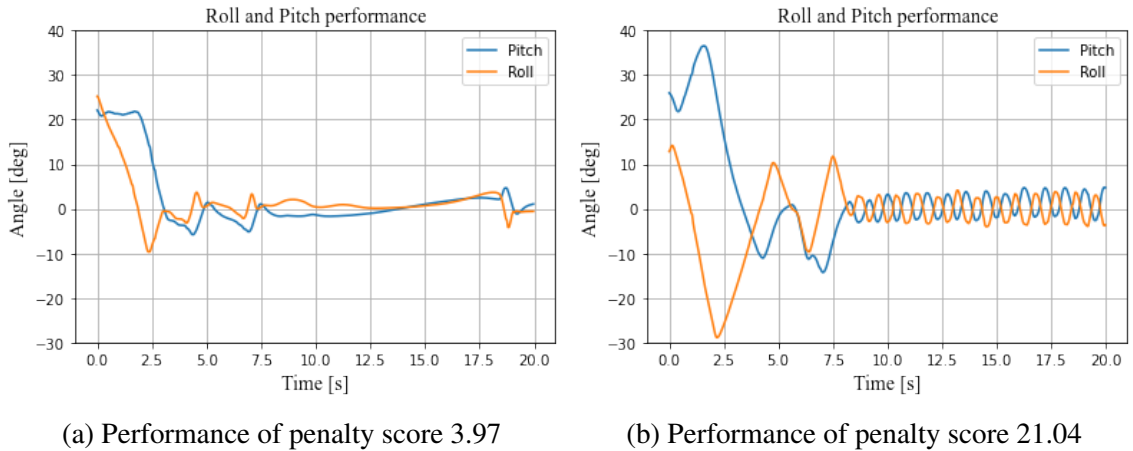


Figure 7.7: Performances of X-plane Cessna 172 samples.

From Figure 7.7, The sample performance plot is provided with different performance penalty scores to show how the performance score difference appears in the simulation itself. In Figure 7.7a, the pitch and roll performance converged to zero angles and stabilizes, and it showed a relatively lower penalty performance value, 3.97, from the last 20 seconds of simulation performance. But in Figure 7.7b, the pitch and roll angle approaches zero angles, but never stabilizes and consumes bigger control inputs that produce large angular rates in both state measures. Therefore, it showed with a bigger penalty score of 21.04.

7.3.4 Model Verification

In the model verification of the x-plane simulation, the model from Figure 7.7a was selected with a performance score in the algorithm test and it is verified with varying environment conditions.

Table 7.3: X-plane model verification parameter information

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

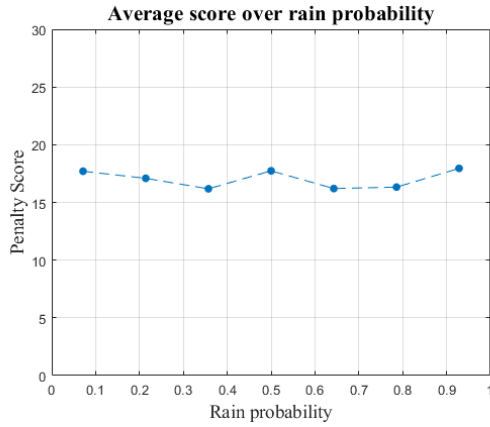
In this test, as described in Table 7.3, the weather profile including wind turbulence, cloud, and rain probability were chosen as changing environmental parameters. A total of 500 tests were performed with varying parameter combinations, tabulating the penalty score for each run of the simulation.

Table 7.4: X-plane correlation coefficient with performance penalty score

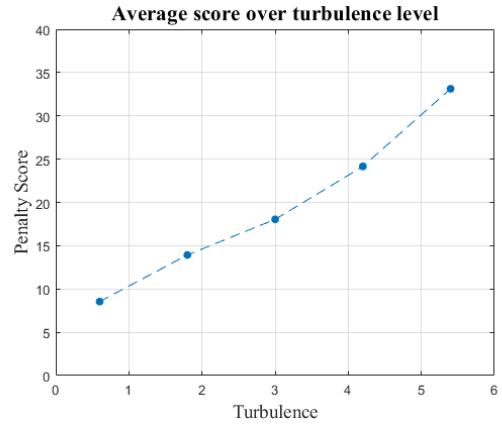
Parameters	Correlation coefficient
Cloud level	0.552
Rain probability	0.065
Turbulence	0.924

From the result described in Table 7.4, the turbulence showed the highest correlation to the penalty score which is an obvious result. Rain probability has the least correlation to the performance, while cloud showed bigger than rain percentage. Since rain can only be present when the cloud level is 4 or higher, this probably affected the correlation of cloud level to the model performance score.

The Figure 7.8 shows plots of average penalty scores with changing rain probability and turbulence measures, which showed the biggest and lowest correlations in Table 7.4. The turbulence factor almost showed linear growth in penalty with the larger turbulence, while the rain probability factor just showed flat random behavior between the 15 and 20



(a) Penalty score with rain probability



(b) Penalty score with turbulence

Figure 7.8: Score comparison with different environment parameters

score range. This demonstrates how the suggested verification approach may aid in the process of identifying the environment parameter that is most accountable for the simulation performance outcomes and helps in verifying the proposed control model.

CHAPTER 8

CONCLUSION

In this study, we developed a verification process for attack mitigating control AI with the integration of the VERIFAI toolkit with an RL algorithm. It is suggested with interconnection between off-policy RL, on-off switching adversarially robust mechanism, and verification toolkit 'VERIFAI'. First, from off-policy learning, it handles the unknown, nonlinear system to approximate optimal policy from the behavioral policy. From the approximation from cost and optimal controller equation, it provides actor and critic weight iteration to converge for optimality from quadratic value function. To defend against the adversarial attacks to this constructed system, actuator switching was applied to the behavioral policy where the adversarial gain is applied. the implementation of an on-off switching mechanism provides an MTD strategy to switch through different actuator gains. Lastly, VERIFAI and simulation were connected to provide the verification test-bed framework with several simulation application options.

To test the designed verification framework, the cart-pole model from OpenAI gym and Cessna 172 model were simulated with different control objectives. The performance analysis of the changing adversarial gains provided effective measures of the performance. The scores provided with performance traces showed weakness over the negative range of adversarial attacks, giving a detailed overview of the algorithm performances over the attack profiles. This successfully evaluated the performance response of the adversarially robust RL control algorithm. Additionally, the performances in varying system dynamics with environment parameters showed correlation value with penalty score, providing a major component of the environment that affects the performance. This evaluated the effectiveness of the model that is learned through the proposed RL method, demonstrating which factor affects the performance by correlation analysis from verification data. Thus,

we established a verification process that can be used in different dynamical systems.

The possible future work for this research is improving the performances by re-training of RL algorithm based on the verification result. The analysis of the verification test provides the weakness of the model in different simulations. The current study provides results for the performance based on different parameters, but the model can be improved further by designing a re-training mechanism. Re-training based on verification results can fill up the data need of the algorithm. From the analysis of behavioral movement to performance, the toolkit can provide information on which behavioral data is lacking, leading to a re-train of the model to improve the performance.

Future studies may also focus on constructing more complicated dynamic systems, such as multi-agent networked systems, and more advanced scoring mechanisms for verification. Currently, only trivial topics like attitude control have been evaluated with the verification method. The application of the suggested technique will be expanded by connecting with more complex control problems and involving additional systems. In terms of the scoring mechanism, the current vector norm has its limitations. For instance, the present vector norm simple scoring method may penalize movement toward the aiming point more severely than it would penalize staying in the unwanted rest position. For mechanism enhancement, it may be possible to apply the cost function that was employed in RL to verification scoring.

Appendices

Listing 1: X-plane parameter file

```
import numpy as np

'''ENV parameters'''

n = 4          # number of dynamic state parameters
m = 3          # number of control inputs in state space
               representation

'''Model parameters'''

target_equilibrium = [0.0, 0.0, 0.0, 0.0]
target_nums = 1
exploration = True
simtime = 1200
opt_simtime = 20
setup_time = 5
max_iteration = 200
sample_nums = 3500
t_sample = simtime/sample_nums
Nc = 26 # Critic basis size
Na = 16 # Actor basis size
gamma = 0.9

u_behavioral_gains = np.array([1.0, 0.5])
noise_gain = 1*np.array([[0.5], [0.5], [1]])
```

```

'''On-Off switching parameters'''
obstacle = None
adv_perturb = True
adv_type = "specific"
adv_gain = 1.0
switch = True
num_actuator = 12
num_switphase = 20 # number of switch phases
time_phase = simtime / num_switphase

'''Learning coefficients'''
S_coeff = 1*np.array([1.0, 1.0, 0.1, 0.1])
R_coeff = 1*np.array([1,1,1]) # input penalty coefficient
S = np.diag(S_coeff)*np.eye(n) # terminal state penalty matrix
Q_m_coeff = 0.5*np.array([1.0, 0.3, 0.1, 0.1])
Q_m = np.diag(Q_m_coeff)*np.eye(n) # # state penalty matrix
R = np.diag(R_coeff)*np.eye(m) # input penalty matrix

input_bound = np.array([1.0,1.0,1.0]) #

```

Listing 2: Cart-pole parameter file

```
import numpy as np
import gym
import gym_cartpole_qac as qac

env = gym.make('cartpole_qac-v0')

'''ENV parameters'''
sim = 'gym'
masscart = 1.0      # mass of the cart [kg]
masspole = 0.1     # mass of the pole [kg]
length = 0.5       # half-length of the pole [m]
gravity = 9.81     # Gravitational acceleration constant [m/s^2]
n = 4              # number of dynamic state parameters
m = 1              # number of control inputs in state space
    representation

'''Model parameters'''

init_eq = np.array([0.0, 0.0, np.pi, 0.0])
reset_init = True
reset_init_eq = np.array([0.0, 0.0, np.pi, 0.0])
aim_eq = np.array([[0.0, 0.0, 0.0, 0.0]])
aim_nums = 1
timeframe = [0,5]
iter_total = 108
iter_init_u = 100
```

```

max_converge = 200
num_samples = 50
Nc = 26
Na = 16

init_weight = False # Initial policy with initial weight matrix
wa_i = np.random.uniform(-0.3,-0.2, (16,1))
Finit = np.array([1,1,15,1])
init_gain = 10
noise_gain = 3
switch_gain = 5

dt = 0.01          # Simulation time step
gamma = 0.8

'''On-Off switching parameters'''
obstacle = None
adv_perturb = True
adv_type = "random"
adv_gain = 1.0
switch = True
num_actuator = 12
switch_token = np.random.randint(1,5, (iter_init_u,))

'''Learning coefficients'''
S_coeff = 1*np.array([1,1,100,60]) # state reward coeff
R_coeff = np.array([[0.01]]) # input reward coeff
S = np.diag(S_coeff)*np.eye(n) # state reward matrix
Q_m_coeff = 1*np.array([1,1,100,60])

```

```
Q_m = np.diag(Q_m_coeff)*np.eye(n)
R = np.diag(R_coeff)*np.eye(m) # input reward matrix

input_bound = 40.0 # -20 ~ 20
```

Table 1: X-plane 11 - Cessna 172 Skyhawk specifications

Engine:	-
Model	1 x Lycoming IO-360-L2A (piston)
Power	180 horsepower @ 2,700 rpm
Propeller	McCauley, 2-Bladed Fixed Pitch
Fuel:	-
Capacity	53 Gallons / 318 Lbs.
Recommended fuel	100 Octane Low Lead (100LL)
Fuel Burn (average)	8 Gallons per hour / 30 Liters per hour
Weight and Capacities:	-
Max. Takeoff Weight	2,550 lb. (1,157 kg)
Max. Landing Weight	2,550 lb. (1,157 kg)
Basic Empty Weight	1,640 lb. (744 kg)
Max. Gross Weight	918 lb. (416 kg)
Maximum Payload	910 lb. (413 kg)
Performance:	-
Cruise Speed	124 KIAS
Stall Speed (Clean)	48 KIAS
Stall Speed (Landing Configuration)	40 KIAS
Best Climb Rate	730 ft. pm (223 m. pm)
Maximum Structural Speed	129 KIAS
Landing Distance	1,335 ft. (407 m)
Service Ceiling	14,000 ft. (4,267 m)
Takeoff Distance	1,630 ft. (497 m)

Table 2: X-plane 11 - Cessna 172 Skyhawk speed profile

Profile	Unit (KIAS - Knots Indicated AirSpeed)
Stall speeds, Flaps down, Power Off	40 KIAS
Stall Speed, Flaps Up, Power Off	48 KIAS
Best Angle of Climb	62 KIAS
Best Climb Speed	74 KIAS
Takeoff (rotate) Speed	55 KIAS
Best Glide Speed	68 KIAS
Maximum flaps Extended Speed - (10/30 degree)	110/85 KIAS
Maneuvering Speed	99 KIAS
Maximum Structural Speed	129 KIAS
Never Exceed Speed	163 KIAS
Enroute Climb Speed	75-85 KIAS
Maximum Demonstrated Crosswind	15 KIAS
Maximum Glide Speed	68 KIAS
Precautionary Landing with Engine Power	65 KIAS
Landing without Engine Power - Flaps Up	70 KIAS
Landing without Engine Power - Flaps Down	65 KIAS

REFERENCES

- [1] W. Gao, Z. P. Jiang, and K. Ozbay, “Data-Driven Adaptive Optimal Control of Connected Vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, 2017.
- [2] L. Chen, Y. Chen, X. Yao, Y. Shan, and L. Chen, “An adaptive path tracking controller based on reinforcement learning with urban driving application,” in *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2019-June, 2019.
- [3] C. Li and K. Czarnecki, “Urban driving with multi-objective deep reinforcement learning,” in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 1, 2019.
- [4] Z. Ni and S. Paul, “A Multistage Game in Smart Grid Security: A Reinforcement Learning Solution,” *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, 2019.
- [5] F. Fotiadis, C. K. Verginis, K. G. Vamvoudakis, and U. Topcu, “Assured Learning-Based Optimal Control subject to Timed Temporal Logic Constraints,” in *IEEE Conference on Decision and Control*, 2021, pp. 750–756.
- [6] P. P. Sahoo and K. G. Vamvoudakis, “On-Off Adversarially Robust Q-Learning,” *IEEE Control Systems Letters*, vol. 4, no. 3, pp. 749–754, Jul. 2020.
- [7] T. Dreossi *et al.*, “Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems,” in *International Conference on Computer Aided Verification*, 2019, pp. 432–442.
- [8] A. Adadi and M. Berrada, “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI),” *IEEE Access*, vol. 6, 2018.
- [9] B. Haskins, J. Stecklein, B. Dick, G. Moroney, R. Lovell, and J. Dabney, “8.4.2 Error Cost Escalation Through the Project Life Cycle,” *INCOSE International Symposium*, vol. 14, no. 1, 2004.
- [10] W. Xiang *et al.*, “Verification for Machine Learning, Autonomy, and Neural Networks Survey,” Oct. 2018.
- [11] Q. Zhu and L. Guo, “Stable adaptive neurocontrol for nonlinear discrete-time systems,” *IEEE Transactions on Neural Networks*, vol. 15, no. 3, pp. 653–662, 2004.
- [12] R. Kamalapurkar, P. Walters, J. Rosenfeld, and W. Dixon, *Reinforcement learning for optimal feedback control*. Springer, 2018.

- [13] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” *Advances in neural information processing systems*, vol. 30, 2017.
- [14] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, “Sensitivity and generalization in neural networks: an empirical study,” *arXiv preprint arXiv:1802.08760*, 2018.
- [15] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, “Simulation-based adversarial test generation for autonomous vehicles with machine learning components,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1555–1562.
- [16] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, “Robust adversarial reinforcement learning,” in *International Conference on Machine Learning*, 2017, pp. 2817–2826.
- [17] A. Zribi, M. Chtourou, and M. Djemel, “A new PID neural network controller design for nonlinear processes,” *Journal of Circuits, Systems and Computers*, vol. 27, no. 04, p. 1 850 065, 2018.
- [18] K. Tanaka, “An approach to stability criteria of neural-network control systems,” *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 629–642, 1996.
- [19] J. H. Gillula and C. J. Tomlin, “Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 2723–2730.
- [20] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, “Hamilton-jacobi reachability: A brief overview and recent advances,” in *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, vol. 2018-January, 2018.
- [21] R. Baheti and H. Gill, “Cyber-physical systems,” *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.
- [22] S. A. Seshia, D. Sadigh, and S. S. Sastry, “Toward Verified Artificial Intelligence,” *Commun. ACM*, vol. 65, no. 7, pp. 46–55, Jun. 2022.
- [23] H.-D. Tran *et al.*, “NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems,” in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds., Cham: Springer International Publishing, 2020, pp. 3–17, ISBN: 978-3-030-53288-8.
- [24] H.-D. Tran, F. Cai, M. L. Diego, P. Musau, T. T. Johnson, and X. Koutsoukos, “Safety verification of cyber-physical systems with reinforcement learning control,” *ACM*

- Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–22, 2019.
- [25] C. Lv, X. Hu, A. Sangiovanni-Vincentelli, Y. Li, C. M. Martinez, and D. Cao, “Driving-style-based codesign optimization of an automated electric vehicle: A cyber-physical system approach,” *IEEE Transactions on Industrial Electronics*, vol. 66, no. 4, pp. 2965–2975, 2018.
- [26] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving Target Defense: creating asymmetric uncertainty for cyber threats*. 2019.
- [27] F. L. Lewis, D. L. Vrabie, and V. L. Syrmos, *Optimal Control*. 3rd ed. John Wiley & Sons, 2012, ISBN: 9780470633496.
- [28] K. J. Åström and B. Wittenmark, *Adaptive Control (2nd Edition)*. 2013.
- [29] T. Degris, M. White, and R. S. Sutton, “Off-Policy Actor-Critic,” *CoRR*, vol. abs/1205.4839, 2012.
- [30] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, *Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers*, 2012.
- [31] K. G. Vamvoudakis, “Q-learning for continuous-time linear systems: A model-free infinite horizon optimal control approach,” *Systems and Control Letters*, vol. 100, pp. 14–20, Feb. 2017.
- [32] R. Song, F. L. Lewis, Q. Wei, and H. Zhang, “Off-Policy Actor-Critic Structure for Optimal Control of Unknown Systems with Disturbances,” *IEEE Transactions on Cybernetics*, vol. 46, no. 5, pp. 1041–1050, May 2016.
- [33] S. A. Seshia and D. Sadigh, “Towards Verified Artificial Intelligence,” *CoRR*, vol. abs/1606.08514, 2016.
- [34] D. J. Fremont, X. Yue, T. Dreossi, A. L. Sangiovanni-Vincentelli, S. Ghosh, and S. A. Seshia, “Scenic: A language for scenario specification and scene generation,” in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019.
- [35] G. Brockman *et al.*, “OpenAI Gym,” Jun. 2016.
- [36] Laminar Research, *X-plane 11*, 2022.

- [37] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, “Comparing popular simulation environments in the scope of robotics and reinforcement learning,” *arXiv preprint arXiv:2103.04616*, 2021.
- [38] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, “Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators,” in *Towards Autonomous Robotic Systems*, M. Giuliani, T. Assaf, and M. E. Giannaccini, Eds., Cham: Springer International Publishing, 2018, pp. 357–368, ISBN: 978-3-319-96728-8.
- [39] R. V. Florian, “Correct equations for the dynamics of the cart-pole system,” *Center for Cognitive and Neural Studies (Coneural), Romania*, 2007.
- [40] J. R. Dormand and P. J. Prince, “A family of embedded Runge-Kutta formulae,” *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19–26, Mar. 1980.
- [41] Julian Lockwood, *X-Plane 11 Cessna 172 Pilot’s Operating Manual*, 2017.