In: AIAA SCITECH 2024 Forum, 8-12 January 2024, Orlando, USA. Paper number AIAA2024-1697
DOI:10.2514/6.2024-1697

Adversarial Proximal Policy Optimisation for Robust Reinforcement Learning

Bilkan Ince* and Hyo-Sang Shin†
School of Aerospace, Transport and Manufacturing, Cranfield University, MK430AL, U.K.

Antonios Tsourdos‡,
School of Aerospace, Transport and Manufacturing, Cranfield University, MK430AL, U.K.

Robust reinforcement learning (RL) aims to develop algorithms that can effectively handle uncertainties and disturbances in the environment. Model-free methods play a crucial role in addressing these challenges by directly learning optimal policies without relying on a pre-existing model of the environment. This abstract provides an overview of model-free methods in robust RL, highlighting their key features, advantages, and recent advancements. Firstly, we discuss the fundamental concepts of RL and its challenges in uncertain environments. We then delve into model-free methods, which operate by interacting with the environment and collecting data to learn an optimal policy. These methods typically utilize value-based or policy-based approaches to estimate the optimal action-value function or the policy directly, respectively. To enhance robustness, model-free methods often incorporate techniques such as exploration-exploitation strategies, experience replay, and reward shaping. Exploration-exploitation strategies facilitate the exploration of uncertain regions of the environment, enabling the discovery of more robust policies. Experience replay helps improve sample efficiency by reusing past experiences, allowing the agent to learn from a diverse set of situations. Reward shaping techniques provide additional guidance to the RL agent, enabling it to focus on relevant features of the environment and mitigate potential uncertainties. In this paper, a robust reinforcement learning methodology is adapted utilising a novel Adversarial Proximal Policy Optimisation (A-PPO) method integrating an Adaptive KL penalty PPO. Comparison is made with DQN, DDQN and a conventional PPO algorithm.

Keywords — model-free, Airsim/Unreal Engine, robust reinforcement learning (RL), robust optimization

I. INTRODUCTION

Reinforcement learning (RL) is focused on enhancing an agent's actions based on the rewards received from its interaction with the environment. The trained agent takes actions, receives rewards from the environment, and adjusts its policy accordingly. By continuously interacting with the simulated environment, the agent can autonomously learn the optimal policy that maximizes cumulative rewards. RL utilizes the Markov Decision Process (MDP) framework to solve the current problem and find an optimal policy. MDP determines the probabilities of transitioning between states, which are often estimated using state-transition probabilities. However, estimating these probabilities can introduce inaccuracies that may limit the application of MDP in new environments or when the model deteriorates. To address this, the use of finite states and actions becomes crucial in decision-making, considering uncertainties within the state transition probability matrix. An example of a challenge in applying reinforcement learning is using drones for various tasks.

The AI research community has shown considerable interest in investigating the vulnerabilities of deep reinforcement learning, particularly in light of recent successes in accomplishing diverse robust multi-agent learning tasks. While existing work often relies on the traditional adversarial learning framework, which makes strong assumptions about the adversary's capabilities, this may not be practical in real-world domains like autonomous driving. Assumptions that attackers can easily manipulate input images or interfere with the learning process of victim agents may not hold. [1]

While attacks involving manipulation of the environment have proven effective in causing a well-trained agent to fail at a given task, these attacks are often impractical in real-world scenarios.[2] For instance, in online video games, a pre-trained master agent receives input in the form of snapshots of current game scenes. It is challenging for attackers to breach game servers, secure permission to manipulate the environment, influence specific pixels in the input image, and subsequently launch an expected adversarial attack. Consequently, a novel method has been proposed in recent research to attack a proficiently trained agent. [3]
Unlike attacks relying on environment manipulation, this new approach is tailored for two-agent competitive games, where two participant agents vie against each other. The objective of this attack is to thwart one well-trained agent in the game by manipulating the behaviours of the other. Compared to methods involving environment manipulation, this novel attack on reinforcement learning is deemed more practical. To exploit the vulnerability of the victim agent, this attack does not assume complete control over the environment or observation of the victim agent. Instead, it relies on only the free access of the adversarial agent.[4]

Contributions

While previous research has primarily focused on static obstacles, the objective of this study is to conduct a thorough comparison of four distinct reinforcement learning (RL) algorithms—DDQN, A - PPO, and Td3(PPO)—for navigating both static and dynamic obstacles. Each algorithm possesses unique characteristics: DQN utilizes Q-learning and discrete action spaces for RL, A -PPO employs continuous action spaces for RL, and PPO is an on-policy, policy gradient RL method that also operates with continuous action spaces. The research was conducted within a simulated environment provided by AirSim, utilizing Unreal Engine 4 to create diverse training and testing environments. The novelty of this study lies in its comprehensive analysis, providing valuable insights into the strengths and weaknesses of different RL techniques. Thus, in this paper, we introduce the first data-efficient, robust, model-free RL method-based reinforcement learning policy.

In particular, these are our contributions:

- We design a new proposed black-box attack training that trains an adversarial agent to exploit the protagonist agent in an effective and efficient fashion.
- Demonstrate a baseline method of PPO to compare with the SA-PPO in a robustness demanding scenario.
- We compare different model-free methods to the developed model-free method.
- We show how our approach outperforms non-robust policy and the relative model-free methodologies.

Related Work

A. Robustness Control

Recently, [5] the concept of robustness has risen significant interest in scenarios driven by data, leading to the emergence of robust, model-free reinforcement learning (RL). Robust reinforcement learning (RL) has been the subject of investigation from various angles due to the potential presence of uncertainty in each element of RL, including observations, actions, transition dynamics, and rewards. One approach in robust RL is the Robust Markov decision process (RMDP) [6], which considers the worst-case perturbations in transition probabilities. This concept has been further expanded to encompass distributional settings and partially observed MDPs. Robust Markov decision processes examine the RL problem in situations where the transition model is affected by recognized and limited uncertainties. Parametric uncertainties are often considered, predominantly in the context of model-free approaches. Consequently, it is crucial to incorporate these uncertainties into a reinforcement learning agent's formulation in order to attain a more resilient dynamic system.

B. Backgrund on Adversarial Attacks

Adversarial attacks have their origin in Adversarial Example attacks to supervised classification systems. Let x be an input and f be a classifier model. An Adversarial Example to f can be crafted through solving the following optimization problem:

$$\min_{\delta} d(x, x+\delta) \qquad subject\ to\ f(x) does\ not\ equal\ f(x+\delta)$$

*Doctoral Researcher , School of Aerospace, Transport and Manufacturing, B.G.Ince@cranfield.ac.uk
†Professor, School of Aerospace, Transport and Manufacturing, A.Tsourdos@cranfield.ac.uk
‡Professor , School of Aerospace, Transport and Manufacturing, H.Shin@cranfield.ac.uk

where d is a similarity metric. The goal is to look for a minimal perturbation, δ, of an input, x, that can change the class assignment of f. An adversarial attack is considered a success if it makes the f outputs any wrong class. According to the different stages of the target model f, the attacks can be divided into two categories: attacks in the training stage, and attacks in the testing/deploying stage, depending on whether the attacks are performed during or after the learning of the model f [7]. However, in this paper we only focus on attacks in the deployment stage. In this stage, adversarial attacks can be divided into white-box attacks and black-box attacks:

- **White-box attacks:** In white-box attacks, the adversaries have total knowledge about the target model f, including algorithm train, data distribution, model parameters. Given a benign input, the attacker can compute adversarial examples as an optimization algorithm. The accessing of the model's internal data for white-box attacks corresponds to a strong adversarial capability, but such amount of knowledge of the victim's model is unrealistic in real world applications.
- **Black-box attacks:** they assume a more realistic threat model, where the adversaries have no knowledge about target model f the adversary is restricted from interacting with f via queries and classification outputs. It uses this information about input/output pairs to analyse the model's vulnerability. Although most of the attacks are formulated to mislead a classifier in supervised learning, the same principles apply in the context of RL. Instead of misleading the classifier f to produce a different label prediction for a given input x, the aim is misleading the policy π to produce a different action a for a given observation s. In a RL context, the adversary attacks a victim trained by a RL algorithm by perturbing the observations to make it take non-preferred actions that can result in reduction of the accumulated future rewards.

Gaining insight into models is often impractical in numerous real-world applications due to concerns related to intellectual property (IP) and support issues. In contrast, black box attacks don't necessitate comprehensive visibility into the models but encounter inefficiency and demand an excessive number of queries to generate an adversarial sample capable of compromising the evaluated model. In our study, the adversary's objective is to use a Reinforcement Learning (RL) agent, with a different training workflow that can learn a policy to make an adversarial attack with fewer queries and with a good success rate while maintaining other metrics like distortion at a minimum. To accomplish this, we focus on a limited adversary that only has access to the output of the deep neural network (DNN). The adversary lacks knowledge about the architectural decisions underlying the DNN, such as the number, type, and size of layers, as well as the training data employed to establish the DNN's parameters. These attacks fall into the category of black box attacks, wherein adversaries do not require familiarity with the internal specifics of a system to undermine its functionality.

C. Robust Policy Optimization

Most of the research in reinforcement learning (RL) primarily emphasizes its application in competitive games rather than focusing on robustness. Nevertheless, the RL framework is equally applicable to robust RL. The key distinction lies in the formulation of the opposing player, commonly referred to as the adversary. By incorporating strategies to handle uncertainty and disturbances through the adversary, it becomes possible to develop robust agents [8]. Introducing adversarial reinforcement learning, which operates on the concept of minimizing the maximum potential loss, enables us to address this aspect of robustness.

Our goal is to learn the policy of the RL agent (protagonist) denoted by, π, such that it has higher reward and better robust (generalizes better to variations in test settings). [9] In the standard reinforcement learning setting, for a given transition function $\mathcal{P}$ we can learn policy parameters θ^{π} such that the expected reward is maximized.

$$\rho(\pi; \theta^{\pi}, \mathcal{P}) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^{t} r(s_t, A_t) | s_0, \pi, \mathcal{P}\right] \tag{1}$$

In standard-RL settings, the transition function is fixed (since the physics engine and parameters such as mass, friction is fixed). However, in our setting, we assume that the transition function will have modeling errors and that there will be differences between training and test conditions. Therefore, in our general setting, we should estimate policy parameters θ μ such that we maximize the expected reward over different possible transition functions as well.

$$\rho(\pi; \theta^{\pi}) = \frac{\mathbb{E}}{\mathcal{P}}\left[\mathbb{E}\left[\sum_{t=0}^{T} \gamma^{t} r(s_t, A_t) | s_0, \pi, \mathcal{P}\right]\right] \tag{2}$$

II. THE APPROACH TO REINFORCEMENT LEARNING

The Deep Q-Network (DQN) has demonstrated the ability to be directly trained using raw images, making it a viable option for a detect and avoid (DAA) algorithm. However, this architecture tends to overestimate action values and requires a lengthy training process to achieve reasonable results. The DQN consists of two networks with identical hyperparameters: the evaluation network, which approximates the action value function using $Q(s, A; \theta)$ and the target network, which employs $Q(s, A; \theta^{-})$. To address the overestimation issue, the Q-values are updated using the following equations:

$$Q(s, a) \leftarrow Q^{\pi}(s, A) + \alpha[R_{t+1} + \gamma Q(s', A') - Q(s, A)\,] \tag{3}$$

$$a = \max_{a'} Q(s', A') \tag{4}$$

$$q_{estimated} = Q'(s', A) \tag{5}$$

In traditional DQN, a single stream of fully connected layers is utilized to estimate the Q-value for each action-state pair after the convolution layers. However, the dueling network approach introduces two separate streams of fully connected layers to compute the value and advantage functions independently [10]. Another variation, known as double DQN (DDQN), employs two different Deep Neural Networks: The Deep Q-Network (DQN) and the target network, Q_{tnet}.

$$Q_{qnet}(s, A) \leftarrow R_{t+1} + \gamma Q(s', A') \tag{6}$$

$$a = \max_{a'} Q_{qnet}(s', A') \tag{7}$$

$$q_{estimated} = Q_{tnet}{}'(s', A) \tag{8}$$

The DDQN updates the Q -values using equations (6) and (7), and $Q_{\text{estimated}}$ is obtained using the target network as shown in equation (8). By combining the concepts of the duelling network and double DQN, both the value and advantage functions are computed separately using two streams of fully connected layers, which are then combined to calculate the Q-values [11]. This architecture typically consists of three convolutional layers and three fully connected layers. The convolutional layers have specific filter sizes in terms of height, width, and channel, while the fully connected layers are part of the duelling architect.

Actor-critic structure. The actor-critic reinforcement learning method integrates both value-based and policy-based techniques [11]. This approach involves two neural networks and an advantage function, A^{π_θ}. The advantage function computes the agent's TD Error or Prediction Error. The actor network functions as a policy gradient algorithm, selecting actions at each time step. Conversely, the critic network assesses the Q-value or offers guidance on adjustments. While the critic network discerns superior or inferior states, the actor utilizes the critic's findings to instruct the agent in seeking favourable states and avoiding unfavourable ones. Using an action-value function $Q^{\pi_\theta}(s, a)$, the critic measures how good the action taken is.

Actor-critic diagram overview can be demonstrated as shown in Figure 1. The solid lines depict a fundamental behavioural sequence in which the actor, guided by the policy, selects an action and executes it in the environment. Subsequently, the environment provides a new state and a potentially null reward. The dotted lines represent the learning process, involving sending the states at two consecutive time points and the reinforcement to the critic. The critic utilizes this information to calculate the TD error, which, in turn, is employed by the critic to update V_{s_t} and relayed to the actor for policy update. In cases where the critic is founded on state-action pairs, it also necessitates actions from times t and t+1 to compute the TD error and update $Q(s, a)$.

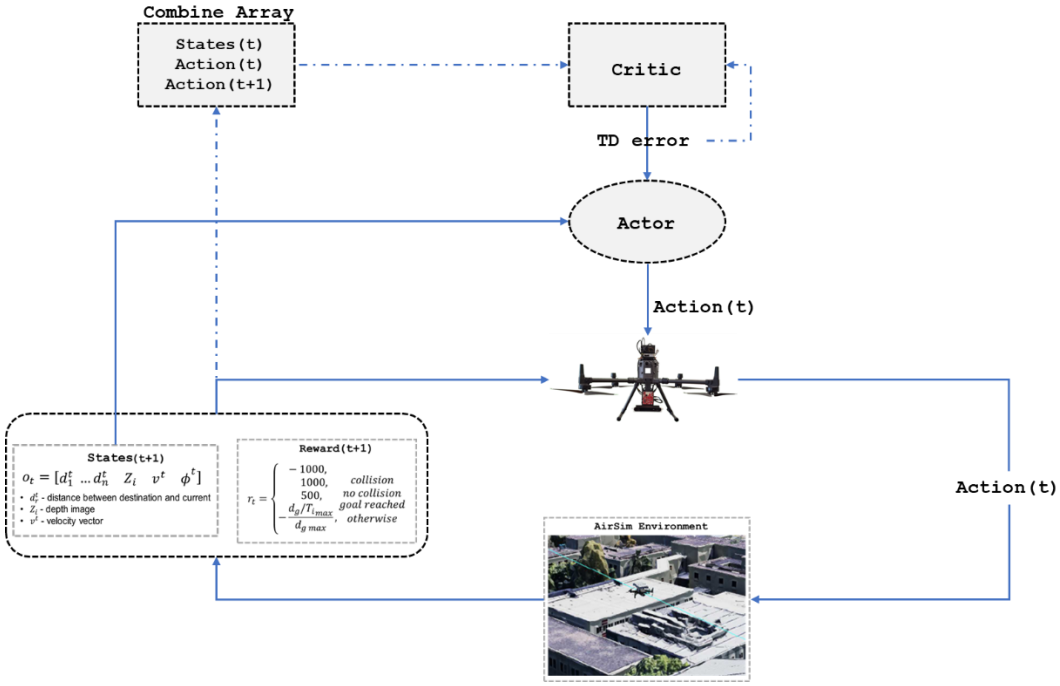


Fig. 1: Actor-Critic Structure Layout

[11] Through a policy network π_θ, the actor controls how the agent behaves. Utilising both these concepts we can reproduce the policy gradient as

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - v_{\pi_\theta}(s) \tag{9}$$

where, $A^{\pi_\theta}(s,a)$ represents the advantage function, which measures the difference between the Q value for action, a in state, s and the average value of that state.

Proximal Policy Optimisation (PPO) Algorithm. Employing the actor-critic framework for agent training has shown that the actor often encounters significant variability during training, impacting the performance of the trained agent. To address this instability in actor training, recent studies suggest the use of the PPO algorithm. This algorithm introduces a novel objective function known as the "Clipped surrogate objective function" to stabilize actor training. The implementation of this new objective function helps limit policy changes to a narrow range.

PPO follows an on-policy learning strategy, where the decision-making policy is updated using a small sample of experience obtained from interacting with the environment. After using this batch of experiences to update the policy, the experiences are discarded, and a fresh set is collected using the most recent policy revision. The policy parameter is updated based on the clipped loss function described in equation (10). This loss function includes a hyperparameter e, which determines the extent to which the updated policy should deviate from the previous policy.

$$L^c(\theta) = E_{\pi_{\theta_t}}[\min(\sigma(\theta)a^{\pi_{\theta t}}, clip(\sigma(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta t}})] \tag{10}$$

where $\sigma(\theta)$- probability ratio, $A^{\pi_{\theta t}}$ – advantage function, ϵ- hyperparameter.

An alternative approach to the clipped surrogate objective, or even in conjunction with it, is to incorporate a penalty on the KL divergence and adjust the penalty coefficient to achieve a specific target value of KL divergence (dtarg) during each policy update. Although in our experiments we observed that the KL penalty yielded inferior results compared to the clipped surrogate objective, we have included it here as an important baseline. In its simplest form, the algorithm follows these steps for each policy update.

III. ROBUST REINFORCMENT LEARNING APPROACH

Several architectures DQN is shown to be trainable directly benefit from raw images [12], it provides a valid option for a detect and avoid (DAA) algorithm. Although this architecture can eventually achieve reasonable results, it tends to overestimate the action, Q, values and takes a long time to train. Given a policy $A_t = \pi(s_t)$, the action value in terms of a state-action can be defined below.

$$Q(s,a) = E[R_t|s_t, A_t, \pi] \tag{11}$$

[13] combines the ideas of dueling network and double Q-network for two streams of fully connected layers which are built to compute the value and advantage functions separately, which are finally combined together for computing Q-values.

This work aims to train and test both on simulation and experimentally robust reinforcement learning for a single agent. Given the Markov decision equation for reinforcement learning,

$$R(\pi, \boldsymbol{P}) = \mathbb{E}^{\pi,\boldsymbol{P}}\left[\sum_{t=0}^{\infty} \lambda^t r_{s_t a_t} | s_0 \sim p_0\right] \tag{12}$$

where s_0 and p_0 the robust policy can be shown as below,

$$\pi^* = arg\max_{\pi} \min_{\bar{\pi}} R(\pi, \boldsymbol{P}) \tag{13}$$

where a transition kernel $\boldsymbol{P}$ which gives transition probabilities $P_{sa} \in R_+^{|\mathbb{S}|}$ for all state-action pair *(s, a),* some rewards r_{sa} for each state-action pair *(s, a)* and a discount factor $\lambda \in (0, 1)$.

A-PPO. In the Adversarial MDP (A-MDP), we introduce an adversary $v(s): S \rightarrow S^{-1}$. The role of the adversary is to perturb only the state observations of the agent, while the action is still determined by the agent's policy $\pi(A|v(s))$ It is important to note that the environment still transitions from the true state s, rather than the perturbed state ν(s), to the next state. As a result, since $v(s)$ may differ from s, the action taken by the agent based on $\pi(A|v(s))$ may not be optimal, allowing the adversary to decrease the reward. In real-world RL problems, this adversary can be seen as representing worst-case noise in measurements or uncertainty in state estimation. It is worth noting that this scenario differs from a two-player Markov game in which both players observe the unperturbed true states of the environment and directly interact with it, and the opponent's actions can impact the true state of the game.

As in most general cases where the policy is stochastic, the total variation is hard to compute for most distributions. We can use an upper bound using KL divergence. [14] has regularised the KL-divergence over all s from sampled trajectories leading to the following adversarial regulizer for PPO,

$$\mathcal{R}_{PPO} = \frac{1}{2}\sum_{s} \max_{\bar{s}\in B(s)} \frac{\left(\mu_{\theta_\mu}(\bar{s}) - \mu_{\theta_\mu}(s)\right)^T}{\sum\left(\mu_{\theta_\mu}(\bar{s}) - \mu_{\theta_\mu}(s)\right)} := 0.5\sum_{s} \max_{\bar{s}\in B(s)} \mathcal{R}_s(\bar{s}, \theta_\mu) \tag{14}$$

Black-Box A-PPO. Our threat model encompasses a diverse range of use-cases, and therefore, we assess our attack across various RL algorithms employing different objective functions. While the evaluation may not cover all possible use-cases, it demonstrates the practical feasibility of conducting fully Black-box attacks. Additionally, we establish the generalizability of our work by demonstrating the capability to model an agent with an unknown objective function, allowing predictions about its future behaviour. Given sufficient observation time, we may anticipate the agent's behaviour to an extent that enables disruption, and in certain instances, disruption may occur after a known delay. We utilise the seq2seq [15] algorithm for implementation as shown in the algorithm below.

Input: $N_{iter}, N_{epi}, \theta_{\pi}, \theta_{\pi_w}, w_0$
Initialize action-value function Q with parameters θ and policy π
Initialize target action value function Q' with parameters θ' and policy π^*
for $i=1,2\ldots N_{iter}$ **do**
  $\theta_i^{\pi} \leftarrow \theta_{i-1}^{\pi}$
  for $k=0,1, \ldots, N_{epi}$ **do**
    Sample trajectory $\tau_k = \{s_t^i, w_t, a_t, s_{t+1}\}$
    $(s_t^i, a_{1i_t}, a_{2i_t}, r_{1i_t}, r_{2i_t}) \leftarrow (\overline{\pi}_{\theta_i^{\pi}}, \pi_{\theta_i^{\pi}}, \tau_k)$
  end for
  $\theta_i^{\overline{\pi}} \leftarrow \theta_{i-1}^{\overline{\pi}}$
  for $k=0,1, \ldots, N_{epi}$ **do**
    Sample trajectory $\tau_k = \{s_t^{i\prime}, w_t{}', a_t{}', s_{t+1}{}'\}_{t=0}$ from the augmented MDP
    $\{(s_t^{i\prime}, a_{1i_t}{}', a_{2i_t}{}', r_{1i_t}{}', r_{2i_t}{}')\} \leftarrow (\overline{\pi}_{\theta_i^{\pi}}, \pi_{\theta_i^{\pi}}, \tau_k)$
  end for
Update Q' with $\{\tau_0, \ldots, \tau_{N_{epi}}\}$
Set $s_{t+1} = s_t, a_t$
Return: $\theta_{N_{iter}\pi}, \theta_{N_{iter}\pi_w}, \tau_{N_{epi}}$

Fig. 2: Algorithm Logic 'seq2seq'

Figure 3 depicts the training workflow for both the protagonist and the black box adversary. This workflow allows the adversary to depict and predict the protagonist behaviour with the minimal amount of information.

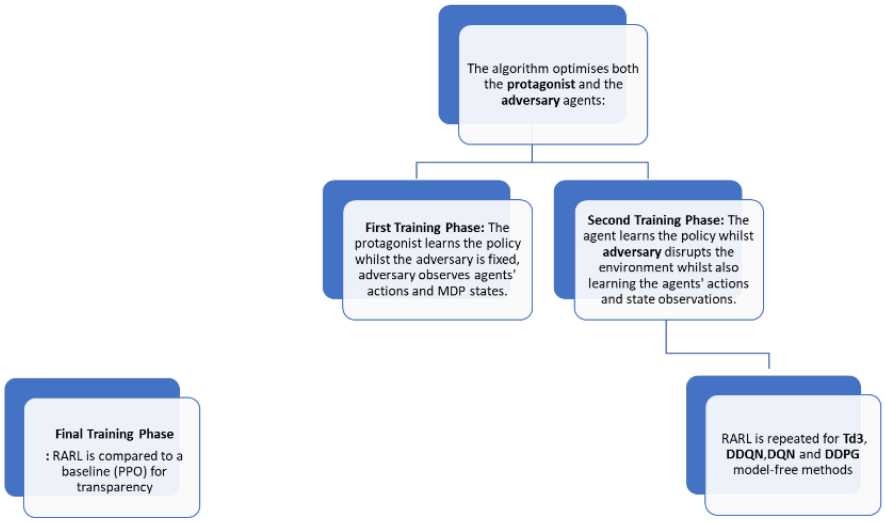


Fig. 3: Training workflow

IV. SIMULATION

A. *Mission Environment*

The objective of this research is to teach drones how to independently navigate around objects that are stationary or in motion. To accomplish this goal, a reinforcement learning approach was utilized. The training of a drone took place within a virtual setting, and Figure 4 provides a visual representation of the system's architecture and its components. The study employed the AirSim simulator's drone, which was integrated into an Unreal Engine 4-based simulated world. The models were constructed using Stable Baseline3 (SB3), a pre-existing implementation of reinforcement learning algorithms in PyTorch. AirSim facilitated the integration of Python with Unreal Engine 4 by offering a client-server plugin. It also provided functionalities for accessing real-time sensor inputs and drone control signals. During training, the drone captured in depth images of the environment, transmitting them to the Python client as input data for the model. The model, in turn, predicted the next action for the drone, which was executed using Python APIs developed by AirSim for drone control.

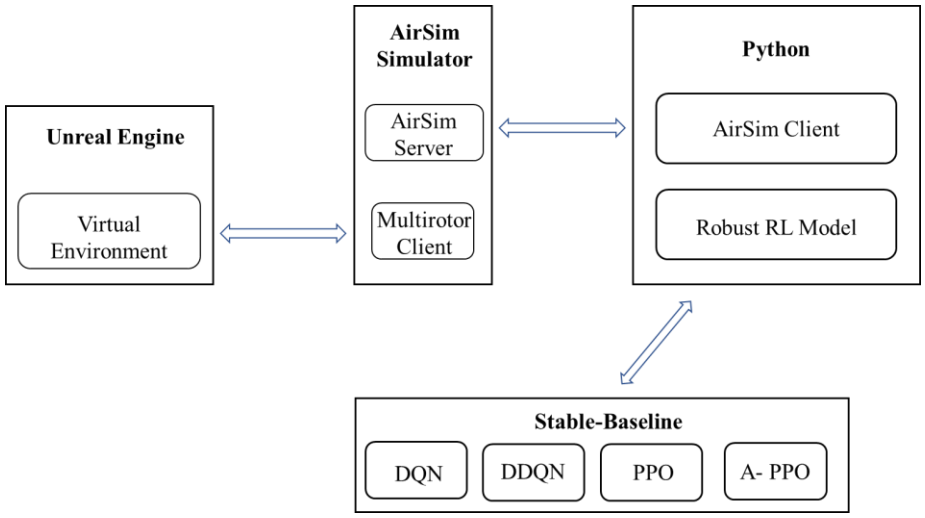


Fig. 4: Unreal Engine/Airsim/Python System Architecture

For simulation purposes Unreal Engine/Airsim (Cesium World Dynamic) are used to build a custom 3D dynamic world map platform for the UAS, linked to Python. Utilising Airsim platform, three accurate quadcopter models (dynamic and aero-propulsion specifications), representative urban scenes (3D OSM Buildings) including sample urban objects (buildings, vegetation, vehicles) and airspace intruders (sUAS) can be modelled. The complex geometry and noise of real-world environments greatly impacts DAA performance (numerous obstacles) and signal propagation. Therefore, the dynamic real-world environment in AirSim/Unreal allows several practical considerations. Weather and noise are a critical enabler for the definition of practical considerations within the dynamic synthetic environment.

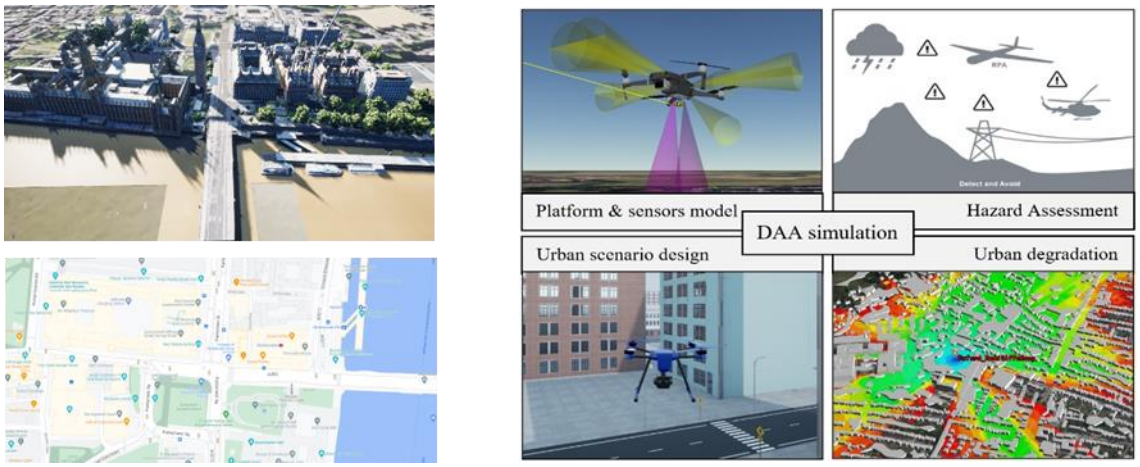


Fig. 5: (a) Synthetic Simulation Environment, Cranfield (Airsim/Cesium and Bing Maps) (b) DAA simulation overview

The obtained results provide a study over the extension of reinforcement learning to real-life experiments through optimization and robustness. In addition to ensure compliance with operational requirements with the airspace legislation and support sUAS. Software in the loop tests will comprise of several different simulation environments these including locations in Airsim/Cesium starting with DARTeC (Digital Aviation Research and Technology Centre), Cranfield, UK. Experimental tests will be carried out in Snowdonia Aerospace Centre, UK. For each test mission the flight phases will consist of take-off, cruise, and landing.

These factors define each mission's success, which are collected and statistically processed for major DAA events, enabling effective mitigations.

Analysis of the findings and simulation results leads to a holistic approach to implementation of sUAS operations, focusing on extracting critical DAA capability for safe mission completion, like minimum field-of-view and detection probability of the sensor system, and minimum manoeuvrability of the guidance and control system.

B. Practical Considerations

From the obtained results in simulation, the following points are presented to be considered for practical applications of the Detect and Avoid systems:

1. **Noise or bias on sensors to detect obstacles:** The accuracy of the detection algorithm is influenced by the presence of noise and bias in the sensors. Consequently, the performance of the avoidance algorithm is also impacted since it relies on the target information estimated by the detection algorithm of the sUAS. The presence of noise and unpredictable weather conditions can directly affect the detection of obstacles, leading to failures in avoiding them and assessing associated risks.[16]
2. **Computational delay***:* The ability to detect rapidly quickly and accurately approaching obstacles depends on the computational capabilities of the Detect and Avoid (DAA) system. Therefore, it is essential to evaluate its performance in reducing and mitigating delays. In software in the loop systems that require intensive computing, it becomes crucial to minimize computational delays resulting from complex environments. By minimizing lag, the DAA system can achieve improved performance in detecting and responding to obstacles.
3. **Weather effects:** The attitude control of the sUAS is affected by wind; rain can increase the noise on the camera image (detection). Light rain, while within tolerable flight safety conditions, can create noise on light-based sensors, and therefore reduce the DAA system capabilities.

V. DISCUSSION

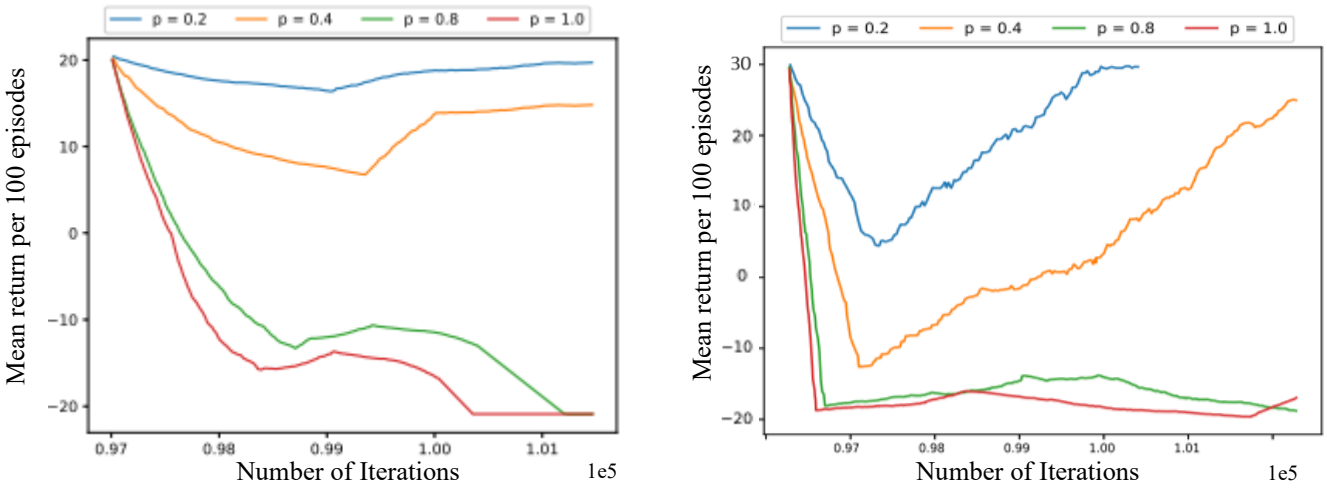
This section focuses on the training process of the RL agent and presents comparisons between the nominal problem and the proposed optimization method based on the simulation results. Additionally, it discusses the challenges encountered during experimental tests and addresses the practical considerations discussed in the previous chapter, all of which contribute to ensuring safer operations.

Changing the learning rate affects the instantaneous reward as the variance rate of exploration increases which may result in several decreased rewards. Overall, less success is achieved. Prior to training, the hyperparameters were defined as given in Table I.

TABLE I
HYPERPARAMETERS OF THE DDQN AGENT

Hyperparameter	Value	Description
training steps	500,000	total number of interactions with environment
minibatch size	32	stochastic gradient descent step size
replay memory	100,000	memory size of the most recent buffer
buffer size	500,000	improve sample efficiency for large buffer
target factor τ	0.01	update frequency from neural network to target
learning rate α	0.00025	optimizer learning rate
discount factor γ	0.98	balance rate of last reward and historical

From initial training of the A-PPO RL algorithm we can observe that the average Q-value (noise) fluctuate between the action points of $Q(s, a) = 1 - 2.2$, and therefore is slowly converging to a value of 1.6. The average Q-value therefore, hasn't fully converged hence further work is undergoing to converge and produce further results regarding the comparative PPO algorithm.



(a) PPO (b) A-PPO

Fig. 6: Fixed adversarial attack for A- Adversarial for PPO and A-PPO

The mean return value per 100 episodes describes the sum of all rewards that the agent expects to receive when following the policy from the state to the end of the episode. In PPO training the values of p equal to 0.8 and 1.0, the performance during test-time significantly declines under non-adversarial conditions. Conversely, when p is set to 0.2 and 0.4, policies trained under adversarial attacks exhibit nearly comparable performance to the original policy in non-adversarial conditions. However, as we can see the A-PPO has been able to very successfully to cope with high adversary attacks of p (Pattack) = 0.8 and 1.0.

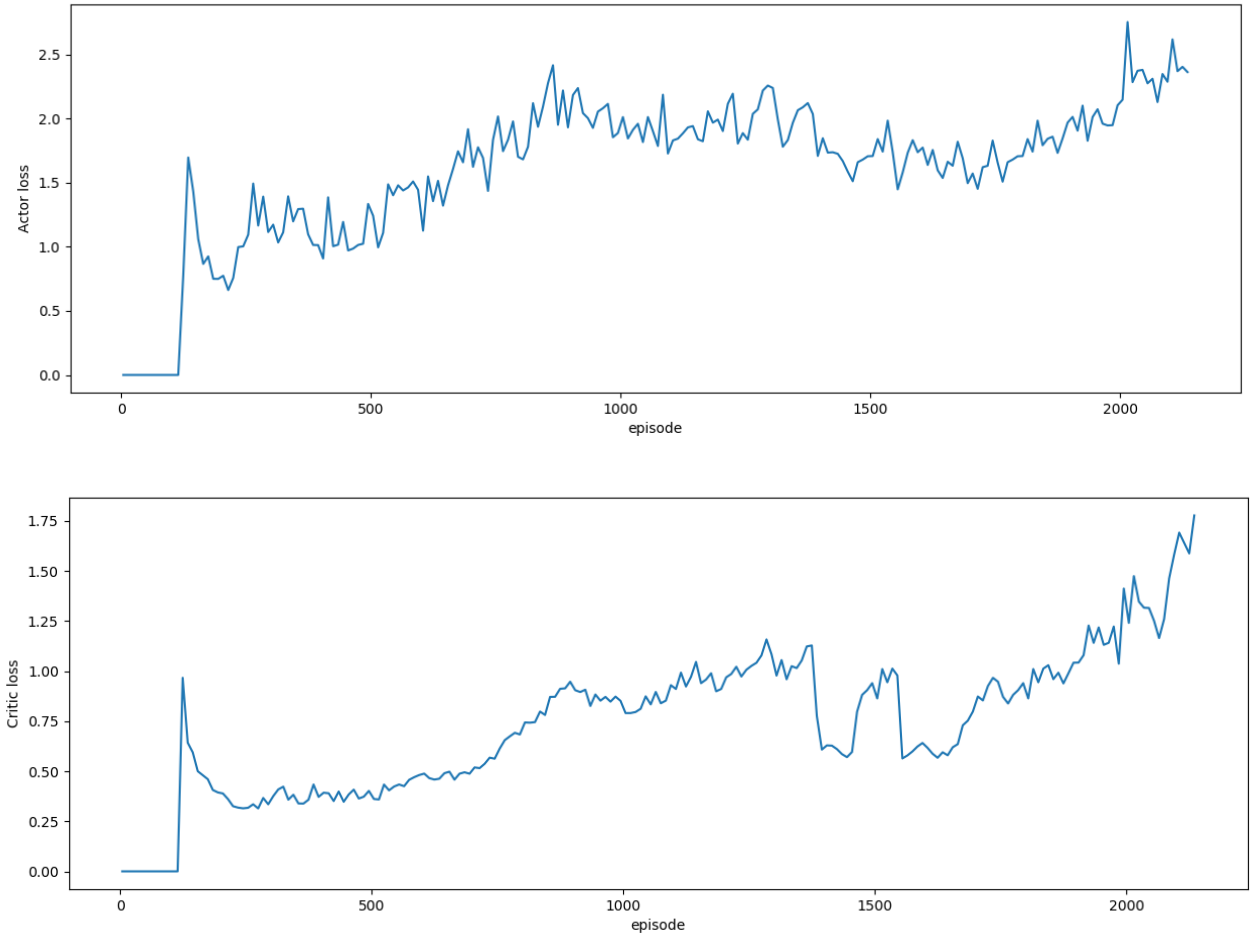


Fig. 7: Actor-Critic Structure

Figure 7 describes the loss of both the actor and the critic. The Actor loss is based on policy gradients with the Critic as a state dependent baseline and computed with single-sample (per-episode) estimates. The loss for the combination of the actor-critic model can be given as,

$$L = L_{actor} + L_{critic} \tag{15}$$

$$L_{actor} = -\sum_{t=1}^{T} log\pi_\theta(a_t|s_t)[G(s_t, a_t) - V_\theta^\pi(s_t)] \tag{16}$$

Where T- the number of timesteps per episode, s_t-state at timestep, a_t-action, π_θ −the policy parameterized by θ, V_θ^π- is the value function (critic) parameterized by θ, G = G_t – the expected return for a given state. [17]

The *G-V* term in the L_{actor} represents the advantage function. which indicates how much better an action is given a particular state over a random action selected according to the policy π for that state. [18-19]

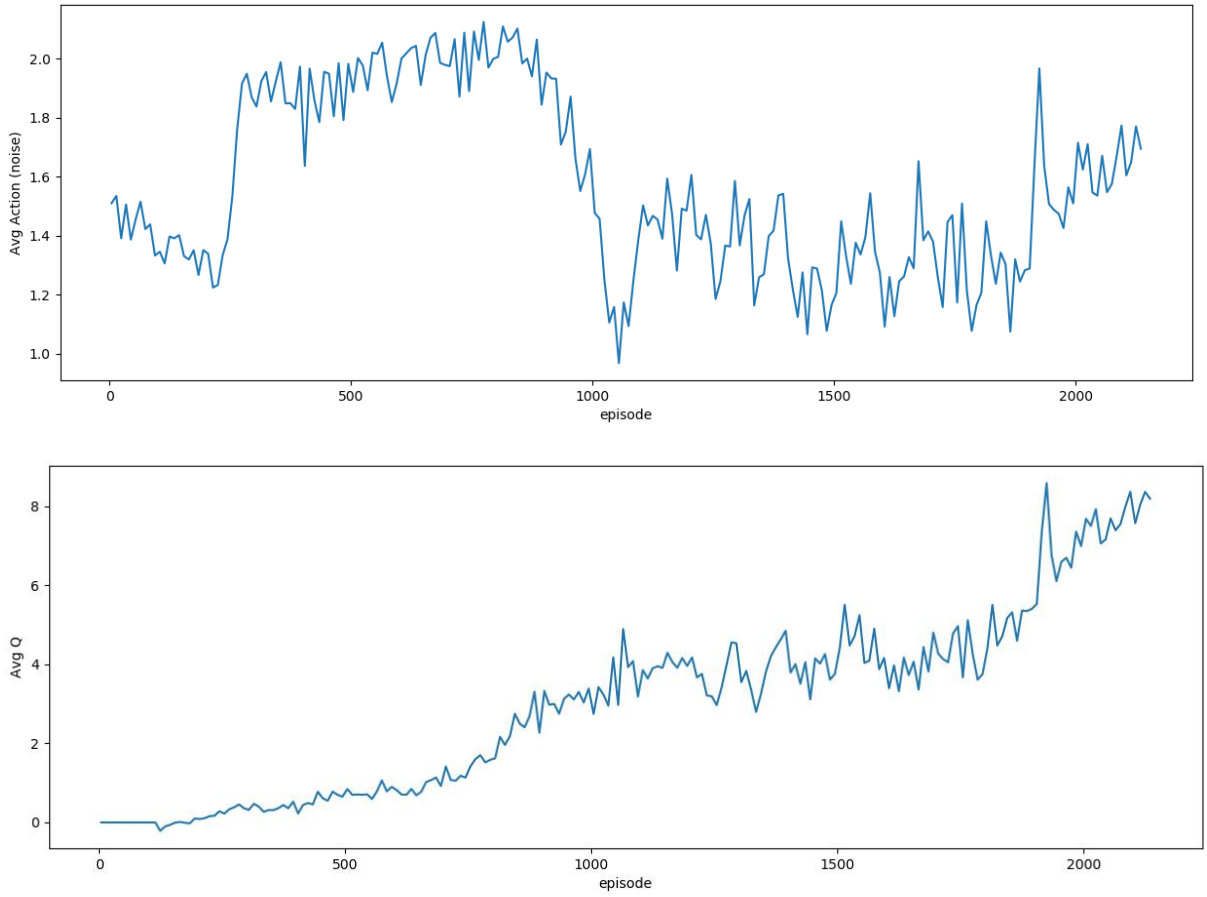


Fig. 8: Actor-Critic Structure

VI. CONCLUSION

This paper presents a comprehensive review of the state-of-the-art adversarial reinforcement learning technologies in conjunction with the simulation of realistic urban scenarios utilising AirSim and Python environments. Utilising PPO and the robust counterpart A-PPO, we observed the factor of robustness is crucial to avoid unknown factors. Through training the robust counterpart using a black-box theory, the adversarial attacks proved to be efficient and successful in manipulating the PPO agent. During the results we observed that high adversarial attacks result in worsened conditions for a non-robust PPO. Different missions are designed and

executed for representative scenes accounting for the common threads for obstacles in the sight of sUAS. Reinforcement learning training incorporated relevant factors such as noise on sensors, and dynamic real-world environments, including obstructed regions, complete the proposed simulation environment of London and Milton Keynes.

Changes in the dynamic environment settings and implementation of dynamic obstacles have a great impact on the manoeuvre of a PPO agent**.** Robust RL has a better collision success rate compared to conventional RL agent.

In the future this work can be extended including **intruder** dynamics and to integrate in the **worst-case** scenarios regarding the intruder collision, by implementing and updating the policy in the adversarial agent. Further improvements are needed for the collision **success rate** of the conventional RL and robust RL aiming for no collision.

VII. ACKNOWLEDGEMENT

This work is supported by Thales UK and EPSRC funding, grant number 2454266.

VIII. REFERENCES

[1] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction Second edition, in progress."

[2] Z. Hu, K. Wan, X. Gao, and Y. Zhai, "A Dynamic Adjusting Reward Function Method for Deep Reinforcement Learning with Adjustable Parameters," *Math Probl Eng*, vol. 2019, 2019, doi: 10.1155/2019/7619483.

[3] M. Marashi, A. Khalilian, and M. E. Shiri, "Automatic reward shaping in Reinforcement Learning using graph analysis," in *2012 2nd International eConference on Computer and Knowledge Engineering, ICCKE 2012*, 2012, pp. 111–116. doi: 10.1109/ICCKE.2012.6395362.

[4] A. Nilim and L. El Ghaoui, "Robust control of Markov decision processes with uncertain transition matrices," *Oper Res*, vol. 53, no. 5, pp. 780–798, Sep. 2005, doi: 10.1287/opre.1050.0216.

[5] X. Wu, W. Guo, H. Wei, and X. Xing, "Adversarial Policy Training against Deep Reinforcement Learning." [Online]. Available: https://github.com/psuwuxian/rl_attack

[6] T. Viet Bui, T. Mai, and T. H. Nguyen, "Imitating Opponent to Win: Adversarial Policy Imitation Learning in Two-player Competitive Games; Imitating Opponent to Win: Adversarial Policy Imitation Learning in Two-player Competitive Games," 2023. [Online]. Available: www.ifaamas.org

[7] "Implementing the Actor-Critic Model of Reinforcement Learning."

[8] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, Association for Computing Machinery, Inc, Apr. 2017, pp. 506–519. doi: 10.1145/3052973.3053009.

[9] S. Sarkar *et al.*, "Robustness with Black-Box Adversarial Attack using Reinforcement Learning," 2023. [Online]. Available: https://tinyurl.com/2p8pnjn6

[10] B. Ince, H.-S. Shin, and A. Tsourdos, "Optimization of a Robust Reinforcement Learning Policy," American Institute of Aeronautics and Astronautics (AIAA), Jan. 2023. doi: 10.2514/6.2023-0967.

[11] J. Wu and Y. Vorobeychik, "Robust Deep Reinforcement Learning through Bootstrapped Opportunistic Curriculum," Jun. 2022, [Online]. Available: http://arxiv.org/abs/2206.10057

[12] A. M. Deshpande, A. A. Minai, and M. Kumar, "Robust Deep Reinforcement Learning for Quadcopter Control." [Online]. Available: https://github.com/adipandas/gym_multirotor

[13] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1. Institute of Electrical and Electronics Engineers Inc., pp. 213–225, Mar. 01, 2020. doi: 10.17775/CSEEJPES.2019.00920.

[14] H. Zhang *et al.*, "Robust Deep Reinforcement Learning against Adversarial Perturbations on State Observations," Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.08938

[15] J. Moos, K. Hansel, H. Abdulsamad, S. Stark, D. Clever, and J. Peters, "Robust Reinforcement Learning: A Review of Foundations and Recent Advances," *Mach Learn Knowl Extr*, vol. 4, no. 1, pp. 276–315, Mar. 2022, doi: 10.3390/make4010013.

[16] J. T. Barron, "A General and Adaptive Robust Loss Function," Jan. 2017, [Online]. Available: http://arxiv.org/abs/1701.03077

[17] T. Hiraoka, T. Imagawa, T. Mori, T. Onishi, and Y. Tsuruoka, "Learning Robust Options by Conditional Value at Risk Optimization."

[18] L. Xie, S. Wang, A. Markham, and N. Trigoni, “Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning.”

[19] M. U. De Haag, C. G. Bartone, and M. S. Braasch, “Flight-test evaluation of small form-factor LiDAR and radar sensors for sUAS detect-and-avoid applications,” in *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Dec. 2016. doi: 10.1109/DASC.2016.7778108.

2024-01-04

Adversarial proximal policy optimisation for robust reinforcement learning

Ince, Bilkan

AIAA

Ince B, Shin HS, Tsourdos A. (2024) Adversarial proximal policy optimisation for robust reinforcement learning. In: AIAA SCITECH 2024 Forum, 8-12 January 2024, Orlando, USA.
Paper number AIAA 2024-1697
https://doi.org/10.2514/6.2024-1697