Hijacking an autonomous delivery drone equipped with the ACAS-Xu system

Adrien Gauffriau, David Bertoin, Jayant Sen Gupta

Hijacking an autonomous delivery drone equipped with the ACAS-Xu system

Adrien Gauffriau*†, David Bertoin†§, Jayant Sen Gupta‡†
*Airbus Operations, † IRT Saint-Exupery, ‡Airbus AI Research, §Institut de Mathématiques de Toulouse

***Abstract*—In this paper, we want to show that automated anti-collision systems in aeronautical industry such as ACAS-Xu are vulnerable to hijacking threats in a urban environment which is less controlled than conventional airspace. Using reinforcement learning methods, we demonstrate the possibility to hijack the mission of a delivery drone equipped with the ACAS-Xu system in a simulated environment. Our objectives are first, to illustrate the security (interception) vulnerabilities of autonomous system and secondly, to enrich reinforcement learning benchmarks with a new one that comes from an industrial aeronautical application.**

***Keywords*—Autonomous and connected systems, Resilience, Artificial Intelligence, Reinforcement Learning, Security, ACAS-Xu**

I. Introduction

Autonomy is one of the hot topics where the potential of artificial intelligence, both for perception and decision making tasks, opens new possibilities. Nevertheless, autonomous systems also raise public acceptance and certification challenges. If autonomous cars are one of the best known examples, we see the emergence of autonomous systems in aeronautics, including delivery drones, autonomous air cabs, or airplanes [16]. This paper will focus on these particular systems.

Whereas most important concerns of the aeronautic certification are dependability and safety, the ability of a system to resist malevolent attack is also a major issue. Our work mainly focus on this topic with the development of an attack (interception) of an avionics system. Aviation security is mostly focused on ensuring that airports are secured by controlling people and goods passing through. Once the aircraft is flying, security is obtained by monitoring specific parts of the airspace all aircraft should respect, like airways. Airways are designed to ensure separation between pairs of aircraft, and any aircraft not respecting these rules is identified, tracked, and eventually neutralized. Nevertheless, what is valid to ensure security from malevolent attackers for standard aviation will no longer be applicable in the urban air mobility (UAM) context. Even if the concept of airway is extended in UAM, the distance to the ground, the distance between airways, and the potential number of threats will make it extremely difficult for humans to supervise all the traffic.

Systems like autonomous avoidance systems are thus needed to ensure, among others, the safety, and security of UAM. Nevertheless, new methods for attacks inspired by video game testing (for instance [2] and [18]) may change the paradigm that was traditionally used. In this work, we illustrate this statement by developing an attack on the ACAS-Xu avoidance system (see [10]) that will possibly be used by future autonomous vehicles.

We consider the following setting: an autonomous delivery drone (target), equipped with the ACAS-Xu system for collision avoidance, whose mission is to reach a predetermined delivery area, is attacked by another drone (attacker) that tries to hijack it and lead it to a different delivery area. When no risk of collision is detected, it follows the heading to the delivery area. When an aircraft (drone) enters the risk zone around the target, its heading is updated according to ACAS-Xu recommendation. The attacker policy (its strategy) is coded in a neural network which is trained to bring the target to the alternate delivery area using reinforcement learning (RL). In this paper, we want to show how vulnerable is a drone that uses a systematic way to avoid collisions. We limited our study to the horizontal recommendations of ACAS-Xu as it is preferred to deal with non-cooperative traffic (see [13]).

This paper is organized as follows. Section II presents the ACAS-Xu avoidance system. Section III presents the related works. Section IV introduces the principles of reinforcement learning (RL). Section V presents an experimental setup demonstrating the effectiveness of such attacks. Finally, section VII summarizes and concludes this paper on future perspectives.

II. Overview of the ACAS system

Among the family of ACAS X, the ACAS-Xu is dedicated to drones and urban air mobility. It provides a horizontal resolution for conflicts using in real-time a set of lookup tables (LUT) that were computed offline. Using geometric parameters, the ownship consults the LUT on collision's probability for five different advisories : COC (Clear of Conflict), WL (Weak Left), WR (Weak Right), L (Left), R (Right). The system does not rely on the fact that the intruder (the attacker in our setting) also applies the same avoidance system. Therefore, this system can also be used to avoid any static object (tower, crane) or birds as

The selected advisory is the one that minimizes the probability of conflicts. The geometry of a conflict is given in figure 1, the parameters definition stands as:

- ρ (ft): Distance from ownship to the intruder
- θ (rad): Angle to intruder relative to ownship heading
- ψ (rad): Heading angle of intruder relative to ownship heading direction
- v_{own} (ft/s): Speed of ownship
- v_{int} (ft/s): Speed of intruder

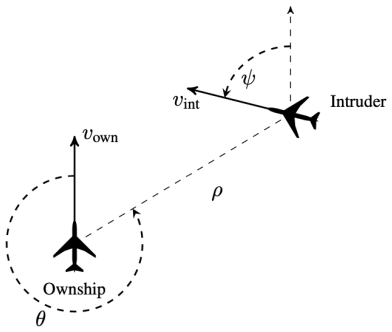

Fig. 1: ACAS-Xu geometry [20]

- τ (s): Time until loss of vertical separation
- R_{n-1}: Last recommendation provided by the ACAS-Xu

The 23 LUT provides the transitions costs between the previous advisory and the next advisory. When the ownship is not in the COC state, it has to initiate the turn given by the advisory. Otherwise it can continue its mission. More information on the ACAS-Xu system can be found in [25].

III. Related works

The use of RL methods to search for attack and interception scenarios is relatively classical. The originality comes from the use of these methods to find security weaknesses of an avionic systems. This will be illustrated in the following state of the art targeting the security in avionics, the safety of the ACAS-Xu system and the use of RL for unmanned aerial vehicle.

A. Security in avionics

The interest in the security of aeronautic systems is not new. In [39], the authors exploit vulnerabilities of the ACARS network to upload new flight plans in the Flight Management System (FMS) of aircraft. Nevertheless, the pilots still keep control of the aircraft, the attack mainly leads to increases in their workloads.

It is also possible to attack ground infrastructure like Instrument Landing System (ILS) [35]. The authors developed two different attacks of the ILS using radio signals. They demonstrate a systematic success rate with offset touchdowns of 18 meters to over 50 meters in lateral and longitudinal. Therefore, an attacked aircraft that performs a fully automatic landing miss the runway.

The literature is not limited to hacking the global system. [1] [34] focus on attacking aircraft networks. Nevertheless, most of these results remain theoretical or academic due to the complexity and cost of deploying such attacks. Moreover, none of these attacks enable complete control of the aircraft.

Even if there is an active research on safety in avionics, the results obtained are often mitigated by the presence of humans in the loop. Moreover, the popularization of drones and their accessibility has led to the emergence of new safety issues. In the future, the introduction of low complex autonomous systems may change this paradigm. Our contribution aims to make the avionic community aware of this future challenge.

B. ACAS-Xu

Detect and avoid is a task that guarantees the safety of flying vehicles, completed by the intervention of the pilot. For large aircraft, it is still the responsibility of the pilot that may be helped by systems that give advisory and recommendations for avoidance. The most famous one is the TCAS [31] that requires both planes to be equipped. The ACAS system [10] was developed for autonomous vehicles and mainly based on [22]. Several methods were explored to guarantee that this system is safe among Petri model [30] or formal methods [17]. The memory size (more than 4 GBytes) required by the look-up table of the ACAS-Xu system may be incompatible with the electronics of small drones. Thus, [19] developed training of neural networks that replace look-up tables with a small memory footprint. This raises the question of the safety of the neural network, which is still an open problem. [20] [7] propose formal methods to analyze neural networks and prove avoidance properties. Another approach [8] also based on formal methods, demonstrates that trained neural network behaves like the look-up table defined by the ACAS-Xu standard.

Our contribution aims to raise the security question of the ACAS-Xu system that differs from previous work that mainly focus on its safety. Up to our knowledge, this has never been done.

C. Reinforcement learning for Unmanned Aerial Vehicle

Recent successes in reinforcement learning have demonstrated the ability of reinforcement learning agents to outperform humans in many tasks [29], [36], [41], [42], [47]. Several recent works have sought to capitalize on the progress made in RL and Deep RL for the control and navigation of UAV. There are mainly two classical use cases of reinforcement learning for UAVs. The first one is flying control where the RL agents aim at providing stability and control and navigation. The second deals with mission planning where the agent is responsible for the high-level policy while control systems are implemented using classical methods such as Proportional-Integral-Derivative (PID) control systems [9].

While PID control has demonstrated excellent results in stable environments, it is less effective in unpredictable and harsh environments. Recently, several research projects have explored the possibility of using reinforcement learning to address its limitations. [49] compared the efficiency of a model based reinforcement learning controller with Integral Sliding Mode (ISM) control [52].

The authors of [15] trained neural-network policy for quadrotor controllers using an original policy optimization algorithm with Monte-Carlo estimates. The learned policy manages to stabilize the quadrotor in the air even under very harsh initialization, both in simulation and with a real quadrotor.

[21] train autonomous controllers flight control systems with state-of-the-art model free deep reinforcement learning algorithms (Deep Deterministic Policy Gradient [23], Trust

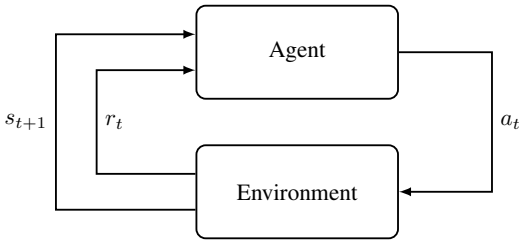

Fig. 2: Standard RL framework

Region Policy Optimization [37], Proximal Policy Optimization [38]) and compare their performance with PID controllers.

In [3], a sequential latent variable model is learned from flying sequences of an actual drone controlled with PID. This latent dynamic model is used as a generative model to learn a deep model-based reinforcement learning agent directly on real drones with a limited number of steps.

In [32], the authors combine a Q-learning [50] algorithm focusing on navigation policy with PID controllers. In [45], the navigation problem is decomposed into two simpler subtasks (collision-avoidance and approaching the target), each of them solved by a separate neural network in a distributed deep RL framework. An active field of research focuses on interception and defense against malicious drones. In a 1 vs 1 close combat situation, [48] demonstrates the effectiveness of an A3C [27] RL agent versus an opponent with Greedy Shooter policy [40]. In a multi-agent context, [51] uses a Multi-Agent Deep Deterministic Policy Gradient algorithm (MADDPG) [24] in an attack-defense confrontation markov game. [26] proposes a ground defense system trained with Q-learning to choose between high-level defense strategies (GPS spoofing, jamming, hacking, and laser shooting). While [12] and [5] use RL to train a drone attacker to intercept a target drone, [6] place the agent in the defender's position and train it with a Soft Actor-Critic algorithm [14] to avoid capture.

Our contribution also aims at intercepting a target UAV using an RL agent. However, it differs on two significant points. First, it highlights the security flaws of a deterministic policy dictated by the ACAS-Xu system for collision avoidance. Second, our attacker does not seek to capture the target directly but to guide it to a specific area where it can potentially be captured. This strategy does not require any attack equipment directly implemented on the attacking UAV and can easily be applied to any UAV.

IV. Reinforcement learning

Reinforcement Learning is a specific field of machine learning considering sequential decision-making problems. In RL, an *agent* interacts with its *environment* during a sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ At each time step t, the agent is provided a representation of the environment *state* $s_t \in \mathcal{S}$, where $\mathcal{S}$ defines a state space. According to s_t, the agent takes an *action* $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of all possible actions. Performing this action in the environment causes the environment to transition from s_t to s_{t+1}, and as a consequence of this transition, the agent receives a numerical *reward* $r_t \in \mathbb{R}$. Figure 2 illustrates the agent-environment interaction. The mapping of a state s to a probability of taking each possible action in $\mathcal{A}$ is called the agent's *policy* and denoted $\pi(a|s) = \mathbb{P}[A_t|S_t = s]$. Considering a discount factor $\gamma \in [0, 1]$, the return is defined as the discounted sum of rewards $R_t = \sum_{k=0}^{T} \gamma^{(k)} r_{t+k}$. Deep Reinforcement learning algorithms aim at finding the policy π_ϕ, represented by a neural network with parameters ϕ, that maximizes the expected return $J(\pi_\phi) = \underset{\tau\sim\pi_\phi}{\mathrm{E}}[R(\tau)]$ with $\tau = (s_0, a_0, ..., s_{T+1})$ the trajectory obtained by following the policy π_ϕ starting from state s_0. For continuous control problems (such as motor speed control), policy gradients methods aim at learning a parametrized policy π_ϕ through gradient ascent on $J(\pi_\phi)$. These methods rely on the policy gradient theorem [44]:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\phi}\left[\nabla_\phi \log \pi_\phi(a \mid s) Q^\pi(s, a)\right],$$

where $Q^\pi(s, a) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\phi}[R_t|s, a]$ is the action-value function. $Q^\pi(s, a)$ represents the expected return of performing action a in state s and following π afterwards.

Policy gradients methods typically require an estimate of $Q^\pi(s, a)$. An approach used in *actor-critic* methods consists in using a parametrized estimator called *critic* to estimate $Q^\pi(s, a)$ (π_ϕ thus represents the *actor* part of the agent). By relying on this principle, [43] propose the deterministic policy gradient algorithm to compute $\nabla_\phi J(\phi)$:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim p_\pi}\left[\nabla_a Q^\pi(s, a)|_{a=\pi(s)} \nabla_\phi \pi_\phi(s)\right].$$

The DDPG algorithm [23] adapts the ideas underlying the success of Deep Q-Learning [28] [29] to estimate Q^π with a neural network with parameters θ. In DDPG the learned Q-function tends to overestimate $Q^\pi(s, a)$, thus leading to the policy exploiting the Q-function estimation errors. Inspired by the Double Q-learning [46], the Twin Delayed DDPG (TD3) [11] addresses this overestimation by taking the minimum estimation between a pair of critics and adding a noise to the actions used to form the Q-learning target. These tricks, combined with a less frequent policy update (one update every d critic updates) result in substantially improved performance over DDPG in a number of challenging tasks in the continuous control setting. Algorithm 1 describes TD3's complete training procedure.

V. Development of an attack on ACAS-Xu

A. Notations

The following notations are used in the next sections:

- D_1: Delivery area. The target's destination objective
- D_2: Alternate delivery area. The attacker destination objective: the interception zone
- I_t: Initial position of the target.
- I_a: Initial position of the attacker.
- v_t: Speed of the target that is constant.
- v_a: Speed of the attacker. v_a^{max} is the maximum possible speed of the attacker
- The target corresponds to the ownship of the ACAS-Xu

Algorithm 1: TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ to T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{t=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics
  $\theta_i \leftarrow \underset{\theta_i}{\text{argmin}} \frac{1}{N} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = \frac{1}{N} \sum \nabla_a Q_{\theta_1}(s, a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end
end

- The attacker corresponds to the intruder of the ACAS-Xu. The attacker is not equipped with the ACAS-Xu.

B. Set up and objective

We consider an environment composed of two agents and two areas. The first agent is the delivery drone that has the mission to reach the delivery area D_1. The target (T) is equipped with the ACAS-Xu system to trigger autonomous avoidance actions by updating its heading. The second agent is the attacker (A) drone. The attacker aims at hijacking the target towards an alternate delivery area D_2, located in a different position than D_1 by exploiting the target's utilization of the avoidance ACAS-Xu system. The attacker's policy is trained for this purpose using Deep Reinforcement Learning. In our setting, for the sake of simplicity, both agents can only move in the same horizontal plan. Ascending and falling are not allowed. The figure 3 provides a graphical representation of the set-up.

The Xu version of the ACAS system is dedicated to drones, and only provides horizontal avoidance recommendations. Among the other version of the ACAS, it exists the Xa version dedicated to large aircraft that provides vertical avoidance like the TCAS. Thus, we restrict the interception to an horizontal plan since ACAS-Xu does not provide vertical avoidance recommendation. Even if, the target may have a vertical flight plan, it is very easy to configure, without reinforcement learning, the attacker to be always on the same horizontal plan of the target. In a future work, we may extend the study by considering a target equipped with the Xu and the Xa version and train an agent for a 3D interception.

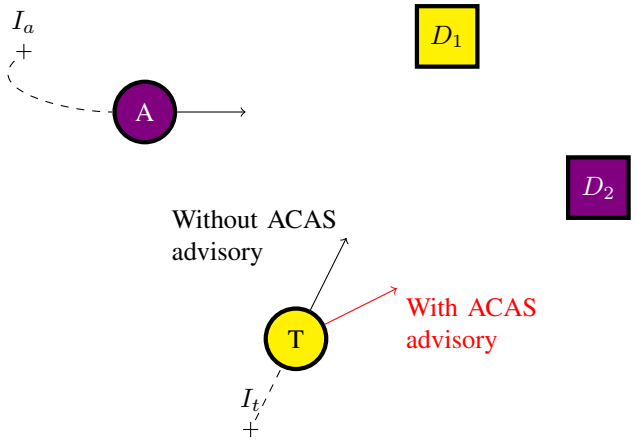

Fig. 3: Set up

C. Training environment

We implemented our training environment with the following settings:

a) State Space: The state of the environment is completely described by the state of the two agents (target and attacker) and the Cartesian positions of delivery areas. Each agent's state is composed of $P = (x, y)$ representing the agent Cartesian positions and a velocity vector $\vec{V} = (v_x, v_y)$ in the horizontal plan. The angle α of $\vec{V}$ is agent's heading.

b) Action Space: The attacker's actions at step n are represented by two updates $\lambda V_n \in [-200, +200]$ and $\lambda\alpha_n \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ representing respectively an update for the velocity and heading.

c) Transitions: The target agent's velocity is constant during the whole episode. Its heading is updated according to the ACAS-Xu system advisory. If the advisory provided is different from COC, the following heading update $\delta\alpha_n$ will be used:

- WL : + 0.15 rad
- WR : - 0.15 rad
- L : + 0.3 rad
- R : - 0.3 rad

If the advisory is COC, the $\lambda\alpha$ update enables the target to reach the heading to the delivery area with a maximum variation of 0.3 rad. We limit this variation to be more representative of a real drone maneuverability and avoid instability due to big turns. For both agents, the update of the speed vector is given by

$$\|V_{n+1}\| = \|V_n\| + \delta V_n$$
$$\alpha_{n+1} = \alpha_n + \delta\alpha_n$$

and position update by $P_{n+1} = P_n + V_{n+1}$.

d) Reward model: The choice of the reward function is not trivial. The training capacity and future policy of the RL agents are deeply impacted by the reward model used during training. The reward model design, often called reward shaping, may lead to strange and unexpected behaviors.

$$R_n = \begin{cases} 0 & \text{if } n = 0 \\ R_{n-1} & \text{if } D_{n-1} \geq D_n \\ R_{n-1} + (D_n - D_{n-1}) & \text{if } D_{n-1} < D_n \end{cases}$$

with $D_n = \|P_n^T - P^E\|$. The reward is increased for each step that globally reduces the distance between the target agent and the interception destination. We used the state-of-the-art Gym [4] framework to implement our training environment.

VI. Experiments and results

This section presents the different scenarios of experiments conducted. In all experiments, the target has a fixed speed of 400 ft/s and is following ACAS-Xu avoidance recommendations in case of presence of another flying object in its vicinity. When there is no obstacle, the drone follows the direction leading to its delivery area.

A. Training setups

We train three different agents depending on the maximum speed we allow the attacker in order to study the influence of the speed ratio between the attacker and the target. Three configurations are tested:

- 300 ft/s
- 600 ft/s
- 1000 ft/s

We fix the size of the playground as a square of 100000 feet. We developed a gym environment for training purposes to simulate the target behavior, following the ACAS-Xu function, and train the attacker policy using our RL algorithm. For each training scenario, we randomly draw the positions of the delivery zone, the alternative delivery zone where the attacker has to bring the target, the initial position of the target, and the initial position of the attacker. At the end of this stage, we get three trained agents designated by $\mathcal{A}_{300}$, $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$.

In every setups, we trained an RL agent using Stable-baselines3 [33] implementations of TD3. In each experiment, the agent is constituted by an actor and two critics. We use a two-layer feedforward neural network of 400 and 300 hidden nodes respectively, with rectified linear units (ReLU) between each layer for both the actor and critic, and a final two neurons output layer with tanh for the actor.

TD3 is an *off-policy* algorithm, during training transitions $(s_t, a_t, s_{t+1}, r_{t+1})$ are stored in a *replay-buffer* [28] and drawn randomly in the form of mini-batches during the weight update phase. We conducted all of our experiments using a replay-buffer of size 50000 and a mini-batch size of 512. TD3 trains a deterministic policy in an off-policy way, which is not favorable for exploration during training. We encouraged exploration of the TD3 agent by adding an action noise drawn from the Ornstein-Uhlenbeck process, as suggested in [23]. For every scenario, we trained our agents on 6 million steps. Table I provides a complete description of the training parameters used during training.

Parameter	Value
Training steps	6,000,000
Learning rate	0.001
γ	0.99
Policy delay	2
τ	0.005
target policy noise	0.2
Ornstein-Uhlenbeck Noise	0.01
Replay buffer size	50,000
Mini-batch size	512

TABLE I: Hyper-parameters used in TD3 agents training

B. Evaluation setups

Once the different models are learned, we evaluate their performance by randomly sampling new testing scenarios where we fix the positions of the delivery zone, the alternative delivery zone where the attacker has to bring the target, and the initial position of the target. For each of these scenarios, we randomly draw a significant number of initial positions of the attacker in the playground. For each of these random initializations, we run the scenario using the three trained policies and to assert whether the attacker succeeds in hijacking the target. For each test scenarios, we plot a map of the successful initial attacker positions and unsuccessful initial positions and estimate a percentage of successful attempts (success rate). We run 1000 different scenarios with the three trained policies.

Depending on the ratio between the speed of the target and the maximum speed of the attacker, we may theoretically know if the interception is possible depending on the initial geometry. We define the disc $\mathcal{D}$ centers on the delivery zone with a radius of R that is computed taken into account the distance d_{TD} between the initial position of the target (I_t) and the delivery zone (D_1)and the ratio $\frac{v_a^{max}}{v_t}$ between the speeds of the attacker and the target.

$$R = \frac{v_a^{max}}{v_t}\, d_{I_t D_1} \tag{1}$$

When the attacker is in $\mathcal{D}$, the interception is theoretically achievable since the attacker has the possibility to reach the delivery zone before the target and can thus interact with it. For a given scenario where the delivery area, the alternative delivery area and the initial position of the target are fixed, $\mathcal{D}$ is also fixed for all episodes.

To evaluate the performance of the hijacking policy, we can empirically estimate the reconstructed probability density function of the success rate on all test scenario. We also define following metrics:

M1: Number of successes over the number of episodes for initial target position in playground.

M2: Number of successes over the number of episodes for initial target position in $\mathcal{D}$.

M3: Number of successes over the number of episodes for initial target position not in $\mathcal{D}$.

$\overline{n_{step}}$: Mean of the number of steps when the episode is a success

$\overline{r_{dist}}$: Mean of the distance covered by the target over the initial distance from the target to the alternate delivery area when the episode is a success.

M1, M2 and M3 are success rates depending on the initial position of the target. $\overline{n_{step}}$, and $\overline{r_{dist}}$ are complementary metrics that evaluate the difficulty for the attacker to hijack the target.

C. Results

1) ***Step 1*** *Exploration of scenarios:* Firstly, we generate 1000 random scenarios by fixing I_t and D_2 as described in the previous section. In order to ease analysis, D_1 is always in the center of the playground. This choice does not change initial geometry exploration, although it may mask some corner cases. For each scenario, we run for the three trained agent $\mathcal{A}_{300}$, $\mathcal{A}_{600}$ and, $\mathcal{A}_{1000}$ 1000 episodes with random position for I_t. As expected during training, the agent $\mathcal{A}_{300}$ was not able to perform interceptions. Thus, we only focus on $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$ in the rest of this section.

We present on figure 4 the distribution of the success rate for the 1000 scenarios for agent $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$ with histograms. We see that the maximum speed of the attacker has a huge impact on the capacity of the attacker to intercept the target. This is confirmed by an average success rate of 39.2% for $\mathcal{A}_{600}$ and 91.4% for $\mathcal{A}_{1000}$.

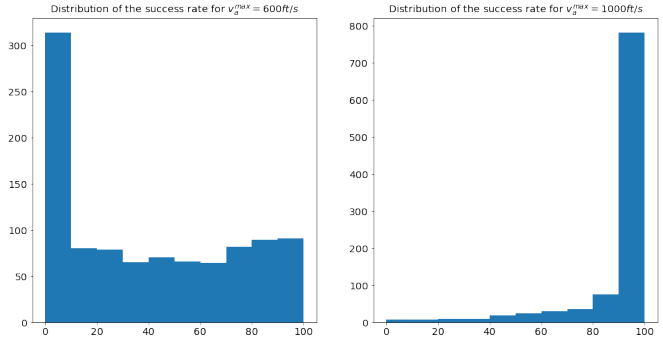

Fig. 4: Distribution of the scenario's success rate for $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$

Then, in figure 5, we present D_2 and I_t for the scenarios that have a good success rate ($> 90\%$) and a bad success rate ($< 10\%$) for trained agents $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$.

These figures highlight the impact of the initial geometry (relative position of the different elements) on the success rate. We recall that the D_1 area is in the center of the playground for all scenarios. When the ratio $\frac{v_a^{max}}{v_t}$ is high (top 2 of Figure 5), the distance $\overline{I_t D_1}$ has a huge impact on the success rate of a scenario. The closer the target is to the first delivery area D_1, the more the interception is difficult. Interpretation is not that straightforward when the ratio is low (bottom 2 of Figure 5). It seems that the learned policy is more efficient when the alternate delivery area D_2 is between or almost between the initial position of the target I_t and the initial delivery area D_1.

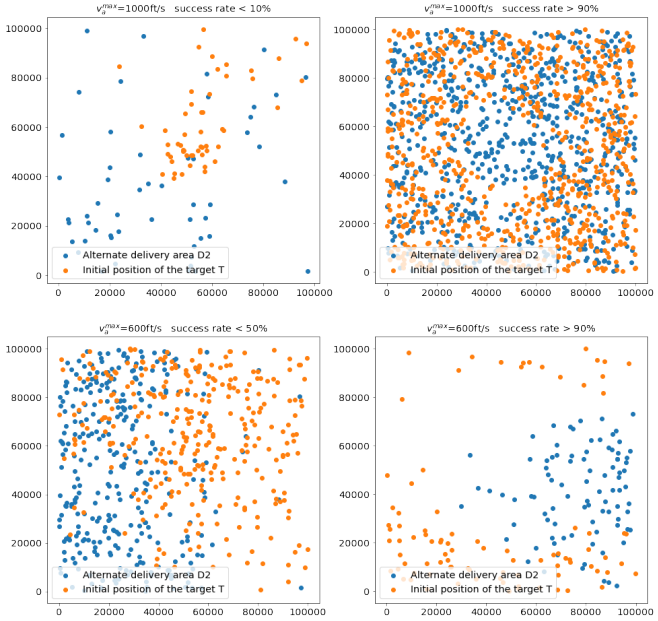

Fig. 5: Alternate delivery areas and initial positions of the target

2) ***Step 2*** *Exploration of selected scenarios :* this section focuses on some selected scenarios to provide a deeper look into how behave the different policies. In Figure 6, we plot the success rate of $\mathcal{A}_{1000}$ vs the success rate of $\mathcal{A}_{600}$. From this graph, we select 3 scenarios $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$.

- $\mathcal{S}_1$ is a scenario where both trained agent have a bad success rate;
- $\mathcal{S}_2$ is a scenario with a good success rate for $\mathcal{A}_{1000}$ and a relatively poor for $\mathcal{A}_{600}$;
- and finally $\mathcal{S}_3$ where both agents have a good success rate.

Each scenario correspond to a specific value of D_1, D_2 and I_t.

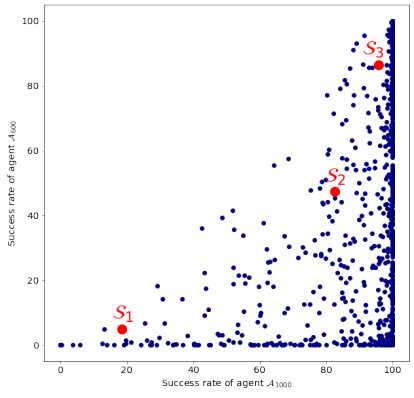

Fig. 6: Scenarios selected for exploration

For each couple $(\mathcal{S}_x, \mathcal{A}_y)_{x\in(1,2,3),y\in(1,2)}$, we launch 10000 episodes with random I_a positions. Then we compute in table

II metrics described in VI-B. The different graphs of Figure 7 present successes and failures for all initial positions of the attacker. We also plot D_1, D_2 and $\mathcal{R}$, the circle inside which the attacker should be able to hijack the target.

Scenario	Agent	m_1	m_2	m_3	$\overline{n_{step}}$	$\overline{r_{dist}}$
$\mathcal{S}_1$	$\mathcal{A}_{1000}$	.19	.56	.16	218	1.39
	$\mathcal{A}_{600}$	.06	.27	.05	346	2.21
$\mathcal{S}_2$	$\mathcal{A}_{1000}$	.84	.96	.73	197	1.47
	$\mathcal{A}_{600}$	.46	.79	.41	423	3.16
$\mathcal{S}_3$	$\mathcal{A}_{1000}$	.96	.96	NA	219	1.27
	$\mathcal{A}_{600}$	.86	.87	.77	370	2.15

TABLE II: Metrics for $(\mathcal{S}_x, \mathcal{A}_y)_{x\in(1,2,3),y\in(1,2)}$

For all scenarios, $\mathcal{A}_{1000}$ performs better than $\mathcal{A}_{600}$. This confirms the global success rate metric and highlights the necessity for the attacker to have a higher velocity than the target. The average ratio also shows that the task hijacking is more accessible when the maximum speed of the attacker is higher.

The success rate inside $\mathcal{R}$ (m_3) is consistently higher the than global one (m_1), (even if not always equals to 100% as we could expect).

From Figure 7, one characteristic which seems discriminant is the distance between the initial position of the target I_t and the delivery zone D_1: the smallest it is, the hardest it is to perform the hijacking. Even inside the $\mathcal{R}$, the success rate is poor, especially in $\mathcal{S}_1$. This can be explained by the fact that the circle has been defined considering the attacker goes directly to D_1. In practice, the initial heading the attacker is random so it has to change direction to head towards D_1. When there is not a lot of time, the change of heading is too long and the attacker cannot hijack. When time is longer, the effect is less visible, especially for $\mathcal{A}_{1000}$.

Failures inside $\mathcal{R}$ for $\mathcal{S}_3$ seem to form a pattern that would require additional investigations. It may show that our policy is not perfect or that our training scenarios did not explore enough certain configurations. This is the object of next step.

3) ***Step 3*** *Exploration of trajectories :* In this section, we present some representative observed trajectories to understand the red area of $(\mathcal{S}_3, v_{1000}^{max})$ and the red area that is inside $\mathcal{R}$ in $(\mathcal{S}_2, v_{1000}^{max})$.

Figure 8 presents the trajectories for two close initial positions of the attacker for scenario $(\mathcal{S}_3, v_{1000}^{max})$. This corresponds to the red area upper right of D_1 on figure 7. With the first position, the interception is not achieved. We can observe that the attacker has a trajectory that intercepts the target after D_1. On the contrary, for the second episode, the attacker intercepts the target before D_1. Failures of the $(\mathcal{S}_3, v_{1000}^{max})$ scenario have similar behaviors. The attacker tries to intercept the target after the D_1 area. This is mainly due to the reward function used during the training of the attacker agent. We used an heuristic that rewards the attacker when the target reduces the distance with D_2 in order to ease the learning of the policy. In order to push the attacker to hijack before the target reaches D_1, we would need to modify the reward to alleviate this undesired behavior. The proportion of episodes leading to such cases is low but definitely the reward function should be updated to avoid such phenomena.

Figure 9 shows the trajectories for two close initial positions of the attacker for the scenario $(\mathcal{S}_2, v_{1000}^{max})$. This corresponds to the red area inside $\mathcal{R}$. In the first episode, the initial velocity of the attacker is small and its acceleration is not strong enough to be able to interact with the target before it reaches D_1. When the initial velocity of the attacker is higher, in the second episode, the attacker is able to interact with the target almost from the beginning, and it is able to hijack it. These situations can occur when the attacker is opposite to the target compared to D_1. It could be solved by increasing the minimal initial speed of the attacker, fixing the initial heading of the attacker towards D_1 or increasing the attacker maximum acceleration.

VII. Conclusion

This work highlights the possibility of using reinforcement learning to train a malicious agent to hijack a delivery drone equipped with the ACAS-Xu avoidance system. Up to our knowledge, we are the first to demonstrate the possibility of fuzzing the algorithm of an autonomous avoidance system. With the current ambition in the aerospace industry for the development of autonomous systems (specifically air taxis and autonomous delivery drones), we are highlighting a security breach. The acceptance of these autonomous vehicles requires the resolution of these security issues.

As the ACAS-Xa system uses the same principle of tables, we believe that our method also applies.

Although we have made simplifying assumptions within the simulation environment, we believe that RL is efficient for attacking the ACAS-Xu system. An extension could be focused on adding complexity to the environment by adding static and dynamic obstacles and considering the vertical dimension. This should imply an extension of the action state of the attacker (new dimension) and an increase of the complexity of the reward function to handle obstacles.

In a future work, we consider training both agents with reinforcement learning in a zero-sum two-player games (as in [41], [42]) to produce collision avoidance system policies robust to malicious attacks.

We believe that, in order to improve the security, a reference use case should be provided with the ACAS-Xu. Access to Look Up Tables is unfortunately not possible for organizations that are not part of the RTCA/EUROCAE, the authority responsible for the normalization of the ACAS-Xu. Nevertheless, thanks to [20], neural networks approximating ACAS-Xu look-up tables are available. Therefore, we would like to replace the ACAS-Xu look-up table with these neural networks inside the gym environment and free it through our git repository (will be available for the conference). This would enable Reinforcement Learning and Security communities to work on this topic with the final objective of improving security in aeronautics.

Acknowledgement

This project received funding from the French "Investing for the Future – PIA3" program within the Artificial and Natural

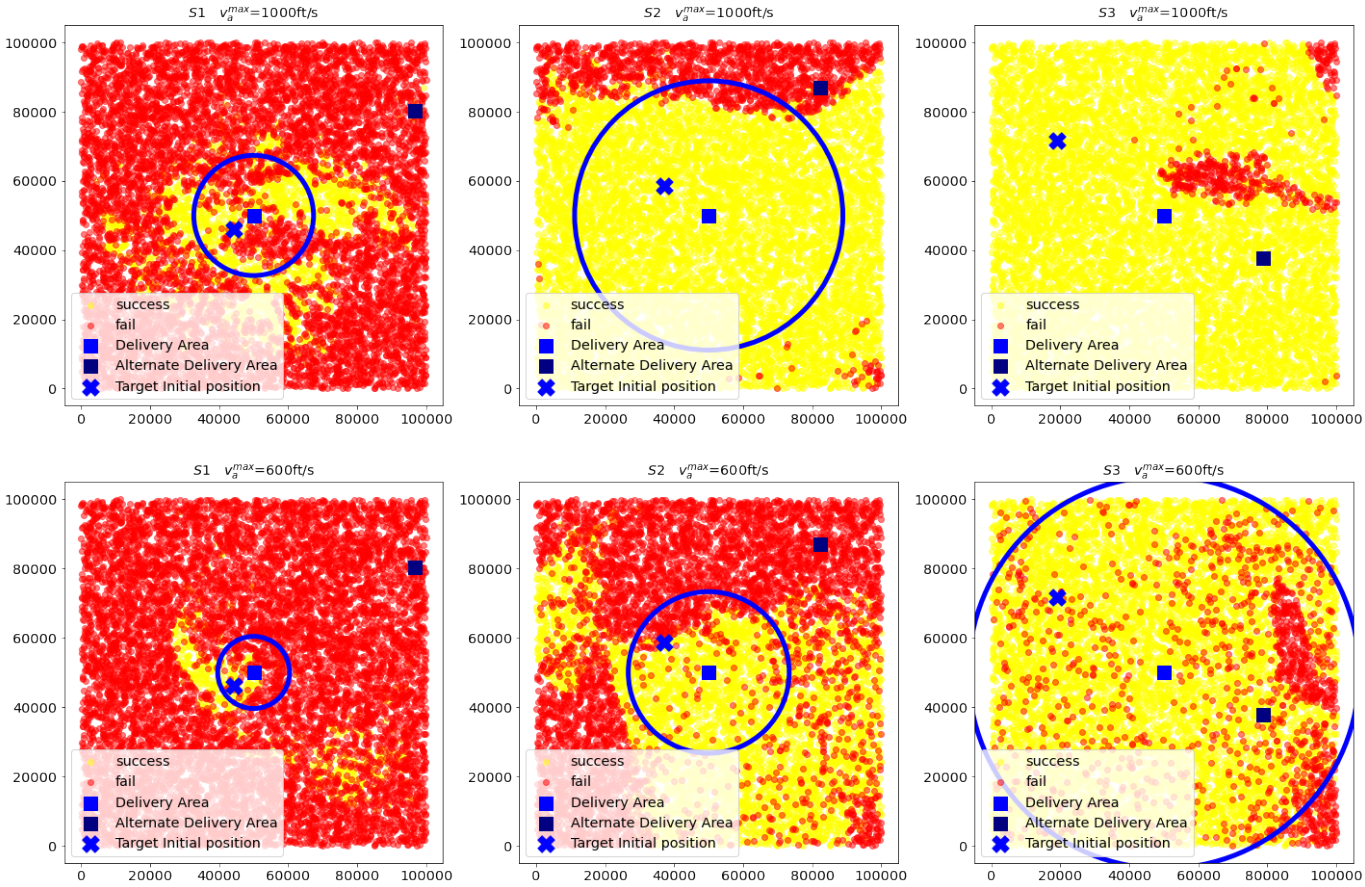

Fig. 7: Success/Failures for different initial position of the attacker among selected scenarios

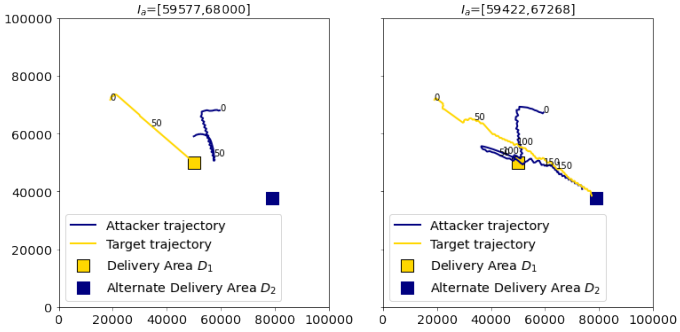

Fig. 8: Analyse of trajectories in S3 scenario for v_{1000}^{max}

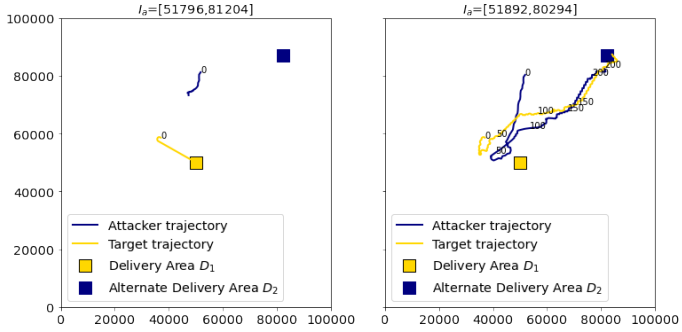

Fig. 9: Analyse of trajectories in S2 scenario for v_{1000}^{max}

Intelligence Toulouse Institute (ANITI). The authors gratefully acknowledge the support of the DEEL project[1].

REFERENCES

[1] R. N. Akram, K. Markantonakis, R. Holloway, S. Kariyawasam, S. Ayub, A. Seeam, and R. Atkinson. Challenges of security and trust in avionics wireless networks. In 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), pages 4B1–1–4B1–12, 2015.

[2] B.-C. A. S. E. Ariyurek S. Automated video game testing using synthetic and human-like agents. arxiv, 1906.00317v1, 2019.

[3] P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Learning to fly via deep model-based reinforcement learning. arXiv preprint arXiv:2003.08876, 2020.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

[5] E. Çetin, C. Barrado, and E. Pastor. Counter a drone in a complex neighborhood area by deep reinforcement learning. Sensors, 20(8):2320, 2020.

[6] Y. Cheng and Y. Song. Autonomous decision-making generation of

[1] https://www.deel.ai/

uav based on soft actor-critic algorithm. In 2020 39th Chinese Control Conference (CCC), pages 7350–7355. IEEE, 2020.

[7] A. Clavière, E. Asselin, C. Garion, and C. Pagetti. Safety Verification of Neural Network Controlled Systems. In 7th International Workshop on Safety and Security of Intelligent Vehicles (SSIV 2021), 2021.

[8] M. Damour, F. D. Grancey, C. Gabreau, A. Gauffriau, J.-B. Ginestet, A. Hervieu, T. Huraux, C. Pagetti, L. Ponsolle, and A. Clavière. Towards certification of a reduced footprint acas-xu system: A hybrid ml-based solution. In International Conference on Computer Safety, Reliability, and Security, pages 34–48. Springer, 2021.

[9] R. C. Dorf and R. H. Bishop. Modern control systems. Pearson Prentice Hall, 2008.

[10] EUROCAE WG 75.1 /RTCA SC-147. Minimum Operational Performance Standards For Airborne Collision Avoidance System Xu (ACAS Xu), 2020.

[11] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In International Conference on Machine Learning, pages 1587–1596. PMLR, 2018.

[12] M. Gnanasekera, A. V. Savkin, and J. Katupitiya. Range measurements based uav navigation for intercepting ground targets. In 2020 6th International Conference on Control, Automation and Robotics (ICCAR), pages 468–472. IEEE, 2020.

[13] Y. J. Guido Manfredi. An introduction to acas xu and the challenges ahead. In 35th Digital Avionics Systems Conference, Sep 2016, Sacramento, United States. IEEE/AIAA, 2016.

[14] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning, pages 1861–1870. PMLR, 2018.

[15] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters, 2(4):2096–2103, 2017.

[16] W. B. III. Airbus concludes attol project that featured world-first automated takeoffs and landings. Aviation Today, 2019.

[17] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer. Formal verification of acas x, an industrial airborne collision avoidance system. In 2015 International Conference on Embedded Software (EMSOFT), pages 127–136, 2015.

[18] K. T. L. G. Ì. Joakim Bergdahl, Camilo Gordillo. Augmenting automated game testing with deep reinforcement learning. arxiv, 2103.15819v1, 2021.

[19] K. D. Julian, J. Lopezy, J. S. Brushy, M. P. Owenz, and M. J. Kochenderfer. Deep neural network compression for aircraft collision avoidance systems. 35th Digital Avionics Systems Conference (DASC), 2016.

[20] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. CoRR, abs/1702.01135, 2017.

[21] W. Koch, R. Mancuso, R. West, and A. Bestavros. Reinforcement learning for uav attitude control. ACM Transactions on Cyber-Physical Systems, 3(2):1–21, 2019.

[22] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In 4th International Conference for Learning Representations, 2016.

[24] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in Neural Information Processing Systems, 30:6379–6390, 2017.

[25] G. Manfredi and Y. Jestin. An introduction to acas xu and the challenges ahead. In 35th Digital Avionics Systems Conference (DASC'16), pages 1–9, 2016.

[26] M. Min, L. Xiao, D. Xu, L. Huang, and M. Peng. Learning-based defense against malicious unmanned aerial vehicles. In 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), pages 1–5. IEEE, 2018.

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015.

[30] F. Netjasov, A. Vidosavljevic, V. Tosic, M. H. Everdij, and H. A. Blom. Development, validation and application of stochastically and dynamically coloured petri net model of acas operations for safety assessment purposes. Transportation Research part C: emerging technologies, 33:167–195, 2013.

[31] U. D. of transportation Federal Aviation Administration. Introduction to tcas ii version 7. https://www.faa.gov/documentlibrary/media/advisory_circular/tcas%20ii%20v7.1%20intro%20booklet.pdf, 2000.

[32] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen. Autonomous uav navigation using reinforcement learning. arXiv preprint arXiv:1801.05086, 2018.

[33] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[34] R. santamarta. Arm ida and cross check: Reversing the 787's core network. https://act-on.ioactive.com/acton/attachment/34793/f-cd239504-44e6-42ab-85ce-91087de817d9/1/-/-/-/-/Arm-IDA%20and%20Cross%20Check%3A%20Reversing%20the%20787%27s%20Core%20Network.pdf, 2019.

[35] H. Sathaye, D. Schepers, A. Ranganathan, and G. Noubir. Wireless attacks on aircraft instrument landing systems. In 28th {USENIX} Security Symposium ({USENIX} Security 19), pages 357–372, 2019.

[36] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. Nature, 588(7839):604–609, 2020.

[37] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In International conference on machine learning, pages 1889–1897. PMLR, 2015.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[39] P. A. Series. Aircraft hacking. 2013.

[40] R. L. Shaw. Fighter combat. Tactics and Maneuvering; Naval Institute Press: Annapolis, MD, USA, 1985.

[41] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. nature, 529(7587):484–489, 2016.

[42] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science, 362(6419):1140–1144, 2018.

[43] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In International conference on machine learning, pages 387–395. PMLR, 2014.

[44] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In NIPs, volume 99, pages 1057–1063. Citeseer, 1999.

[45] G. Tong, N. Jiang, L. Biyue, Z. Xi, W. Ya, and D. Wenbo. Uav navigation in high dynamic environments: A deep reinforcement learning approach. Chinese Journal of Aeronautics, 34(2):479–489, 2021.

[46] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[47] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, 575(7782):350–354, 2019.

[48] B. Vlahov, E. Squires, L. Strickland, and C. Pippin. On developing a uav pursuit-evasion policy using reinforcement learning. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pages 859–864. IEEE, 2018.

[49] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin. Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning. In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3712–3717. IEEE, 2005.

[50] C. J. Watkins and P. Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.

[51] S. Xuan and L. Ke. Uav swarm attack-defense confrontation based on multi-agent reinforcement learning. In Advances in Guidance, Navigation and Control, pages 5599–5608. Springer, 2022.

[52] K. D. Young, V. I. Utkin, and U. Ozguner. A control engineer's guide to sliding mode control. IEEE transactions on control systems technology, 7(3):328–342, 1999.