Autonomous drone interception with Deep Reinforcement Learning

David Bertoin†[1,2,3], Adrien Gauffriau†[1,4], Damien Grasset†[5] and Jayant Sen Gupta[1,6]

[1]*IRT Saint-Exupery, Toulouse, France*
[2]*ISAE-SUPAERO, Toulouse, France*
[3]*Institut de Mathematiques de Toulouse, France*
[4]*Airbus Operations, Toulouse, France*
[5]*IRT Saint-Exupery Montreal, Canada*
[6]*Airbus AI Research, Toulouse, France*

Abstract
Driven by recent successes in artificial intelligence, new autonomous navigation systems are emerging in the urban space. The adoption of such systems raises questions about certification criteria and their vulnerability to external threats. This work focuses on the automated anti-collision systems designed for autonomous drones evolving in an urban context, less controlled than the conventional airspace and more vulnerable to potential intruders. In particular, we highlight the vulnerabilities of such systems to hijacking, taking as example the scenario of an autonomous delivery drone diverted from its mission by a malicious agent. We demonstrate the possibility of training Reinforcement Learning agents to deflect a drone equipped with an automated anti-collision system. Our contribution is threefold. Firstly, we illustrate the security vulnerabilities of these systems. Secondly, we demonstrate the effectiveness of Reinforcement Learning for automatic detection of security flaws. Thirdly, we provide the community with an original benchmark based on an industrial use case.

Keywords
Automated anti-collision system, Autonomous Drone, Reinforcement Learning, Urban Mobility, UAV, Security

1. Introduction

Spurred on by the recent advances in Machine Learning, autonomous vehicles are destined to become an integral part of tomorrow's mobility. So far limited to land conveyance, a new category of autonomous vehicles (e.g., air cabs, delivery drones) is about to take place in the urban mobility space. In a similar way to self-driving cars, unmanned aerial vehicles (UAVs) cause new public acceptance and certification challenges. Guaranteeing the non-collision of these UAVs with the obstacles present in the urban landscape and other flying objects is part of these challenges. If flight plans and airways are used to prevent collisions in the traditional airspace, they most often require human supervision. The potential amount of UAVs circulating simultaneously in the urban air mobility contexts makes their utilization unfeasible. Thus,

ATT'22: Workshop Agents in Traffic and Transportation, July 25, 2022, Vienna, Austria
david.bertoin@irt-saintexupery.com (D. Bertoin†); adrien.gauffriau@airbus.com (A. Gauffriau†); damien.grasset@irt-saintexupery.com (D. Grasset†); jayant.sengupta@airbus.com (J. S. Gupta)

many UAVs are currently equipped with automatic collision avoidance systems. These systems are subject to drastic controls ([1], [2] [3], [4]) to obtain the certifications allowing them to be integrated into UAVs. Although these certifications guarantee these systems' proper functioning and efficiency, the deterministic aspect of their decisions makes their behavior predictable. We argue that the predictability of their behaviors exposes ownships equipped with such systems to malicious attacks. In this paper, we take advantage of Reinforcement Learning (RL) agents' exploration and exploitation capabilities to demonstrate the possibility of diverting an autonomous drone to an interception zone by hacking its obstacle avoidance system. We implement a simulator based on the scenario of an autonomous delivery drone trying to reach a delivery area and being hijacked by another drone. We use RL to train an autonomous attacker to position itself to deceive the anti-collision system and therefore, modify his target's trajectory towards the desired zone. Our work thus highlights a critical security flaw of UAVs and opens the way to the use of RL in the certification procedures of such systems.

This paper is organized as follows. Section 2 presents the related works. Section 3 introduces background knowledge on the ACAS-X anti-collision system family and Reinforcement Learning. Section 4 presents the general interception scenario and the simulator design. Section 5 shows empirical results and discusses the critical components of successful hijackings. Finally, Section 6 summarizes and concludes this paper on future perspectives.

2. Related works

Security in avionics. Ongoing research into security vulnerabilities plays a fundamental role in the fight against hijacking. In [5], the authors exploit vulnerabilities of the Aircraft Communication Addressing and Reporting System (ACARS) network to upload new flight plans in the aircraft's Flight Management System (FMS). Nevertheless, since the pilots still carry out the control of the aircraft, the attack mainly leads to an increase in their workload. [6] directly targets the ground infrastructure and designs two different attacks, using radio signals, on the Instrument Landing System (ILS). The authors develop two different attacks on the ILS using radio signals that prevent the aircraft from landing successfully. They demonstrate a consistent success rate with offset touchdowns from 18 to over 50 meters laterally and longitudinally. [7] highlights the vulnerabilities of the Avionics Wireless Networks (AWN). [8] describes a detailed, plausible attack reaching the avionics network effectively on a commercial airplane. Due to the complexity and cost of deploying such attacks, most of these results remain theoretical. To date, none of these attacks enable complete control of the aircraft. Their impact is often mitigated by the presence of humans in the loop. In the future, the introduction of low complex autonomous systems, combined with the popularization of drones and their accessibility, may change this paradigm. Our work focuses on this new generation of UAVs, which is about to occupy the urban airspace, and highlights their vulnerabilities to malicious attacks.

Automatic anti-collision systems Although supplemented by pilots' interventions, automated collision detection and avoidance plays a crucial role in the safety of flying vehicles. These systems are mainly used for airliners to provide maneuvering recommendations to the pilot and thus lighten his general workload. The most commonly used is the Traffic Collision

Avoidance System (TCAS). Unlike newer systems, the TCAS requires both aircrafts to be equipped to be effective. Based on [9], the Airborne Collision Avoidance System X (ACAS X) [10] has been designed for autonomous vehicles. The ACAS systems rely upon probabilistic models to represent the various sources of uncertainty and upon computer-based optimization to provide the best possible collision avoidance recommendations. Its robustness has been demonstrated notably in [11] using Petri model and [12] using formal methods. Within the ACAS X family, the ACAS-Xu is specifically designed for unmanned aircraft systems (UAS). Due to the UAS reduced capabilities, this variant solves conflicts using horizontal motions. However, the cost table size ($\sim$ 5GB) makes it challenging to be embedded within small UAVs. [13], [14], and [15] study the possibility to replace these look-up tables with surrogate models to reduce the system's footprint. Nonetheless, the use of surrogates raises the problem of their certification. [16],[17], and [18] employ formal methods to provide guarantees on these models. Our contribution highlights security flaws of the ACAS-Xu but could as well be extended to other ACAS-X versions.

RL for Unmanned Aerial Vehicle. Recent successes in Reinforcement Learning have demonstrated the ability of autonomous agents to outperform humans in many tasks [19, 20, 21, 22, 23] and several works have applied it for the control and navigation of UAVs. RL provides a promising alternative to classical stability and control methods such as Proportional-Integral-Derivative (PID) control systems [24]. 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. [25] compares the efficiency of a model based Reinforcement Learning controller with Integral Sliding Mode (ISM) control [26]. The authors of [27] train a 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 initializations, both in simulation and with a real quadrotor. [28] trains autonomous controllers flight control systems with state-of-the-art model-free deep Reinforcement Learning algorithms (Deep Deterministic Policy Gradient [29], Trust Region Policy Optimization [30], Proximal Policy Optimization [31]) and compares their performance with PID controllers. In [32], 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 RL agent directly on real drones with a limited number of steps.

Mission planning is another classical use case of RL for UAVs. In [33], the authors combine a Q-learning [34] algorithm focusing on navigation policy with PID controllers. In [35], the navigation problem is decomposed into two simpler sub-tasks (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, [36] demonstrates the effectiveness of an A3C [37] RL agent versus an opponent with Greedy Shooter policy [38]. In a multi-agent context, [39] uses a Multi-Agent Deep Deterministic Policy Gradient algorithm (MADDPG) [40] in an attack-defense confrontation Markov game. [41] proposes a ground defense system trained with Q-learning to choose between high-level defense strategies (GPS spoofing, jamming, hacking, and laser shooting). While [42] and [43] use RL to train a drone attacker to intercept a target drone, [44] places the agent in the defender's position and trains it with a Soft Actor-Critic algorithm [45] to avoid capture. [46] propose an adaptive stress testing based on Monte Carlo Tree Search [47] to find the most likely state trajectories leading to near mid-air collisions.

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.

3. Background

3.1. Overview of the ACAS system

The Airborne Collision Avoidance System X (ACAS X) is a collection of rules providing an optimized decision logic for airborne collision avoidance. Among the family of ACAS X, the ACAS-Xu is specifically designed for drones and urban air mobility. By requesting a set of lookup tables (LUT) computed offline, the ACAS-Xu provides, in real-time, a horizontal resolution of conflicts. Using data collected from its surrounding environment, the ownship queries the LUT on the collisions probabilities for five different directional recommendations: COC (Clear of Conflict for the current heading), WL (Weak Left), WR (Weak Right), L (Left), R (Right). Therefore, the ACAS-Xu is used to avoid any static object (tower, crane, antenna) or any moving object (e.g., birds, UAV) whose behavior could be unpredictable or even using the same avoidance system. No particular maneuver is required when the ownship is in the COC state. Otherwise, the autonomous agent may decide to follow the advisory that minimizes the probability of conflicts according to the geometric parameters. We describe, in Table 1, these parameters along with their unit of measure and illustrate in Figure 1 the example of an intruder crossing the ownship's trajectory.

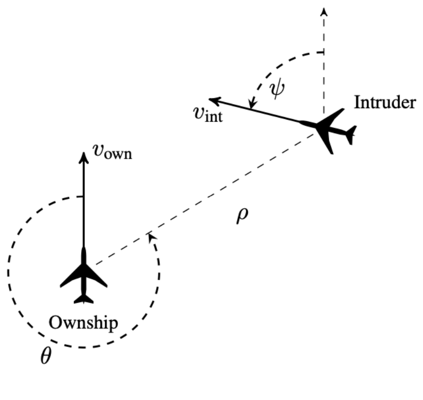


Figure 1: ACAS-Xu geometry [16]

Table 1
Geometric parameters

Param. (units)	**Description**
ρ (ft)	Distance from ownship to intruder
θ (rad)	Angle between ownship and intruder relative to the ownship's heading
ψ (rad)	Heading angle of intruder relative to the ownship's heading
v_{own} (ft/s)	Speed of the ownship
v_{int} (ft/s)	Speed of the intruder

A complete description of the ACAS-Xu system can be found in [48].

3.2. Reinforcement Learning

Within the field of Machine Learning, Reinforcement Learning is particularly suited for 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 $\mathcal{S}$ is the space of all possible states. Considering s_t, the agent takes an *action* $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the space of all possible actions (including inaction). 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* $r_t \in \mathbb{R}$.

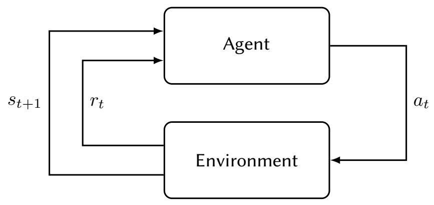


Figure 2: Agent-environment interaction loop

Figure 2 taken from [49] illustrates the agent-environment interactions. 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 $R_t = \sum_{k=0}^{K} \gamma^{(k)} r_{t+k}$. Deep Reinforcement Learning's goal consists in 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 $\tau = (s_0, a_0, ..., s_{K+1})$ 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 parameterized policy π_ϕ through gradient ascent on $J(\pi_\phi)$. These methods rely on the policy gradient theorem [50]:

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

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 parameterized estimator called *critic* to estimate $Q^\pi(s, a)$ (π_ϕ thus represents the *actor* part of the agent). By relying on this principle, [51] 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 Deep Deterministic Policy Gradient (DDPG) algorithm [29] adapts the ideas underlying the success of Deep Q-Learning [52] [19] 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 [53], the Twin

Delayed DDPG (TD3) [54] addresses this overestimation by taking the minimum estimation between a pair of critics and adding noise to the actions used to form the Q-learning target. Combined with a less frequent policy update (one update every d critic updates), these tricks result in substantially improved performance over DDPG in a number of challenging tasks in the continuous control setting. Algorithm 1 describes TD3's training procedure.

Algorithm 1: TD3

```
Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network $\pi_\phi$ with random parameters
  $\theta_1, \theta_2, \phi$
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 $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
    Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} \frac{1}{N} \sum (y - Q_{\theta_i}(s, a))^2$
    if $t$ mod $d$ then
        Update $\phi$ 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
```

4. Drone interception scenario

We consider an environment composed of two agents (T, A) and two areas of interest (D_1, D_2) inside a delimited playground. The first agent T, which we refer to as *the target*, is a delivery drone that has the mission to reach the delivery area D_1. This *target* drone is equipped with the ACAS-Xu system, introduced in section 3.1. The target starts in position I_t, has a fixed velocity v_t, and uses the autonomous avoidance system to modify its trajectory when a possible conflict is detected. The second agent A, referred to as *the attacker*, aims at hijacking the target towards an alternate delivery area, called the *interception* area D_2. To do so, the *attacker* exploits the potential flaws induced by the target's use of the avoidance ACAS-Xu system to deflect its trajectory toward the interception area D_2. The attacker starts in position I_a and can continuously adjust its velocity v_a within the range $[0, v_{max}]$.

Both agents interact in a 2-dimensional playground. Gravity is not considered, and the scene does not contain any obstacles. Figure 3 provides a schematic representation of the set-up.

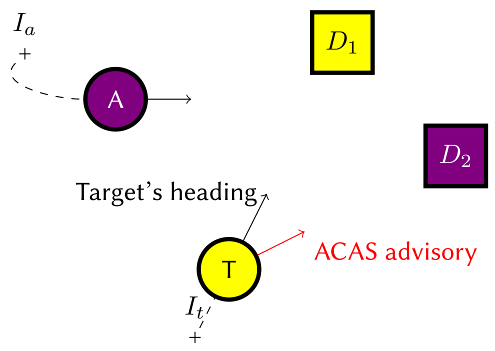


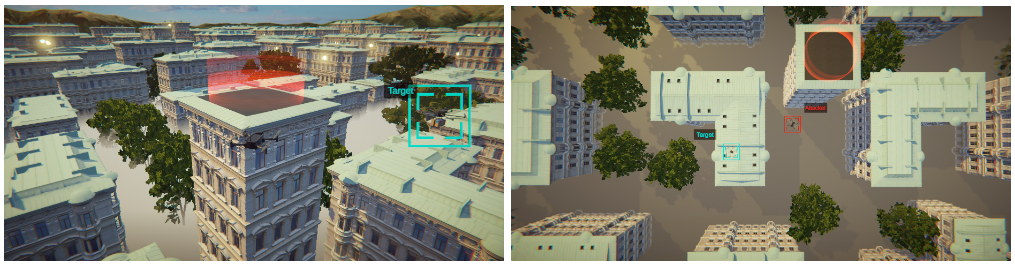
Figure 3: The interception scenario: The target heads toward its delivery zone while the attacker seeks to divert it to the interception zone by taking advantage of the recommendations given by the ACAS-Xu.

One can note that the Xu version of the ACAS system is dedicated to drones and only provides horizontal avoidance recommendations. Among the other versions of the ACAS, the Xa version is dedicated to large aircraft and provides vertical avoidance. For the sake of simplicity, we restrict the interception to a horizontal plan and the use of the ACAS-Xu avoidance system solely. Adding a vertical dimension would not necessarily complexify the learning task (given some adjustments to the reward model). We let the study of a 3D interception with a target equipped with both the Xu and the Xa version to future work.

The simulator environment

We implemented this scenario in an original open-source simulator[1]. In the following, we describe its main characteristics.
State Space : The state of the environment at step n is fully described by the state of the two agents (target and attacker) as well as the cartesian positions of the delivery and interception areas. Each agent's state is composed of its cartesian positions $P_n^{agent} = (x, y)$ and its velocity vector in the horizontal plan $\vec{V}_n^{agent}$. We denote by α (resp. V), the heading's angle (resp. the norm) of the velocity vector, and use $\vec{V} = (\alpha, V)$ as its representation. The last recommendation AX_{n-1} of the ACAS-Xu system is also included in the state.
Action Space : The attacker's action vector at step n is composed of two updates $\delta V_n \in [-200, +200]$ (ft/s) and $\delta\alpha_n \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ (rad) representing respectively an update of its velocity and its heading.
Transitions: The target agent's velocity is constant during the whole episode. Its heading is updated according to the ACAS-Xu recommendations. If the advisory provided is different from COC, the following heading update $\delta\alpha_n$ is performed: WL $\rightarrow$ + 0.15 (rad), WR $\rightarrow$ - 0.15 (rad), L $\rightarrow$ + 0.3 (rad) and R $\rightarrow$ - 0.3 (rad). If the advisory is COC, the $\lambda\alpha$ update allows the target to reach the optimal heading pointing to the delivery area with a maximum variation of 0.3 (rad). This variation is constrained to represent actual drone maneuverability and avoid instability

[1]Since the ACAS-Xu lookup tables are exclusively available to organizations that are part of RTCA/EUROCAE (the authority responsible for the standardization of ACAS-Xu), a mode allowing to run the simulator with a surrogate model emulating the tables is available. Link available by the time of the conference.



(a) Attacker's point of view (b) Top view

Figure 4: The simulator environment

due to big turns. Both agents' positions P are updated by $P_{n+1} = P_n + \vec{V}_{n+1}$ with the update of the speed vector given by:

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

Reward model : RL agents are strongly impacted by the reward model used during training. The environment implements the following reward schema:

$$r_n = \begin{cases} r_{n-1} + (d_n^{D_2} - d_{n-1}^{D_2}) & \text{if } d_n^{D_1} \geq d_{min}^{D_1} \quad \& \quad d_n^{D_2} < d_{min}^{D_2} \\ 0 & \text{otherwise} \end{cases}$$

with $d_n^{D_1}$ the distance between the target and the delivery zone, $d_n^{D_2}$ the distance between the target and the interception zone, and $d_{min}^{D_1}$ and $d_{min}^{D_2}$ their respective minimum since the episode's beginning. The agent is rewarded when $d_{min}^{D_2}$ is reduced. To avoid the degenerate case where the attacker makes the target move away and closer continuously, the value of the reward at each step is increased as long as the target gets closer to the interception zone. It is reset to 0 otherwise. This reward model choice encourages the agent to divert the target with the most direct trajectory possible. Figure 4 shows two visualizations of the simulator.

5. Experimental results and discussion

This section exhibits the efficiency of RL agents in hijacking the target delivery drone and discusses the different factors impacting the success of an interception. For the sake of simplicity, we fixed the target's velocity to 400 ft/s. We trained three attacker agents A_{300}, A_{600} and A_{1000} capable of reaching the maximum speed of 300 ft/s, 600 ft/s, and 1000 ft/s (75%, 150% and 250% of the target's speed) respectively. We used stable-baselines3 [55] implementation of TD3 to train the attackers. Each agent was composed of an actor and two critics, and we used a two-layer feedforward neural network of 400 and 300 hidden nodes for both the actor and the critics. 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* [52] and drawn randomly in the form of mini-batches during the weight update phase. We used a replay buffer of size 5.10^4 and a mini-batch size of 512. As suggested in [29], we added an action noise drawn from the Ornstein-Uhlenbeck process to promote exploration. We trained all agents for 6 million steps. Table 2 provides a complete description of the hyper-parameters used during training.

Table 2
Hyper-parameters used in TD3

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 3
Overall performance

Agents	Success Rate
A_{300}	>1%
A_{600}	37%
A_{1000}	94%

Hijacking success rate. We assessed the performance of the three attackers by Monte-Carlo sampling on 10^5 simulations varying all the initial positions in each episode. Results reported in Table 3 show that the interception of the target is possible, but highly dependent on the attacker's speed relative to the target's. The A_{1000} agent obtains a 94% success rate on its attacks. Conversely, the A_{300} agent only very rarely succeeds in performing an interception. The A_{600} agent manages to intercept the target in only 34% of the scenarios, although it goes slightly faster than the target.

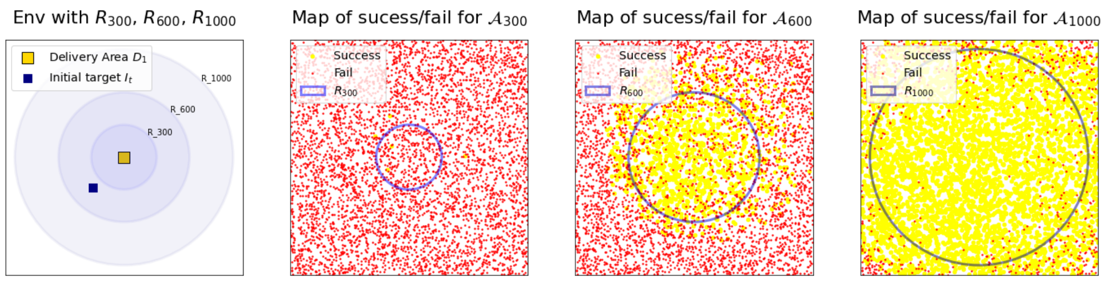


Figure 5: Empirical estimation of the attacker's influence area.

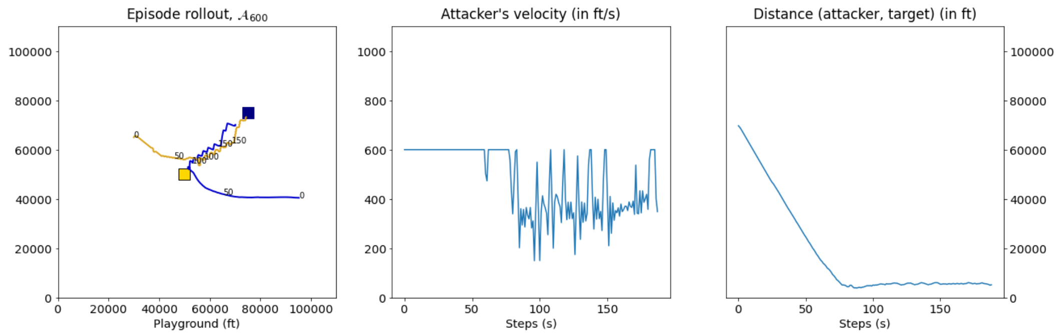


Figure 6: Example of successful hijacking. **Left**: trajectories of the attacker and the target. **Middle**: attacker's velocity during the episode. **Right**: distance between the attacker and the target during the episode.

Impact of the speed. The previous experiment showed the importance of speed on the success rate of the attack. In some situations, a too low velocity does not allow the attacker to position himself in such a way that the ACAS-Xu detects a possible collision and thus have an opportunity to interfere with his target. We conducted a similar Monte Carlo experiment to evaluate the attacker's area of influence according to its speed, fixing the delivery area's position and its distance to the target. We compared the estimated influence area with a simplified theoretical one consisting of a disk $\mathcal{D}$ centered on the delivery area, with a radius $R = \frac{v_a^{max}}{v_t} d^{D_1}$ the distance between the target and the delivery area weighted by the ratio between the maximum speed of the attacker and the speed of the target. Figure 5 shows the estimated areas for the three agents. The agents A_{600} and A_{1000} almost always succeed in hijacking the target in the theoretical influence area. Their estimated area is even slightly larger than the theoretical one. Conversely, agent A_{300} almost never succeeds in intercepting the target, even in theoretically favorable situations. This reveals that the ratio between the speeds of the attacker and his target does not only impact the attacker's surface of influence. When the attacker's maximum speed is lower than the target's, the agent cannot influence the trajectory sufficiently. We demonstrate the need for the attacker to be able to fly faster than his target by looking at an example of a successful hijacking on the agent A_{600}. Figure 6 shows the trajectory of the attacker and its speed variations during a successful attack. For this example, the agent first heads with maximum speed towards the target to penetrate its close vicinity. In this example, the agent first heads towards the target at maximum speed to penetrate its immediate vicinity. In a second time, he approaches his target's speed to be able to divert it with more precision. We can then observe through some peaks that the agent sometimes needs to suddenly increase its speed above 400ft/s to keep control over the target. Figure 7 shows an example of attempt of hijacking when the attackers are in the close vicinity of the target. The fastest agent A_{1000} smoothly divert the target towards the interception area while the A_{600} agent succeeds in his attack in a less straightforward fashion. The A_{300} agent fails in diverting the target. It is easily outpaced by the target which avoids it before resuming its trajectory towards the delivery zone.

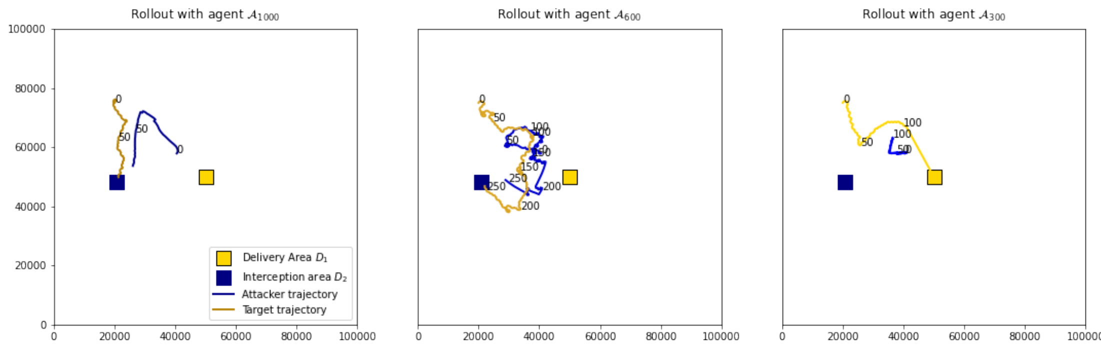


Figure 7: Examples of hijacking attempt trajectories when the attacker is in the close vicinity of the target. **Left**: A_{1000}. **Middle**: A_{600}. **Right**: A_{1000}.

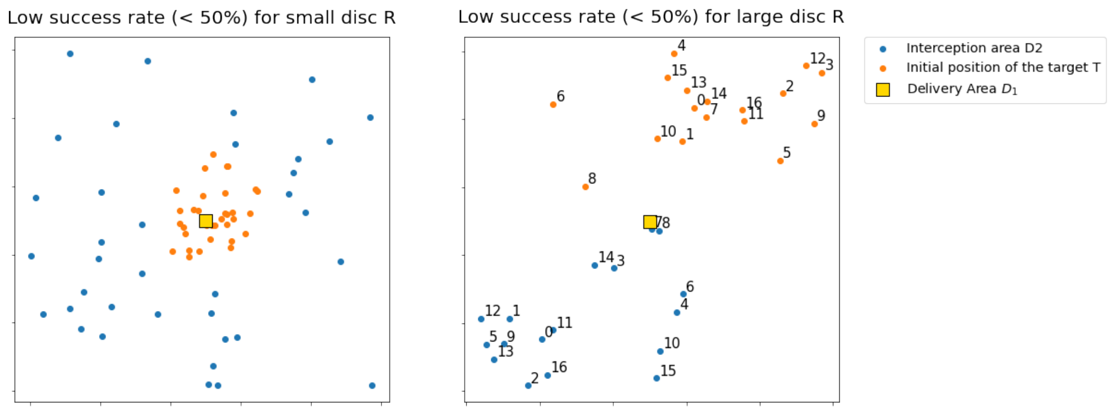


Figure 8: Pairs of scenarios (I_t, D_2) leading to a poor success rate for agent A_{1000}. **Left**: clusters of scenarios where target's initial positions are too close to the delivery area. **Right**: clusters of scenarios where the delivery area is in between I_t and D_2.

Impact of the geometry. With speed much higher than the target, the agent A_{1000} almost systematically succeeds in the interception. We can distinguish two cases where the hijacking does not succeed despite the speed difference. Figure 8 shows examples of these situations. The first group consists of situations where the target is already too close to the delivery area and where the agent cannot interact quickly enough with the target. The second group involves situations where the delivery area is located between the target and the interception zone. It may happen, in these situations, that the policy learned by the attacker does not succeed in deviating the target's trajectory enough to prevent it from reaching the delivery zone. We believe that fine-tuning the reward function would improve the agent's performance in these situations. Since our goal is to demonstrate the feasibility of the attack and not to obtain a better score, we consider the search for the best policy outside the scope of this work.

6. Conclusion

This work highlights a security flaw in automatic collision avoidance systems designed to equip future unmanned aerial vehicles. Through the example of a delivery drone equipped with the ACAS-Xu avoidance system, we demonstrate the possibility of diverting it from its trajectory to an interception zone using an agent trained with Reinforcement Learning. Although we made simplifying assumptions in modeling the problem (a single avoidance system on a horizontal plan), we believe that our method would generalize to other configurations with additional degrees of freedom. Our contribution is not limited to a new hacking method. We believe that it demonstrates the effectiveness of Reinforcement Learning in finding security holes for these autonomous systems, and thus opens the door to future certification processes based on RL adaptive stress testing. Since these subjects are essential for accepting and developing future autonomous vehicles, we make our simulator available to the community. In a future work, we consider training both agents with Reinforcement Learning in a zero-sum two-player game (as in [21, 22]) to produce collision avoidance policies robust to malicious attacks.

Acknowledgement

This project received funding from the French "Investing for the Future – PIA3" program within the Artificial and Natural Intelligence Toulouse Institute (ANITI). The authors gratefully acknowledge the support of the DEEL project[2].

References

[1] Code of Federal Regulations CFR 91.221, General operating and flights rules, Subpart C–Equipment, Instrument, and Certificate Requirements, 2016.

[2] Code of Federal Regulations CFR 135.180, Operating requirements: commuter and on demand operations and rules governing persons on board such aircraft; Subpart c - aircraft and equipment ;135.180 Traffic Alert and Collision Avoidance System, 1994.

[3] TSO-C119B; Traffic Alert and Collision Avoidance System (TCAS) Airborne Equipment, TCAS II., 1998.

[4] RTCA - DO-185A Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System II (TCAS II) Airborne Equipment Volume I, 2013.

[5] H. Teso, Aircraft hacking, 4th annual Hack in the Box (HITB) Security Conference in Amsterdam (2013).

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

[7] R. N. Akram, K. Markantonakis, R. Holloway, S. Kariyawasam, S. Ayub, A. Seeam, R. Atkinson, Challenges of security and trust in avionics wireless networks, in: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), 2015, pp. 4B1–1–4B1–12. doi:10.1109/DASC.2015.7311416.

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

[8] 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.

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

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

[11] F. Netjasov, A. Vidosavljevic, V. Tosic, M. H. Everdij, 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 (2013) 167–195.

[12] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, A. Platzer, Formal verification of acas x, an industrial airborne collision avoidance system, in: 2015 International Conference on Embedded Software (EMSOFT), 2015, pp. 127–136. doi:10.1109/EMSOFT.2015.7318268.

[13] I. Lahsen-Cherif, H. Liu, C. Lamy-Bergot, Real-time drone anti-collision avoidance systems: an edge artificial intelligence application, in: 2022 IEEE Radar Conference (RadarConf22), IEEE, 2022, pp. 1–6.

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

[15] W. Xiang, Z. Shao, Approximate bisimulation relations for neural networks and application to assured neural network compression, arXiv preprint arXiv:2202.01214 (2022).

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

[17] A. Clavière, E. Asselin, C. Garion, C. Pagetti, Safety verification of neural network controlled systems, in: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), IEEE, 2021, pp. 47–54.

[18] M. Damour, F. D. Grancey, C. Gabreau, A. Gauffriau, J.-B. Ginestet, A. Hervieu, T. Huraux, C. Pagetti, L. Ponsolle, 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, Springer, 2021, pp. 34–48.

[19] 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 (2015) 529–533.

[20] 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 (2019) 350–354.

[21] 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 (2016) 484–489.

[22] 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 (2018) 1140–1144.
[23] 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 (2020) 604–609.
[24] R. C. Dorf, R. H. Bishop, Modern control systems, Pearson Prentice Hall, 2008.
[25] S. L. Waslander, G. M. Hoffmann, J. S. Jang, 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, IEEE, 2005, pp. 3712–3717.
[26] K. D. Young, V. I. Utkin, U. Ozguner, A control engineer's guide to sliding mode control, IEEE transactions on control systems technology 7 (1999) 328–342.
[27] J. Hwangbo, I. Sa, R. Siegwart, M. Hutter, Control of a quadrotor with reinforcement learning, IEEE Robotics and Automation Letters 2 (2017) 2096–2103.
[28] W. Koch, R. Mancuso, R. West, A. Bestavros, Reinforcement learning for uav attitude control, ACM Transactions on Cyber-Physical Systems 3 (2019) 1–21.
[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971 (2015).
[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International conference on machine learning, PMLR, 2015, pp. 1889–1897.
[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).
[32] P. Becker-Ehmck, M. Karl, J. Peters, P. van der Smagt, Learning to fly via deep model-based reinforcement learning, arXiv preprint arXiv:2003.08876 (2020).
[33] H. X. Pham, H. M. La, D. Feil-Seifer, L. V. Nguyen, Autonomous uav navigation using reinforcement learning, arXiv preprint arXiv:1801.05086 (2018).
[34] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (1992) 279–292.
[35] G. Tong, N. Jiang, L. Biyue, Z. Xi, W. Ya, D. Wenbo, Uav navigation in high dynamic environments: A deep reinforcement learning approach, Chinese Journal of Aeronautics 34 (2021) 479–489.
[36] B. Vlahov, E. Squires, L. Strickland, C. Pippin, On developing a uav pursuit-evasion policy using reinforcement learning, in: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2018, pp. 859–864.
[37] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International conference on machine learning, PMLR, 2016, pp. 1928–1937.
[38] R. L. Shaw, Fighter combat, Tactics and Maneuvering; Naval Institute Press: Annapolis, MD, USA (1985).
[39] S. Xuan, L. Ke, Uav swarm attack-defense confrontation based on multi-agent reinforcement learning, in: Advances in Guidance, Navigation and Control, Springer, 2022, pp. 5599–5608.
[40] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, Advances in Neural Information Processing Systems 30 (2017) 6379–6390.

[41] M. Min, L. Xiao, D. Xu, L. Huang, M. Peng, Learning-based defense against malicious unmanned aerial vehicles, in: 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), IEEE, 2018, pp. 1–5.
[42] M. Gnanasekera, A. V. Savkin, J. Katupitiya, Range measurements based uav navigation for intercepting ground targets, in: 2020 6th International Conference on Control, Automation and Robotics (ICCAR), IEEE, 2020, pp. 468–472.
[43] E. Çetin, C. Barrado, E. Pastor, Counter a drone in a complex neighborhood area by deep reinforcement learning, Sensors 20 (2020) 2320.
[44] Y. Cheng, Y. Song, Autonomous decision-making generation of uav based on soft actor-critic algorithm, in: 2020 39th Chinese Control Conference (CCC), IEEE, 2020, pp. 7350–7355.
[45] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International conference on machine learning, PMLR, 2018, pp. 1861–1870.
[46] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, M. P. Owen, Adaptive stress testing of airborne collision avoidance systems, in: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), IEEE, 2015, pp. 6C2–1.
[47] R. Coulom, Efficient selectivity and backup operators in monte-carlo tree search, in: International conference on computers and games, Springer, 2006, pp. 72–83.
[48] G. Manfredi, Y. Jestin, An introduction to acas xu and the challenges ahead, in: 35th Digital Avionics Systems Conference (DASC'16), 2016, pp. 1–9.
[49] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.
[50] 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, Citeseer, 1999, pp. 1057–1063.
[51] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: International conference on machine learning, PMLR, 2014, pp. 387–395.
[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013).
[53] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.
[54] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International Conference on Machine Learning, PMLR, 2018, pp. 1587–1596.
[55] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann, Stable baselines3, https://github.com/DLR-RM/stable-baselines3, 2019.