# Using Deep RL to improve the ACAS-Xu policy - concept paper

Filipo Studzinski Perotto

# Using Deep RL to improve the ACAS-Xu policy – concept paper

Filipo Studzinski Perotto
*ONERA – The French Aerospace Lab – Toulouse, France*
*filipo.perotto@onera.fr*

Keywords: Collision Avoidance, Unmanned Aerial Vehicles, Deep Reinforcement Learning, Aeronautics

Abstract: ACAS-Xu is the future collision avoidance system for UAVs, responsible for generating evasive maneuvers in case of imminent risk of collision with another aircraft. In this concept paper, we summarize drawbacks of its current specification, particularly: (a) the fact that the resolution policy, calculated through dynamic programming over a stochastic process, is stored as a huge Q-table; (b) the need of forcing a correspondence between the continuous observations and the artificially discretized space of states; and (c) the consideration of a single aircraft crossing the drone's trajectory, whereas the anti-collision system must potentially deal with multiple intruders. We suggest the use of recent deep reinforcement learning techniques to simultaneously approximate a compact representation of the problem in their continuous dimensions, as well as a near-optimal policy directly from simulation, in addition extending the set of observations and actions.

## 1 Introduction

ACAS-Xu is the new generation of *Airborne Collision Avoidance System* for remotely piloted fixed-wing aerial vehicles (UAVs – Figure 1). It must generate and execute evasive maneuvers in the case of imminent risk of mid-air collision. The resolution policy is calculated beforehand through *Dynamic Programming* over a stochastic *Markovian Decision Process* (MDP) representing an artificially discretized space of states and actions. The resulting policy is stored into immense tables of expected state-action long-term costs. Considering horizontal and vertical resolution maneuvers (which are defined separately), these tables contain billions of parameters ($q$-values). This high dimensionality constitutes an important practical issue for deploying the solution as an industrial embedded system. In the literature, one strategy to compress those tables is training a neural network to approximate the Q-function.

In this concept paper, we suggest another approach: the use of recent *Deep Reinforcement Learning* (Deep RL) techniques, which can natively deal with continuous dimensions and learn a policy of actions directly from simulation. Those techniques optimize the policy of actions at the same time than learning a compact and consistent representation of the underlying process using *Deep Neural Networks* (DNN).

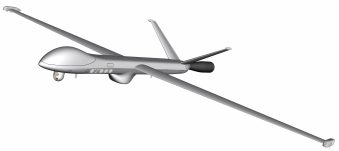

Figure 1: A fixed-wing unmanned aerial vehicle (UAV) is a large drone remotely operated.

Instead of trying to compress the standard table, we would like to come back to the base problem and learn an improved solution, both by considering an extended set of observations and actions that can be included in the system to enhance flight security, as well as by taking into account encounters involving multiple aircraft. The original ACAS-Xu policy considers only a single intruder crossing the drone's trajectory, and multi-threats are solved using ad-hoc strategies. Furthermore, vertical and horizontal resolutions are modeled separately, than combined in operation time. Using Deep RL techniques, both action dimensions can be assessed together.

Following the paper, Section 2 overviews Reinforcement Learning and Deep RL concepts, Section 3 overviews Collision Avoidance Systems, Section 4 presents our proposition and some discussion, and Section 5 concludes the paper.

This work is supported by a French government grant managed by the National Research Agency under the France 2030 program with the reference ANR-23-DEGR-0001 – DeepGreen Project.

## 2 Reinforcement Learning

RL agents must learn from observation, exploring the environment and exploiting their knowledge to improve their performance, eventually converging to a policy of actions that maximizes the expected future rewards, or that minimizes the expected future costs (an optimal policy). Since what can be learned depends on the agent's behavior, a necessary trade-off must be found between *exploration* (trying different actions) and *exploitation* (choosing the actions with best expected return, given the actual knowledge).

Classically, the RL loop is characterized by an interleaved interaction between an agent that perceives a state and executes an action, and an environment that returns a new state and a reward (Sutton and Barto, 2018). The environment can be described as a stochastic *Markovian Decision Process* (MDP) which evolves discretely over time (Bertsekas, 2019; Puterman and Patrick, 2010; Wiering and Otterlo, 2012).

### 2.1 Tabular RL Methods

When the "model" is known (i.e. when $R$ and $P$ are given), an MDP can be exactly solved using *linear* or *dynamic programming* (DP) techniques (Howard, 1960), which generalize over graph search strategies and classical planning, allowing to find optimal sequential decisions for discrete rewarded stochastic processes. This is the procedure used in the ACAS-X definition, solved using the *Value Iteration* method (Kochenderfer and Chryssanthacopoulos, 2011; EUROCAE, 2020).

In contrast, model-free RL algorithms try to learn a policy directly from the experience. Those methods rely on local approximations of the state-action values (Q-values) after each observation based on the Bellman's equation (Bellman, 1952), which defines the value for a policy $\pi$ from state $s$:

$$V_\pi(s) = R(s,\pi(s)) + \gamma \sum_{s'} P(s'|s,\pi(s)) V_\pi(s') \qquad (1)$$

The value of an optimal policy $\pi^*$ is given by:

$$V_{\pi^*}(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{\pi^*}(s') \right\} \qquad (2)$$

A simple trade-off between exploration and exploitation can be done by introducing some randomness into the decision-making process (undirected exploration). In the $\varepsilon$-*greedy* strategy (Sutton and Barto, 2018), at each time step, the agent either executes a random action with probability $\varepsilon$, or follows the current estimated optimal policy, otherwise, with probability $1-\varepsilon$. To avoid a linear expected regret due to a constant exploration, $\varepsilon$ can be dynamically and gradually decreased (Auer et al., 2002).

A smarter approach to solve the exploration-exploitation dilemma is the *optimism in the face of uncertainty*. The insight is to be curious about potentially good but insufficiently frequented state-action pairs (Meuleau and Bourgine, 1999). In tabular methods, it can be done by simply initializing the Q-values optimistically, and then following a completely greedy strategy. Actions that lead to less-observed situations increase their chances of being chosen. The more a state-action pair is tried, the closer its estimated utility approximates its true value. More rigorous strategies lean on the *upper-confidence bound* principle to solve the exploration-exploitation dilemma (Auer and Ortner, 2006; Poupart et al., 2006).

*Q-learning* (Watkins and Dayan, 1992) is a traditional model-free algorithm that implements an off-policy immediate temporal-difference (TD) strategy for learning a policy on environments with delayed rewards. The estimated Q function is stored in memory as a table in the form $S \times A \rightarrow \mathcal{R}$, where the entries represent the estimated utility of each state-action pair $\{s,a\}$. The $\varepsilon$-greedy decision making technique is used to induce some random exploration. The stored Q-table is updated after each transition through a simple *value-iteration* step, based on the Bellman equation, using, for ensuring learning stability, the $\alpha$-weighted average of the previous and the new values:

$$Q'_{s,a} \leftarrow Q_{s,a} + \alpha\big(r + \gamma \max_{a'}[Q_{s',a'}] - Q_{s,a}\big) \qquad (3)$$

Unfortunately, the need of storing a Q-table limits the practical application of tabular RL methods to complex real-world problems. The combinatorial nature of an exhaustive enumeration of states can make impossible to maintain a Q-table in memory, unless of strongly limiting the number of observations and imposing a rough discretization of continuous dimensions. Furthermore, because no correlation exists between the state-action pairs, the learning experiences cannot be generalized to similar states.

### 2.2 Deep Reinforcement Learning

Following remarkable results on learning to play complex games (Mnih et al., 2013; Silver et al., 2016), *deep* RL algorithms have been showing impressive capabilities in learning robust solutions for diverse applications, such as controlling autonomous cars, drones, and airplanes (Liu et al., 2023; Plasencia-Salgueiro, 2023; Razzaghi et al., 2022; Kiran et al., 2022; Pankiewicz et al., 2021), robotics (Singh et al., 2022; Polydoros and Nalpantidis, 2017), multi-robot coordination (Orr and Dutta, 2023), etc.

The rise of *Deep Neural Networks* (DNN) provided new possibilities for function approximation and representation learning (LeCun et al., 2015). Deep RL combines deep learning algorithms with classic RL methods, allowing to solve scalability and complexity issues faced by tabular methods, and enabling decision making in high-dimensional state and action spaces (François-Lavet et al., 2018). Deep RL relies on approximating the state-action values $Q^*$, or even directly approximating the optimal policy $\pi^*$, using deep neural networks. The advantage is that, when available, gradients provide a strong learning signal that can be estimated through sampling.

Deep Q-Networks (DQN) (Mnih et al., 2013; Mnih et al., 2015) is the deep version of Q-Learning. DQN uses two DNNs in the learning process: the *policy neural network*, represented by the weight vector $\theta$, models the Q-function for the current state $s$ and action $a$, and is updated frequently; the *target neural network*, represented by $\theta'$, models the Q-function for the upcoming state $s'$ and action $a'$. During a specific number of iterations, the policy network is updated while the target network remains frozen. Then, the weights of the primary network are copied or merged (using a soft update) into the target network, effectively transferring the acquired knowledge. The use of two networks helps to mitigate instability and divergence issues by providing a stable target for the Q-value updates. Training the policy and target networks corresponds to minimize the following loss function:

$$L(\theta) = \mathbb{E}\left[(r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2\right] \quad (4)$$

A set of observed transitions $\{s_t,\ a_t, s_{t+1}, r_{t+1}\}$ is stored in memory to create a replay buffer, used to sample and train the neural network in an offline manner. By breaking temporal correlations, this *experience replay* technique reduces the learning updates variance, ensuring a better stability of the learned policy and improving the overall performance of the agent. An improvement over DQN is the Double DQN algorithm, which addresses the issue of overestimation bias by using separated networks for action selection and for action evaluation. DQN and DDQN can handle continuous state spaces, but require discrete actions.

In contrast, policy search methods can deal with continuous action spaces, constituting a class of techniques that directly optimize the policy parameters instead of relying on a value function to derive it. Most of policy search methods compute the gradient of the expected reward with respect to the policy parameters and use this gradient to update the policy. It is the case of REINFORCE (Koutník et al., 2013), which relies on Monte Carlo estimation. *Trust Region Policy Optimization* (TRPO) focuses on optimizing policies with a trust region to ensure stable updates, constraining their step size, then ensuring that the new policy does not deviate excessively from the current one. *Proximal Policy Optimization* (PPO) (Schulman et al., 2017) proposes a more efficient stability method by using a clipped objective function to maintain updates within a trust region.

Actor-Critic methods mix policy search with value function approximation. One of these methods is the *Deep Deterministic Policy Gradient* (DDPG) (Silver et al., 2014). Parallel computation techniques, including asynchronous gradient updates, are employed to enhancing learning efficiency. Building on DDPG, the *Twin Delayed Deep Deterministic Policy Gradient* (TD3) algorithm further improves stability and reduces overestimation bias by using two critics and policy delay.

The use of an *advantage function* to update the policy can reduce the variance of the gradient estimates and provide a more reliable direction for policy improvement. The standard advantage function is defined as the difference between the action-value function $Q(s,a)$ and the value function $V(s)$, using the average action. *Advantage Actor-Critic* (A2C) combines advantage updates with actor-critic formulation, using multiple agents to update a shared model. *Asynchronous Advantage Actor-Critic* (A3C) updates the policy and value function networks in parallel, using multiple agents to update the model independently, which enhance exploration (Mnih et al., 2016). *Soft Actor-Critic* (SAC) is an off-policy actor-critic algorithm that encourages exploration through entropy regularization.

# 3 Collision Avoidance Systems

After a series of catastrophic mid-air collisions occurred in the 1950s and 1960s, the *American Federal Aviation Administration* (FAA) decided to support the development of collision avoidance systems aiming to introduce an additional safety resource for the cases in which the *Air Traffic Control* (ATC) fails in ensuring minimal aircraft separation during flight.

Early solutions, available in the 1960s and 1970s and relying on radar transponders, were able to identify the surrounding traffic and alert the pilot to their presence. However, the generation of excessive unnecessary alarms compromised the usability of those systems. Furthermore, passive and independent systems (without communication between aircraft) proved to be impractical for resolution advi-

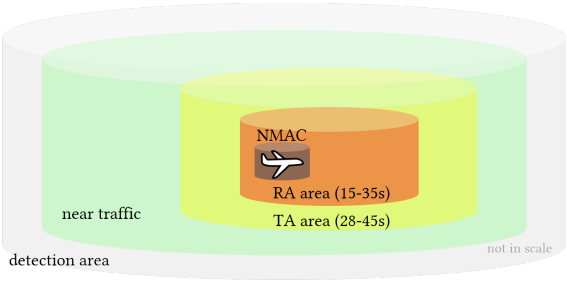


Figure 2: ACAS protected volume.

sory due to the need for coordinated avoidance maneuvers (FAA, 2011).

## 3.1 ACAS

In the 1980s, FAA launched the research and development of the current generation of onboard instruments designed to prevent mid-air collisions, called TCAS (*Traffic alert and Collision Avoidance System*), later included in the *International Civil Aviation Organization* (ICAO) recommendations as ACAS (*Airborne Collision Avoidance System*). For this reason, the term ACAS generally is used to refer to the standard or concept, and TCAS to refer to the equipment actually implementing the corresponding standard (ICAO, 2014; EUROCONTROL, 2022).

ACAS operates autonomously and independently of other aeronautical navigation equipment or ground systems used in air traffic coordination. It replaced the previous BCAS (*Beacon Collision Avoidance System*). Using the antennas of the *Secondary Surveillance Radar* (SSR), already deployed on the aircraft skin, ACAS interrogates the transponders of other nearby aerial vehicles using modes C and S to obtain their altitude (normally sent in the answer) and to estimate their slant range (distance) and relative bearing based, respectively, on the delay and angle of response reception. In mode S, the two aircraft can exchange messages to coordinate non-conflicting maneuvers (Sun, 2021).

Once an aircraft is detected in the surrounding airspace, ACAS tracks its displacement and velocity. For near threats, it interrogates the intruder transponder every second. The ACAS alarm logic is based on minimal vertical and horizontal separation, combined with the time to reach the CPA (*Closest Point of Approach*), given by the slant range divided by the horizontal closure rate, and the time to co-altitude, given by the altitude separation divided by the vertical closure rate. Those boundaries create a protection envelope around the airplane (Figure 2). ACAS is configured with different sensitivity levels, depending on the ownship altitude.

Three categories of ACAS have been defined by ICAO: ACAS I only gives *Traffic Advisories* (TA), ACAS II recommends, in addition, vertical *Resolution Advisories* (RA), and ACAS III should give resolution advisories in both horizontal and vertical directions. In Europe, the TCAS II v7.1 implementation (FAA, 2011) is mandatory since 2015 for all civil turbine-powered aircraft above 5700 kg or with capacity for more than 19 passengers (EU, 2011). In contrast, the study concerning ACAS III was suspended, initially because estimating the horizontal position of the intruder based on the directional radar antenna was considered too inaccurate to define horizontal resolutions without the risk of inducing new conflicts, then definitively abandoned after the rising of ACAS-X concept.

## 3.2 ACAS-X

ACAS-X (Kochenderfer et al., 2008; Kochenderfer and Chryssanthacopoulos, 2011; Kochenderfer et al., 2012; Holland et al., 2013; EUROCONTROL, 2022) is the *New Generation Airborne Collision Avoidance System*, still in development and not yet operational. In this new approach, instead of using a set of scripted rules, a table of expected long-term costs, called look-up table (LUT), is calculated beforehand using *Dynamic Programming* over a *Markovian Decision Process* (MDP), which is a discrete probabilistic dynamic model that represents the near airspace and traffic evolution.

In the underlying ACAS-X model, immediate costs are associated to specific events, implementing the desired operational and safety considerations, making it possible to adapt the obtained resolution strategy to particular procedures or configurations of the airspace. That optimization method is able to determine the preferable sequences of actions for different types of encounters. For each particular context, the expected long-term costs of each possible evasive action is calculated, following different possible scenarios and their probabilities. This method produces a large LUT containing all state-action Q-values.

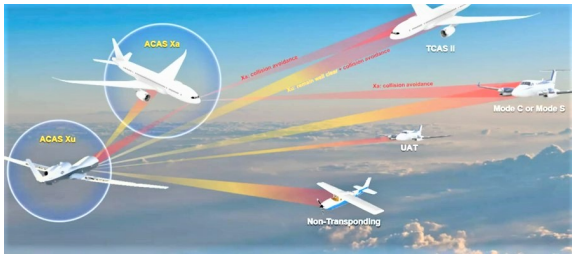


Figure 3: The ACAS-X family.

ACAS-X can access the information communicated periodically by the surrounding traffic through *Automatic Dependent Surveillance-Broadcast* (ADS-B), without the need for interrogation. Those messages typically include the aircraft position, determined by *Global Navigation Satellite Systems* (GNSS) and air data systems, the velocity, derived from the GNSS and the inertial measurement system, and the barometric altitude. ACAS-X constitutes a family of specific collision avoidance systems, which includes ACAS-Xa for civil aviation, ACAS-Xr for helicopters and rotorcrafts, ACAS-Xu for fixed-wing remotely piloted UAVs, ACAS-sXu for small drones, and others (Figure 3).

## 3.3 ACAS-Xu

The ACAS-Xu (*Airborne Collision Avoidance System for Unmanned Aircraft*) is the specific version of the ACAS-X designed for fixed-wing remotely piloted drones (RPASs or UAVs) (Manfredi and Jestin, 2016; Holland and Kochenderfer, 2016; Owen et al., 2019; Cleaveland et al., 2022; EUROCAE, 2020). It can take into account non-cooperative sources, customized logic for drone performance, horizontal and vertical resolutions, coordination adapted to the intruder's equipment, and the automation of resolution advisory implementation.

The immediate configuration of the airspace, relative to an encounter constitutes the *state* of the environment (Figure 4). For the horizontal resolution, that state is defined by the combination of 7 observations:

$$\begin{cases} \rho & \text{the horizontal distance to the intruder (ft),} \\ v_{own} & \text{the own horizontal speed (ft/s),} \\ v_{int} & \text{the horizontal speed of the intruder (ft/s),} \\ \theta & \text{the angle at which the intruder is seen,} \\ \psi & \text{the heading angle difference,} \\ \tau & \text{the time before vertical intrusion (s),} \\ a_{prev} & \text{the previous resolution advisory.} \end{cases}$$

In the standard implementation, that input space was discretized as follows: 39 values for horizontal distance ($\rho$), arranged non-linearly between 499 and 185,318 ft; 12 values for speeds ($v_{own}$ and $v_{int}$) arranged non-linearly, between 0 and 1200 ft/s; 41 values for angles ($\theta$ and $\psi$), arranged linearly between $-\pi$ rad and $+\pi$ rad; 10 values for altitude separation time, arranged non-linearly between 0 and 101 s. It makes $5\times12\times12\times41\times41\times39\times10 = 472,024,800$ states to represent the horizontal problem.

The kernel of the ACAS-Xu strategy is presented in the form of a large tables (LUTs - *look-up tables*) containing the $q$-costs $Q(s,a)$ associated with each

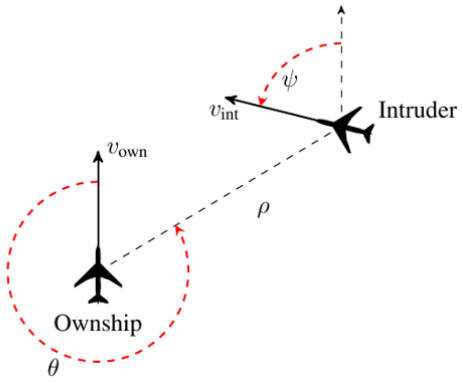


Figure 4: ACAS-Xu is a collision avoidance system for remotely piloted aircraft, developed using an MDP

possible resolution advice $a$ for each discrete state $s$ of the system. Those tables have been calculated offline through optimization of the corresponding MDP, and are used online for the selection of the optimal resolution advice by choosing the action which minimizes the $q$-cost given the probabilities associated with the different possible discrete states of the system at the current time.

In fact, ACAS-Xu must find a correspondence between the current state, given in continuous dimensions for the seven observation variables, and the discrete states represented into the LUT, in order to get the relevant $q$-costs. For the vertical resolution, those values are estimated using multilinear interpolation over the surrounding table points. For the horizontal resolution, it is done by getting the values associated to the nearest neighbor (EUROCAE, 2020).

In the horizontal plan, for each state, five possible actions (evasive maneuvers) can be executed by the drone:

- SL: strong left turn ($-3^\circ/s$),
- WL: weak left turn ($-1.5^\circ/s$),
- COC (*Clear of Conflict*): no action ($0^\circ/s$) ,
- WR: weak right turn ($+1.5^\circ/s$),
- SR: strong right turn ($+3^\circ/s$).

In the vertical plan, the observations are not the same, and correspond to the vertical separation, the rate of loss of vertical separation, and the time to loss of horizontal separation. Other actions are considered, representing different options for climbing or descending. Horizontal and vertical resolutions are chosen separately, then combined afterwards.

Since the generated Q-table (LUT) is significantly large, there is an issue concerning their practical, industrial, deployment into a drone. Several works in the literature use machine learning to compress those

tables using neural networks, reducing the memory footprint of the embedded code and even potentially improving the anti-collision behavior through smoothing, typically (in the horizontal case) using a fully connected neural network with 7 inputs corresponding to the state observations and 5 outputs corresponding to the Q-values for the 5 possible actions, with a compression in an order of $1/1000$ (Damour et al., 2021; Julian et al., 2016; Katz et al., 2017).

# 4 Discussion

Creating a robust collision avoidance system is difficult: the sensors available to the system are imperfect and noisy, resulting in uncertainty in the current positions and velocities of the aircraft involved, variability in pilot behavior and aircraft dynamics makes it difficult to predict the evolution of surrounding traffic trajectories, and the system must balance multiple objectives (Kochenderfer et al., 2012), avoiding to pass dangerously near to other aircraft, but not deviating unnecessarily, always considering a possible imprecision in the positioning data, and an uncertainty in the intruder's behavior. The increase of air traffic density, the inclusion of unmanned aerial vehicles, and the intention of making flight routes less constrained for future airspace operations requires trustful and performing collision avoidance methods. There is also a need to enhance safety but considering the economical acceptability of the proposed solutions by the industry.

The contribution of this paper, beyond the overview concerning reinforcement learning methods (Section 2) and onboard collision avoidance systems (Section 3), is to propose the idea of using Deep RL techniques on the ACAS-Xu problem. To do so, a simulator must be used as the training environment for learning robust policies. At least one implementation of the ACAS-Xu simulator is publicly available[1] and was used for realizing the experiments in (Bak and Tran, 2022). That simulator can be modified to comply with the standard *Gymnasium*[2] environments (Brockman et al., 2016), widely used to implement RL problems, and compatible with several RL libraries such as *SB3*[3] (Raffin et al., 2021).

Technically, a *Near Mid-Air Collision* (NMAC) incident occurs when two aircraft fly closer than 100 feet vertically and 500 feet horizontally. In the standard implementation of the ACAS-Xu problem, the immediate reward function corresponds to:

- Clear of Conflict: $+0.0001$,
- Strengthen: $-0.009$,
- Reversal: $-0.01$,
- NMAC: $-1.0$.

However, a modified reward function could be imagined, considering, for example, the distance to the intruders, the smoothness of the maneuvers, and the deviation of the original trajectory, i.e. positive rewards for maintaining the aircraft in its original trajectory, reducing deviations, and negative rewards that grows up when the drone approximates to a conflict (interception zone with the intruder).

Furthermore, we could consider an integrated action space, where horizontal resolutions (turning left or right) and vertical resolutions (climbing or descending) are taken together, and in their native continuous dimensions. An extended set of actions can also be imagined, including the possibility of changing the ownship velocity (accelerating or decelerating). The set of observations can also be extended to provide more detailed information about other aircraft current actions and intentions, for example, the next scheduled waypoint or maneuver.

Finally, the use of RL techniques to solve the ACAS-Xu problem can allow to handle its complexity, taking into account the drone performance, facing episodes with multiple aircraft, in different configurations, and where the avoidance of a first intruder can occasionally create a new conflict with another one, and including the need to avoid terrain obstacles.

# 5 Conclusion

The idea of using of Deep RL to build a policy for the ACAS-Xu problem is technically sound. The next steps of this research include developing a modified ACAS-Xu simulator implementing the rewards, actions and observations imagined in this paper, using the Gym architecture. Then, using this simulator as environment, experiment training the modified problem with different state-of-the-art Deep RL algorithms, and verify their effective capability to improve flight safety and collision avoidance efficiency compared to TCAS II and to the standard ACAS-Xu implementation.

# REFERENCES

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256.

[1] https://github.com/stanleybak/acasxu_closed_loop_sim

[2] https://gymnasium.farama.org/

[3] https://github.com/DLR-RM/stable-baselines3

Auer, P. and Ortner, R. (2006). Logarithmic online regret bounds for undiscounted reinforcement learning. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 49–56. MIT Press.

Bak, S. and Tran, H.-D. (2022). *Neural Network Compression of ACAS Xu Early Prototype Is Unsafe: Closed-Loop Verification Through Quantized State Backreachability*, page 280–298. Springer International Publishing.

Bellman, R. (1952). On the theory of dynamic programming. *Proc Natl Acad Sci USA*, 38(8):716–719.

Bertsekas, D. (2019). *Reinforcement Learning and Optimal Control*. Athena Scientific.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Cleaveland, R., Mitsch, S., and Platzer, A. (2022). Formally Verified Next-Generation Airborne Collision Avoidance Games in ACAS X. *ACM Trans. Embed. Comput. Syst.*, 22(1).

Damour, M., De Grancey, F., Gabreau, C., Gauffriau, A., Ginestet, J.-B., Hervieu, A., Huraux, T., Pagetti, C., Ponsolle, L., and Clavière, A. (2021). Towards Certification of a Reduced Footprint ACAS-Xu System: a Hybrid ML-based Solution. In *Computer Safety, Reliability, and Security (40th SAFECOMP)*, pages 34–48. Springer.

EU (2011). Commission regulation (eu) no 1332/2011 of 16 december 2011 laying down common airspace usage requirements and operating procedures for airborne collision avoidance. *Official Journal of the European Union*, pages L336/20–22.

EUROCAE (2020). Ed-275: Minimal operational performance for airborne collision avoidance system xu (acas-xu) – volume i. Technical report, EUROCAE.

EUROCONTROL (2022). Airborne Collision Avoidance System (ACAS) guide. EUROCONTROL.

FAA (2011). Introduction to TCAS II version 7.1 (February 2011) – booklet. Technical report, FAA.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354.

Holland and Kochenderfer, M. (2016). Dynamic logic selection for unmanned aircraft separation. In *35th IEEE/AIAA Digital Avionics Systems Conference, DASC*.

Holland, Kochenderfer, M., and Olson (2013). Optimizing the next generation collision avoidance system for safe, suitable, and acceptable operational performance. *Air Traffic Control Quarterly*, 21(3).

Howard, R. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.

ICAO (2014). Annex 10 to the convention on international civil aviation – volume iv: Surveillance and collision avoidance systems. Technical report, International Civil Aviation Organization (ICAO).

Julian, K. D., Lopez, J., Brush, J. S., Owen, M. P., and Kochenderfer, M. J. (2016). Policy compression for aircraft collision avoidance systems. In *35th IEEE/AIAA Digital Avionics Systems Conference, DASC*, pages 1–10. IEEE.

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

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Pérez, P. (2022). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926.

Kochenderfer, M. and Chryssanthacopoulos, J. (2011). Robust airborne collision avoidance through dynamic programming. project report atc-371. Technical report, MIT, Lincoln Lab.

Kochenderfer, M., Espindle, L., Kuchar, J., and Griffith, J. (2008). Correlated encounter model for cooperative aircraft in the national airspace system. project report atc-344. Technical report, MIT, Lincoln Lab.

Kochenderfer, M., Holland, J., and Chryssanthacopoulos, J. (2012). Next-generation airborne collision avoidance system. *Lincoln Lab Journal*, 19(1):17–33.

Koutník, J., Cuccu, G., Schmidhuber, J., and Gomez, F. (2013). Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, page 1061–1068, New York, NY, USA. ACM.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–444.

Liu, H., Kiumarsi, B., Kartal, Y., Koru, A. T., Modares, H., and Lewis, F. L. (2023). Reinforcement learning applications in unmanned vehicle control: A comprehensive overview. *Unmanned Syst.*, 11(1):17–26.

Manfredi and Jestin (2016). An introduction to ACAS-Xu and the challenges ahead. In *35th IEEE/AIAA Digital Avionics Systems Conference, DASC*.

Meuleau, N. and Bourgine, P. (1999). Exploration of multi-state environments: Local measures and backpropagation of uncertainty. *Mach. Learn.*, 35(2):117–154.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. and Weinberger, K. Q., editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48, pages 1928–1937. ICLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M.

(2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Orr, J. and Dutta, A. (2023). Multi-agent deep reinforcement learning for multi-robot applications: A survey. *Sensors*, 23(7):3625.

Owen, M., Panken, A., Moss, R., Alvarez, L., and Leeper, C. (2019). ACAS Xu: Integrated Collision Avoidance and Detect and Avoid Capability for UAS. In *38th IEEE/AIAA Digital Avionics Systems Conference, DASC*.

Pankiewicz, N., Wrona, T., Turlej, W., and Orlowski, M. (2021). Promises and challenges of reinforcement learning applications in motion planning of automated vehicles. In Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., and Zurada, J. M., editors, *Artificial Intelligence and Soft Computing - 20th International Conference, ICAISC 2021, Virtual Event, June 21-23, 2021, Proceedings, Part II*, volume 12855 of *Lecture Notes in Computer Science*, pages 318–329. Springer.

Plasencia-Salgueiro, A. (2023). Deep reinforcement learning for autonomous mobile robot navigation. In Azar, A. T. and Koubaa, A., editors, *Artificial Intelligence for Robotics and Autonomous Systems Applications*, volume 1093 of *Studies in Computational Intelligence*, pages 195–237. Springer.

Polydoros, A. S. and Nalpantidis, L. (2017). Survey of model-based reinforcement learning: Applications on robotics. *J. Intell. Robotic Syst.*, 86(2):153–173.

Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An analytic solution to discrete bayesian reinforcement learning. In *Proc. of the 23rd ICML*, pages 697–704. ACM.

Puterman, M. and Patrick, J. (2010). Dynamic programming. In *Encyclopedia of Machine Learning*, pages 298–308. Springer.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Razzaghi, P., Tabrizian, A., Guo, W., Chen, S., Taye, A., Thompson, E., Bregeon, A., Baheri, A., and Wei, P. (2022). A survey on reinforcement learning in aviation applications. *CoRR*, abs/2211.02147.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Silver, D., Huang, A., Maddison, C., and et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 387–395. JMLR.

Singh, B., Kumar, R., and Singh, V. P. (2022). Reinforcement learning in robotic applications: a comprehensive survey. *Artif. Intell. Rev.*, 55(2):945–990.

Sun, J. (2021). *The 1090 Megahertz Riddle – A Guide to Decoding Mode S and ADSB Signals*. TU Delft OPEN Publishing, 2nd edition.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, 2 edition.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Wiering, M. and Otterlo, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement Learning: State-of-the-Art*, pages 3–42. Springer.