Proceedings of AIAA SCITECH 2022 Forum, 3-7 January 2022, San Diego, CA and Virtual Event
Paper number AIAA 2022-2102. DOI:10.2514/6.2022-2102

Towards Safe Deep Reinforcement Learning for Autonomous Airborne Collision Avoidance Systems

Christos Panoutsakopoulos*
Cranfield University, Cranfield, Buckinghamshire, MK43 0AL, United Kingdom

Burak Yuksek†
Cranfield University, Cranfield, Buckinghamshire, MK43 0AL, United Kingdom

Gokhan Inalhan‡ and Antonios Tsourdos§
Cranfield University, Cranfield, Buckinghamshire, MK43 0AL, United Kingdom

In this paper we consider the application of Safe Deep Reinforcement Learning in the context of a trustworthy autonomous Airborne Collision Avoidance System. A simple 2D airspace model is defined, in which a hypothetical air vehicle attempts to fly to a given waypoint while autonomously avoiding Near Mid-Air collisions (NMACs) with non-cooperative traffic. We use Proximal Policy Optimisation for our learning agent and we propose a reward engineering approach based on a combination of sparse terminal rewards at natural termination points and dense step rewards providing the agent with continuous feedback on its actions, based on relative geometry and motion attributes of its trajectory with respect to the traffic and the target waypoint. The performance of our trained agent is evaluated through Monte-Carlo simulations and it is demonstrated that it achieves to master the collision avoidance task with respect to safety for a reasonable trade-off in mission performance.

I. Introduction

Advanced Air Mobility (AAM) is an umbrella term which defines various future aviation concepts, covering new modes of urban, suburban and rural air transportation of people and goods [1]. AAM includes new aviation applications and markets served by innovative aerial vehicles, such as Unmanned Aerial Vehicles (UAV), electric Vertical Take-off and Landing (eVTOL) and electric Short Take-off and Landing (eSTOL) vehicles. Therefore, it comes with many challenges with respect to the required technological advances, as well as the analysis of the viability, profitability and sustainability of these new markets and business models. For example, in a recent socioeconomic study [2] UKRI (United Kingdom Research and Innovation - non-departmental public body of the UK Government) examines use cases such as power line inspection, cargo delivery and urban air taxis, comparing the proposed new aerospace application against the current state of 'business-as-usual' and analysing cost drivers that would impact the viability of the new business cases. Flight autonomy emerges as a key enabler for all these new aerospace applications, as it would eliminate operational limitations imposed by the presence of a human pilot, allowing operations to be safer and more scalable.

One of the hardest challenges on the way to autonomous flight is autonomous aircraft collision avoidance. Airborne Collision Avoidance Systems (ACAS) have been in use in military and civil aviation for decades. Designed on the basis that a pilot is on board commanding the aircraft, on-board CAS were originally designed to detect potential mid-air collisions and issue alerts and advisories to the pilots for them to take corrective actions. Paper [3] provides some historical context on the long development process of such systems for civil large aircraft, covering Beacon Collision Avoidance Systems (BCAS - 1970s), Traffic Alert and Collision Avoidance Systems (TCAS - 1980s to 2000s) and finally the ACAS X program, which transitioned to more advanced configurations accommodating new sensor modalities, thus enabling collision avoidance protection for new aircraft classes. ACAS systems can be categorised with respect to different aspects of the system design. Review [4] proposes a framework that articulates the basic functions of Conflict Detection and Resolution (CDR) and uses it to categorize various CDR methods based on criteria such as: Dimensions of state information, methods of dynamic state propagation, conflict detection thresholds, conflict resolution methods,

*Ph.D. Student, Centre for Autonomous and Cyber-physical Systems, AIAA Student Member.

†Postdoctoral Research Fellow, Centre for Autonomous and Cyber-physical Systems, AIAA Member.

‡BAE Systems Chair, Professor of Autonomous Systems and Artificial Intelligence, AIAA Associate Fellow.

§Head of Centre for Autonomous and Cyber-physical Systems, AIAA Member.

maneuvering dimensions and methods for management of multiple aircraft conflicts. Survey [5] provides a comparative analysis of ACAS specifically for for UAVs, emphasizing on the basic components of any ACAS system; techniques and hardware used for perception and approaches for collision avoidance.

An autonomous ACAS has to detect potential collisions based on sensory information and take appropriate control actions to avoid them as efficiently as possible, balancing mission safety and performance. This setting fits perfectly with the Reinforcement Learning (RL) framework; in RL, an agent learns to successfully perform a task within an environment. The agent perceives the state of the environment and takes actions based on the state information. The agent's actions affect the way the environment transitions into new states and generate a reward signal that represents how well the agent is doing with respect to the given task. The agent continues this interaction cycle indefinitely or until a natural termination point aiming to maximise the total reward signal it receives. RL algorithms get the agent to iteratively interact with the environment, balancing between exploiting what it already knows to maximise its reward and exploring new actions aiming to discover even better options, until the agent learns to take actions that sufficiently accomplish the given task. In this work, we consider ACAS in the context of RL as follows: An autonomous ACAS is an agent who learns to perform the task of flying the aircraft to its original controlled trajectory waypoint while staying safe from collisions, based on sensory information about the surrounding airspace and through direct control of its trajectory.

In this paper we consider the application of Safe Deep Reinforcement Learning in the context of a trustworthy autonomous Airborne Collision Avoidance System. In section 2 we define a simple 2D airspace model in which a hypothetical air vehicle attempts to fly to a given waypoint while autonomously avoiding Near Mid-Air Collisions (NMAC) with non-cooperative traffic. We start by defining the characteristics of our airspace and aircraft models and providing a summary of some basic relative geometry and kinematics concepts that we use in our analysis. Then, we define mission and safety performance metrics that we will use to evaluate our system. Finally, we describe the encounter model that will be used to generate the aircraft encounters, both during the training phase of our learning agent and for the evaluation of the system and we demonstrate the performance of a baseline fly-straight-to-goal agent, which will be used as a baseline for evaluation. In section 3, we start by providing some background on RL, Deep Reinforcement Learning (DRL) and the Proximal Policy Optimisation (PPO) algorithm, which we use for our learning agent. Then, we formulate our ACAS 2D problem using the standard RL Markov Decision Process (MDP) framework. Last but not least, we introduce our reward engineering approach of incorporating safety requirements into the agent's reward signal, which we later show that helps the agent learn to successfully avoid NMAC autonomously. Finally, in section 4, we demonstrate our training results, and provide a thorough evaluation of our trained agent through Monte Carlo simulations. We demonstrate the performance of our agent in the collision avoidance task, highlighting the balance between mission performance and safety.

II. ACAS 2D Model

In this section we define our simple 2D simulation environment. We define the characteristics of our airspace and aircraft models and list some basic relative geometry and kinematics concepts that we use in our analysis. Then, we define mission and safety performance metrics that we will use to evaluate our system. Finally, we describe the encounter model used to generate the aircraft encounters, for training as well as evaluation of our system and we demonstrate the performance of a baseline fly-straight-to-goal agent in this simulation environment.

A. Simulation Environment

1. 2D Airspace and Hypothetical Air Vehicles

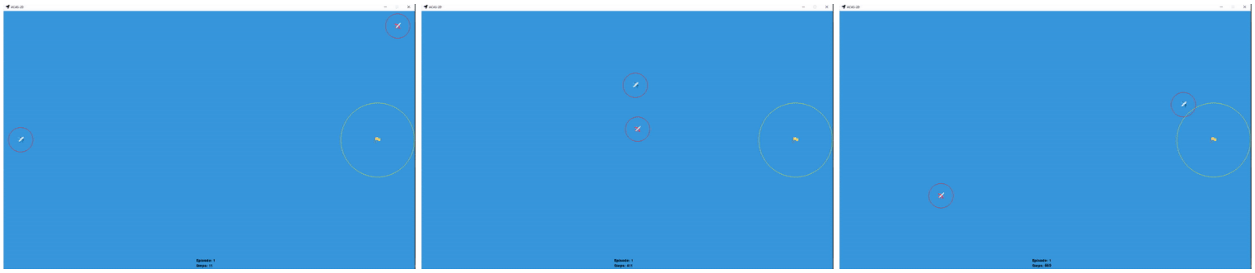
At the beginning of each simulation episode our agent-controlled aircraft (referred to as 'agent aircraft' or just 'agent' hereafter) is located at the Western end of a rectangular airspace section heading at a waypoint placed at the Eastern end. At the same time, a non-cooperative traffic aircraft (referred to as 'traffic aircraft' or just 'traffic' hereafter) with the same performance characteristics is located at the North-East or South-East corner of the airspace section heading towards the general direction of our agent aircraft. The traffic aircraft follows a straight-line trajectory, and the agent aircraft has to control its own trajectory in order to reach the desired waypoint within a defined time limit, while avoiding NMACs (referred to as 'collisions' hereafter) with the traffic aircraft. The aircraft are assumed to be of the same type, a hypothetical type of AAM vehicle, and have the same performance characteristics. They are simulated as point-mass objects moving on the 2D planar aerospace and they have the same kinematic profile. Collisions are defined using a common collision radius around each aircraft. An episode can have one of the following outcomes:

GOAL: The agent reaches the target waypoint when it arrives within goal radius around it, or mathematically, when: $d_{goal} < R_{goal}$

COLLISION: A collision occurs when the aircraft violate each other's collision radius, or mathematically when: $d_{sep} < 2 \cdot R_{collision}$

TIMEOUT: A timeout occurs when the time limit has been reached.

Figure 1 demonstrates a successful simulation episode.



(a) Episode start. **(b) Collision avoidance.** **(c) Arrival to target waypoint.**

Fig. 1 Simulation environment.

2. Assumptions

The following assumptions have been made for our simulation environment allowing us to provide isolated and computationally tractable solutions for 1-1 encounters. This also allows to generate the solution mechanism which can further be scaled up through various learning methods such as transfer learning for more complex scenarios. Let us briefly review the assumptions:

Assumption 1: Both aircraft maintain a constant altitude and airspeed (V_{air}) throughout the episode.
Assumption 2: The airspace section is air-traffic-controlled, and it can be safely assumed that no other aircraft will occur throughout a simulation episode.
Assumption 3: It is assumed that the agent has full awareness of its state and environment through perfect sensors.
Assumption 4: It is assumed that the agent can control its trajectory by commanding its lateral acceleration (a_{lat_c}) within given defined limits through perfect actuators.
Assumption 5: Both aircraft have sufficient fuel/battery life to last the entire episode.

3. Kinematics, Relative Geometry and Relative Motion

The kinematic model that governs the motion of the aircraft in the 2D airspace is defined in the following equations.

$$\dot{x}(t) = V_{air} cos\psi(t) \tag{1}$$

$$\dot{y}(t) = V_{air} sin\psi(t) \tag{2}$$

$$\dot{\psi}(t) = \frac{a_{lat_c}}{V_{air}} \tag{3}$$

We also define the following relative geometry and motion parameters that we use later on in our analysis. Specifically position, velocity and heading of agent aircraft correspond to

$$\vec{p}_a = (x_a, y_a)\,, \quad \vec{v}_a = (\dot{x_a}, \dot{y_a})\,, \quad \psi_a \tag{4}$$

and the position, velocity and heading of traffic aircraft is denoted as

$$\vec{p}_t = (x_t, y_t)\ , \quad \vec{v}_t = (\dot{x_t}, \dot{y_t})\ , \quad \psi_t \tag{5}$$

The position of target waypoint is denoted as

$$\vec{p}_g = (x_g, y_g) \tag{6}$$

and this can be utilized to describe the distance between agent and target waypoint (distance to goal) via

$$d_{goal} = \|\vec{p}_a - \vec{p}_g\| \tag{7}$$

In addition, the distance between agent and traffic (separation) is

$$d_{sep} = \|\vec{p}_a - \vec{p}_t\| \tag{8}$$

and the relative angle between agent and target waypoint (heading to goal) is geometrically defined as

$$\phi_g = \arctan 2(\frac{y_g - y_a}{x_g - x_a}) \tag{9}$$

Here the relative angle between agent and traffic (heading to traffic) is

$$\phi_t = \arctan 2(\frac{y_t - y_a}{x_t - x_a}) \tag{10}$$

The distance from straight-line-to-target plan (plan deviation) is denoted as

$$d_{dev} = d_{goal} \sin \phi_g \tag{11}$$

and agent's velocity relative to traffic is

$$\vec{V}_{at} = (V_{at_x}, V_{at_y}) = (V_{air} cos\psi_a - V_{air} cos\psi_t, V_{air} sin\psi_a - V_{air} sin\psi_t) \tag{12}$$

and agent's heading relative to traffic is

$$\phi_{at} = \arctan \frac{V_{at_y}}{V_{at_x}} \tag{13}$$

Thus, the distance at point of closest approach between agent and traffic:

$$d_{cpa} = d_{sep} \sin(\phi_t - \phi_{at}) \tag{14}$$

and the closing speed between agent and traffic is

$$v_{closing} = \frac{(\vec{v}_a - \vec{v}_t) \cdot (\vec{p}_a - \vec{p}_t)}{d_{sep}} \tag{15}$$

4. Encounter Model

A probabilistic encounter model has been used to generate the traffic trajectories for our simulations. At the start of each episode, the traffic aircraft is randomly initialised with the following approach :

- Position: One of two positions, at the North-Eastern or South-Eastern corner of the airspace segment.
- Heading: Random heading, within a defined range, towards the general direction of the agents straight-line-path to target.

These initial settings are sufficient to define the traffic aircraft's straight-line, constant-speed trajectory.

5. Simulation Parameters

For reproducability of the results, we provide a complete list of the parameters of this simulation setting are given in Table 1.

Table 1: Simulation Parameters

Parameter	**Value or Range**	**Description**
v_{air}	$40\ m/s$	Aircraft airspeed.
a_{lat_c}	$[-4g, 4g]\ m/s^2$	Aircraft lateral acceleration limits.
$\psi_p(0)$	$[0, 3] \cup [357, 360]\ degrees$	Player's initial heading range - towards goal.
$\psi_t(0)$	$[115, 175] or [185, 245]\ degrees$	Traffic's initial heading range - towards agent (depends on initial traffic position).
$T_{episode}$	$10\ s$	Episode time limit.
$R_{collision}$	$9.6\ m$	Aircraft collision radius.
R_{goal}	$28.8\ m$	Goal radius.

B. Performance Metrics

We will be using a number of metrics in order to assess our trained agent's performance. These metrics are sub-categorised in mission performance metrics, which evaluate the efficiency of the agent's trajectory, and safety performance metrics, which evaluate the safety of the agent's trajectory with respect to mid-air collisions.

1. Safety Performance Metrics

Outcome percentages: The percentage of successful arrivals, timeouts and collisions for a given number of simulation episodes.

Collision Risk Ratio (CRR): The probability of NMAC of the trained agent (the mitigated probability) normalized by the probability of NMAC of the fly-to-goal strategy (without collision avoidance logic) (the unmitigated probability). This is a common metric for ACAS evaluation. For example, [6] uses collision risk ration to evaluate different CA system architectures for encounters between small UAS and manned aircraft, and [7] uses the same metric to demonstrate the improvements of ACASx over TCAS.

$$CollisionRiskRatio = \frac{Pr(NMAC)_{mitigated}}{P(NMAC)_{unmitigated}} \tag{16}$$

Separation: The distance between the agent and traffic aircraft.

Time to Safety: The time that the agent is under of collision threat. That is when the aircraft are flying towards the general direction of one another, mathematically: $v_{closing} < 0$

2. Mission Performance Metrics

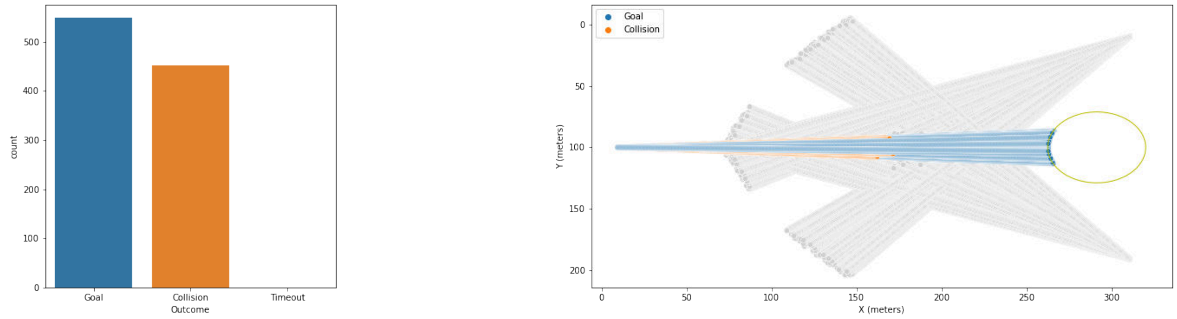
Path Time: The duration of a successful arrival at the target way-point.

Path Length: The distance covered from the starting position to the target way-point.

C. Fly-to-goal Agent Baseline

A simplistic baseline agent was used to comparatively assess the performance of our trained agent, as well as evaluate the fidelity of our encounter model. This baseline agent aircraft always follows a straight line path to the target, regardless of the traffic trajectory. This agent was tested for 1000 simulation episodes as shown in Fig. 2.

Clearly, the performance of the baseline agent is poor with respect to safety, as there are unacceptably too many NMACs. However, successful arrivals occur with straight-line paths which are optimal in terms of time and distance. This observation already demonstrates the trade-off between mission performance and safety that an autonomous ACAS has to balance.



(a) Episode outcomes

(b) Agent-Traffic encounters

Fig. 2 Baseline agent simulation

III. Deep Reinforcement Learning for ACAS 2D

In this section, we start by providing some background on RL, DRL and the Proximal Policy Optimisation (PPO) algorithm, which we use for our learning agent. Then, we formulate our ACAS 2D problem using the standard RL Markov Decision Process (MDP) framework. Finally, we provide a detailed analysis of our reward engineering approach.

A. Background

1. Reinforcement Learning

This sub-section aims to provide a brief summary of key Reinforcement Learning concepts and is based on [8], which is considered one of the best reference resources for RL.

Markov Decision Processes (MDP) are a theoretical tool for framing problems that involve sequential decision making, where actions influence not only current rewards, but also subsequent situations and through those, future rewards. In this context, a decision maker entity, the agent, interacts with its environment continually, selecting actions that lead to new states of the environment. The environment also provides numerical rewards to the agent based on its actions, which the agent seeks to maximise over time through its actions.

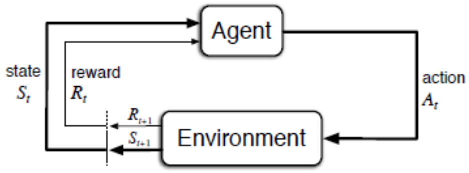


Fig. 3 Agent-environment interaction in a MDP [8]

This agent-environment interaction generates trajectories of the form $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, ...$ These trajectories are governed by the dynamics of the MDP.

$$p(s', r|s, a) \doteq PrS_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a \tag{17}$$

The MDP framework is the basis or the RL agent-environment interaction framework, but it has been used in ACAS research not just in the contect of RL. As mentioned in [3], formulating the collision avoidance problem as an MDP to be solved with Dynamic Programming was the basis of the TCAS logic optimisation introduced by the ACASx program. [9] provides MDP based collision avoidance approaches for unmanned aircraft with different sensor modalities.

In RL the goal of the learning agent is to maximise the expected value of the cumulative sum of received rewards. This expected return is usually expressed as the discounted sum of future returns.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{18}$$

where γ is a discount factor and captures the present value of future rewards.

Most RL algorithms involve estimating the value of states and actions; how good it is for the agent to be in a given state or to take an action at a given state. Since the rewards the agent receives depend on the actions it takes, these values are defined with respect to the agent's action selection policy $\pi(s|a)$; a mapping from states to probabilities of selecting actions at those states.

The value of a state s under policy π (state-value function) is defined as follows

$$u_\pi(s) \doteq E_\pi[G_t|S_t = s] \tag{19}$$

Similarly, the value of taking action a in state s under policy π (action-value function) is defined as follows

$$q_\pi(s,a) \doteq E_\pi[G_t|S_t = s, A_t = a] \tag{20}$$

The goal of the RL agent is, through experience obtained through interaction with the environment, to learn an optimal policy $\pi_*(s)$; a policy that maximises its expected return. RL algorithms determine how the agent updates its policy based on its experience. This is where one of the biggest challenge of RL lies: the trade-off between exploration and exploitation i.e. the agents dilemma between exploiting the information it already has and select the best actions to its current knowledge, or explore different actions that may lead to higher rewards.

2. Deep Reinforcement Learning

This sub-section covers some key Deep Reinforcement Learning (DRL) concepts and is based on [10] and [11], which are very useful resources for theoretical as well as practical aspects of DRL.

Ideally, a RL agent would fully learn the state/action value functions and use them to optimise its policy by always picking the best actions. For simple environments, this might be achievable, but is quickly becomes infeasible as the environment complexity grows. This is were Deep Learning (DL) comes to the rescue. The basic idea behind DRL is that a DL algorithm can learn to abstract away the details of specific situations and provide the agent with a representation it can use to drive its action selection.

There are three main types of RL functions that can be learned (approximated) using DL models: policies, value functions and environment models (MDP dynamics). Correspondingly, there are three main families of DRL methods: policy based, value based and model based. There are also combined methods that learn more than one of these functions simultaneously.

Actor-Critic algorithms is a family of combined DRL methods that learn a policy and a value function; the policy acts and the value function critiques the actions. There is active research going on in the area with many interesting developments in recent years, including Trust Region Policy Optimization (TRPO) [12], Deep Deterministic Policy Gradients (DDPG) [13], Soft Actor-Critic (SAC) [14], and Proximal Policy Optimization (PPO) [15], which is the algorithm we used for our DRL agent.

3. Proximal Policy Optimisation

As mentioned in [16], policy based methods for are fundamental to recent DRL breakthroughs, but they suffer from the inherent challenge of being sensitive to the choice of update step size — too small, and progress is hopelessly slow; too large and the signal is overwhelmed by the noise, or one might see catastrophic drops in performance. They also often have very poor sample efficiency, taking too many times steps to learn relatively simple tasks. Researchers have tried to tackle these problems by constraining or otherwise optimizing the size of a policy update but these methods have their own challenges. Proximal Policy Optimization [15], is a policy based method, which alternates between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent, which strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

4. Safe Reinforcement Learning

Safe Reinforcement Learning (SRL) is a sub-domain of RL where the agent aims to learn policies that maximize the expected return in problems where it is important to ensure reasonable system performance and safety during the learning and/or deployment phase. [17] provides a survey of SRL approaches classifying them in two main methodologies: The first is based on the modification of the optimality criterion (the discounted sum of expected rewards) with a safety factor and the second is based on the modification of the exploration process through the incorporation of external knowledge or a risk metric. Naturally, safety is paramount for any ACAS and an autonomous ACAS based on RL is well suited for the SRL paradigm.

B. ACAS 2D MDP Formulation

We formulated the ACAS 2D agent-environment interaction as an episodic task in the following form:

Time steps: A time step is a single interaction between the agent and the environment and can be thought of as the resolution of our agent's decision making capacity. We have set this resolution to 100 time steps per second, which means that our agent receives an observation and a reward signal from the environment and takes an action every 0.01 seconds.

Observations: The observations of our agent include information about the agent's state (heading, distance and heading to goal and deviation from straight-line-to-goal flight plan), information about the traffic aircraft relative to the agent (distance, closing speed and projected distance at point of closest approach) and the current episode time step t (see [18] for the importance of including a notion of the remaining time in agent's input to avoid violation of the Markov property in time limited tasks). Mathematically, the agent's observation vector is defined as follows:

$$O = [t, \psi_a, d_{dev}, d_{goal}, \phi_g, d_{sep}, d_{cpa}, v_{closing}] \tag{21}$$

This observation vector is designed for training efficiency, as it aims to provide the agent just enough information for it learn to solve the collision avoidance task. Moreover, by providing relative information (e.g. relative distance and heading to goal) rather than absolute information (e.g. positions of agent and goal) this observation vector represents a form of feature engineering; the agent will not have to waste training time in order to learn well known associations between absolute and relative information and will be able to focus on solving the collision avoidance task directly.

Actions: Our agent flies in the 2D airspace at a constant air speed and controls its heading by commanding its lateral acceleration. Therefore, we have a continuous action space (a_{lat_c} can take any value within the defined limits) and the agent's action vector is defined as follows:

$$A = [a_{lat_c}] \tag{22}$$

Rewards: The reward signal that we designed for our agent is a combination of a time discounted dense reward signal provided to the agent at each time step a form of instant feedback on its actions and sparse reward signals provided to the agent when it reaches episode terminal states (goal or collision). The reward signal at each time step t is defined as follows:

$$R = d_t \cdot R_{step} + R_{terminal} \tag{23}$$

where:

d_t, the time discount factor aiming to urge the agent to solve the task as soon as possible, is defined as follows:

$$d_t = 1 - \frac{t}{T_{episode}} \tag{24}$$

$R_{terminal}$, the sparse terminal state reward signal, is refined as follows:

$$R_{terminal} = \begin{cases} 1000 & \text{if episode ends with outcome GOAL at time step } t \\ -1000 & \text{if episode ends with outcome COLLISION at time step } t \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

R_{step}, the dense, informative reward signal, is crucial to our system design and is described in detail in the following sub-section.

C. Reward Engineering

Our ACAS 2D environment provides natural terminal states that can easily be represented by a sparse reward signal, only provided at those terminal states. Although such a sparse reward is important for the training process, it is terribly sample-inefficient. As mentioned in Ref [16], reward sparsity may even make the problem unsolvable, because the agent receives so little feedback that it is unable to discover a sequence of good actions. To address this problem, we designed our reward function as a combination of natural terminal rewards and informative step rewards providing the agent with continuous feedback on the effectiveness of its actions.

Our step reward signal attempts to address the main challenge of the collision avoidance problem: balancing safety and mission performance. First, it divides the collision avoidance task in two distinct phases: escaping the collision threat and correcting the trajectory to arrive at the target waypoint. Then, it provides the agent with a bounded non-negative reward signal at each time step, based on the task phase and certain parameters of its trajectory. Mathematically, the step reward function is defined as follows:

$$R_{step} = \begin{cases} R_{\phi_g} \cdot R_{d_{cpa}} \cdot R_{d_{dev}} & \text{if } v_{closing \leq 0} \\ R_{\phi_g} \cdot R_{d_{goal}} & \text{otherwise} \end{cases} \tag{26}$$

This step reward signal is constructed as the product of separate reward components, each of which represents an aspect of the agent's trajectory that we want to control. The closing speed between the agent and the traffic aircraft is used to tell the two phases of the task apart. While $v_{closing \leq 0}$ the two aircraft are moving towards the general direction of each other, which represents a state of collision threat and the agent attempts to maximise the projected distance of closest approach with the traffic, whilst maintaining a heading as close as possible to the heading-to-goal and deviating as less as possible from the straight-line-to-goal trajectory. On the contrary, when $v_{closing > 0}$ the two aircraft are moving away from each other, therefore the collision threat has been resolved and the agent attempts to correct its heading to direct itself to the target and minimise the distance between itself and the target. This intuition is instilled in the functional form of the reward components, as follows:

Goal heading reward

$$R_{\phi_g} = (1 - \frac{\Delta\phi_{goal}}{180})^4 \tag{27}$$

where $\Delta\phi_{goal} = min(|\psi_a - \phi_g|,\ 360 - |\psi_a - \phi_g|)$ is the agent's heading deviation from heading-to-goal.

The functional form of R_{ϕ_g} is shown in Fig. 4 (a).

Distance of Closest Approach reward

$$R_{d_{cpa}} = min(1, (\frac{d_{cpa}}{D_{safe}})^4) \tag{28}$$

where $D_{safe} = 4 \cdot R_{collision}$ is design parameter of our reward function, selected so that it creates a safety buffer zone for d_{cpa} beyond the collision distance $(2 \cdot R_{collision})$.

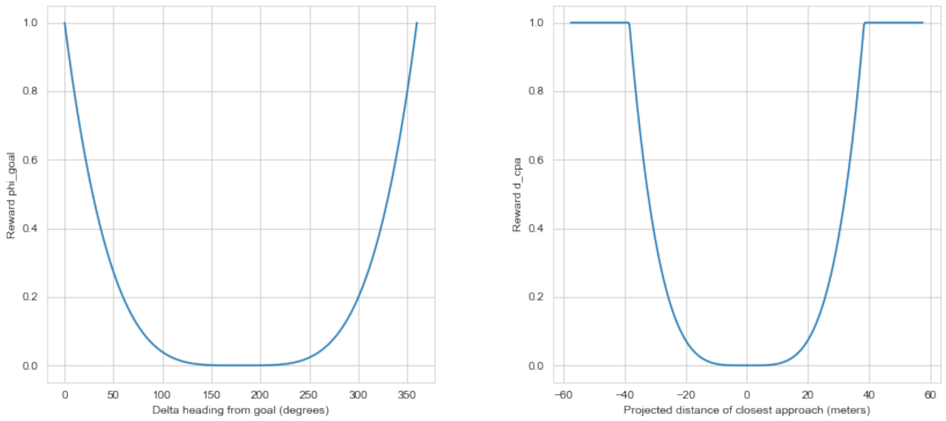
The functional form of $R_{d_{cpa}}$ is shown in Fig. 4 (b).

Straight-line-to-goal plan deviation reward

$$R_{d_{dev}} = \begin{cases} (1 - \frac{d_{dev}}{D_{dev_{max}}})^{0.5} & \text{if } d_{dev} \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

where $D_{dev_{max}} = d_{goal}(0)/2$ is a design parameter of our reward function, selected so that it constrains the agent's trajectory close to the straight-line-to-goal plan.

The functional form of $R_{d_{dev}}$ is shown in Fig. 5 (a).



(a)
Heading reward.

(b)
Closest approach distance reward.

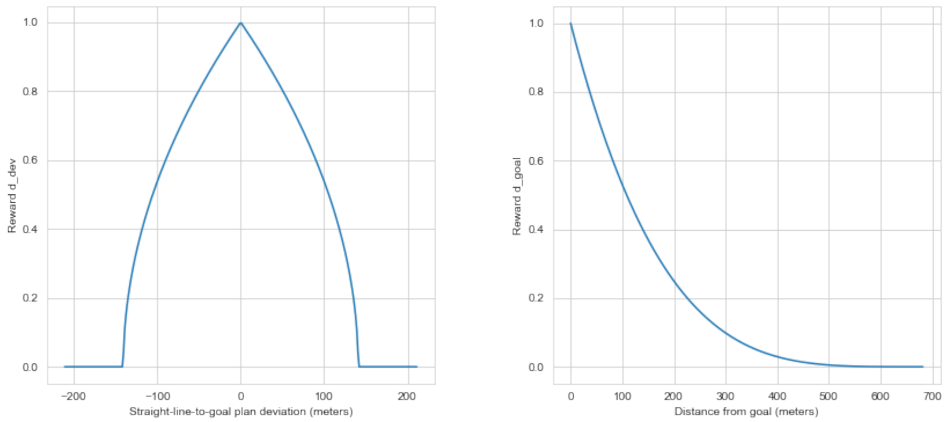
Fig. 4 Step reward components - Heading and closest approach.

Goal distance reward

$$R_{d_{goal}} = min(1, (\frac{d_{goal}}{d_{goal_{max}}})^4) \tag{30}$$

where $d_{goal_{max}} = d_{goal}(0) + V_{air}T_{episode}$ is the maximum distance from goal the agent can achieve over the course of an episode.

The functional form of $R_{d_{goal}}$ is shown in Fig. 5 (b).



(a)
Plan deviation reward.

(b)
Goal distance reward.

Fig. 5 Step reward components - Plan deviation and goal distance.

Our step reward function design has some significant advantages. Firstly, its functional form is intuitive, as the product of targets that the agent is trying to achieve simultaneously. Moreover, it is bounded between 0 and 1, which comes with convenient mathematical properties; for instance, it is easy to derive upper bounds for the total step reward over the course of an episode, which is very helpful for defining meaningful terminal rewards. Last but not least, the very few tunable parameters of our reward function all have a straightforward natural meaning, such as D_{safe} and $D_{dev_{max}}$ (in contrast, for example, to tunable weight parameters) making the reward function and the policies it generates more interpretable.

IV. Agent Implementation, Training and Evaluation

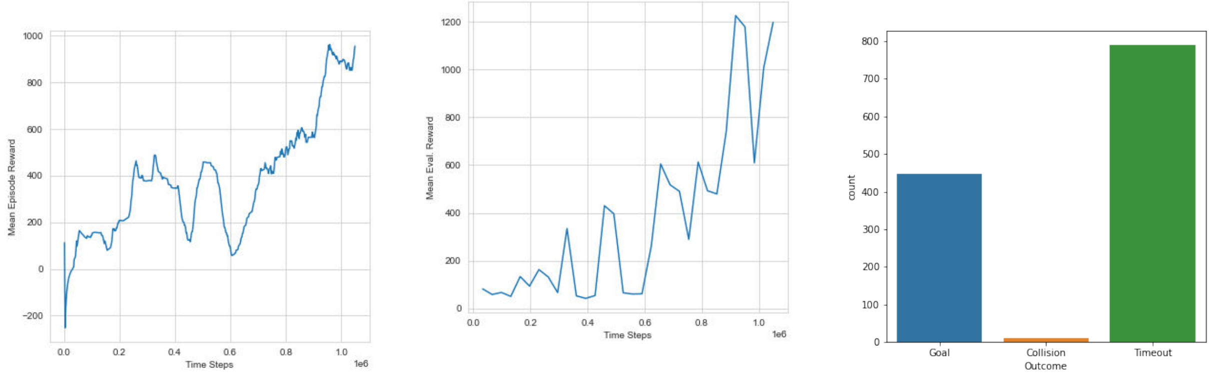
In this section, we demonstrate our training results, and provide a thorough evaluation of our trained agent through Monte Carlo simulations. We demonstrate the performance of our agent in the collision avoidance task, highlighting its safety performance and the trade-off between safety and mission performance.

A. Implementation

Our environment was implemented using Open AI Gym ([19] and [20]); a toolkit for developing and comparing RL algorithms. Our learning agent was implemented using Stable Baselines 3 (SB3) [21], the PyTorch [22] version of Stable Baselines (SB) [23], which in turn is based on OpenAI Baselines ([24]). These libraries provide a set of high-quality implementations of reinforcement learning algorithms, aiming to make it easier for the research community to produce reproducible research results. We used the PPO algorithm using the default settings and hyper-parameters as specified in SB3 documentation. This was because the agent was able to demonstrate "signs-of-life" early on during training with the default hyper-parameters and therefore hyper-parameter tuning was neither necessary nor within the scope of our analysis.

B. Training Results

Our agent was allowed a budget of 1,048,576 (2^{20}) total training time steps, which translated in approximately 4 hours of running time using CPU device on a laptop computer (Intel® Core™ i7-8565U Processor - 8GB RAM). The performance of the agent was monitored during training and the best policy was saved periodically, at evaluation cycles of 10 simulation episodes every 32,768 (2^{15}) time steps (32 evaluation points). The training results are summarised in Fig. 6.



(a) Mean episode reward over time.

(b) Mean reward of 10 evaluation episodes over time.

(c) Training episode outcomes.

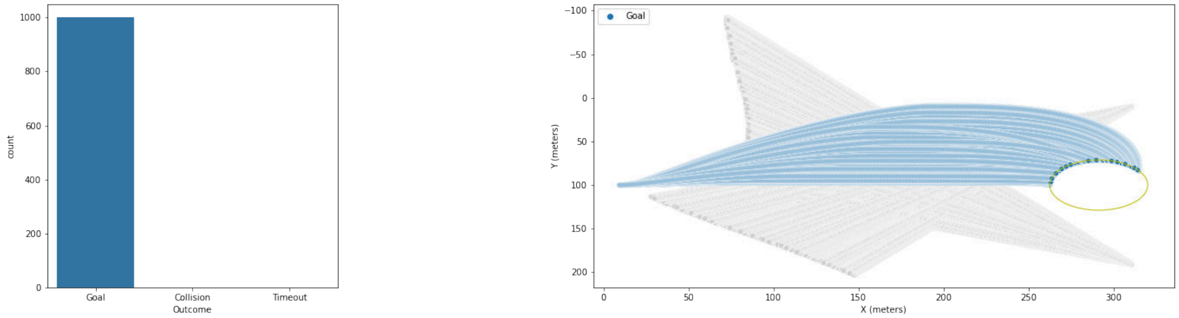
Fig. 6 Training results.

Clearly, the training curves demonstrate that the agent made steady progress in learning to tackle the collision avoidance task (therefore maximising its episode rewards) over time and the evaluation process was able to mirror this progress and capture the best policies that the agent was able to discover throughout the training process. Interestingly, the episode outcomes demonstrate that there were very few collisions throughout the training process, which shows that our agent found it relatively straightforward to learn to avoid collisions, but found it significantly more challenging to do so in an efficient manner, that would allow it to arrive to target on time.

C. Performance Evaluation

1. Simulation Test Summary

Our trained agent was tested for 1000 simulation episodes as shown in Fig. 7. It is important to mention that the stochasticity of our simulation environment was controlled via common random number generator seeds, ensuring that our trained agent was tested on the exact same encounters as the baseline agent.



(a) Episode outcomes

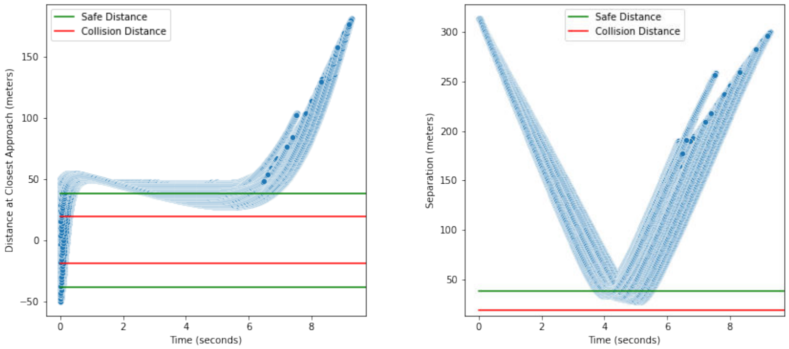
(b) Agent-Traffic encounters

Fig. 7 Trained agent simulation

As shown in the figure, the agent has learnt to solve the collision avoidance task; it achieved to avoid the NMAC and arrive at the target waypoint safely in all test episodes. This can be expressed as: $Outcome_{\%}(GOAL) = 100\%$ and $CRR = 0$. Although, clearly this is a perfect score in terms of encounter outcomes, in order to fully understand the performance of our system we need to analyse these successful trajectories with respect to mission performance and safety metrics.

2. Safety Performance

Although all of our agent's trajectories are successful, it is important to understand the level of safety with respect to collision that they achieve. To this end, we analyse these trajectories with respect to their safety properties. Figure 8 provides a summary of the agent's simulation performance with respect to separation metrics.



(a) Projected Closest Approach

(b) Separation

Fig. 8 Agent's separation performance

Figure 8 (a) captures the projected distance of closest approach between the agent and traffic aircraft over time for each episode. This parameter captures the agent's perception of collision risk and can be positive or negative, depending on the relative position of the aircraft at the point of closest approach. The figure demonstrates how the D_{safe} parameter of our step reward function creates a safe buffer zone before the collision distance is reached. Trajectories that are projected to penetrates this buffer zone generate less step reward, and this leads the agent to learn to avoid such trajectories. As a result, the agent follows trajectories that maintain a safe projected separation, minimising collision risk. As a result, Fig. 8 (b) shows that the agent achieves trajectories that penetrate the D_{safe} buffer zone but never reach the collision threshold.

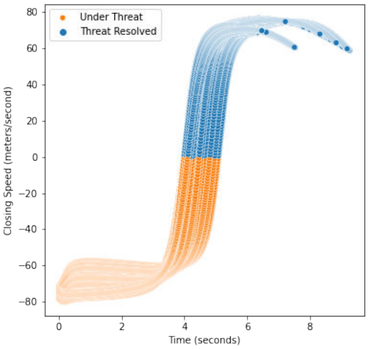
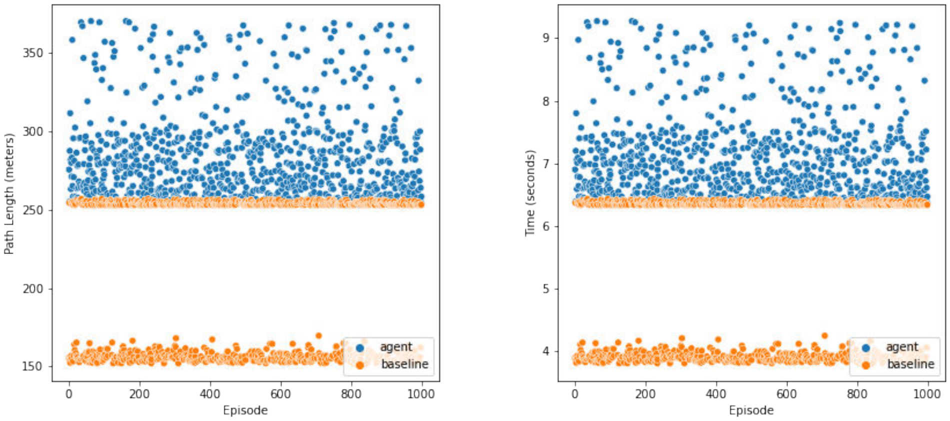


Fig. 9 Agent's threat resolution time

Another important aspect of the performance of our collision avoidance system is conflict resolution time. When $v_{closing} < 0$ the aircraft are getting flying towards the general direction of each other, therefore this condition represents a state of collision threat. The first goal of the agent is to resolve this danger before it proceeds to fly to the target waypoint. In Fig. 9 we see how long it took the agent to resolve the collision threat in each episode. As shown in the figure, the agent took roughly 4 to 5 seconds (half the episode time limit) to resolve the conflict and escape the collision threat, not only achieving safety quickly, but also allowing enough time to correct its trajectory and arrive at the target waypoint on time.



(a) Trajectory lengths

(b) Trajectory times

Fig. 10 Agent's mission performance

3. Mission Performance

Another important consideration is the efficiency of our agent's collision avoidance policy with respect to mission performance. Figure 10 compares the trajectories of our agent to the successful trajectories of the baseline fly-straight-to-goal agent with respect to path length and time. As shown in the figure, the baseline agent's path lengths and times are split in two clusters, corresponding to goal and collision episode outcomes, and the path lengths/times are fairly constant for each outcome (around 250 meters/6.3 seconds long for goal outcomes and around 150 meters/4 seconds for collision outcomes). The small deviations can be attributed to the initial heading range of the agent and the different traffic trajectories in the case of collisions. On the contrary, for the trained agent there is only one single cluster, since all trajectories lead to a goal outcome, however the path lengths and times are longer and scatter a lot more. Clearly, safety comes at a reasonable price in mission performance.

V. Conclusion

In this study, we used Deep Reinforcement Learning (DRL) to design an autonomous Airborne Collision Avoidance System (ACAS) for a simple 2D airspace and aircraft encounter model. Using a composite reward function that incorporates collision risk information via relative geometry and motion attributes of the agent's trajectory, we were able to train an agent that performed the collision avoidance task perfectly with respect to safety (Collision Risk Ratio = 0), whilst maintaining a reasonable mission performance. In future work, we will attempt to transition to an increasingly more realistic setting, by migrating to a 3D environment and relaxing the assumptions that we made about our system. For instance, this may include more realistic aircraft encounter models, sensor information and aircraft controls. We may also extend our methodology to cooperative approaches, where aircraft coordinate to avoid collisions efficiently. Another interesting research direction would be the integration of our autonomous ACAS with other systems, such as a higher level autonomous control system, or an autonomous Air-Traffic Management (ATM) system. Finally, we may examine different approaches towards establishing the trustworthiness of autonomous ACAS, for example through verification, testing and assessing the explainability of the system's collision avoidance policies. In this area, it will be important to approach trustworthiness as a continuous process over the system's life, where the learning agent can be continuously updated over time using Incremental Learning (IL), Online Learning (OL) or Transfer Learning (TL) and go through the mechanisms of verification, testing and explainability assessment, remaining at a trustworthy state.

References

[1] "UAM Glossary," , 2021. URL https://www.flight-crowd.com/uam-glossary.

[2] UKRI, and PwC, "Future Flight Challenge Socio-Economic Study," , 01 2021. URL https://www.ukri.org/wp-content/uploads/2021/01/UKRI-120121-ISCFFutureFlightChallenge-Socio-economicStudy.pdf.

[3] Kochenderfer, M., Holland, J., and Chryssanthacopoulos, J., "Next-Generation Airborne Collision Avoidance System," *Lincoln Laboratory Journal*, Vol. 19, 2012.

[4] Kuchar, J., and Yang, L., "A Review of Conflict Detection and Resolution Modeling Methods," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 1, 2000, pp. 179–189. https://doi.org/10.1109/6979.898217.

[5] Yasin, J. N., Mohamed, S. A. S., Haghbayan, M.-H., Heikkonen, J., Tenhunen, H., and Plosila, J., "Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches," *IEEE Access*, Vol. 8, 2020, pp. 105139–105155. https://doi.org/10.1109/access.2020.3000064.

[6] Deaton, J., and Owen, M. P., "Evaluating Collision Avoidance for Small UAS Using ACAS X," AIAA Scitech 2020 Forum, 2020. https://doi.org/10.2514/6.2020-0488.

[7] Holland, J. E., Kochenderfer, M. J., and Olson, W. A., "Optimizing the Next Generation Collision Avoidance System for Safe, Suitable, and Acceptable Operational Performance," *Air Traffic Control Quarterly*, Vol. 21, 2013, pp. 275–297. https://doi.org/10.2514/atcq.21.3.275.

[8] Sutton, R. S., and Barto, A. G., *Reinforcement Learning, second edition: An Introduction*, MIT Press, 2018.

[9] Temizer, S., Kochenderfer, M., Kaelbling, L., Lozano-Perez, T., and Kuchar, J., "Collision Avoidance for Unmanned Aircraft Using Markov Decision Processes," AIAA Guidance, Navigation, and Control Conference, 2010. https://doi.org/10.2514/6.2010-8040.

[10] Graesser, L., and Keng, W. L., *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*, Addison-Wesley Professional, 2019.

[11] Zai, A., and Brown, B., *Deep Reinforcement Learning in Action*, Manning, 2020.

[12] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P., "Trust Region Policy Optimization," PMLR, 2015, p. 1889–1897. URL http://proceedings.mlr.press/v37/schulman15.

[13] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous Control with Deep Reinforcement Learning," , 2015. URL https://arxiv.org/abs/1509.02971.

[14] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," PMLR, 2018, p. 1861–1870. URL http://proceedings.mlr.press/v80/haarnoja18b.

[15] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," , 2017. URL https://arxiv.org/abs/1707.06347.

[16] OpenAI, "Proximal Policy Optimization," , 2021. URL https://openai.com/blog/openai-baselines-ppo/#ppo.

[17] García, J., and Fernández, F., "A Comprehensive Survey on Safe Reinforcement Learning," *Journal of Machine Learning Research*, Vol. 16, 2015.

[18] Pardo, F., Tavakoli, A., Levdik, V., and Kormushev, P., "Time Limits in Reinforcement Learning," Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 2018.

[19] "OpenAI," , 2021. URL https://openai.com/.

[20] OpenAI, "Gym: a Toolkit for Developing and Comparing Reinforcement Learning Algorithms," , 2021. URL https://gym.openai.com/.

[21] "Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations," , 2021. URL https://stable-baselines3.readthedocs.io/en/master/index.html.

[22] "PyTorch - Open Source Machine Learning Framework That Accelerates the Path from Research Prototyping to Production deployment." , 2021. URL https://pytorch.org/.

[23] "Stable Baselines Docs - RL Baselines Made Easy," , 2021. URL https://stable-baselines.readthedocs.io/en/master/.

[24] "OpenAI Baselines," , 2021. URL https://openai.com/blog/tags/baselines/.

2021-12-29

Towards safe deep reinforcement learning for autonomous airborne collision avoidance systems

Panoutsakopoulos, Christos

AIAA