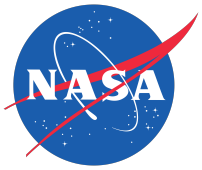
NASA/TM–20250002761



# Simulation Framework for Tactical Separation Assurance Service using Deep Reinforcement Learning

*Gautam Sai Yarramreddy*
*Universities Space Research Association, Moffett Field, California 94035, United States*

*José Ignacio de Alvear Cárdenas*
*San José State University, Moffett Field, California 94035, United States*

*Priyank Pradeep*
*Analytical Mechanics Associates Inc, Moffett Field, California 94035, United States*

*Min Xue and Seungman Lee*
*NASA Ames Research Center, Moffett Field, California 94035, United States*

*Vincent H. Kuo*
*Metis Technology Solutions, Moffett Field, California 94035, United States*

March 2025

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

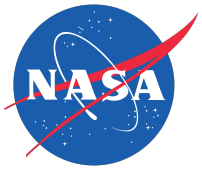
- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, and organizing and publishing research results.

For more information about the NASA STI Program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*
- E-mail your question to *help@sti.nasa.gov*
- Fax your question to the NASA STI Information Desk at 443-757-5803
- Phone the NASA STI Information Desk at 443-757-5802
- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076–1320

NASA/TM–20250002761



# Simulation Framework for Tactical Separation Assurance Service using Deep Reinforcement Learning

*Gautam Sai Yarramreddy*
*Universities Space Research Association, Moffett Field, California 94035, United States*

*José Ignacio de Alvear Cárdenas*
*San José State University, Moffett Field, California 94035, United States*

*Priyank Pradeep*
*Analytical Mechanics Associates Inc, Moffett Field, California 94035, United States*

*Min Xue and Seungman Lee*
*NASA Ames Research Center, Moffett Field, California 94035, United States*

*Vincent H. Kuo*
*Metis Technology Solutions, Moffett Field, California 94035, United States*

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

March 2025

## Acknowledgments

The material is based upon work supported in part by NASA under award number NNA16BD14C and 80ARC024DA007 for NASA Academic Mission Services (NAMS-1 and NAMS-2) respectively.



Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320

National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Available electronically at http://www.sti.nasa.gov

# Simulation Framework for Tactical Separation Assurance Service using Deep Reinforcement Learning

## Summary

Deep Reinforcement Learning (DRL) is a revolutionary Artificial Intelligence (AI) methodology that combines Reinforcement Learning and Deep Neural Networks. This report proposes a simulation framework for training and evaluating DRL approaches for tactical separation assurance services in the Unmanned Aircraft Systems (UAS) Traffic Management (UTM) ecosystem for UAS-to-UAS conflict detection and resolution. The DRL agent is developed in Python using PyTorch and TorchRL. NASA's GPU-enabled Flexible engine for Fast-time evaluation of Flight environments ($Fe^3$) is chosen as the simulation environment for data collection and policy evaluation. $Fe^3$ is responsible for i) simulating different scenario geometries, ii) detecting UAS-to-UAS conflicts, and iii) computing the states and rewards given the actions chosen by the DRL agent. The $Fe^3$ and Python processes share data via NVIDIA's CUDA inter-process communication Application Programming Interface (API) and communicate using named pipes. Named pipes are used to ensure proper data flow and avoid race conditions between the Fe3 and Python processes. The Python process opens two pipes, one that it can only read to and one that it can only write to.

Double Deep Q-Network with experience replay is the reinforcement learning algorithm implemented to assess the functionality of the framework. It is a model-free approach that updates a Q-value function and follows an $\epsilon$-greedy policy during training and a greedy policy at inference. During the training stage, the data samples collected from the interactions with the environment are stored in a replay buffer to improve data efficiency, prevent forgetting experiences, and reduce training data correlation. Furthermore, the Q-value function is represented by two neural networks: the Q-network for selecting the best action and the target network for evaluating it, reducing the oscillatory behavior during training.

The agent dynamics were modeled in Fe3 based on the performance characteristics of DJI Phantom 4.0 UAS, and its action space was constrained to heading changes. The state space used as input for the decision-making process is based on selected UAS states, UAS-to-UAS conflict states, and $Fe^3$'s simulation states. The reward used to guide the learning process consisted of intermediate and termination components that encouraged safety and efficiency, the former in terms of the closest lateral distance and the latter in the form of energy consumption minimization.

Preliminary results from a dataset comprising 150 UAS pairwise (UAS-to-UAS) conflict scenario geometries, run for 50 epochs, demonstrate that the agent is successfully learning. This is evident from an increase in the percentage of deconflicted scenarios and the maximum reward achieved per epoch. In addition, the safety performance of the policy has improved by deconflicting earlier in the flight and at greater distances from the conflict area.

# 1 Introduction

The FAA and NASA envision unmanned aircraft systems (UAS) traffic management (UTM) having a multi-layer conflict management model to ensure the safe, efficient, and scalable operations of UAS between each other. These layers are referred to as strategic deconfliction, tactical deconfliction, and collision avoidance (Ref. 1, 2). At each layer, UAS-to-UAS conflicts are resolved through a series of maneuvers compatible with the operational environment. The objective of the first layer of the conflict management model, i.e., strategic deconfliction, is to i) minimize the likelihood of airborne conflicts between UAS operations and ii) maximize the airspace usage by adjusting the departure times of UAS (Ref. 2, 3). The tactical deconfliction layer consists of two levels. The first level, i.e., tactical separation assurance, consists of executing one or more maneuvers (speed, altitude, and heading changes) to avoid an UAS-to-UAS airborne conflict promptly when strategic deconfliction was not completed or underperformed due to uncertainties (Ref. 4–7). It is activated 3 minutes before the near midair collision (NMAC) between pairwise UAS. The second level of tactical deconfliction, i.e., detect and avoid (DAA), handles UAS-to-UAS conflicts that are not detected until $\leq 1$ min prior to the NMAC or, when successfully detected by the first level, but could not be resolved successfully (Ref. 5, 7). Finally, the last layer of protection in the UTM ecosystem is the onboard collision avoidance system (Ref. 3).

## 1.1 Background and Motivation

In the UTM ecosystem, there is a demand for an autonomous air traffic control system to enable safe, efficient, and scalable UAS beyond visual line-of-sight (BVLOS) operations (Ref. 8–10). The critical challenge is researching and building an autonomous air traffic control system to provide real-time conflict detection and resolution solutions to UAS in UAS-to-UAS conflict. To this end, the Next-Generation Airborne Collision Avoidance System (ACAS-X) (Ref. 11) was built upon Traffic Alert and Collision Avoidance System (TCAS) (Ref. 12), introducing a partially observable Markov decision process (POMDP) (Ref. 13) for the problem formulation. TCAS and ACAS-X are designed to resolve pairwise conflicts between manned aircraft with vertical maneuvers (Ref. 14). The difference is that TCAS uses fixed rules to resolve conflicts, whereas ACAS uses a probabilistic model to represent future aircraft positions. A tailored version of this system for use onboard UAS, ACAS Xu (Ref. 15), incorporates altitude and heading change maneuvers for collision avoidance (Ref. 16).

Tactical separation assurance is an intermediate traffic management layer of the UTM ecosystem for keeping UAS safe from conflict and collision hazards by an appropriate separation criterion, namely the minimum deviation from the original flight path (Ref. 5). Therefore, tactical separation assurance involves preventing an NMAC between UAS in-trail, at intersections, and metering fixes by providing advisory maneuvers to UAS (Ref. 17). In addition to resolving conflicts, it also provides trajectory segments that return the UAS back to its original flight path after the resolution segments of the trajectories have been completed (Ref. 5, 7).

Recently, reinforcement learning (RL) (Ref. 18) approaches have shown potential in conflict detection and resolution problems. Reinforcement learning aims to allow an agent to learn an optimal policy by directly interacting with the environment. Various deep reinforcement learning (DRL) algorithms have been researched for collision avoidance (Ref. 19–22), tactical deconfliction (Ref. 8, 17, 23), and strategic deconfliction (Ref. 24–26) in conventional air traffic, and urban air mobility. However, in the UTM ecosystem, very little research has been performed in the tactical

separation assurance time scale, i.e., three to one minutes before the UAS-to-UAS conflict (Ref. 5).

To enable the study of DRL strategies for tactical separation assurance services, the main contribution of this research is a simulation framework that facilitates the interaction of a Python DRL agent with a fast-time evaluation tool previously developed at NASA Ames Research Center. To test its functionality, a DRL approach from literature — Double Deep Q-Network with experience replay — was implemented for UTM pairwise (UAS-to-UAS) deconfliction within the tactical separation assurance time frame. The model balances safety and efficiency by applying heading change maneuvers and incorporating a higher-fidelity 6-DOF physics-based energy consumption model (Ref. 5, 27, 28) in the reward function. The rest of this report is organized as follows. Section 2 describes the simulation framework, explains RL, and discusses the specific DRL algorithm implemented. Section 3 presents preliminary results in terms of the percentage of deconflicted scenarios, safety and efficiency. Finally, Section 4 provides conclusions, future research directions and recommendations for future work.

# 2 Methodology

## 2.1 Simulation Framework

NASA's Flexible engine for Fast-time evaluation of Flight environments ($Fe^3$) (Ref. 29) is used as the simulation environment modeling flight kinetics in the UTM context. The flight kinematics, flight dynamics, power model, and quadrotor performance models (Ref. 30) have been incorporated in $Fe^3$ to simulate various pairwise scenarios with conflicts where the agent performs heading change tactical maneuvers and learns a policy that optimizes for both safety and efficiency. The training and testing scenarios are randomly generated with a pair of UASs having conflicting 4D (3D space and 1D time) trajectories. $Fe^3$ uses NVIDIA GPUs for accelerated computation performance and parallelized simulations of scenarios (Ref. 31).

This research used the deep learning library PyTorch and the reinforcement learning framework TorchRL to train the agent for tactical deconfliction using heading change maneuvers and execute the policy (Ref. 32, 33). As shown in Figure 1, the $Fe^3$ process calculates simulator and UAS states periodically at every time step, simulates and detects pairwise (UAS-to-UAS) scenarios with conflicts, and executes heading change for the UAS. The Python process uses the simulated states calculated by $Fe^3$ as input to the agent to calculate the optimal heading change maneuver to avoid a conflict. $Fe^3$ then simulates the next time step given the heading change maneuver. This interaction between $Fe^3$ and the RL agent in Python starts when a conflict has been detected and continues back and forth until the conflict has been resolved. CUDA inter-process communication (IPC) and IPC via Unix-named first-in-first-out pipes (FIFO) were used to share the state data between the $Fe^3$ and Python process (Ref. 31).

Named pipes are used to ensure proper data flow and avoid race conditions between the $Fe^3$ and Python processes. The Python process opens two pipes, one that it can only read to and one that it can only write to. The $Fe^3$ process opens the same pipes in the opposite modes, opening the Python read-only pipe in write-only mode and the Python write-only pipe in read-only mode. A process is able to write to a pipe and the other process is able to read data from that pipe. The writing function is non-blocking, whereas the reading function is blocking, meaning that if the pipe is empty, the process waits until another process writes enough data to read. During the main simulation loop, these pipes are used to relay status messages when a process is done using

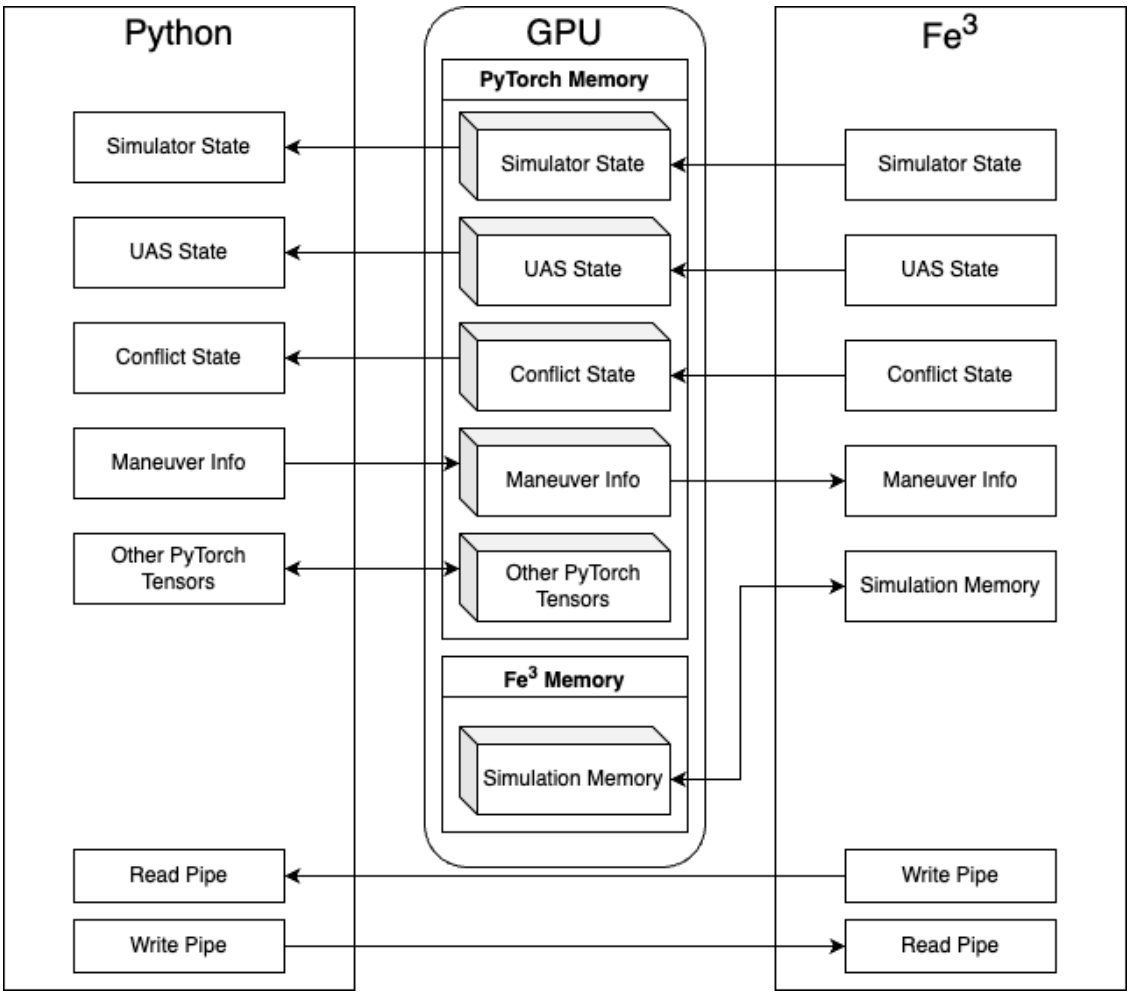



**Figure 1.**—Shared memory and Inter-Process Communication between Python and $Fe^3$ Processes.

the shared memory and allowing the other process to use it. For example when $Fe^3$ is finished calculating the states it sends a message via the named pipes to the Python process letting it know to calculate the best maneuver. In addition, when allocating GPU memory at the beginning of the program, the pipes are also used to send information necessary for opening the shared GPU memory (Ref. 34).

Various states such as UAS, simulator, conflict, and maneuver information are stored on GPU memory during the simulation. Transferring data between the GPU and CPU is costly and can slow down the simulation drastically if conducted often and with large amounts of data. Instead, the data is kept on the GPU, but both the $Fe^3$ and Python processes have access to the same block of memory. This allows both processes to read and write to the same block of memory without little to no overhead. Along with the shared GPU memory, the Python process stores other tensors on the GPU including the weights for the neural network policy and the gradients. $Fe^3$ stores memory on the GPU necessary for the simulation including the ordinary differential equation (ODE) state, waypoint data, wind data, etc.

### 2.2 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning, that involves an agent interacting with an unknown deterministic or stochastic environment through sequential decision making (Ref. 18) to achieve a long-term goal. The components that make up a reinforcement learning problem can be defined by the tuple ($S$, $A$, P, R, $\gamma$). At each time step, t, an agent observes the current state ($S_t \in S$) and selects an action ($A_t \in A$) based on its policy ($\pi$); function that maps each state to a distribution in the action space. $S$ and $A$ represent the set of all possible states and actions in the environment, namely the state and action space, which can be discrete or continuous. Based on the state-action tuple ($S_t$, $A_t$), the state is then updated to $S_{t+1}$ and the agent receives a reward ($R_t$). How the state progresses from $S_t \rightarrow S_{t+1}$ given action $A_t$ is determined by the dynamics of the environment represented by the transition probability function $\mathrm{P}(S_{t+1}|S_t,\, A_t)$. The reward function $\mathrm{R}(R_{t+1}|S_t,\, A_t,\, S_{t+1})$ assigns the reward to the agent for reaching state $S_{t+1}$ from state $S_t$ given action $A_t$.

After training, the optimal policy ($\pi^*$) representing the agent's strategy will take actions that maximize the expected cumulative reward over time, also known as the expected return ($\pi^*(t) = \arg\max_\pi \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1}|\pi]$). Correctly shaping the reward function is essential to align the agent's final strategy with its initial goal within the environment, since the reward tells the agent what we want it to accomplish and not how to achieve it (Ref. 18). This task is particularly challenging as the reward function must also be designed to prevent the agent from getting stuck in local minima, which can impact convergence speed (Ref. 35). Finally, the discount factor $\gamma \in (0, 1)$ determines how much value do future rewards have in the present. As $\gamma \rightarrow 0$, immediate rewards are emphasized, whereas, when $\gamma \rightarrow 1$, future rewards are also considered (Ref. 18). The end product of applying the reinforcement learning paradigm to a problem is a policy that maximizes the cumulative reward over time.

Two main approaches can be distinguished for training an agent to discover its optimal policy, namely policy- and value-based methods. Policy-based methods ($\pi(S_t) = P(A_t|S_t)$) teach the agent directly what action to take at every time step by learning a policy function that maps each state to the best possible action (deterministic) or by defining a probability distribution over the available actions at that state (stochastic). In contrast, value-based methods teach the agent to learn which state is more valuable by learning a value function that maps a state to the expected value or utility of being at that state.

The value function can have two representations. The state value function $V_\pi(S_t)$ is the expected discounted return the agent would obtain if it would start in state $S_t$ and then act according to the policy ($\mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots|S_t]$), whereas the action-value function $Q_\pi(S_t, A_t)$ reflects the expected discounted reward when starting in state $S_t$, taking action

$A_t$ and then following the policy ($\mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t, A_t]$). Both value functions are related to each other by the following equality: $Q_\pi(S_t, A_t) = R(S_t, A_t, S_{t+1}) + \gamma V_\pi(S_t)$, where $S_{t+1} \sim P(S_{t+1}|S_t, A_t)$. An optimal value function ($V^*$ or $Q^*$) is reached when it meets the Bellman optimality equation in Equation 1 (Ref. 18). Once the optimal value function has been computed, a greedy policy that selects the action with the highest utility is usually chosen ($\pi^* = \arg\max_{A_t} Q^*(S_t, A_t) = \arg\max_{A_t} R_{t+1} + \gamma V^*(S_t)$) (Ref. 19).

$$
\begin{aligned}
V^*(S_t) &= \max_{A_t} \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1})] \\
Q^*(S_t, A_t) &= \mathbb{E}[R_{t+1} + \gamma \max_{A_{t+1}} Q^*(S_{t+1}, A_{t+1})]
\end{aligned}
\tag{1}
$$

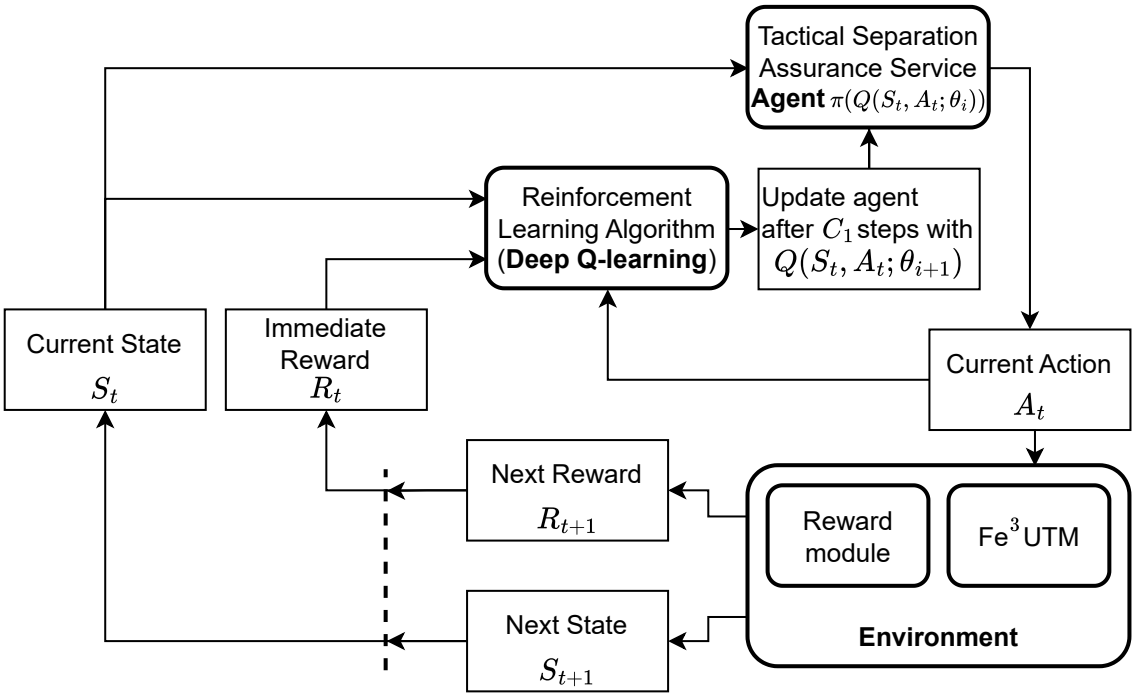



**Figure 2.**—Schematic diagram of state, action, and reward in reinforcement learning framework in UTM context. Adapted from (Ref. 18).

### 2.2.1 Double Deep Q-Network

Q-learning is a model-free reinforcement learning algorithm for estimating action-value functions which falls under the category of temporal difference (TD) value-based learning techniques (Ref. 18, 36, 37). In contrast with Monte Carlo based methods that wait until the end of an episode for computing the return and updating the value function (complete episode environment interactions are required), TD learning techniques only wait for one time step in order to compute an update. Here, an episode refers to a single sequence of interactions between an agent and its environment, starting from an initial state and ending at a terminal state. Since TD methods have

not experienced an entire episode, the expected return is estimated using the immediate reward ($R_t$) and the discounted value of the next state. Hence, in the case of Q-learning, the Q-values are iteratively updated as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{A_{t+1}} Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)], \tag{2}$$

where $\alpha$ is the learning rate, $\gamma$ is the discount factor and $\max_{A_{t+1}}$ is the maximum Q-value for the next state given all possible next state actions (Ref. 18).

To balance the exploration vs exploitation trade-off during training, the action $A_t$ is chosen by an $\epsilon$-greedy policy that selects a random action with probability $\epsilon$ and the action with highest value according to the current Q-value function with probability 1-$\epsilon$. The update to the action-value function in Equation 2 following the learning rate is called the TD error or Q-loss (Ref. 18) and its first two terms within brackets are known together as the TD target because they are the value the output of the Q-value function (third term of the TD error) should aim for to achieve optimality, as seen in Equation 1.

In this research, the Double Deep Q-Network (Double DQN) algorithm variant with experience replay is implemented where the Q-value function is approximated using a neural-network model (NN) (Ref. 38). Experience replay consists of populating a replay buffer ($\mathbf{D}$) at every simulated time step with transition tuples of the form ($S_t$, $A_t$, $R_{t+1}$, $S_{t+1}$) (Ref. 39) and, periodically, sampling from it a mini-batch of transitions for training the NN. Mini-batch refers to a small, randomly selected subset of experiences or transitions of size $B$ that are sampled from a larger dataset of experiences ($\mathbf{D}$) accumulated during the training process. It improves data efficiency by reusing each data sample multiple times during training, prevents catastrophic forgetting of previous experiences, and reduces sequential training data correlation by enabling data batches of diverse past experiences (Ref. 40, 41).

A double NN approach was taken for representing the Q-value function (Ref. 38, 39, 42); one for selecting the best action at the current and next state (Q-network with parameters $\theta$ — NN weights and biases) and another for evaluating the best action (target network with parameters $\theta^-$). The role of each network can be seen from the loss ($\mathcal{L}$) for training the Q-network in Equation 3. The Q-network is updated with frequency $C_1$, every time a mini-batch of transitions is sampled from the replay buffer for training, whereas the target network is kept constant and only updated with the parameters of the Q-network after multiple Q-network updates with frequency $C_2$. The target network was introduced to reduce the emergent oscillatory behavior during training due to the correlation between the TD target and the output of the Q-network, which would otherwise share the same parameters (Ref. 39). Including the Q-network within the target network for selecting the action at the next state counteracts the positive bias previously observed in Q-learning (Ref. 42). Both the data collection and training phases of Double DQN can be seen in Figure 3.

$$\mathcal{L}(\theta) = \frac{1}{2}\left(R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{A_{t+1}} Q(S_{t+1}, A_{t+1}; \theta); \theta^-) - Q(S_t, A_t; \theta)\right)^2 \tag{3}$$

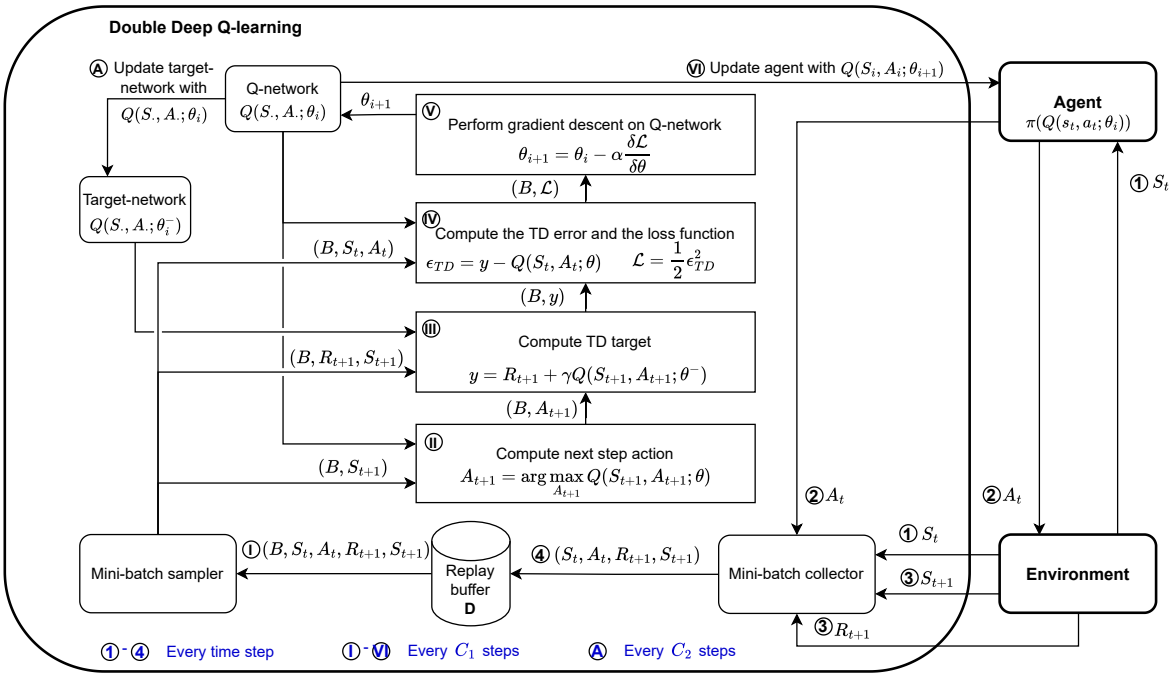



**Figure 3.**—Data collection and training in Double DQN.

## 2.3 Reinforcement Learning Framework

### 2.3.1 Environment: Conflict Scenarios

Scenarios are randomly generated so the agent can learn from diverse experiences. The pair of UAS have routes that intersect in 4D. The ownship UAS always starts with an initial heading of true north, while the intruder UAS approaches from varying bearing angles in each scenario. The relative heading between the two UAS ranges from 20° to 160°, and both UAS operate at the same altitude. Each UAS starts 4,000 meters from the conflict location, i.e., an estimated time of 200 seconds to conflict at a groundspeed of 20 m/s. When the simulation starts, each UAS travels on its respective route towards the final waypoints. In Fe3, by default a conflict is detected when the pair of UAS are within broadcast range of each other (6,000 meters) and are on track to collide. From this point, the DRL agent calculates heading change maneuvers for the ownship UAS to execute until one of the scenario termination conditions is met: the conflict is resolved or the vehicles have collided. A conflict is resolved when the ownship UAS has a clear line-of-sight to the destination.

### 2.3.2 Agent

This research treats the ownship UAS in the UTM airspace as the RL agent. The UAS performance data shown in Table 1 is based on DJI Phantom 4.0 UAS (Ref. 30). These parameters are used to compute the power required ($P_{\text{required}}$) and energy consumption for UAS flights. The instantaneous power required in forward flight is equal to the sum of the induced power, parasite power, climb power, and profile power (Ref. 28, 43, 44). The energy consumption ($E_{\text{consumption}}$) in time ($t$) is as follows (Ref. 30):

$$E_{\text{consumption}} = \int_0^{\text{t}} P_{\text{required}}\, dt \tag{4}$$

**Table 1.**— UAS Performance Parameters Relevant to Deep Reinforcement Learning (Ref. 30).

| UAS Performance Parameter | Value(s) |
|---|---|
| Cruise Airspeed | 20 m/s |
| Cruise Altitude | 121.92 m |
| Mass | 1.410 Kg |
| Rotor Diameter | 0.24 m |
| Equivalent Front Plate Area | 0.012 $m^2$ |
| Equivalent Top Plate Area | 0.03 $m^2$ |
| Maximum Heading Change Rate | 150°/s |

### 2.3.3 State Space

Selecting the appropriate state space dimensions is crucial to providing the agent with sufficient information to effectively navigate the environment optimizing for reward. To minimize computational complexity, unnecessary and redundant states should be avoided, which may be derived from combinations of others. For the dynamic re-routing of two agents, the state space has been defined in Table 2. In order to reduce the state space size, only scenarios where the intruder UAS approaches the ownship UAS from the right are considered. In the case that the trained agent encounters scenarios upon deployment where the intruder UAS is to the left, the intruder UAS is mirrored around the ownship UAS's original route for the state computation and the policy's output is mirrored back before execution in the environment, as can be observed in Figure 4. This method halves the size of the state space.

### 2.3.4 Action Space

The dimensions of the action space dictates how the agent interacts with the environment. To compute the optimal action based on the Q-value function, the action space needs to be discretized, with the number of output neurons in the value networks matching the number of possible actions. For the present research, only heading changes with respect to the original route heading are

**Table 2.**—State space definition.

| **State name** | **Limits** | **Units** | **Dimensions** |
|---|---|---|---|
| Simulation Time | [0, 4100] | s | 1 |
| Time to Conflict | [0, 180] | s | 1 |
| Ownship UAS position | [-4000, 4000] | m | 2 |
| Conflict Location | [-4200, 4200] | m | 2 |
| Relative distance | [-4200, 4200] | m | 2 |
| Relative velocity | [-40, 40] | m/s | 2 |

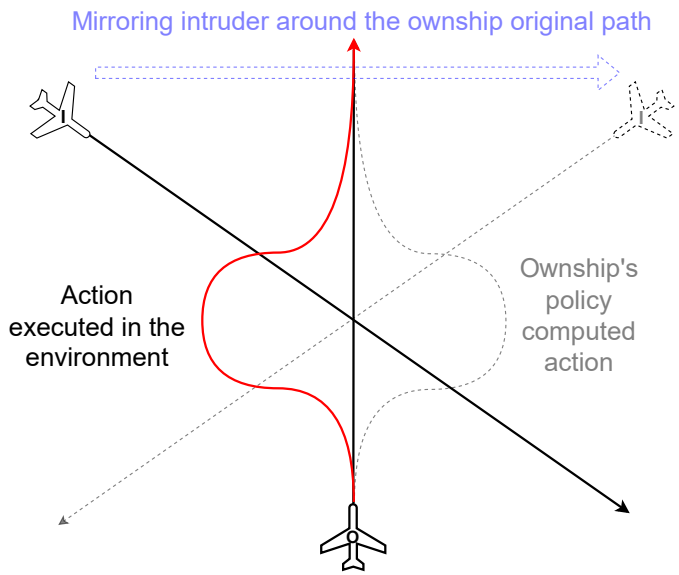



**Figure 4.**—Computation and execution of action when the intruder UAS approaches from the left of the ownship UAS's path.

considered in the range [-45°, 45°] segmented into intervals of one degree (specifically {-45°, -44°, -43°, ..., 44°, 45°}. As an example, if the UAS is heading East at the beginning of the scenario, namely 90° heading if the North is considered to be the reference direction, then it can only change its heading in the range [45°, 135°].

### 2.3.5 Reward Function

The agent's goal is to maintain visual contact with the destination while avoiding conflict with the intruder UAS. The reward function includes a large termination reward, complemented with smaller rewards at each time step, to accelerate the agent's learning of desired behaviors. To balance safety and efficiency, the reward function incorporates the following factors:

1. Intermediate Reward

   (a) Energy consumption penalty: Fe[3] incorporates an energy model of the UAS. Each time step's energy consumption ($\Delta E_s$) is penalized by multiplying it with a negative weight ($w_1$), aimed at minimizing the overall energy usage. This penalty incentivizes the agent to make decisions that are more energy-efficient.

   (b) Destination proximity reward: besides conflict resolution, the agent's goal involves progressing towards its destination. To incentivize this behavior, the distance flown towards the destination ($\Delta d_{OD}$) during each time step is multiplied by a positive weight ($w_2$).

2. Termination Reward

   (a) Base reward: an initial positive reward of 0.75 provides the agent a fixed incentive to reach the goal, signaling that the termination itself is not undesirable.

(b) Conflict resolution time frame penalty: Dynamic re-routing takes place between three minutes and one minute of collision ($1< t_c <3$). To promote conflict resolution before reaching the one-minute mark before impact, the agent incurs a large penalty ($w_3$) if the agent fails to resolve the conflict before that threshold.

$$\lambda_1 = \begin{cases} 1, & \text{if } t_c < 60. \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

(c) Total extra energy consumption penalty: A penalty is added that scales with the extra total energy used by the UAS when compared to the same UAS flying directly on the initial path as if there was no conflict ($\Delta E_t$) by a factor of $w_4$. This is to avoid maneuvers that have high total energy costs.

Considering all the aforementioned components shaping the reward, the intermediate ($R_{t+1}$) and the termination ($R_T$) signals can be formulated as in Equation 6 and Equation 7, respectively, with the weight values in Table 3.

$$R_{t+1} = w_1 \cdot \Delta E_s + w_2 \cdot \Delta d_{OD} \tag{6}$$

$$R_T = 0.75 + w_3 \cdot \lambda_1 + w_4 \cdot \Delta E_t \tag{7}$$

**Table 3.**—Reward function weights.

| $w_1$ | $w_2$ | $w_3$ | $w_4$ |
|---|---|---|---|
| -0.002 | 0.001 | -0.5 | -0.001 |

## 3 Preliminary Results and Discussions

Without performing a sensitivity analysis on the impact of each of the chosen states, actions and reward factors enumerated in Section 2, as well as fine-tuning their chosen values, preliminary results were collected in order to verify the functionality of the complete simulation pipeline and the learning of the DRL agent. The Q-value function consists of a multi-layer perceptron (MLP) with 3-hidden layers, each with 64 neurons. The input layer contains 10 neurons, corresponding to the 10 perceived environmental states (see Section 2.3.3), while the output layer has 91 neurons, representing the 91 possible heading angle changes (see Section 2.3.4). Leaky ReLU is the activation function for all the layers. The probability $\epsilon$ of the $\epsilon$-greedy policy linearly decays from 1 to 0.25 in 500,000 time steps, ensuring that initial actions are random and that there is always at least a 25% chance of taking a random action during training.

The replay buffer has a size of 300,000 samples and for every time step of data collected, 100 update steps with batches of 256 data points are performed on the Q-network. The Adam optimizer is used with a learning rate of $1 \cdot 10^{-5}$ and a discount factor of 0.99. Instead of updating the target network with the weights of the Q-network every $C_2$ steps, its weights are updated at every time step following Equation 8 with $\zeta$=0.99. Thanks to this software update (Ref. 45), the target changes slowly, improving learning stability at the expense of slower learning.

$$\theta_t^- = \theta_{t-1}^- \cdot \zeta + \theta_t^- \cdot (1 - \zeta) \tag{8}$$

Exploiting the automated conflict scenario tool discussed in Section 2.3.1, 150 flight plans were created for training and run for 50 epochs. Figure 5 shows how the percentage of successful deconflicted flight plans increases with the number of epochs for the training and validation scenarios. The validation scenarios were 100 randomly generated pairwise scenarios that were simulated after each epoch. The validation scenarios remained the same across the epochs. In both cases, the validation signal seems to lag behind the training counterpart by one epoch. Additionally, from these plots, it is clear that the policy has not yet converged and it could benefit from more training epochs.

Safety can be assessed by looking at the closest distance of the ownship to the intruder before declaring the conflict as resolved and disconnecting the DRL agent from the environment, as well as the time that has taken the DRL to resolve the conflict. Figure 6 presents the first metric by showing the closest ownship-intruder distance among all the scenarios in each epoch for the training and validation datasets. As the value function is trained with more epochs, the closest distance to the intruder while in conflict increases, showing that the agent is learning to take action early in order to minimize energy consumption. Similar conclusions can be derived from Figure 7, which shows the final ownship-intruder time to conflict remain much greater than the limit failure threshold of 60 seconds. Again, the oscillations in Figure 6 and Figure 7 signals call for more epochs to reach convergence.

Finally, the efficiency of the policy can be assessed by examining the total energy consumed by the agent per epoch, as illustrated in Figure 8. For the training dataset and validation dataset, the total energy consumption remains relatively constant. Again, it is clear that neither of the signals has converged. This suggests that further training of the value function is needed.

Some reasons for why the percent of successful deconflictions becomes very high early in the training, as seen in Figure 5 is that the model learns a suboptimal policy that successfully deconflicts UAS-to-UAS in many scenarios. The model then improves upon that policy by learning maneuvers that are more efficient but still successfully deconflicts UAS as seen as seen in Figure 8 as the total energy used decreases while the percent of successful deconflictions remains high.

To better understand what policy the model learned, Figure 9 shows the first flight plan of epoch 50. As can be seen, the ownship UAS (blue) initially makes a left turn of 35°, followed by a right turn of 57°, ultimately heading back toward the goal with a 22° left turn.

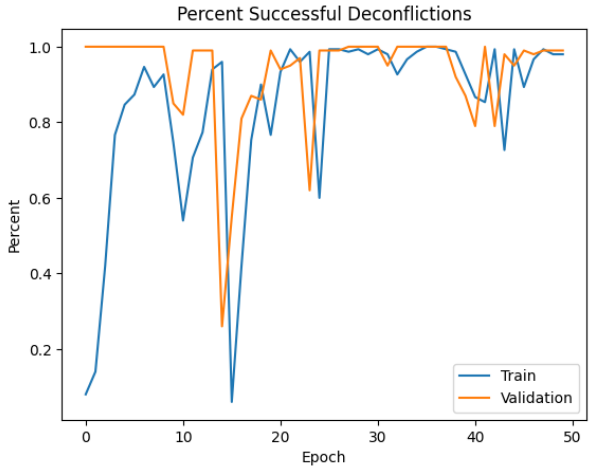



**Figure 5.**—Percentage of successfully deconflicted flight plans per epoch for the training and validation datasets.

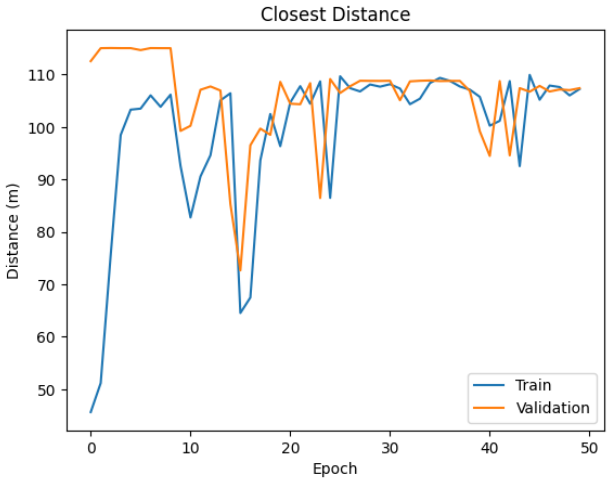



**Figure 6.**—Closest ownship-intruder distance while in conflict among all scenarios per epoch. The blue line reflects the training results whereas the orange line those from the validation dataset.


**Figure 7.**—Ownship-intruder time to conflict prior to conflict resolution among all scenarios per epoch. The blue line reflects the training results whereas the orange line those from the validation dataset.

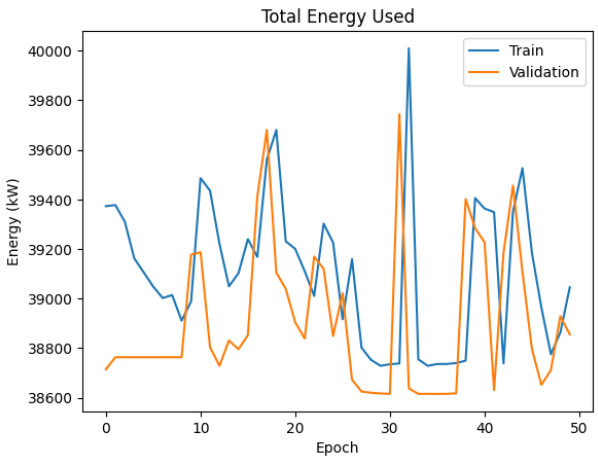



**Figure 8.**—Total energy consumed per epoch. The blue line reflects the training results whereas the orange line those from the validation dataset.

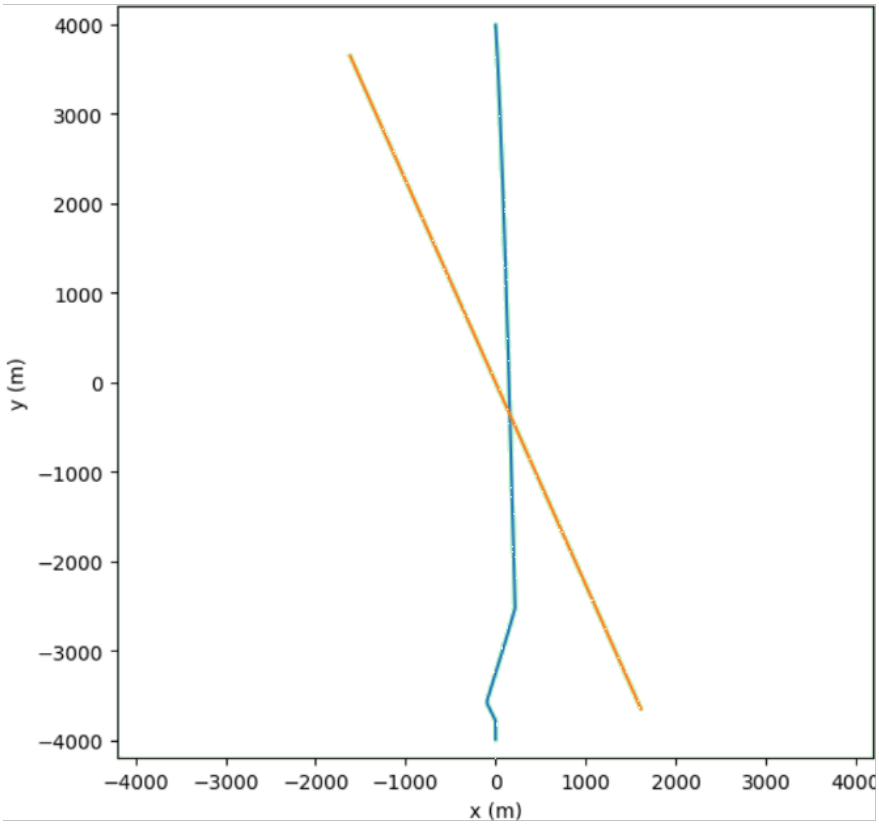



**Figure 9.**—First deconflicted scenario of epoch 50. The blue line represents the ownship vehicle's flight track and the orange represents the intruder's flight track.

# 4 Conclusions

This report proposes a simulation framework for training and evaluating Deep Reinforcement Learning (DRL) approaches for tactical separation assurance services in Unmanned Aircraft Systems (UAS) Traffic Management (UTM). To assess its functionality, the Double Deep Q-Network with experience replay, a DRL algorithm from literature, was implemented as the policy for the ownship agent, constrained to heading changes, in a pairwise UAS conflict scenario. The reward used to guide the learning process encouraged safety and efficiency, the latter in the form of energy consumption minimization.

Preliminary results from a dataset comprising 150 conflict scenario geometries, run for 50 epochs, demonstrate that the agent is successfully learning. This is evidenced by an increase in the percentage of deconflicted scenarios per epoch. Additionally, the safety performance of the policy has improved by deconflicting with the other UAS earlier in the flight and at greater distances from the conflict area.

# References


1. Prevot, T.; Rios, J.; Kopardekar, P.; III, J. E. R.; Johnson, M.; and Jung, J.: UAS Traffic Management (UTM) Concept of Operations to Safely Enable Low Altitude Flight Operations. *16th AIAA Aviation Technology, Integration, and Operations Conference*, 2016.

2. Rios, J. L.; Homola, J.; Craven, N.; Verma, P.; and Baskaran, V.: Strategic Deconfliction Performance: Results and Analysis from the NASA UTM Technical Capability Level 4 Demonstration. Technical Memorandum NASA/TM-2020-5006337, NASA Ames, August 2020.

3. Federal Aviation Administration NextGen Office: UTM ConOps Version2. March 2020. URL `https://www.faa.gov/sites/faa.gov/files/2022-08/UTM_ConOps_v2.pdf`, accessed on 05-05-2024.

4. Erzberger, H.; Paielli, R. A.; Isaacson, D. R.; and Eshow, M. M.: Conflict detection and resolution in the presence of prediction error. *1st USA/Europe Air Traffic Management R&D Seminar, Saclay, France*, Citeseer, 1997, pp. 17–20.

5. Johnson, M.; and Larrow, J.: UAS traffic management conflict management model. Technical Memorandum NASA/TM-2020-5002076, NASA Ames, May 2020.

6. Bilimoria, K.: A geometric optimization approach to aircraft conflict resolution. *18th Applied aerodynamics conference*, 2000, p. 4265.

7. Erzberger, H.; and Heere, K.: Algorithm and operational concept for resolving short-range conflicts. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 224, no. 2, 2010, pp. 225–243.

8. Brittain, M.; and Wei, P.: Autonomous air traffic controller: A deep multi-agent reinforcement learning approach. *arXiv preprint arXiv:1905.01303*, 2019.

9. Evans, A. D.; Egorov, M.; and Munn, S.: Fairness in decentralized strategic deconfliction in UTM. *AIAA Scitech 2020 Forum*, 2020, p. 2203.

10. Hunter, G.; and Wei, P.: Service-oriented separation assurance for small UAS traffic management. *2019 Integrated Communications, Navigation and Surveillance Conference (ICNS)*, IEEE, 2019, pp. 1–11.

11. Kochenderfer, M. J.; Holland, J. E.; and Chryssanthacopoulos, J. P.: Next generation airborne collision avoidance system. *Lincoln Laboratory Journal*, vol. 19, no. 1, 2012, pp. 17–33.

12. Kuchar, J.; and Drumm, A. C.: The traffic alert and collision avoidance system. *Lincoln laboratory journal*, vol. 16, no. 2, 2007, p. 277.

13. Kochenderfer, M. J.: *Decision making under uncertainty: theory and application*. MIT press, 2015.

14. Yang, X.; and Wei, P.: Scalable multi-agent computational guidance with separation assurance for autonomous urban air mobility. *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 8, 2020, pp. 1473–1486.

15. Owen, M. P.; Panken, A.; Moss, R.; Alvarez, L.; and Leeper, C.: ACAS Xu: Integrated Collision Avoidance and Detect and Avoid Capability for UAS. *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, 2019, pp. 1–10.

16. Owen, M. P.; Panken, A.; Moss, R.; Alvarez, L.; and Leeper, C.: ACAS Xu: Integrated collision avoidance and detect and avoid capability for UAS. *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, IEEE, 2019, pp. 1–10.

17. Brittain, M. W.; Alvarez, L. E.; and Breeden, K.: Improving autonomous separation assurance through distributed reinforcement learning with attention networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 21, 2024, pp. 22857–22863.

18. Sutton, R. S.; and Barto, A. G.: *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

19. Cone, C.; Owen, M.; Alvarez, L. E.; and Brittain, M. W.: Reward Function Optimization of a Deep Reinforcement Learning Collision Avoidance System. *AIAA SCITECH 2023 Forum*, 2023, p. 2155.

20. Li, S.; Egorov, M.; and Kochenderfer, M.: Optimizing collision avoidance in dense airspace using deep reinforcement learning. *arXiv preprint arXiv:1912.10146*, 2019.

21. Julian, K. D.; and Kochenderfer, M. J.: Reachability analysis for neural network aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, vol. 44, no. 6, 2021, pp. 1132–1142.

22. Mueller, E. R.; and Kochenderfer, M.: Multi-rotor aircraft collision avoidance using partially observable Markov decision processes. *AIAA Modeling and Simulation Technologies Conference*, 2016, p. 3673.

23. Fremond, R.; Xu, Y.; and Inalhan, G.: Application of an autonomous multi-agent system using Proximal Policy Optimisation for tactical deconfliction within the urban airspace. *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, IEEE, 2022, pp. 1–10.

24. Huang, C.; Petrunin, I.; and Tsourdos, A.: Strategic conflict management using recurrent multi-agent reinforcement learning for urban air mobility operations considering uncertainties. *Journal of Intelligent & Robotic Systems*, vol. 107, no. 2, 2023, p. 20.

25. Xie, Y.; Gardi, A.; and Sabatini, R.: Reinforcement learning-based flow management techniques for urban air mobility and dense low-altitude air traffic operations. *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, IEEE, 2021, pp. 1–10.

26. Yang, X.; Egorov, M.; Evans, A.; Munn, S.; and Wei, P.: Stress testing of UAS traffic management decision making systems. *AIAA AVIATION 2020 FORUM*, 2020, p. 2868.

27. Johnson, W.: *Helicopter theory*. Courier Corporation, 2012.

28. Johnson, W.: NDARC NASA design and analysis of rotorcraft. , NASA Ames, 2017.

29. Xue, M.; Rios, J.; Silva, J.; Zhu, Z.; and Ishihara, A. K.: Fe3: An evaluation tool for low-altitude air traffic operations. *2018 Aviation Technology, Integration, and Operations Conference*, 2018, p. 3848.

30. Pradeep, P.; Park, S. G.; and Wei, P.: Trajectory optimization of multirotor agricultural UAVs. *2018 IEEE Aerospace Conference*, IEEE, 2018, pp. 1–7.

31. Nickolls, J.; Buck, I.; Garland, M.; and Skadron, K.: Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, vol. 6, no. 2, 2008, pp. 40–53.

32. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019.

33. Bou, A.; Bettini, M.; Dittert, S.; Kumar, V.; Sodhani, S.; Yang, X.; Fabritiis, G. D.; and Moens, V.: TorchRL: A data-driven decision-making library for PyTorch. 2023.

34. Stevens, W. R.; and Narten, T.: UNIX network programming. *ACM SIGCOMM Computer Communication Review*, vol. 20, no. 2, 1990, pp. 8–9.

35. Ribeiro, M.; Ellerbroek, J.; and Hoekstra, J.: Distributed Conflict Resolution at High Traffic Densities with Reinforcement Learning. *Aerospace*, vol. 9, no. 9, 2022. URL `https://www.mdpi.com/2226-4310/9/9/472`.

36. Watkins, C. J.; and Dayan, P.: Q-learning. *Machine learning*, vol. 8, 1992, pp. 279–292.

37. Razzaghi, P.; Tabrizian, A.; Guo, W.; Chen, S.; Taye, A.; Thompson, E.; Bregeon, A.; Baheri, A.; and Wei, P.: A survey on reinforcement learning in aviation applications. *arXiv preprint arXiv:2211.02147*, 2022.

38. Hasselt, H. v.; Guez, A.; and Silver, D.: Deep reinforcement learning with double Q-Learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, AAAI Press, 2016, p. 2094–2100.

39. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D.: Human-level control through deep reinforcement learning. *Nature*, vol. 518, 2015, pp. 529–533. URL https://api.semanticscholar.org/CorpusID:205242740.

40. Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; Hassabis, D.; Clopath, C.; Kumaran, D.; and Hadsell, R.: Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, 2017, pp. 3521–3526. URL https://www.pnas.org/doi/abs/10.1073/pnas.1611835114.

41. Morales, M.: *Grokking Deep Reinforcement Learning*. Manning Publications, 2020. URL https://books.google.com/books?id=IpHJzAEACAAJ.

42. Hasselt, H.: Double Q-learning. *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds., Curran Associates, Inc., vol. 23, 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

43. Johnson, W.; Silva, C.; and Solis, E.: Concept Vehicles for VTOL Air Taxi Operations. *Conference on Aeromechanics Design for Transformative Vertical Flight, San Francisco, CA*, 2018.

44. Pradeep, P.; Kulkarni, C. S.; Chatterji, G. B.; and Lauderdale, T. A.: Parametric Study of State-of-Charge for an Electric Aircraft in Urban Air Mobility. *AIAA Aviation 2021 Forum*, 2021, p. 3181.

45. 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. *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds., 2016. URL http://arxiv.org/abs/1509.02971.