Curriculum-based Deep Reinforcement Learning for Autonomous AlphaDogFight Air-to-Air Combat*

Rana U. Riaz†
UNSW Canberra at ADFA.

AlphaDogfight Trials (ADT) sought to test the feasibility of Artificial Intelligence (AI) in piloting an F-16 for simulated air-to-air combat. The details of the most successful attempts in ADT are not available in the academic literature. I attempt to recreate the close-range visual air-combat problem defined in ADT. I use JSBSim, a high-fidelity open-source Flight Dynamics Model (FDM) simulation library that models the flight dynamics of an aerospace vehicle. I create a multi-aircraft environment to train agents using deep reinforcement learning (RL) and a rendering environment to visualise and test the agents' performance. The environment is then used to collect air-combat data from a human for training a neural network using an offline RL algorithm. The resultant model is able to compete and win against another forward-moving aircraft in a simulation environment. The results demonstrate the potential of human-guided offline RL in generating competent autonomous Air-to-Air Combat behaviours.

Keywords— Autonomous Maneuvers, Aircraft Autonomy, Air Combat, AlphaDogfight Trials, Curriculum Learning, Deep Reinforcement Learning

I. Introduction

Advances in Artificial Intelligence (AI) in real-time strategy games (Berner *et al.*, 2019, Silver *et al.*, 2016, 2017, Vinyals *et al.*, 2019) and many real-world applications such as autonomous vehicles (Alizadeh *et al.*, 2019, Bouton *et al.*, 2019, Isele *et al.*, 2018, Zhu *et al.*, 2020) have raised the ambition to question whether AI is ready for more complex real-world uses. One domain in particular is the air domain which is characterised by its own unique set of challenges, where agents operate in 3-D physical space at high speed. The mere idea of having a fully autonomous fighter-jet is not new, but the real challenge resides in the question: which complex autonomous behaviours can an AI-enabled fighter-jet perform effectively?

Air force pilots train for years to reach an expert level in air combat, as it is a complex domain including many different facets. The operational requirements of the Air Force warrant a high-performance level resulting in a higher chance of fatality. Pamplona & Alves (2020) conducted a risk analysis of the US Air Force fleet and concluded that military aviation is riskier than commercial aviation with some aircraft having more than a 20% probability of one fatality. This high risk is warranted by the hostile environment and adversaries' capabilities (Ball, 2003). Almost 80% of aircraft crashes are due to human error (Wiegmann & Shappell, 2017). In military aviation, the human element is magnified due to the gravity-induced loss of consciousness (g-LOC) (Burton, 1988). In aggressive manoeuvres, upwards of 9g acceleration, pilots tend to lose consciousness, which has been among the leading causes of aviation mishaps. This phenomenon puts the load of, and expectations from, human pilots at their extreme. Such an environment has escalated the demands on increased autonomy in the air combat domain to reduce the risks of fatality to human pilots, and to push the bounds of performance.

There are automated systems that can automate some functions and facets of flights without the need for human intervention. For example, automation exists to maintain the altitude of an aircraft, and to autonomously fly to a point or to perform a landing (Wiener, 1988). These systems have helped reduce pilots' workload and are indispensable to modern flying. In the same spirit, the United States Defense Advanced Research Projects Agency (DARPA) launched the Air Combat Evolution (ACE) program to analyse potential applications of autonomy in the air combat domain in a bid to decrease risk to human pilots and improve performance.

ACE's AlphaDogfight trials (ADT) grapple with the problem of visual range combat using only guns with an effective range of 3000 feet. These short-range fights are done by tracking the adversary through the cockpit glass

*The work from this project is currently under review by a specialised journal in the field with an impact factor > 3.
†Undergraduate Student, School of Engineering and Information Technology

rather than using radar technology. The visual range lethal interaction is challenging because trained pilots fly against each other in powerful aircraft with fast-paced manoeuvres, which begs for high-performance requirements for decision-making in a split of a second. The costs associated with the wrong decision are very high, both from a human life perspective and from the cost of the platform itself. The ADT competition showcased that Heron Systems' AI agent call-sign Falco defeated the expert USAF pilot 5-0. The match was broadcasted live on Youtube (DARPAtv, 2020). This experience has motivated the need to further explore autonomy in the fighter aircraft control domain.

There is limited transparency regarding the algorithms and techniques used by the teams in the ADT competition. Some teams commented in the videos on their use of reinforcement learning. However the algorithms and training frameworks have not been shared in the public domain. Only Lockheed-Martin published a paper based on their work in ADT (Pope *et al.*, 2021) outlining how their agent was trained using self-play with hierarchical reinforcement learning. The above high-end technological advances in ADT have demonstrated the gap between the academic research and these trials.

This paper attempts to close this gap. I rely on the descriptions from ADT to develop an environment similar to visual range air combat and design a solution for ADT using deep reinforcement learning in a simulation environment using offline RL and reward shaping to improve sample efficiency on the task. The paper demonstrates the challenges in designing this level of autonomy and possible solutions to these challenges. In particular, I

- formulate the problem as a competitive 1-vs-1 multi-agent reinforcement learning problem (Section III),
- recreate the simulation environment used in AlphaDogfight trials where each agent can learn autonomously using RL or using human guidance (Section IV.A),
- develop a rendering component to visualise the agents' behaviours for diagnosing and testing their performance (Section IV.B),
- create a high-speed training pipeline using asynchronous RL to enable autonomous learning (Section IV.C),
- collect 20 hours of human data using the simulation environment to enable learning from demonstrations, and
- validate the usability of the environment in both RL and human-guided learning settings and evaluate the resulting agent performance under these settings (Section V).

II. Background

The background section is structured into four sub-sections. I commence the discussion with an analysis of aircraft dynamics and the basics for computational aircraft simulations in Subsection II.A. Air combat and ADT are then discussed in Subsection II.B. The basic rundown of reinforcement learning is presented in Subsection II.C followed by the multi-agent RL in subsection II.D.

A. Aircraft Dynamics and Control

The differential force between gravity and the lift generated by the interaction of an aircraft body with air at high speed causes an aircraft to fly. Figure 1 shows the four forces required to balance an aircraft during a flight. The aerodynamic shape of the wings is the biggest contributor to lift (upward force), which depends on the speed of the flow above and below the wings and can be altered by introducing disturbance to the flow. The control of aircraft is thus achieved by modifying the shape of the aircraft using control surfaces shown in red, blue and green in the figure below. The elevators on the tail of the aircraft shown in green are moved together to control the aircraft's nose up or down (pitch). Ailerons are moved in the opposite direction to roll and the rudder is used to control the movement about the z-axis (yaw). For a more detailed description, the reader is referred to Anderson & Bowden (2005).

Simulating a physical system relies on mathematical models of the system that can predict the next state of the system given the current state and the input. These mathematical models capture the system's equations of motion. The motion of aircraft is modelled using non-linear equations around the earth (Stevens *et al.*, 2015). This is a six-degrees of freedom (DoF) system; thus, it forms at least six equations from Newton's 2^{nd} law. There are many variants of models for aircraft dynamics in order of increasing complexity and fidelity, and the choice of a simulator should be balanced by the fidelity required in the application and the computing resources available to execute the model.

The equations of motion for control problems are written in a matrix form as $\dot{x}(t) = Ax + Bu$ where x is the current state of the system, u is the control input, A and B are given by the system dynamics. The size and coefficients of matrices A and B are determined based on x and u. The larger the size, the more equations required to simulate the system. For an aircraft, the state vector x comprises the linear displacements $(latitude, longitude, altitude)$, rotational displacements or Euler angles (θ, ϕ, ψ), linear velocities (u, v, w), rotational velocities (p, q, r) and linear and rotational accelerations. The control inputs for aircraft include $u = [\delta_{aileron}, \delta_{elevator}, \delta_{rudder}, \delta_{throttle}]$ corresponding to the deflection of control surfaces (aileron, elevator, rudder), and the throttle controls engine power output. The coefficients of matrices A and B depend on the aircraft geometry, environmental conditions and the altitude, and

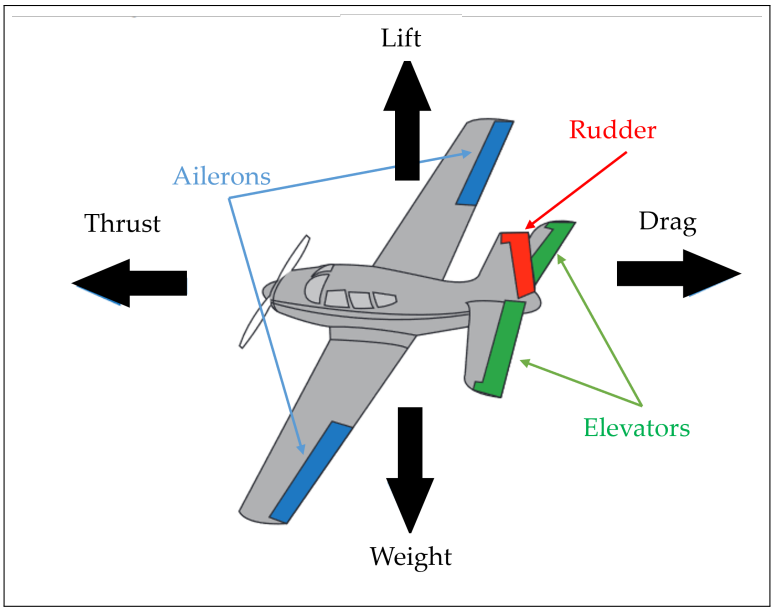


Figure 1: The aircraft flight in motion has four forces acting on it which can be varied using control surfaces.

are calculated for each aircraft in wind tunnels and actual flight tests. This data is needed for designing a simulated version of a physical aircraft around various flight conditions to model the aircraft to be as close as possible to its behaviour in the real-world.

B. Simulating Air Combat

There has been much research on automating air combat at various times, the modelling of aircraft dynamics and an intelligent controller are the main components of such work. Dong *et al.* (2019) show that the intelligent controller has usually been done using three main approaches: mathematical solutions (Breakwell & Merz, 1977, Meier, 1969, Merz, 1975, Virtanen *et al.*, 2006), knowledge-encoded (Austin *et al.*, 1987, Burgin, 1986, Burgin *et al.*, 1975, Burgin & Owens, 1975, Burgin & Sidor, 1988, GOODRICH & MCMANUS, 1989, Lee *et al.*, 2017), and learning-based (Akbari & Menhaj, 2001, Karli *et al.*, 2017, Liu & Ma, 2017, McMahon, 1990, Teng *et al.*, 2012). Isaacs (1951, 1955) represents a mathematical approach where air combat was modelled as a pursuit-evasion game. Subsequent works improved the models but many unrealistic assumptions (2D space, instant turning) limited their practical utility. Knowledge-encoded approaches emerged mainly through NASA's work on adaptive manoeuvring logic (Burgin & Sidor, 1988); the required feature creation by fighter pilots was high while the generalisation still remained low. McGrew (2008), McGrew *et al.* (2010) applied the air combat problem to unmanned autonomous systems by using value iteration and dynamic programming to learn a 2-dimensional flight trajectory and validated this approach in an indoor environment, while the effect of altitude change was ignored.

There has been a trend of working towards autonomy in air combat, but most works have simplified the simulation environments. This was changed by DARPA's AlphaDogfight trials in 2020, which used a full non-linear dynamics model in an attempt to be very close to the real world. AlphaDogfight trials invited eight teams from industry and academia to bring their best models to solve the visual range air combat problem and to showcase the state-of-the-art in artificial intelligence research in the air combat domain. The problem and simulation environment were defined by John-Hopkins APL in collaboration with DARPA. The provided environment was using the JSBSim flight dynamics modelling (FDM) library as the simulation backend. JSBSim is an open-source FDM that models the aircraft's motion with six-DoF non-linear equations of motion. This library does not have a rendering engine and the detail of rendering used by APL has not been provided.

The competition, which occurred in Aug 2020, comprised three rounds, and it was aired online on YouTube (DARPAtv, 2020). In round one, the teams fought simple agents provided by APL. The second round was a round-robin tournament among the eight competitors. The Heron systems' agent Falco won the first two rounds. In round three, the agent Falco beat an expert pilot from USAF's weapons training school. Lockheed's agent PHANG-MAN positioned second in the competition and was also able to beat the expert human pilot 5-0 (Pope *et al.*, 2021). They used hierarchical reinforcement learning to train the agent. Higher-level policy chose between three low-level policies, which perform dedicated tasks like control zone, aggressive shooter and conservative shooter. They use a reward function based on six different parameters, and a self-play architecture was used to train the agent.

This competition had some assumptions that should be highlighted. The human agent played against the AI-enabled agents using a virtual reality (VR) system, where the effects of the dynamics in a real-world environment on the human's body cannot be represented with sufficient fidelity to match the real-world, and hence the pilot cannot efficiently use his/her experience and intuition based on the sense of gravity. The RL agents had the perfect state of the environment which includes own aircraft position, velocities, acceleration, orientation, health and opponent's position and orientation. While there are systems available that calculate relatively correct own aircraft properties, there are usually delays in all aircraft instruments. The information about the intruder aircraft is a bit tricky because the current radar-based systems are not as accurate as the information provided in this experiment, and success actually depends upon the pilot's ability to predict opponent's trajectory based on experience to attempt to out-manoeuvre the opponent.

C. Reinforcement Learning

Air combat is a control problem where the next action u is conditioned on the current state x_t, and the next state x_{t+1} emerges from the system dynamics of A and B. This control problem has been frequently modelled as a Markov Decision Process (MDP) (Baron & Kleinman, 1971, Feigin *et al.*, 1984). The latter is defined as a tuple $M = (S, A, T, d_o, r, \gamma)$ where S denotes the state space, A the available control actions, T the transition dynamics analogous to A and B, d_0 the initial state distribution, r the reward, and γ a discount factor. This formulation of MDP to learn a policy $\pi(a_t|s_t)$ forms the basis of reinforcement learning. The learning is achieved by collecting a trajectory distribution p_π in a horizon of length H and optimising the objective function $J(\pi)$. The reader is referred to (François-Lavet *et al.*, 2018, Sutton & Barto, 2018) for a detailed description of reinforcement learning.

$$p_\pi(\tau) = d_0(s_0) \prod_{t=0}^{H} \pi(a_t|s_t) T(s_{t+1}|s_t, a_t) \tag{1}$$

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} [\sum_{t=0}^{H} \gamma^t r(s_t, a_t)] \tag{2}$$

The objective function in Equation 2 is then optimised to find a policy. The key challenges include: 1. trajectories should be representative of the overall state space, but problems with large state spaces suffer from the curse of dimensionality. 2. the distribution should have enough high reward scenarios; reward sparsity 3. Markov chain defined by $\pi(a_t|s_t)T(s_{t+1}|s_t, a_t)$ assumes a stationary distribution - the same actions from same states achieve same next state- however, this assumption is violated in the multi-agent settings. 4. Countering reward sparsity requires supplementing the true reward with reward shaping. However, that has an undesired effect of value misalignment (Amodei *et al.*, 2016).

Various reinforcement learning algorithms have been proposed successfully to solve difficult control problems in games (Mnih *et al.*, 2013), critical infrastructure (Degrave *et al.*, 2022) and robotic autonomy (La *et al.*, 2014). This has prompted the use of reinforcement learning in a complex domain of air combat. However, the challenges discussed above are magnified.

D. Multi-Agent Reinforcement Learning

This problem has two aircraft in the environment, so the MDP formulation needs to be extended. The extensions in the literature depend on the synchronisation and the information difference between the agents are: multi-agent MDPs (Boutilier, 1996), Dec-POMDPs (Bernstein *et al.*, 2002), and partially observable stochastic ("Markov") games (Shapley, 1953). Shapley's partially observable stochastic Markov game requires all agents to step simultaneously, observe together and receive a reward together. This formulation is close to our scenario of dogfighting and hence I model the environment as a Markov game, as shown in Figure 2. The agents in this environment have identical attributes and step simultaneously.

Training RL agents in a multi-agent setting require a special treatment due to the opponent's induced complexity. For example, games with relatively simple rules, such as chess, can be very hard to play and excel because the complexity of a round depends also on the adversary's strategy. The opponent can use very advanced strategies to make the environment significantly complex. This level of non-stationarity and complexity introduced by opponent needs to be carefully managed in a training system. There has been some treatment of this effect in game theory. Nash Jr (1950) theorised that two-player zero-sum games converge to an equilibrium point, the best policy agents can have is in fact the Nash equilibrium (Bowling & Veloso, 2001) from which agents would be disadvantaged if they deviate. Training against deterministic agents, self-play (Brown, 1951), fictitious self-play (Heinrich *et al.*, 2015) and league-based play (Vinyals *et al.*, 2019) have been used in order of increasing complexity and generalisation.

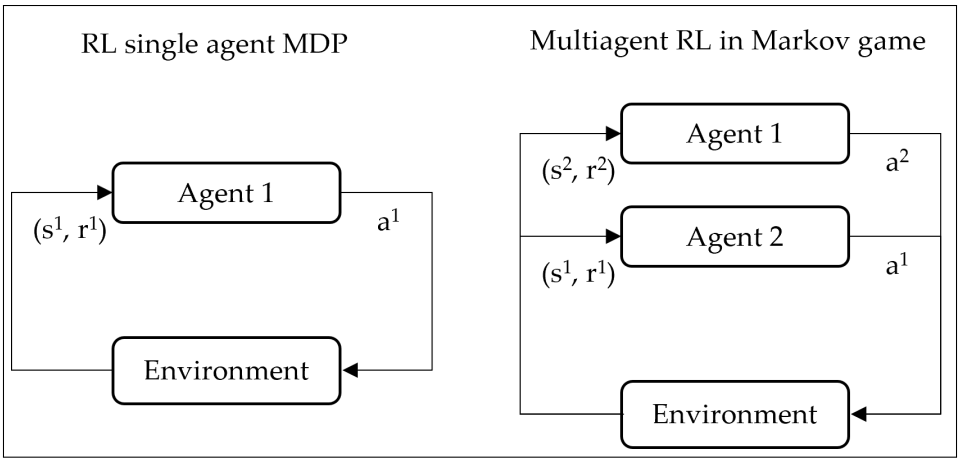


Figure 2: The framework of reinforcement learning as a Markov decision process, and multi-agent as a Markov game

III. Problem Definition

I model the air-to-air combat with two aircraft deployed in an open environment. Each aircraft is given a state information vector S which contains aircraft's geographical position(latitude, longitude, altitude) , Euler angles (θ, ϕ, ψ), linear velocities (u, v, w), angular velocities (p, q, r), accelerations $(\dot{u}, \dot{v}, \dot{w}, \dot{p}, \dot{q}, \dot{r})$, aerodynamic angles (α, β), current surface deflections, and fuel load(f_l) of own aircraft along-with the position and velocities of the opponent aircraft.

$$\begin{aligned} S = [&OWN(latitude, longitude, altitude, \theta, \phi, \psi, u, v, w, p, q, r, \\ &\dot{u}, \dot{v}, \dot{w}, \dot{p}, \dot{q}, \dot{r}, \alpha, \beta, \delta_{left\ aileron}, \delta_{right\ aileron} \delta_{elevator}, \delta_{rudder}, \delta_{throttle}, f_l, Health), \\ &OPPONENT(latitude, longitude, altitude, \theta, \phi, \psi, u, v, w, p, q, r, Health)] \end{aligned} \tag{3}$$

Using this state information, each aircraft should output the control vector U to be used as input to the F-16s flight control system at a maximum of 50Hz frequency.

$$U = [\delta_{aileron}, \delta_{elevator}, \delta_{rudder}, \delta_{throttle}] \tag{4}$$

An agent should maintain its adversary within its weapons engagement zone (WEZ) as shown in Figure 3. The WEZ is defined by a 2° cone from the nose of the aircraft, and it extends out from 500 ft – 3000 ft. When the target is within this WEZ, it is said to be locked by the agent and can also be called a "gun snap". If an aircraft is within the WEZ of the other aircraft, the damage per second is given by the formula:

$$d_{WEZ} = \begin{cases} 0 & r < 500ft \\ \frac{3000-r}{2500} & 500 \leq r \leq 3000 \\ 0 & r > 3000ft \end{cases} \tag{5}$$

A single episode represents a single engagement and starts with both aircraft having full health of value one. The health is then decreased by multiplying damage per second by the time an agent stayed in the WEZ of its adversary. The engagement terminates if (1) the duration of the simulation reaches 300 seconds, or (2) one of the agents wins. An agent wins when its opponent reaches zero health or falls below a hard deck (minimum altitude) of 1000 feet.

IV. Methodology

Reinforcement learning aims to train an agent on a given task by maximising the accumulated reward function that encapsulates the task objective. In this paper, the objective of an agent (aircraft) is to dominate the opponent's aircraft by decreasing its health as defined in the previous section. Before learning this task, it is necessary to create the system that will be used for the training and evaluation. Figure 4 shows the proposed system architecture, which consists of the multi-agent environment, the two aircraft agents, and the rendering component. The multi-agent environment defines the environment rules and incorporates and synchronises the aircraft dynamics model. An agent represents the controller used to generate the control input U for its aircraft. Three types of agents are used in this

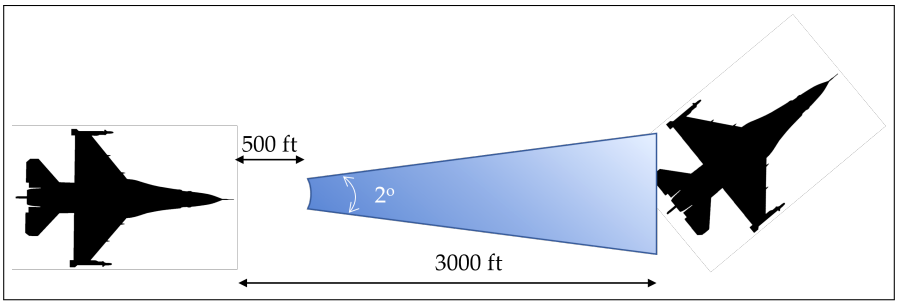


Figure 3: Weapon engagement zone (WEZ) for the aircraft gun

paper: rule-based, RL-based, or human controlled. The environment needs to support very high training speeds to enable collecting huge amounts of training experience, similar to those used by the successful agents in ADT (Pope *et al.*, 2021). This is achieved by using asynchronous training architecture. In addition to autonomous RL, supporting human-guided learning is desirable in such a high-complexity domain. This is enabled by allowing a human to take control over one of the agents such that human control data can be later used by the agent to learn an autonomous policy. The rest of this section goes into the details for each system components.

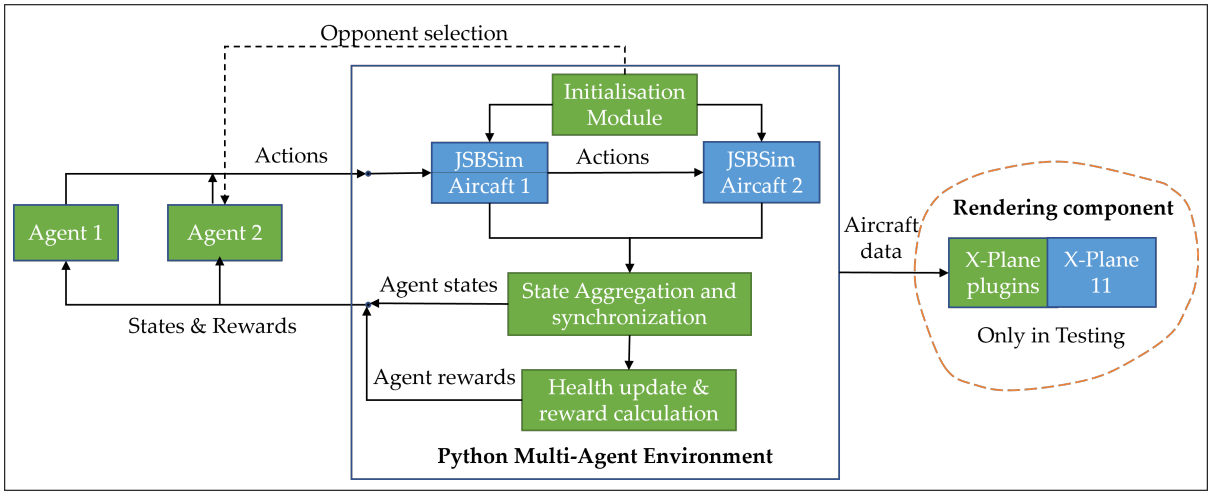


Figure 4: The system overview

A. Python Multi-agent Environment

The air combat problem includes two aircraft agents with identical state space topologies, action space and reward function. The action space is made of the deflection of control surfaces (rudder, elevator and aileron) and the throttle control, as described in Equation 4. The control actions are continuous variables with surface deflections in the range [-1, 1], and throttle in [0, 1]. The state, defined in equation 3, consists of 40 continuous variables forming the observation space of an agent, which I transform into a smaller space for more efficient use of resources and algorithmic efficiency.

This work models the multi-agent problem as a Markov game, so both agents step simultaneously and receive the next state and reward at the same time. The environment is implemented as a Python class based on PettingZoo's simultaneous environment structure (Terry *et al.*, 2021), and consists of the following components: an initialisation module, flight dynamics models for both aircraft, a state and timestep matching module, and a health calculation module that estimates the health and reward for each agent. This is a competitive multi-agent setting of a partially observable stochastic game. The agents are modelled as decentralised with no parameter sharing to maintain true competition in the environment. I abstract the task from the environment based on the approach in Rennie (2018), which was designed for single-agent systems. The opponent aircraft is not learning simultaneously; the opponent agent can be using its existing policy or a deterministic motion. It can be chosen from a pool of agents at the start of each episode which can facilitate fictitious self-play and league-based training (Vinyals *et al.*, 2019).

1. *Initialisation module*

The initialisation module resets the state of both agents at the start of each episode. The agents start at a randomly chosen altitude in the range of 5000 to 15000 feet. The initial distance between the agents is set randomly between 1500 ft to 3000 ft with their noses facing in opposite directions. This setting is shown in Figure 5. Both agents start with the same initial velocity randomly chosen between 300 to 400 ft/s. An agent should learn to undertake a 180 degrees turn to try to attack the opponent.

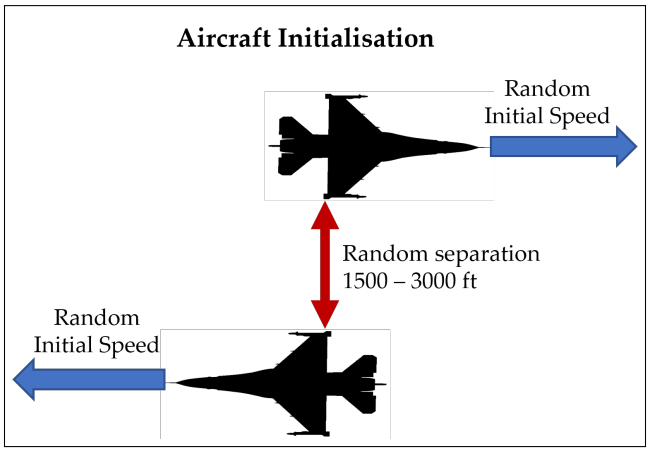


Figure 5: The task initialisation

Initialisation module offers the opportunity to choose an agent with different targets (different rewards), different architectures and training algorithms. The initial conditions for altitude, distance, speeds and directions can be varied to provide flexibility during the training phase. This initialisation module also instantiates two aircraft objects with JSBSim physics backend on their appropriate location.

2. *Flight Dynamics Modelling*

Problems that model real-world physical systems require a physics engine to simulate the real-world. In Aerospace, these physics engines are called flight dynamics modelling(FDM) software. The key requirements for FDM to be suitable for reinforcement learning are to have sufficient fidelity to model the problem at hand, and to be computationally efficient. The latter refers to the computational running speed of simulation of the FDM. The complexity of the problem results in the need to collect a lot of experience. The winning agent in ADT trained on 31 years of continuous data. Therefore, computational efficiency is an important limiting factor. The neural networks have been quite optimised using GPUs. In many reinforcement learning problems, the simulators become the bottleneck, which is magnified by the aircraft 6-DoF non-linear equations as they require more time to calculate. The high-fidelity issue has been explained in the previous discussion in section II; hence I am conscious of the need for a simulation environment that closely replicates the real environment. The FDM should therefore use at least the non-linear 6DoF equations of motion and have a small timestep on the order of at least 100ms. With these minimum fidelity requirements, the simulation should be considerably faster than in real-time, preferably also having an option to run multiple environments in parallel to further speed up the training.

Addressing these requirements, the JSBSim flight dynamics library (Berndt, 2004) has been selected as the FDM. JSBSim uses spherical rotating earth equations of motion to simulate the aircraft dynamics as a 6-DoF non-linear model. The flight model was tested by NASA and they reported that JSBSim calculated the aircraft dynamics with high accuracy (NASA, 2015), it has also been used in commercial flight simulators such as flight Gear and Outerra. JSBSim is reasonably efficient and high fidelity. Using a state calculation frequency of 100Hz, and eight instances of JSBSim, a simulation of 200 times faster than real-world can be achieved on a 12 Core laptop with a 16GB RAM. JSBSim thus is scalable and reasonably fast for this application. However, JSBSim is more geared towards single aircraft applications and not artificial intelligence agents training. So, a significant amount of work is required to make a training pipeline that communicates with this FDM. Additionally, no rendering environment is available with it. Therefore, the selection and integration of independent rendering options need to be explored.

The second aspect related to modelling flight dynamics is the choice of aircraft. A realistic model for a fighter jet is needed. The flight control system, weight and balance, aircraft geometry, and aerodynamic coefficients have significant effects on aircraft behaviour during flight. A good model reflects all these appropriately. Choosing an aircraft then requires that such data is available. For military aircraft, it can be challenging to acquire and validate this data. Nguyen *et al.* (1979) have provided data for F16 aircraft in a technical report from NASA. An F16 model created from this data is open-source and available with JSBSim. The quality of this model was tested in NASA's

verification of the JSBSim flight model (NASA, 2015) and further evidenced by the model's use by expert pilots in DARPA's AlphaDogfight Trials.

3. State and health calculation

The environment is coded with maximum flexibility in mind; this is required to improve the agents'learning. The state can be aggregated in any way that seems desirable for better RL training. For example, rather than providing raw values of the 40 available state variables, relative position and velocity information can be used to decrease the state space size to 26 dimensions, which can have a significant improvement in learning. Similarly, varying pseudo-rewards can be provided in the training to mitigate reward sparsity.

Algorithm 1 Pseudocode for a training episode in environment

```
Instantiate an environment with aircraft
Choose the opponent
Link learning agent to aircraft and a policy to opponent
Initiate FDM for each aircraft
Set initial conditions for aircraft
Get the observations from the simulator and provide them to associated agents
repeat
    Get next actions for the agents
    Pass the actions to the FDM for each agent
    Get the new observations from associated simulators
    Aggregate and transform the state information
    check if the state is terminal
    Calculated rewards described in tasks
    Store the experience to the buffer
    Perform a training step on minibatch from the buffer
until In terminal state
```

The environment contains two aircraft simulated in two different JSBSim simulations. Thus, the environment needs to combine the information from these FDM simulations when constructing the state vector of each agent. The environment queries the FDMs for their state data, and then matches the simulation time of the two aircraft. Once the state of both aircraft is known to the environment, it calculates the health of both aircraft and adds it to the state. Health is not a property of the dynamics of the aircraft and has not been modelled in the FDM so the calculation is done based on the time aircraft spends in the adversary's WEZ. The WEZ is modelled as 2° cone extending out from the nose of the agent. The environment then provides an option if transformation of the state data is required before feeding it to the opponent. The reward is also calculated based on the state. The reward and state are then provided to the reinforcement learning agent, which then provides an action for the agent. The environment passes the actions to the aircraft. The pseudo code for a training run is shown in Algorithm 1.

B. Environment Rendering

Visualisation is an essential part of simulating physical systems. We need to visualise the agent's performance to monitor and evaluate their behaviour and adjust as required. JSBSim does not have a built-in rendering component, so I needed to integrate another tool or software to visualise the flight behaviour of the agents. I have chosen X-Plane 11 (Laminar Research, 2021) for this purpose. X-Plane 11 is a commercial flight simulator that is widely used by pilots and engineers in testing aircraft. The X-Plane simulator is versatile software with good rendering and simulation capabilities, but it has not been chosen as an FDM in training the system because it is very slow compared to JSBSim. This meant I still needed to use JSBSim as a backend FDM and visualise its data in X-Plane. This required sharing of data from the environment to X-Plane.

X-Plane 11 is a closed-source software. However, it is extensible, using the provided software development kit (SDK) to develop plugins. Plugins can be used to significantly enhance the simulator as per the user's need. I have developed a plugin that is capable of sending data to and from X-Plane, which was required as I am using an external FDM, essentially providing a python interface to X-Plane 11. The X-Plane plugin listens for information on a user datagram protocol (UDP) port. The plugin then shares the data with the X-Plane, which draws the aircraft on-screen. Figure 6 shows the two aircraft in simulation. Moreover, X-Plane is heavily geared towards single aircraft and most default renderings settings favour a first-person view of the aircraft. In this case, I needed to support both the first-person view and the console view. The console view is required to visualise the two aircraft in the same screen to understand their relative performance in the ADT. For this reason, I added some camera manipulations to

the plugin; first: I calculate the mid-point between the two agents, and then focus the camera on that point from above. The camera position is bound to the mouse so I can adjust the viewport accordingly.

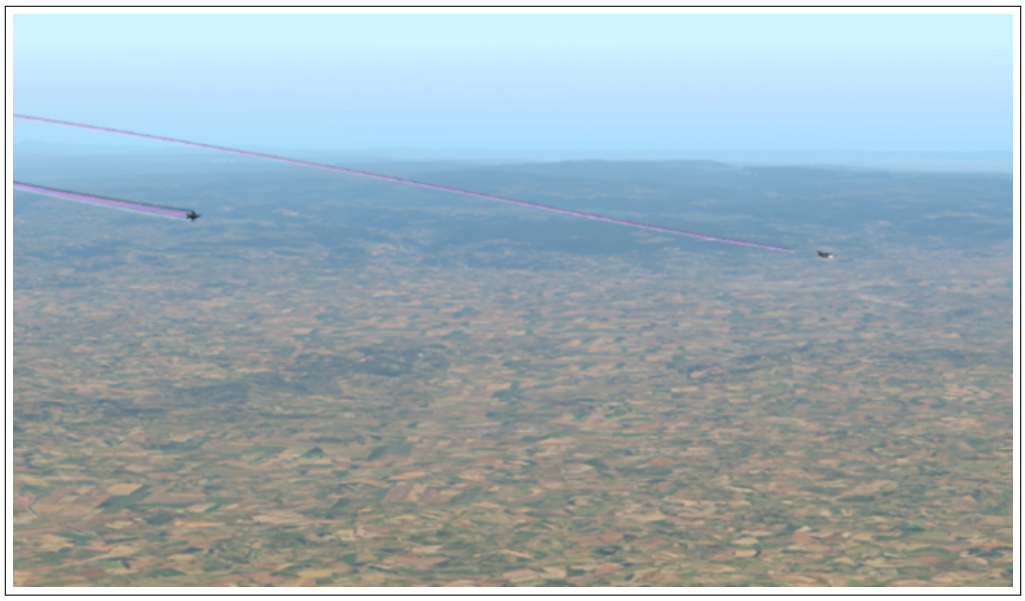


Figure 6: The rendering of the environment

C. Scaling the Training using Asynchronous PPO Algorithm

Air combat is quite complex; an agent needs enormous amounts of training samples to converge to an acceptable policy. This is evidenced by the fact that the main agent of Heron systems took over 31 years of real-world flight time. Collecting experience, however, is expensive because the FDM solves multiple non-linear equations in every timestep. Both preceding statements suggest high computational requirements and supporting training infrastructure to parallelise a complex environment. The training environment described so far was slow compared to the required training speed, as it cannot provide enough data for the agent to learn in a reasonable amount of time. The simulation environment can be scaled up with various methods in reinforcement learning.

The most common method is to create similar instances of the environment in various subprocesses and collate the data and send it to a single learner. Most available open-source libraries support multiprocessing (e.g., (Raffin *et al.*, 2021, Weng *et al.*, 2021) etc.). However, the traditional multiprocessing approach is limited by the number of processor cores on the machine. This does not scale very well because all the cores are exhausted by simulating the environment, sending data, and sampling new trajectories simultaneously and the slowest environment can make others wait; this results in a lot of context switching and decreased performance due to waiting. The asynchronous approach from networking has been proposed to overcome this challenge where, we dedicate most CPU cores to simulating the environments and some cores to outputting the next actions with the given policy (Petrenko *et al.*, 2020, Weng *et al.*, 2022). The environments then save the trajectories in a buffer. This buffer is read asynchronously by the learner on GPU using faster-FIFO (Petrenko & Kumar, 2020) C++ based queue system. The learner does not have to wait for simulation and can learn continuously as shown in Figure 7. This provides a significant speedup in training.

The speedup using asynchronous architecture however needs to be done carefully because the policy used to collect the data lags behind the current policy on learner and the performance of RL algorithms suffer if we are collecting data with an old policy (Babaeizadeh *et al.*, 2016). This significantly impacts the stability of training and the sample efficiency of the algorithm. Hence, we can not merely use many cores and have a linear decrease of training time while maintaining performance. For this reason, Espeholt *et al.* (2018) proposed V-trace loss that reduces the tendency of the current policy update to diverge dramatically from the policy used for sampling trajectories, which helps maintain the sample efficiency and training stability.

The reinforcement learning algorithm tested on this task was the asynchronous PPO (Petrenko *et al.*, 2020) which is the addition of V-trace loss to the proximal policy optimisation algorithm (Schulman *et al.*, 2017). PPO is an on-policy algorithm that collects experience based on the current policy and performs an optimisation step on fresh experience. Table 1 shows the experimental results of using a single training environment, multiprocessing with 16 training environments, and asynchronous training with 512 environments. Comparing the three approaches in benchmark Atari environments as well as in the proposed ADT environment, it is evident that asynchronous training

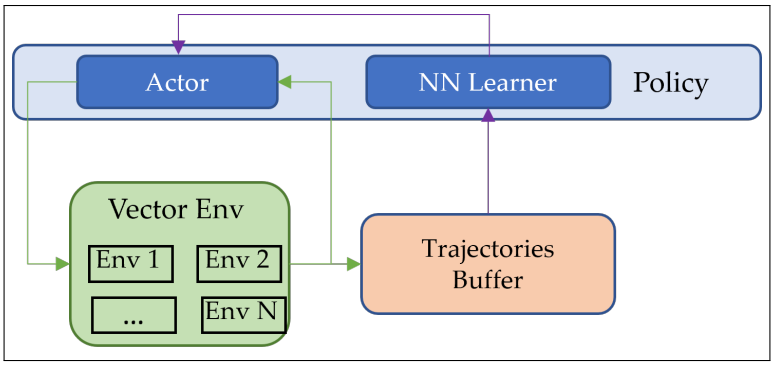


Figure 7: The asynchronous PPO architecture

speeds up the simulation significantly.

Table 1: Environment speed of simulation (expressed in steps/second) using different training settings.

Option	**Atari Breakout**	**Our ADT environment**	**Days for 30 years of real-world experience**
Single Env	600	215	510
Subprocess 16 Envs	4000	1450	76
Async 512 Envs	100000	9500	11.5

V. Experimental Setup and Results

The environment and asynchronous PPO explained in the previous section are used for training the agent on the task. The setting is shown in Figure 5, with aircraft facing away from each other, at random altitudes and distances, with the true reward function given in ADT. However, the complexity of the task was soon evident, and the agent could not improve its performance during training. Upon further investigation, reward sparsity is found to be the main reason for this behaviour. In its initial flight time, the agent could spend years of real-world flight and only find the opponent a handful of times because the physical world is quite vast. Significantly damaging the opponent required the agent to maintain a direction for a while, which was very rare to have happened during exploration. This led us to consider two pathways to enable the learning. The first pathway uses reward shaping and curriculum learning, whereas the second pathway uses human-guidance to allow for learning from demonstrations.

A. Reinforcement Learning with Reward Shaping and Curriculum Learning

The initial experiments highlighted the importance of reward shaping to tackle reward sparsity. I wanted the agent to win quickly, cause more damage to its adversary without crashing. The reward shaping was done by adding reward for closure $R_{closure}$ (getting closer to the adversary), track angle R_{track_θ} (angle between own nose and the centre of opponent aircraft), gunsnap $R_{gunsnap}$ (keeping the opponent in the two degrees nose cone regardless of relative distance),decreased health R_{health}, crashing R_{crash} and winning the engagement R_{win}.

Table 2: Reward structure

Element	**Weighting**	**Conditions**	**Received at step**
Closure	-0.003 * relative distance	Nil	every step
Track angle	-0.001 * θ	$0 \leq \theta \leq 180$	every step
Gunsnap	0.05	if in 2 degrees cone	every step
Health	$h_{t-1} - h_t$	Nil	every step
Crash	-10	$if altitude \leq 1000ft$	terminal
Win	15	$if opponenthealth \leq 0$	terminal

Another promising avenue of progress is the curriculum learning approach (Narvekar *et al.*, 2020). This approach is based on the ubiquitous curricula in human development and education. Many studies have shown the usefulness of this idea by sequencing tasks in a simple-to-complex curriculum in order for RL agents to learn on a problem that is otherwise too complex to learn from scratch (Narvekar *et al.*, 2020). I changed the initialisation scheme to create the task as a curriculum-based task. Figure 8 below shows the new initialisation; the opponent moves in a straight line and within the weapon engagement zone of the learning agent.

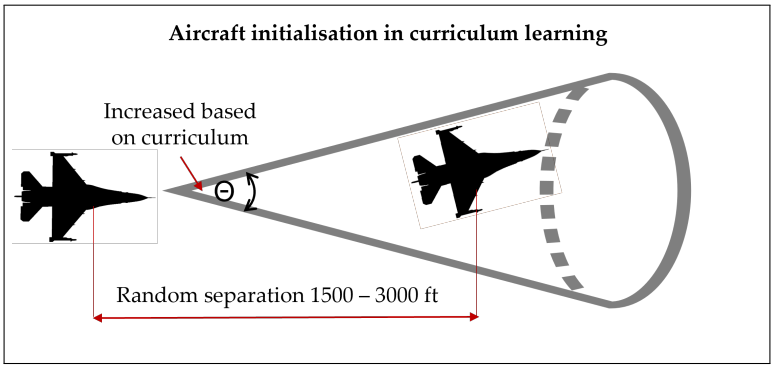


Figure 8: The simplified initialisation scheme

This new setting provided denser rewards to the agent and the agent learned using the same reward structure as defined before. The neural network that learned from this reward had six fully connected layers and four LSTM recurrent layers of 1024 hidden units each. This large neural network was selected to match the complexity of the problem. The recurrent layers were especially helpful to the agents' learning because this is a physical problem and the state in the current step is heavily dependent on more than one previous states. These layers serve as the agents' memory. The plot in Figure 9 shows the training reward of the agent in this setting.

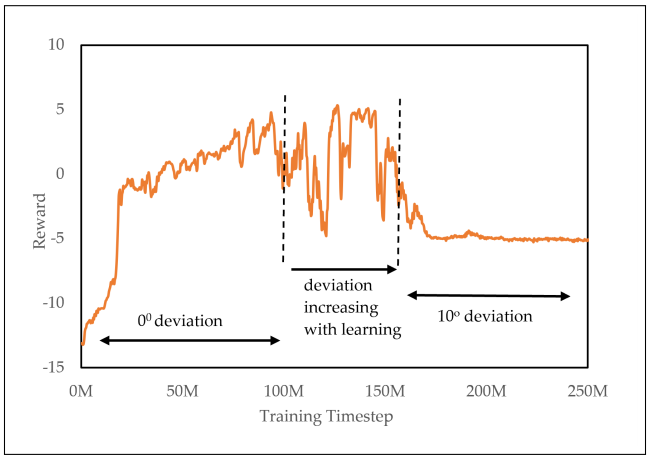


Figure 9: The training reward of the agent in the simpler setting

Once the agent showed signs of improvement, I started randomising the opponent's initial position at the face of the cone with the apex on the nose of the aircraft (maximum deviation). The maximum deviation from this position increased slowly as the agent kept learning. After ten degrees maximum deviation, the reward started to decrease, showing that the current approach is not providing more useful data and improvement in learning. This is because the environment was becoming too difficult to be learnt autonomously, and I started exploring options to learn from demonstrations.

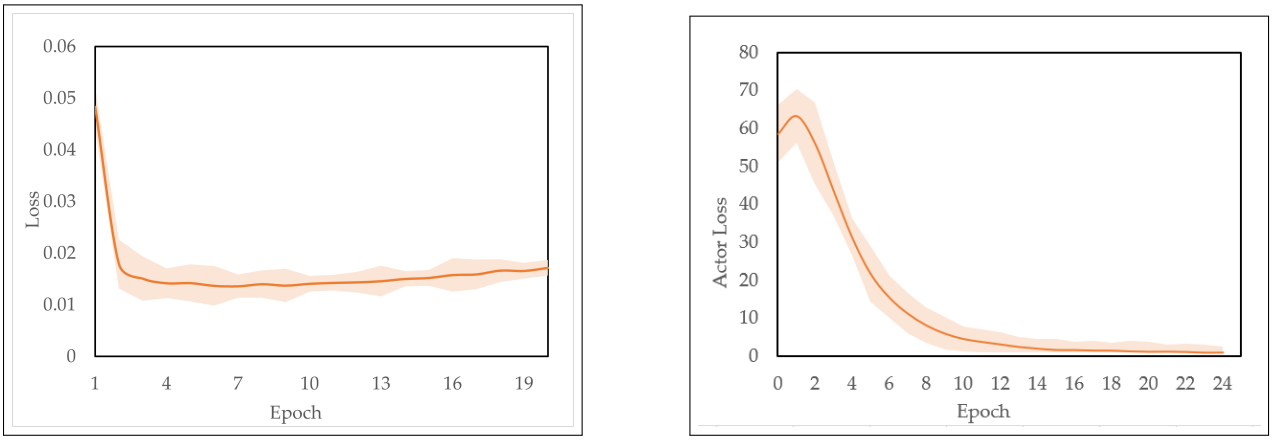
B. Learning from Demonstrations

Learning from demonstrations is a scheme that supports learning from data collected by a human expert when performing a task (Argall *et al.*, 2009, Schaal, 1996). Data normally collected include state information, actions, and possibly rewards at each step. In this work, I did not have access to an expert human pilot, however, the I did have some experience with flight simulators. Even though my collected data was going to be suboptimal, it could still help the agent bootstrap a reasonable policy.

The rendering system built was aimed to visualise dogfight engagements of the two aircraft. I needed to replace one of the agents with a human to collect data. To achieve this, a Meta Quest 2 VR headset was used so that the human could look around and have a full view of the opponent. Using some state augmentation, the opponent was enlarged so that the human could see it on screen as it would appear in real engagements. Then using this system, I collected 20 hours worth of flight data in the format $\mathcal{D} = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\}$, where i is the trajectory index.

Learning from demonstrations has two key targets: first staying close to the training data and second optimising the long-term reward. To achieve these objectives two key directions emerged in the literature: behaviour cloning and offline reinforcement learning. Behaviour cloning is a supervised learning setting with the objective of imitating the policy using which the dataset was collected (Torabi *et al.*, 2018). The agent learns by minimising the difference between its actions and the expert's actions. The loss is defined as $L(\theta) = \mathbb{E}_{a_t, s_t \tilde{D}}[(a_t - \pi_\theta(s_t))^2]$ which is the mean squared deviation of the action in dataset a_t and the chosen action by the agent π_θ in the state s at timestep t. The agent tries to closely match the actions taken by the expert, totally discarding the main objective of RL: the reward, making the algorithm much simpler however its performance is strongly limited by the quality of whole trajectories in the dataset distribution. Small errors in upstream trajectory can lead to a very different state downstream making it hard for behaviour cloning to generalise under suboptimal data (Kumar *et al.*, 2021).

Offline RL uses traditional RL algorithms with the aim of maximising long-term reward, as defined in Equation 2, with the caveat of being non-interactive (Levine *et al.*, 2020). If the expert's trajectory has some sub-optimal actions, but contains some high reward states, these rewards can be used for improving the learning. Therefore, offline reinforcement learning is expected to be useful given that the data is collected from a non-expert. Conservative Q-learning (CQL) is used which is a state-of-the-art algorithm for offline RL (Kumar *et al.*, 2020). CQL augments the standard Bellman error objective with a simple Q-value regulariser on top of the existing deep Q-learning (Fan *et al.*, 2020). This regulariser calculates a lower-bound on the Q-value, which prevents the overestimation of the Q-value which is common in offline RL due to out-of-distribution actions and function approximation errors.



(a) BC (b) CQL

Figure 10: The training loss of the agent using conservative Q learning.

The agents were trained using both behaviour cloning and CQL. Figure 10 shows the loss of the actor during training. The loss decreased in behaviour cloning but did not drop significantly. This was because the data I collected was suboptimal; some of the actions in the dataset were not optimised. Visual inspection of the resulting agent's performance showed that it has learnt to use three out of four controls successfully. It was noted that the agent has not learned how to use the throttle. On checking the data, I realised that the throttle signal was very sparse compared to the variation in the other three controls, hence showing why the agent struggled to learn it. The algorithm weighted all actions equally in the loss resulting in this catastrophic error that lead to aircraft crash in each evaluation episode. The performance is much better using CQL, as it did not discard the reward. This confirms that offline RL performs better in the case of suboptimal data.

C. Evaluation

There is a lack of standard evaluation metrics for this task in the academic literature due to the limited attempts in using AI agents for air-to-air dogfight combat. The air-combat problem is inherited from the defence domain without the availability of much public information. The true evaluation of autonomous agents can be estimated by competing against a human expert. In the absence of a human expert (air-combat pilot in this case), the alternative is to define various flight conditions and compare the performance against held-out validation agents as proposed by Jaderberg *et al.* (2019).

The agent trained using CQL is evaluated against an agent flying straight in the opposite direction (see video). The CQL agent achieved an average of 17.4 wins with a standard deviation of 0.5 wins in 20 different test conditions evaluated five times. This shows that the agent has learned significantly. However, evaluating the agent against itself in a self-play setting did not provide reasonable results, as none of the agent could achieve wins in any of the evaluation experiments. This highlights limitation of the collected data and raises the need for more diverse expert demonstrations in episodes with more challenging adversaries.

VI. Conclusion and Future Work

This work aims to advance the utilisation of AI algorithms for aviation, particularly in the combat domain, by providing a reproducible environment for AI-enabled air-to-air dogfight combat. Air-to-air combat is a high-risk environment lined with fatal risks to pilots. This paper attempts to recreate the simulation environment used in DARPA to allow exploring the feasibility of cutting-edge RL techniques and their potential in this domain in order to decrease the risk for human pilots. This will help in decreasing the risk in air combat and costs associated with large-scale human pilots' training. To this end, this work implements a simulation environment for training RL agents, using the 6-DoF high-fidelity simulator (JSBSim) which models the system close to the real world. Furthermore I implement a visual environment to evaluate the performance of the agents on the proposed task. The agents have been trained using traditional reinforcement learning, behaviour cloning and offline reinforcement learning. I have shown that RL could be applied to the very complex decision-making domain of air combat. Given an access to an expert pilot, this work can be scaled-up to train better agents that can perform on par with human experts.

The performance of the trained agent has shown the viability of the use of reinforcement learning in air combat. Learning from scratch is not a very promising avenue due to the high complexity of dynamics. Expert data however can be leveraged to train reasonable agents, and further training in multi-agent setting is required to have agents that can surpass human performance. The unavailability of information and expertise on this task is a significant obstacle.

The domain complexity has warranted us to start training with a simple opponent, the opponents need to scale-up. However, due to the unavailability of optimal expert data, I was unable to scale-up the opponents. Future steps would include collecting demonstrations by an expert against more complex deterministic adversaries and and to continue to supplement the data with data collected from the trained agents. This slow build up can then be used to train further using fictitious self-play to achieve superhuman performance. Moreover, similar to other problems in machine learning, once an autonomous agent reaches an expert level on one aircraft, transfer learning could be used to generalise it ot other aircraft with far lesser costs. To undertake this generalisation systematically, machine education (Abbass *et al.*, 2021) could be applied.

Acknowledgements

I would like to acknowledge my supervisors Dr Heba El-Fiqi and Dr Aya Hussein for their continued support throughout this project. I would like to acknowledge Dr Hussein Abbass for help in final review and submission to MDPI Drones journal.

References

Abbass, H., Petraki, E., Hussein, A., McCall, F., & Elsawah, S. (2021). A model of symbiomemesis: machine education and communication as pillars for human-autonomy symbiosis. *Philosophical Transactions of the Royal Society A*, *379*(2207), 20200364.

Akbari, S., & Menhaj, M. (2001). A fuzzy guidance law for modeling offensive air-to-air combat maneuver. In *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, (pp. 3027–3031). IEEE.

Alizadeh, A., Moghadam, M., Bicer, Y., Ure, N. K., Yavas, U., & Kurtulus, C. (2019). Automated lane change decision making using deep reinforcement learning in dynamic and uncertain highway environment. In *2019 IEEE intelligent transportation systems conference (ITSC)*, (pp. 1399–1404). IEEE.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Anderson, J. D., & Bowden, M. L. (2005). *Introduction to flight*. McGraw-Hill Higher Education New York.

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, *57*(5), 469–483.

Austin, F., Carbone, G., Falco, M., Hinz, H., & Lewis, M. (1987). Automated maneuvering decisions for air-to-air combat. In *Guidance, navigation and control conference*, (p. 2393).

Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., & Kautz, J. (2016). Ga3c: Gpu-based a3c for deep reinforcement learning. *CoRR abs/1611.06256*.

Ball, R. E. (2003). *The fundamentals of aircraft combat survivability: analysis and design*. American Institute of Aeronautics and Astronautics.

Baron, S., & Kleinman, D. L. (1971). A markovian approach to dynamic games of combat. In *1971 IEEE Conference on Decision and Control*, (pp. 464–469). IEEE.

Berndt, J. (2004). Jsbsim: An open source flight dynamics model in c++. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, (p. 4923).

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., *et al.* (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, *27*(4), 819–840.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *TARK*, vol. 96, (pp. 195–210). Citeseer.

Bouton, M., Nakhaei, A., Fujimura, K., & Kochenderfer, M. J. (2019). Cooperation-aware reinforcement learning for merging in dense traffic. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, (pp. 3441–3447). IEEE.

Bowling, M., & Veloso, M. (2001). Rational and convergent learning in stochastic games. In *International joint conference on artificial intelligence*, vol. 17, (pp. 1021–1026). Citeseer.

Breakwell, J. V., & Merz, A. W. (1977). Minimum required capture radius in a coplanar model of the aerial combat problem. *AIAA journal*, *15*(8), 1089–1094.

Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, *13*(1), 374–376.

Burgin, G. H. (1986). Improvements to the adaptive maneuvering logic program. Tech. rep., NASA.

Burgin, G. H., Fogel, L. J., & Phelps, J. P. (1975). An adaptive maneuvering logic computer program for the simulation of one-on-one air-to-air combat. volume 1: General description. Tech. rep., NASA.

Burgin, G. H., & Owens, A. (1975). An adaptive maneuvering logic computer program for the simulation of one-to-one air-to-air combat. volume 2: Program description. Tech. rep., NASA.

Burgin, G. H., & Sidor, L. (1988). Rule-based air combat simulation. Tech. rep., NASA.

Burton, R. R. (1988). G-induced loss of consciousness: definition, history, current status. *Aviation, space, and environmental medicine*.

DARPAtv (2020). Alphadogfight trials final event.
URL https://www.youtube.com/watch?v=NzdhIA2S35w

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., *et al.* (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, *602*(7897), 414–419.

Dong, Y., Ai, J., & Liu, J. (2019). Guidance and control for own aircraft in the autonomous air combat: A historical review and future prospects. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, *233*(16), 5943–5991.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., *et al.* (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, (pp. 1407–1416). PMLR.

Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020). A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, (pp. 486–489). PMLR.

Feigin, P., Pinkas, O., & Shinar, J. (1984). A simple markov model for the analysis of multiple air combat. *Naval research logistics quarterly*, *31*(3), 413–429.

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

GOODRICH, K., & MCMANUS, J. (1989). Development of a tactical guidance research and evaluation system (tgres). In *Flight simulation technologies conference and exhibit*, (p. 3312).

Heinrich, J., Lanctot, M., & Silver, D. (2015). Fictitious self-play in extensive-form games. In *International conference on machine learning*, (pp. 805–813). PMLR.

Isaacs, R. (1951). *Games of Pursuit*. Santa Monica, CA: RAND Corporation.

Isaacs, R. (1955). Differential games iv: Mainly examples. Tech. rep., RAND CORP SANTA MONICA CA.

Isele, D., Rahimi, R., Cosgun, A., Subramanian, K., & Fujimura, K. (2018). Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 2034–2039). IEEE.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., *et al.* (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, *364*(6443), 859–865.

Karli, M., Efe, M. Ö., & Sever, H. (2017). Air combat learning from f-16 flight information. In *2017 IEEE international conference on fuzzy systems (FUZZ-IEEE)*, (pp. 1–6). IEEE.

Kumar, A., Hong, J., Singh, A., & Levine, S. (2021). Should i run offline reinforcement learning or behavioral cloning? In *International Conference on Learning Representations*.

Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, 1179–1191.

La, H. M., Lim, R., & Sheng, W. (2014). Multirobot cooperative learning for predator avoidance. *IEEE Transactions on Control Systems Technology*, *23*(1), 52–63.

Laminar Research (2021). X-plane 11.55.
URL https://www.x-plane.com/manuals/desktop/#abouttheversionsofthex-planesimulator

Lee, B.-Y., Han, S., Park, H., & Tahk, M.-J. (2017). One-versus-one air combat algorithm considering direction of the lift vector. In *2017 25th Mediterranean conference on control and automation (MED)*, (pp. 265–270). IEEE.

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Liu, P., & Ma, Y. (2017). A deep reinforcement learning based intelligent decision method for ucav air combat. In *Asian Simulation Conference*, (pp. 274–286). Springer.

McGrew, J. S. (2008). *Real-time maneuvering decisions for autonomous air combat*. Ph.D. thesis, Massachusetts Institute of Technology.

McGrew, J. S., How, J. P., Williams, B., & Roy, N. (2010). Air-combat strategy using approximate dynamic programming. *Journal of guidance, control, and dynamics*, *33*(5), 1641–1654.

McMahon, D. C. (1990). A neural network trained to select aircraft maneuvers during air combat: a comparison of network and rule based performance. In *1990 IJCNN international joint conference on neural networks*, (pp. 107–112). IEEE.

Meier, L. (1969). A new technique for solving pursuit-evasion differential games. *IEEE Transactions on Automatic Control*, *14*(4), 352–359.

Merz, A. (1975). Application of differential game theory to role-determination in aerial combat. Tech. rep., NASA.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., & Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*.

NASA (2015). Check-cases for verification of six-degree-of-freedom flight vehicle simulations. Tech. rep., NASA. URL https://nescacademy.nasa.gov/flightsim/

Nash Jr, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, *36*(1), 48–49.

Nguyen, O., Gilbert, B., Kibler, & Deal (1979). Simulator Study of Stall/Post-Stall Characteristics of a Fighter Airplane with Relaxed Longitudinal Static Stability. Tech. rep., NASA Langley Research Center Hampton, VA, United States.

Pamplona, D. A., & Alves, C. J. P. (2020). Does a fighter pilot live in the danger zone? a risk assessment applied to military aviation. *Transportation research interdisciplinary perspectives*, *5*, 100114.

Petrenko, A., Huang, Z., Kumar, T., Sukhatme, G., & Koltun, V. (2020). Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, (pp. 7652–7662). PMLR.

Petrenko, A., & Kumar, T. (2020). A faster alternative to python's multiprocessing.queue. https://github.com/alex-petrenko/faster-fifo.

Pope, A. P., Ide, J. S., Mićović, D., Diaz, H., Rosenbluth, D., Ritholtz, L., Twedt, J. C., Walker, T. T., Alcedo, K., & Javorsek, D. (2021). Hierarchical reinforcement learning for air-to-air combat. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, (pp. 275–284). IEEE.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1–8. URL http://jmlr.org/papers/v22/20-1364.html

Rennie, G. (2018). *Autonomous Control of Simulated Fixed Wing Aircraft using Deep Reinforcement Learning*. Department of Computer Science Technical Report Series. University of BATH.

Schaal, S. (1996). Learning from demonstration. *Advances in neural information processing systems*, *9*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the national academy of sciences*, *39*(10), 1095–1100.

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

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354–359.

Stevens, B. L., Lewis, F. L., & Johnson, E. N. (2015). *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second ed. URL http://incompleteideas.net/book/the-book-2nd.html

Teng, T.-H., Tan, A.-H., Tan, Y.-S., & Yeo, A. (2012). Self-organizing neural networks for learning air combat maneuvers. In *The 2012 international joint conference on neural networks (IJCNN)*, (pp. 1–8). IEEE.

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., *et al.* (2021). Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, *34*, 15032–15043.

Torabi, F., Warnell, G., & Stone, P. (2018). Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*.

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

Virtanen, K., Karelahti, J., & Raivio, T. (2006). Modeling air combat by a moving horizon influence diagram game. *Journal of guidance, control, and dynamics*, *29*(5), 1080–1091.

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, H., & Zhu, J. (2021). Tianshou: A highly modularized deep reinforcement learning library. *arXiv preprint arXiv:2107.14171*.

Weng, J., Lin, M., Huang, S., Liu, B., Makoviichuk, D., Makoviychuk, V., Liu, Z., Song, Y., Luo, T., Jiang, Y., *et al.* (2022). Envpool: A highly parallel reinforcement learning environment execution engine. *arXiv preprint arXiv:2206.10558*.

Wiegmann, D. A., & Shappell, S. A. (2017). *A human error approach to aviation accident analysis: The human factors analysis and classification system*. Routledge.

Wiener, E. L. (1988). Cockpit automation. In *Human factors in aviation*, (pp. 433–461). Elsevier.

Zhu, M., Wang, Y., Pu, Z., Hu, J., Wang, X., & Ke, R. (2020). Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving. *Transportation Research Part C: Emerging Technologies*, *117*, 102662.