Towards Explainable Automation for Air Traffic Control using Deep Q-learning from Demonstrations and Reward Decomposition

M.C. Hermans

Thesis Report

TU Delft Delft University of Technology

Challenge the future

Towards Explainable Automation for Air Traffic Control using Deep Q-learning from Demonstrations and Reward Decomposition

by

M.C. Hermans

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	4475038	
Project duration:	May 11, 2020 – May 18, 2021	
Thesis committee:	Prof. dr. ir. M. Mulder,	TU Delft, Chairman
	Dr. ir. E. van Kampen,	TU Delft, supervisor
	Dr. ir. C. Borst,	TU Delft, supervisor
	ir. T. Nunes,	supervisor
	Dr. A. Sharpanskykh,	TU Delft, External examiner

An electronic version of this thesis will be available at `http://repository.tudelft.nl/`.

TU Delft

Preface

First of all, I would like to thank my supervisors Erik-Jan van Kampen, Clark Borst and Tiago Nunes, for their guidance in conducting this thesis. It has been a pleasure to work closely together with my supervisors and brainstorm on the various subjects related to this thesis. They have allowed me to explore problems on my own, which was difficult at times, but has taught me a lot. Their critical attitude towards the project has certainly contributed to the overall quality of my work.

Deep learning is computationally demanding, Reinforcement Learning even more so. Setting up the software architecture and building the reinforcement learning agents was a major effort. Since the techniques I wanted to use were not readily available in any Python library, I had to build the learning agents myself. One of the goals of this thesis was to increase the explainability of the automation, of which I now fully understand its relevance. Half-way through the thesis, the scope of the research had to be adjusted as implementation of the various concepts proved to be challenging.

A word of gratitude goes out to all my friends, who I have met at the faculty or through extra-curricular activities, who have contributed in making my time at the TU Delft, a fruitful one. Most importantly, I would like to thank my family for their unconditional support throughout my studies.

"*The greatest benefit of machine learning may ultimately be not what the machines learn but what we learn by teaching them.*" - Pedro Domingos

Max Hermans

Delft, April 2021

Contents

List of Figures

List of Tables

Nomenclature

List of Abbreviations

ANN	Artificial Neural Network
ATC	Air Traffic Control
ATCo	Air Traffic Controller
ATM	Air Traffic Management
ATS	Air Traffic Services
CD&R	Conflict Detection and Resolution
CNN	Convolution Neural Network
DARPA	Defense Advanced Research Projects Agency
DP	Dynamic Programming
DQfD	Deep Q-Learning from Demonstrations
DQN	Deep Q-Networks
drDuel-DQN	Decomposed Dueling Deep Q-Networks
FANS	Future Air Navigation Services
GPI	General Policy Iteration
ICAO	International Civil Aviation Association
IRL	Inverse Reinforcement Learning
MDP	Markov Decision Processes
MLP	Multilayer Perceptrons
MVP	Multi-Voltage Potential
PPO	Proximal Policy Optimisation
PVD	Plan View Display
RDX	Reward Decomposition
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SSD	Solution Space Diagram
TD	Temporal-Difference
XAI	Explainable Artificial Intelligence
XRL	Explainable Reinforcement Learning

List of Symbols

δ	Temporal-difference error
ϵ	Exploration rate
γ	Discount factor
θ	Network weight parameters
d_{CPA}	Distance to closest point of approach
t_{CPA}	Time to closest point of approach
a	Action
C	Set of reward components
G	Return
J	Loss/cost function
Q	State-Action value function
r	Reward
s	State
V	State-value function

1

Introduction

To cope with future demand, the Air Traffic Management (ATM) system must be more efficient in both operational as well as economic context (Raja Parasuraman, Molloy, and I. L. Singh 1993). The biggest limitation for coping with an increased air traffic density is the current Air Traffic Control system. Whereas one might expect Air Traffic Control (ATC) to be highly automated, the current system still entirely relies on air traffic controllers (ATCos) to ensure safe separation between aircraft. The workload of ATCos, specifically, has been identified as a key limiting factor to increase sector capacity (Djokic, Lorenz, and Fricke 2010). The mid-term view on the solution is to increase the productivity of ATCos by introducing new support tools and separation modes (European Commission 2009). It is expected that free-flight solutions using trajectory-based operations will be incorporated in Future Air Navigation Services (FANS), in which air traffic controllers will purely have a supervisory role. To facilitate this transition, implementing a high degree of automation is crucial to maintain safety.

The most demanding task of ATCos is Conflict Detection and Resolution (CD&R) (EUROCONTROL 1996). Therefore, this will be the focus of this research. The goal of the ATCos is to maintain safe and orderly operations. Currently, only automated systems to assist lower-level cognitive tasks as the visualisation of flight data and tracks, and collision warning systems have been implemented. To make the transition towards FANS, the support of ATCos for increased automation is crucial. Acceptance of automation by ATCos is seen as a main limitation for the introduction of novel automation in ATC (Westin, Borst, and Hilburn 2015). In (Bekier, Molesworth, and Williamson 2012), it was concluded that ATCos have a 'tipping point' at which they are no longer willing to cooperate with automation. This tipping point was reached in case the decision authority shifted away from ATCos towards the automation. In an effort to increase the acceptance of automation in ATC, strategic conformal automation has been investigated as a possible solution (Regtuit et al. 2018; Van Rooijen 2019). It is, however, questionable whether emulating human strategies is desirable since the automation is trying to improve on human performance (Westin, Borst, and Hilburn 2015). Furthermore, with an increase in air traffic complexity and density, implementing current strategies of ATCos for CD&R might lead to inefficient automation.

In addition to strategic conformal automation, research in the past decade has focused on automating the CD&R task to make it fully autonomous. Currently, the modified voltage method which has been developed in (J.M Hoekstra, van Gent, and Ruigrok 2002) achieves the highest performance in maintaining safe separation (Ribeiro, Joost Ellerbroek, and Jacco Hoekstra 2020). In more recent efforts, reinforcement learning (RL) has shown promising results in automating the CD&R task (Puca et al. 2014; Brittain and Wei 2018; Brittain and Wei 2019; Brittain, Yang, and Wei 2020; Pham et al. 2019; Tran et al. 2019). These papers indicate that RL can successfully be implemented to automate ATC, although it must be noted that the traffic scenarios analysed remain limited in terms of complexity.

The effectiveness of implementing reinforcement learning, which is a form of artificial intelligence (AI), for ATC remains questionable since these AI solutions are yet unable to explain decisions. Explainable models should be made to increase trust and allow users to understand and effectively manage artificially intelligent partners. To reach this goal, the Defense Advanced Research Projects Agency (DARPA) launched the explainable artificial intelligence (XAI) program (Gunning 2017). Models should be able to provide an explanation

for the decisions it makes in order for the human operator, either the designer or user, to know when the automation is successful and when it fails. An example is that of a supervised learning algorithm classifying a cat. Instead of outputting the probability that an image contains a cat, it should provide clues as: 'it has fur, whiskers and claws'; it has ears similar to the following images of a cat. Extending this ideology of developing explainable AI to ATC, the automation should be designed in a way that the human operator understands why an agent would take certain actions. Since implementing RL in ATC has shown to be promising, the focus of this research shall be on developing an explainable reinforcement learning agent (XRL) to automate ATC. Furthermore, by careful design of the reward function, the behaviour of the agent can be steered to be conform with strategies employed by ATCos.

To support ATCos during the CD&R task, decision support tools have been created, such as the separation monitor and the Solution Space Diagram (SSD) (Mercado Velasco, Mulder, and van Paassen 2010). These tools assist ATCos in making decisions. The solution space in the SSD is constructed from all vector commands (heading and altitude) that satisfy constraints of safety, productivity, and efficiency. For an individual aircraft, it is hypothesised that this display contains all the relevant information to be able to form a decision on what action to take (Mercado Velasco et al. 2015; Mercado Velasco, Mulder, and van Paassen 2010; Van Dam et al. 2004). The strength of the SSD is that no matter the complexity or airspace density, the ATCo is able to see a solution to the detected conflict. One of the limitations of reinforcement learning is that the state and action space have to remain limited in order to successfully find an optimal policy. Using the SSD as state- input for a learning agent might be promising as the size of the state remains fixed and, in a sense, naturally portrays the solution space.

1.1. Problem Formulation

The research objective is to automate ATC by developing an explainable reinforcement learning algorithm that uses the Solution Space Diagram as input. The SSD is a visual representation of how ATCos solve conflicts and it provides the learning agent with an overview of the surrounding traffic. The research question is linked to this objective:

"How can reinforcement learning be applied to the ATC task of conflict detection & resolution by exploiting features from the Solution Space Diagram and contribute to the explainability of the automation?"

There are six subquestions to this main research question:

1. What traffic scenarios and ATC tasks are relevant to analyse?
2. What type of reinforcement learning agent can be used to automate CD&R for a single controlled aircraft in a multiple-aircraft traffic scenario by using the SSD as state input?
    - What states and actions are used by an ATCo to perform their tasks?
    - What information can be extracted from the SSD?
    - Which deep RL algorithms are suitable for a two-aircraft traffic scenario?
3. What metrics can be used to evaluate the performance of the automation?
4. How can the RL based automation be implemented in a two-aircraft traffic scenario with a controlled and observed aircraft?
    - What factors should be incorporated in the reward function?
5. What factors can contribute to the explainability?
    - What techniques are there available?
    - How can strategy be incorporated?

In Figure 1.1, the research questions are shown and it is indicated in what part of the research these will be answered.

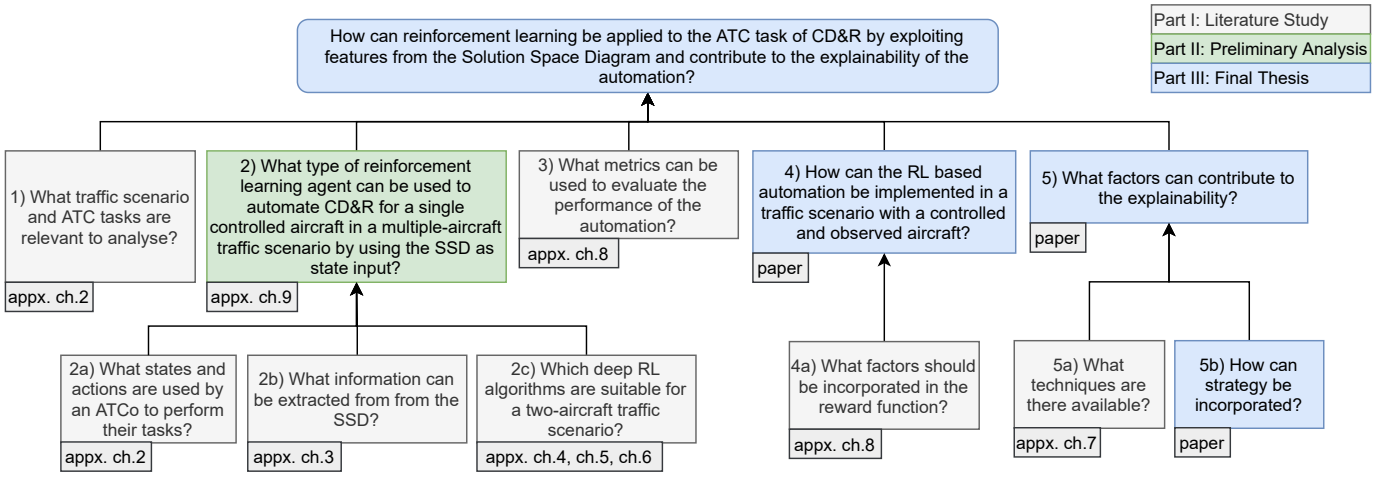


Figure 1.1: Research Questions.

1.2. Related Work

Two areas of research are of interest to this research, namely reinforcement learning to automate ATC and the development of explainable reinforcement learning. In (Brittain and Wei 2018), a deep double Q-Network was used for autonomous aircraft sequencing and separation. The agent had a limited action space in which solely the speed could vary discretely between six different values. The research showed that proper separation could be obtained by using this learning algorithm. One of the disadvantages was, however, that the state space increased in dimensions when more aircraft were added to the state space system. (Brittain and Wei 2019) and (Brittain, Yang, and Wei 2020) implemented a multi-agent learning algorithm which significantly improved the results. The researchers were able to ensure safe operations in more complex scenarios, although with an action space that was limited to a discrete amount of velocity alterations. Furthermore, (Pham et al. 2019) and (Tran et al. 2019) both explored the concept of having a controlled and observed aircraft. In these papers, the route of the controlled aircraft is changed by adding a way point, the 'trajectory change point'. In (Tran et al. 2019), the reinforcement learning agent was able to efficiently learn by taking human demonstrations into account.

Explainable reinforcement learning (XRL) remains a relatively unexplored area of research. (Juozapaitis et al. 2019) was the first to consider decomposing the reward function to identify why an agent behaves as it does. Decomposing the reward function as such allows the operator to see why an agent chooses to perform a certain action, i.e. due to safety, fuel efficiency, or to uphold airline preference in the case of ATC.

Reinforcement learning has proven to be effective for automating CD&R in simplified environments. However, it has not yet been investigated how using RL can contribute to the explainability of the automation.

1.3. Contribution

In (Van Rooijen 2019), a strategic conformal model was developed by training a supervised model which uses the SSD as input to increase the understanding of the agent's actions. Although the model developed was able to mimic human strategies, it will never improve on the human operator. Through the use of RL, an optimal policy in terms of maximising a reward function can be achieved. Reinforcement learning has already been applied to ATC in recent studies with limited state-action spaces. The SSD is said to offer the human operator with an overview of the solution space for an aircraft and has actually been shown that the use of it especially increases the fault detection performance (Borst et al. 2017). This work is the first to explore whether a RL agent benefits from having a support tool to represent its state.

The acceptance of automation, due to trust and transparency issues, is a burden for the implementation of higher degrees of automation in the current ATC system. Therefore, this work will also focus on the explainability of the automation. In the context of explainable AI, this work is the first to use reinforcement learning to contribute to the explainability of the automation for ATC.

1.4. Report Structure

Part I: This part of the thesis contains **the scientific article**.

Part II: This part of the thesis holds **the preliminary thesis**.

Part III: This part of the thesis details **appendices** to the scientific article of presented in part I.

I

Scientific Paper

Towards Explainable Automation for Air Traffic Control Using Deep Q-Learning from Demonstrations and Reward Decomposition

M.C. Hermans, *MSc. Student*
Supervisors: dr. ir. E. van Kampen, dr. ir. C. Borst, ir. T. Nunes
Section Control & Simulation, Department Control and Operations, Faculty of Aerospace Engineering,
Delft University of Technology, Delft, The Netherlands

***Abstract*—The current ATC system is seen as the most significant limitation to coping with an increased air traffic density. Transitioning towards an ATC system with a high degree of automation is essential to cope with future traffic demand of the airspace. In recent studies, reinforcement learning has shown promising results automating Conflict Detection and Resolution (CD&R) in Air Traffic Control. The acceptance of automation by Air Traffic Controllers (ATCos) remains a critical limiting factor to its implementation. This work explores how automation can be developed using Deep Q-Learning from Demonstrations (DQfD), which aims to be transparent and conforms with strategies applied by ATCos to increase acceptance of automation. Reward decomposition (RDX) is used to monitor the learning and to understand what the agent has learned. This study focuses on two-aircraft conflicts, in which the state of the controlled and observed aircraft is represented by raw pixel data of the Solution Space Diagram. It was concluded that pre-training on demonstrations speeds up learning and can increase strategic conformance between the solutions provided by the RL agent and the demonstrator. Next to increasing conformance, results also show that DQfD can improve its policy with respect to the suboptimal demonstrations used during training. Finally, RDX has allowed the designer to examine the policy learned by the RL agent in more detail.**

***Keywords*: Air Traffic Control, Solution Space Diagram, Conflict Detection & Resolution, strategic conformance, Reinforcement Learning, Deep Q-learning from Demonstrations, Reward Decomposition, Acceptance of Automation, BlueSky, Decision Support Systems**

I. INTRODUCTION

Today, human Air Traffic Controllers (ATCos) are responsible for maintaining safe and orderly operations within the air space. Conflict Detection & Resolution (CD&R) is seen as the most demanding task of ATCos [1]. To perform CD&R, ATCos acquire information from the Plan View Display (PVD). Apart from lower-level cognitive tasks such as the visualisation and presentation of traffic data, few tasks of an ATCo are automated. From the PVD, ATCos need to integrate lower order aircraft state information, presented on individual flight labels, to determine what actions can be taken to avoid a conflict [2]. The CD&R task is complex, especially in high-density traffic scenarios, since coordination is needed between different aircraft. Although autonomous methods to assure safe separation exist, acceptance of automation by ATCos is seen as a main limitation for the introduction of novel automation in ATC [3].

In an effort to increase the acceptance of automation in ATC, strategic conformal automation has been investigated as a possible solution [4], [5]. In [5], Van Rooijen confirmed the existence of a strategy heterogeneity between ATCos, which is a crucial assumption of strategic conformal automation. Even so, it remains questionable whether emulating human strategies is desirable as human errors might propagate through the automation [3]. Alternatively, trust can be increased by developing automation that is understandable to the individual [6].

Decentralised automation methods, such as the Modified Voltage Potential (MVP) method, have shown to be effective in automating CD&R [7]. A limitation of the MVP method is that resolutions may oppose the flight direction proposed by the flight plan, as it solely uses conflict geometry for its resolution, and therefore is not an optimal solution method for all traffic scenarios [7]. In pursuit of optimised automation methods for ATC, more recent efforts explore a data-driven approach. Reinforcement Learning (RL) has achieved promising results in automating CD&R [8]–[13]. RL is a computational approach to learn from interaction, mimicking the way that biological agents, like humans, learn. The potential of RL is that it is a way of programming agents based on reward and punishment without having to specify how the agent needs to achieve a task [14]. Applying RL for ATC, therefore, poses the opportunity to learn a policy that can account for a wide range of air traffic scenarios. Studies into applying RL for CD&R explore both centralised and decentralised control by implementing single agent, as researched by Brittain et al. [9], and multi-agent reinforcement learning techniques [10], [11].

The effectiveness of implementing RL, which is a form of artificial intelligence (AI), for ATC remains questionable since these AI solutions are yet unable to explain decisions. Explainable models should be made to increase trust and allow users to understand and effectively manage artificially intelligent partners. For Machine Learning methods, explainability remains a relatively unexplored area of research. In [15], Juozapaitis et al. developed a RL method with *reward decomposition* (RDX) which can be used to expedite the

explanation of action selection.

The main contribution of this article is that it focuses on the development of automation for CD&R using RL, with an aim to contribute to the explainability of the automation for the designer. This work proposes the use of Deep Q-learning from Demonstrations to bridge the gap between strategic conformance and optimal control, and uses Reward Decomposition to contribute to the explainability of the automation. Furthermore, it explores how a RL agent can benefit from having the Solution Space Diagram (SSD) to represent the state of a conflict pair. Explainability in this research relates to the mental model a designer has of the automation. Essentially, explainable models aim to increase the 'what will it do' prediction [6]. Experiments performed by Juozapaitis et al. show that decomposing the reward function into meaningful components allows the designer to gain significant insights into the agent's behaviour, e.g. identifying unwanted behaviour [15]. In an effort to highlight the individual contributions of the aforementioned methods, four case studies will be performed. The first is aimed at researching the capability of a DQfD agent to perform CD&R in a two-aircraft traffic scenario; the second case study explores how artificially generated demonstrations can be utilised to enhance the strategic conformance of the automation; the third investigates how reward decomposition can contribute to the transparency of the RL agent; and, finally, the fourth case study explores how reward shaping can contribute to the development of strategic conformal automation.

ATCos identify conflicts pairwise [16]. Therefore, this work focuses on resolutions in a two-aircraft traffic scenario with a controlled and observed aircraft. The RL task is formulated as a continuous control problem, rather than a strategic planning tool. To represent the state of an aircraft, this work explores whether a RL agent can benefit from the raw pixel data of the Solution Space Diagram (SSD). The SSD is an ecological decision support tool which visualises the locomotion constraints in terms of heading and velocity of the controlled aircraft. By using the SSD to represent the controlled aircraft's state, the dimension of the state space remains constant with varying aircraft density.

Section II elaborates on this study's fundamental concepts and theoretical motivations. Thereafter, section III formulates the RL problem and explains the setup of the four case studies. Section IV presents the results of the four case studies. Subsequently, in section V, a sensitivity analysis is performed. Section VI discusses the results of the study and its broader context. Finally, some final remarks on the research are presented in section VIII.

II. Background

A. Conflict Detection & Resolution

CD&R entails preventing violations of minimum separation standards, of which the horizontal separation standards are 5 nautical miles [17]. An ATCo is presented with the position of each aircraft in the sector, in terms of longitude and latitude, on an electronic radar screen. Each aircraft has a flight label which contains the call sign, heading and speed of the aircraft. Based on this information, an ATCo has to detect possible conflicts and resolve these. In general, ATCos work in one certain sector for the entirety of their professional life. Therefore, ATCos become specialists in detecting and resolving potential conflicts just from looking at a radar screen. To do so, the operators apply various strategies:

1) Detection: Two aircraft are in conflict if a vertical or horizontal loss of separation will occur in the near future [16]. Whenever the vertical separation standards are not maintained, an ATCo checks whether the horizontal separation standards are met [18]. An ATCo approaches the detection task by identifying potential conflicts pair-wise [18]. ATCos estimate the Closest Point of Approach (CPA) to see whether two aircraft are in conflict. This is the case when the distance of closest approach (d_{CPA}) is smaller than 5 nm. Other factors defining a conflict are the time to the closest point of approach [16], t_{CPA}, the conflict angle [19] and speed difference [20].

ATCos predict future conflicts with a prediction time varying between 5 and 10 minutes [16]. Studies on how a human detects conflicts show that the conflict angle, speed difference and air traffic density are negatively correlated with the accuracy of conflict prediction [20]–[22]. Furthermore, Xu and Rantanen [23] showed that ATCos tend to judge potential aircraft at smaller spatial distance but with equal t_{CPA} more urgent. This phenomenon is referred to as the "distance-over-speed bias".

2) Resolution: To resolve a conflict, ATCos can instruct pilots to perform an altitude, heading or velocity change. An altitude change is preferred since it is a "non-radar" manoeuvre which does not require constant monitoring [18]. If an altitude change is not possible, a heading change is preferred instead of a speed change as speed changes are mentally challenging to represent for controllers [21] and have less visual impact [18]. Furthermore, ATCos avoid resorting to speed changes for en-route ATC as the narrow flight envelope of airliners limits the velocity window at cruise altitude. In line with this hierarchy, the main influences on the expected utility of the controller's decision are [24]:

- *Expediency*: manoeuvres that resolve a conflict faster are preferred over time-consuming ones.
- *Preservation of airspace*: the least disruptive manoeuvres to the overall traffic flow are preferred.
- *Visualisation*: manoeuvres that resolve the conflict with more visual impact are preferred.

Furthermore, ATCos try to minimise the amount of requests to the pilot [25]. Next to the strategies shared between ATCos, ATCos apply personal strategies to resolve conflicts. The resolutions vary due to the conflict geometry, and depend on the safety factor and the t_{CPA} at which ATCos tend to solve conflicts. These strategies can, therefore, be identified from the parameters defining a conflict by examining resolutions as provided by ATCos.

For the CD&R task, both global and pairwise resolution methods have been developed. Global resolution methods may be more robust as it considers the entire traffic scenario at

once, but also more complex [26]. In [27], Van der Hoff has taken a multi-agent RL approach for the control task of ATC. Results showed that due to a lack of global coordination, this model is unable to resolve complex traffic scenarios. Moreover, the study into CD&R shows that ATCos detect and solve conflicts pairwise. This research will, therefore, take a pairwise approach and focus on two-aircraft conflicts with a controlled and an observed aircraft. In real-life, an ATCo first determines which pair of aircraft is in conflict. Then, the operator solves the conflict using shared and personalised strategies. This final stage will be the focus area of this work, as shown in Figure 1.

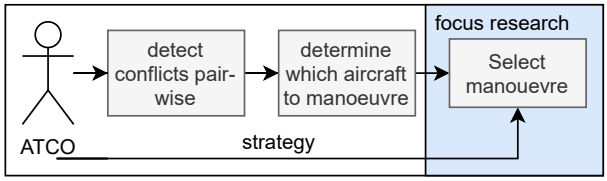


Figure 1. Control task of an ATCo.

B. Solution Space Diagram

In [28], Van Dam et al. researched the solution space of an aircraft from a pilot's perspective to increase situational awareness. The solution space visualises the locomotion constraints in terms of heading and velocity of the aircraft. Hermes et al. [29] used this approach to develop an ecological decision support tool for ATCos, called the solution space diagram. In Figure 2, the step-by-step construction of the SSD is visualised. First, all relative velocity vectors that lead to a loss of separation are calculated to define the Forbidden Beam Zone (FBZ) per observed aircraft. The FBZ is, then, displaced by the velocity of the observed aircraft to define the *no-go zone* for the controlled aircraft. The velocity vector of the controlled aircraft should be outside of this no-go zone to avoid a loss of separation. In a final step, the solution area is limited by V_{min} and V_{max} of the controlled aircraft. In this research, the colour of the FBZs is defined by the t_{CPA}, which can be calculated for a conflict pair using Equation 1 [30]. In this equation, p_{0_A} and p_{0_B}, and v_A and v_B are the current position coordinates and velocity vectors of the controlled and observed aircraft. The FBZ of the observed aircraft has a red colour for a $t_{CPA} < 60s$, an orange colour for a $60 < t_{CPA} < 120s$ and a grey colour for a $t_{CPA} > 120s$. An example of the final SSD used in this research is shown in Figure 4.

$$t_{CPA} = -\frac{(p_{0_A} - p_{0_B}) \cdot (v_A - v_B)}{||(v_A - v_B)||^2} \quad (1)$$

All relevant parameters to perform CD&R are contained in the SSD and their representations are listed in Table I and visualised in Figure 2.

Whether or not the FBZ of an observed aircraft is included in the SSD is dependent on the look-ahead time of the detection algorithm, and the absolute distance between the observed and the controlled aircraft. As ATCos have a look-ahead time of between 5 and 10 minutes [16], the look-ahead time of the detection algorithm is set at 10 minutes. Furthermore, the minimal absolute distance at which a conflict can be detected is set at 65 nautical miles.

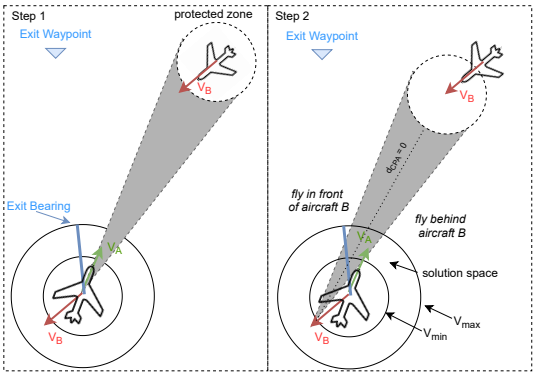


Figure 2. Step-by-step construction of the SSD for an upcoming conflict with a single observed aircraft. In this figure, *A* and *B* are the controlled and observed aircraft respectively.

Table I
PARAMETERS INCORPORATED IN THE SSD. ADAPTED FROM [5].

Parameter	Feature in the SSD
Velocity	Length of the velocity vector.
Velocity envelope	Inner and outer circle on the SSD.
Exit waypoint	Strikingly blue coloured heading vector.
Heading	Direction of the velocity vector.
Conflict in terms of heading and speed	FBZ.
Distance observed aircraft	Width of FBZ.
t_{CPA}	(1) Colour coding. (2) Combination of tail FBZ, green velocity vector and width FBZ.
d_{CPA}	Reflected by relative velocity vector in the FBZ.
Conflict angle	Inclination of the FBZ.
Traffic Density	Amount of FBZs represented in the SSD.
Velocity observed aircraft	Position tail FBZ.
Traffic Complexity	With a higher complexity, the FBZs of individual observed aircraft overlap less and solution space becomes smaller.
Objects	Represented as a FBZ covering entire range of velocities in a certain direction.

C. Reinforcement Learning

The model-free RL problem consists of three main elements: a policy, a reward signal and a value function. The policy, $\pi(a|s)$, is a mapping from the state to the action. The agent thus uses the policy to decide what action to take in a particular state. The performance of an action is measured by the reward signal and the state value function, v_π(s), indicates the expected cumulative sum of future rewards under the current policy π. Next to the state-value function, a state-action-value function, $Q_\pi(s, a)$, which relies on both the state

and the action can be learned. At every timestep, the agent chooses an action, $a_t \in A(s)$, based on the state , $s_t \in S$. It receives a reward signal, $r_t \in R$, as a consequence of the previous interaction with the environment. The goal of the agent is to maximise the reward it accumulates over time, which is referred to as the return. This is shown in Equation 2, in which γ is the discount factor. The discount factor weighs the importance of immediate and future rewards.

$$G_t = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2}$$

D. Deep Q-Networks

For this research it is chosen to implement a value-based RL method, in which actions are selected based on the learned values, as this would make the algorithm suitable for reward decomposition [15]. A deep RL algorithm, in which the value function is approximated by a deep neural network, is implemented. The pixel data of the SSD will be used to represent the state of the controlled aircraft. Each pixel in an image is a feature to the neural network. For image data, it is common to use a Convolutional Neural Network (CNN). This is because CNNs can effectively reduce the number of parameters, compared to an artificial neural network (ANN), without losing the quality of the model [31]. Furthermore, CNNs are not spatially dependent. This allows the network to learn certain features, such as the FBZ in the SSD, regardless of the position of the feature in the image [31]. For these reasons, a CNN will be used to represent the value function.

Deep Q-Networks (DQN) [32] is an online value-based approximate solution method which was the first to show that policies could be learned from high-dimensional sensory inputs using end-to-end learning. Such an approximator for the value function is essential to learning and generalising to large state spaces. The downside of the DQN algorithm compared to the tabular Q-learning algorithm, in which the representation of the Q-function is exact, is that convergence is no longer guaranteed [33]. In DQN, the loss function at each iteration is composed of the temporal difference (TD) error, as shown in Equation 3. In this equation, θ', represent the network weights of the target network, D represents the replay buffer and s' represents the next state.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[\overbrace{(r + \gamma \max_{a'} Q(s', a'; \theta_i')}^{\text{learned value}} - Q(s, a; \theta_i))^2] \tag{3}$$

The two concepts of a target network and experience replay buffer are both related to removing correlation between data samples used for updates, and prove to be critical for convergence. The agent interacts with the environment and saves the experience to a replay buffer. More formally put, the agent's experiences are stored at each time step t in the data set $D_t = e_1, ..., e_t$, in which $e_t = (s_t, a_t, r_t, s_{t+1})$. The Q-learning training updates are performed on randomly drawn samples, $(s, a, r, s') \sim U(D)$. To avoid chasing a moving target, a separate target network, which is updated every τ steps, is used. During each update of the target network, the target network copies the weights of the current policy.

E. Dueling Deep Q-Networks

In this work, a Dueling DQN algorithm shall be used to represent the value function. Wang et al. [34] showed that Dueling DQN achieves a better performance than DQN on most RL problems. The only difference between DQN and Dueling DQN is that the approximation of the state-action value function is decomposed in an advantage and state-value stream, which is shown in Equation 4.

$$Q(s, a) = A(s, a) + V(s) \tag{4}$$

The advantage function is defined as the advantage of a particular action in a state against all other possible actions. During every update step, the weights of the network are updated by back-propagating the loss between the learned value and the value approximation. The size of these steps is dependent on the optimiser used. In this research, the *Adam optimiser* [35] will be used as this optimiser achieves the best performance for most RL algorithms and is easy to configure [36]. The Adam optimiser maintains an adaptive learning rate per network parameter.

F. Learning from Demonstrations

Most RL algorithms are data inefficient which in general is not a problem for RL tasks for which an accurate simulation environment is present. However, for real-world applications, an accurate simulation environment is often not available. To speed up learning, Hester et al. [37] presented a method in which the RL agent can learn as much as possible from demonstration data before exposing the RL agent to the task at hand. This demonstration data can be acquired from a human performing the RL task at hand, or by another form of automation that is capable of performing the task. The algorithm employs a pre-training phase which aims to learn a policy that imitates the demonstration data with a value function satisfying the Bellman equation so that it can be used for TD updates during the normal training phase.

The loss function is composed of four components: 1-step double Q-learning loss, an n-step double Q-learning loss (n-step loss), a supervised large margin classification loss (Expert loss), and a L2 regularisation loss on the network weights and biases. It is shown in Equation 5, in which λ_1 and λ_2 are the n-step return and supervised loss weight, whose values are taken from the original paper [37]. The n-step and Expert loss only apply on samples from the demonstrations data set. The n-step loss aims to propagate the trajectory of the demonstrator to all earlier states. Essentially, the n-step loss component ensures that the estimation of the current state is not only affected by the estimation of the next state, but also by the estimation of the next n-states [37]. It is implemented with a forward view, just as in [38]. The Expert loss is critical for the pre-training phase and is used for the classification of demonstrator's actions, whilst Q-learning is used to ensure that

the Bellman equation is met. Lastly, L2 regularisation is added to prevent overfitting on the often limited demonstrations data set.

$$L(Q) = L_{DQN}(Q) + \lambda_1 L_{n-step}(Q) + \lambda_2 L_E(Q) + \lambda_3 L_{L2}(Q) \quad (5)$$

The Expert loss is shown in Equation 6. In this equation, a_E is the action of the demonstrator and $l(a_E, a)$ is the margin function that is 0 when the action is of the demonstrator, and a positive value otherwise. This margin function ensures that the values of all the other actions are at least a value lower than that of the demonstrator.

$$L_E = \max_{a \in A}[Q(s,a) + l(a_E, a)] - Q(s, a_E) \quad (6)$$

To ensure that the RL agent can still benefit from demonstrations during the normal training phase, these samples cannot be removed from the experience replay. Furthermore, they have a higher probability of being selected as sample. This is explained in more detail in subsection II-G. The pseudo-code of the algorithm is shown in Algorithm 1.

One of the challenges in RL concerns the exploration-exploitation trade-off. Agents seek to learn action values conditional on subsequent optimal behaviour, but also need to behave non-optimally to explore all actions and possibly find the optimal solution [39]. In this research, an ϵ-greedy approach is taken. This means that the RL agent has a probability of ϵ to select an exploratory action (random action). This ϵ-greedy approach has also been used in original paper on DQfD [37].

G. Prioritised Experience Replay

Schaul et al. [40] improved DQN and Dueling DQN in terms of performance and sped up the learning on the Atari 2600 games by adding *prioritised replay* instead of uniform experience replay. When using uniform experience replay, the agent randomly samples a batch from the replay buffer. The basic idea of prioritised replay is to draw important samples more often instead of randomly drawing samples. During optimisation, the algorithms try to minimise the temporal-difference (TD) error. The basic idea of prioritised sweeping is to increase the replay probability for samples in the set of experiences which have a high expected learning progress, as measured by the magnitude of the TD error. The TD error for Dueling DQN is given in Equation 7.

$$\delta_i = \overbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i')}^{\text{learned value}} - Q(s, a; \theta_i) \quad (7)$$

In DQfD, prioritised replay is used to balance the amount of demonstration data and new experiences contained in a mini-batch during training. The probability of selecting a sample, P(i), is determined by how surprising or expected a sample is, following the TD-error [37], which is shown in Equation 8. In this equation, α determines how much prioritisation is used, k is the mini-batch size and p_i is the priority. Setting α to zero corresponds with the uniform case. In the paper [37], in which DQfD is first described, Hester et al. implement proportional priority as shown in Equation 9. In this equation, ϵ_a is small positive constant to ensure all transitions are sampled with some probability. An additional demonstration priority bonus, ϵ_d, is added to p_i of the demonstration data to increase the frequency at which they are selected.

Algorithm 1. The pseudo-code of Deep Q-Learning from Demonstrations (DQfD) [37]. The behaviour policy $\pi^{\epsilon Q_\theta}$ is ϵ-greedy with respect to Q_θ.

Require: $\mathbb{D}^{replay}$: initialised with demonstration data set; θ: weights for initial behaviour network (random); θ': weights for target network (random); τ: frequency at which to update target net; k: number of pre-training gradient updates; α: learning rate; $N_{\text{training epochs}}$: number of epochs for training

for steps t $\in \{1, 2, ...k\}$ {pre-training phase} **do**
  Sample a mini-batch of n transitions from D_{replay} with prioritisation
  Calculate loss L(Q) using target network
  Perform a gradient descent step to update θ
  if t **mod** τ = 0 **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
end for
for steps t $\in \{1, 2, ..., N_{\text{training epochs}}\}$ {*normal training phase*} **do**
  Sample action from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
  Play action a and observe (s', r)
  Store (s, a, r, s') into D^{replay}, overwriting oldest self-generated transition if over capacity occurs
  Sample a mini-batch of n transitions from D^{replay} with prioritisation
  Calculate loss L(Q) using target network
  Perform a gradient descent step to update θ (Adam optimiser)
  if t **mod** τ = 0 **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
end for

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad (8)$$

$$p_i = \begin{cases} |\delta_i| + \epsilon_a + \epsilon_d, & \text{for sample from demonstrator} \\ |\delta_i| + \epsilon_a, & \text{else} \end{cases} \quad (9)$$

To account for the change in data-distribution caused by the prioritisation, updates to the network are weighted by sampling weights, $w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$ [37]. In this equation, N is the size of the replay buffer and β determines the amount of

[1]For the implementation, the 'sum-tree' data structure approach is used as detailed in [40]

importance sampling, with $\beta = 0$ for no importance sampling. In this research, β is annealed from 0.6 to 1.0, meaning up to complete importance sampling, over the course of entire training period. The value of 0.6 for β_0 was taken from the original paper on DQfD [37].

H. Reward Decomposition

Many RL tasks have a reward function that can naturally be decomposed into meaningful components. In value-based solution methods, the RL agent takes the action that maximises the expected value function, the Q-function. The agent learns how to maximise the reward function, but the learned Q-function fails to provide any insight into how each component of the reward function contributes to the learned value. In [15], the Juozapaitis et al. show how utilising decomposed rewards can give insight into trade-offs between different reward components. Translating this concept to the ATC task, its reward function can naturally be decomposed into rewards related to avoiding a loss of separation, taking the minimum amount of actions and minimising deviation from the flight path. When considering the ATC task in a more complex setting, rewards on sector disruptiveness and safety margin could also be included. A well-trained agent would then be able to provide an explanation to the user to show which part of the reward function contributes most to taking a certain action. Not only does this have the potential to increase the explainability of the automation for the user, it can assist designers to spot anomalies in the automation. In the original paper on reward decomposition [15], Juozapaitis et al. provide a proof that DQN should be suited for reward decomposition in case the algorithm is run under the standard conditions for almost sure (a.s.) convergence[2]. In this research, DQfD and Dueling DQN is implemented, which uses a Dueling state-advantage convolutional network. As this does not violate the standard conditions for a.s. convergence, reward decomposition can be implemented according to the DQN implementation shown below:

$$L(\theta_c) = \sum_{i=1}^{k} (y_{c,i} - Q_c(s_i, a_i; \theta_c))^2 \tag{10}$$

$$y_{c,i} = \begin{cases} r_c, & \text{for terminal } s'_i \\ r_c + \gamma Q_c(s'_i, a_i^+; \theta'_c), & \text{for non-terminal } s'_i \end{cases} \tag{11}$$

$$a_i^+ = \underset{a'}{argmax} \sum_{c \in C} Q_c(s'_i, a'; \theta'_c) \tag{12}$$

In these equations, C is the set of reward components. Each value of the current network is thus updated based on the current greedy action of the complete target network. Except for a different update method, decomposed Dueling DQN (drDuel-DQN) operates exactly the same as regular Dueling DQN.

[2]Specifically, we must update each state-action pair in-finitely often, and the learning rates $\alpha_t(s, a)$ must satisfy $\sum_t \alpha_t(s, a) = \infty$ and $\sum_t \alpha_t^2(s, a) < \infty$

III. Methodology

This section elaborates on the methodology used to develop automation for ATC that aims at being strategically conformal with ATCos and explainable for both the designer as well as the operator. First, subsection III-A formulates the RL task for 2D CD&R with a single controlled aircraft. Then, the experiment setup for case study 1, case study 2, case study 3 and case study 4 are elaborated on in subsection III-C, subsection III-D, subsection III-E and subsection III-F.

A. Problem Formulation of the RL Task

Reinforcement learning has been shown to be effective in simplified learning environments. This research focuses on en-route ATC in air traffic sectors. Specifically, this research only considers two-aircraft conflicts.

1) Objective: The objectives of the case studies performed in this research are to avoid a loss of separation (LOS) and minimise the flight path using a resolution strategy compliant with that of an ATCo. This entails that the action space should represent one that is also available to an ATCo.

2) Simulation Environment: For this research, BlueSky [41], which is a high-fidelity ATC simulation environment, is used. Whereas BlueSky allows for high-fidelity ATC simulations including uncertainties, it is chosen to exclude these in this research and essentially use it as a deterministic simulation environment. In these simulations, OpenAP [42] was used as performance model.

3) Assumptions: As ATCos solve conflicts pairwise, it has been decided to simplify the ATC task that entails controlling all aircraft in a sector to controlling a single aircraft, the controlled aircraft. Furthermore, the simulation environment is constructed such that it updates once every 10 seconds as ATCos will in general not provide multiple resolution commands within this time window. The main reason for this being that radar updates are provided once every 10 seconds [17]. The aircraft type used in the case studies is the B737, of which the performance is modelled using the OpenAP [42] performance model. Only one aircraft type was considered to simplify the CD&R task without losing too much realism.

4) State Representation: In an effort to increase the generalisation of the automation, the SSD will be used to represent the state of the conflict pair. The advantage of using the SSD is that the size of the state space does not increase with an increase of the number of aircraft in the sector. More specifically, most of the information is contained in the upper half of the SSD [5], in which the current speed vector is directed upwards. An ATCo is unlikely to command an aircraft to perform a heading change of more than 90 degrees. In [5], Van Rooijen supports this hypothesis by having successfully trained a supervised model to perform CD&R using only the upper half of the SSD. An analysis comparing the full SSD and upper half of the SSD to represent the state of the controlled aircraft showed that using the upper half of the SSD increased the data-efficiency of the RL algorithm, without suffering any performance loss. The data-efficiency achieved by limiting the size of the state space outweighs the additional

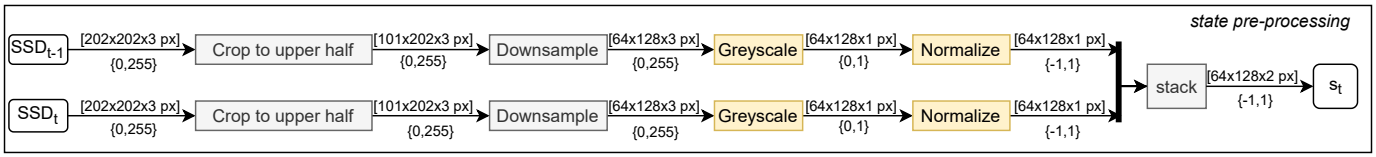


Figure 3. Pre-processing of the state at time-step t in the simulation. The values between the square brackets represent the dimensions of the image whilst the values between curly brackets represent the range of values of a single pixel.

information provided on the lower half of the SSD. On its own, the SSD does not contain enough information for an ATCo to select the most optimal conflict resolution strategy [43]. This is because some features, such as the t_{CPA} and d_{CPA}, are difficult to extract from a single SSD for a human operator as they are disguised in the shape of the FBZ. In deep RL, it is common to stack multiple consecutive frames to incorporate motion information [44]. To ensure information on the motion of the FBZs is present in the state of the RL agent, two consecutive SSDs are stacked in this research, which is visualised in Figure 4. By stacking the images, the optical flow of the features in the image is contained in the state. Stacking two images does, however, double the size of the state-space.

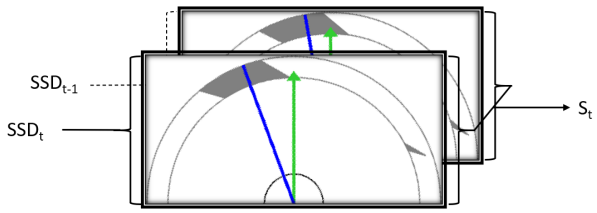


Figure 4. State of the RL agent is represented by stacking two SSDs, of which only the upper half is used.

The SSD contains multiple colours. A single RGB SSD has three feature maps, one for each colour. To reduce the state-space, the SSD image is converted to greyscale, reducing the state-space from two SSDs containing three feature maps to a single feature map per SSD. In general, deep RL does not require the input data to be normalised. However, neural networks work better with normalised data [45]. Therefore, it is chosen to normalise the value of each pixel in the SSD image using the mean and standard deviation calculated after having acquired 10,000 samples. These samples are acquired by using the MVP method to resolve conflicts for traffic scenarios encountered in the first test case. Acquiring samples with random exploration would lead to encountering many SSDs without any FBZ in it. This can result in a large mismatch between the data distribution of the memory buffer from which data statistics are determined and the data encountered during interaction. The full pre-processing sequence is shown in Figure 3.

5) Action Space Definition: This research considers the en-route airspace. In this part of the airspace, the flight envelope of the aircarft limits the possible airspeed to a small velocity window. Moreover, since resolving a conflict with a heading change has more visual impact than altering the velocity, ATCos do not often command a velocity change [24]. One of the goals of this research is to design automation that is conformal to how ATCos solve conflicts. The action space of the RL agent should conform to the ATCo solution space. The resolution commands that ATCos give are in steps of 5 degrees and span between -30 and +30 degrees. Additionally, ATCos can command the controlled aircraft to continue its route to the exit waypoint [46]. Incorporating all these commands in the action space of the agent results in an action space size of 14. One of the major challenges in the field of machine learning is the curse of dimensionality. Adding a dimension exponentially increases the amount of possible solutions [14]. With this consideration in mind, it is decided to limit the action space to 6 output classes, as shown in Equation 13. In the action space, Δ_{DCT} is the Direct To exit waypoint command. Reducing the action space will not affect the ability of the RL agent to solve conflicts compared to the larger action space. Commanding a B737, which is used in this research, to perform a heading change of 30 degrees takes three simulation steps (30 seconds) to complete at cruise altitude. With a limited action space the RL agent can also achieve a 30 degrees heading change by taking three consecutive actions of + 10 degrees. The RL agent can thus achieve the same heading deviation in the same amount of simulation steps.

$$\mathcal{A}_{\text{RL agent}}(s) = \{-10, -5, 0, 5, 10, \Delta_{\text{DCT}}\}[^\circ] \tag{13}$$

6) Reward Function: Following the objective of this research, the agent should be able to avoid a loss of separation and reach the target waypoint with a minimum flight path deviation. En-route ATCos provide an aircraft with a minimal amount of resolution commands [25] and instruct the pilot to resume navigation towards the target waypoint once the conflict has clearly been solved. To incorporate this, a sub-goal for the reinforcement learning agent is to minimise the amount of actions taken. These goals each make up a component from the reward function, as shown in Equation 14.

$$r_{\text{total}}(t) = r_{\text{LOS}}(t) + r_{\text{action}}(t) + r_{\text{flight path}}(t) \tag{14}$$

The component of the reward function related to minimising the flight path can be designed using either continuous reward or by providing it at the end of the episode as a function of the flight path distance. In this research, it is chosen to use a continuous approach as literature shows that the agent is able to learn in a more data-efficient manner when a continuous reward is used. The agent retrieves a reward of + 1 for the flight path if its current heading, ψ_t, is aligned with the

target waypoint and is negatively rewarded for any deviation from this target heading. The value of the different reward components are listed below:

- r_{LOS}(t) = -100 if *loss of separation* else 0.
- r_{action}(t) = -5 for all actions except Δ_{DCT} and 0 degrees.
- $r_{\text{flight path}}$(t) = $1 - \frac{\|\psi_t - \Delta_{TH}\|}{50}$

7) Terminal State: As Dueling DQN is suited for episodic tasks, the learning environment should have clear initial and terminal states. Since the state of the controlled aircraft is not dependent on its geographical location, but on the conflict geometries, the aircraft is initialised at the same coordinates at the start of each episode. If the agent learns to avoid a conflict and reach the target waypoint, it can do so from any geographical location. The terminal state is reached in one of the following cases:

- Loss of separation between controlled and observed aircraft.
- controlled aircraft reaches target waypoint within 5.4 nm.
- The simulation runs out of time ($t_{sim} > 1200s$) without seeing any of the aforementioned terminal states.

The last terminal state is included to speed up the learning process since the agent would be exploring states that are not interesting for conflict resolution.

8) Neural Network Value Function Approximation: The network architecture used in this research is a Dueling DQN architecture [34]. The dueling network has two heads, one for estimating the state value and one for estimating the action advantage, which share the same feature learning module. The feature learning module is composed of the same convolutional neural network as used in the original paper describing DQN [32]. It has three convolutional layers and is described per layer in Table II.

Table II
SETUP OF THE FEATURE EXTRACTOR USED FOR DQFD. THE FIRST AND SECOND CONVOLUTIONAL LAYER HAVE A STRIDE OF 4 AND 2.

Layer #	Type of Layer	Input Size	# Feature Maps	Kernel Size	Output Size
1	CONV(s4)	[64x128x2]	32	8x8	[15, 31, 32]
3	ReLU	[15, 31, 32]	32	-	[15, 31, 32]
4	CONV(s2)	[15, 31, 32]	64	4x4	[5, 14, 64]
6	ReLU	[5, 14, 64]	64	-	[5, 14, 64]
7	CONV(s1)	[5, 14, 64]	64	3x3	[4, 12, 64]
9	ReLU	[4, 12, 64]	64	-	[4, 12, 64]
10	Flatten	[4, 12, 64]	-	-	3072

The feature learning module is extended with a stream for the value and advantage function. The details of these layers can be found in Table III. As one can see, both the value stream and advantage stream are composed of a fully-connected layer with 512 units. Remember that the input of the network is a stack of two images, having a dimension of 64x128x2 pixels. Due to the different convolutions applied to the image, the dimensions are altered. This causes the feature learning module to have an output size of 3072. This is the input size of the two streams. For drDuel-DQN, the network shares the feature learning model. For each reward component, the network is extended with a stream for the value and advantage function related to that particular reward component.

Table III
HEAD OF VALUE AND ADVANTAGE APPROXIMATOR OF THE DUELING DQN AGENT.

	Type	Input Size	Output Size
value stream layer #1	Linear	3072	512
value stream head	Linear	512	1
advantage stream layer #1	Linear	3072	512
advantage head	Linear	512	# actions (6)

To enable the network to update on batches of experiences, a dimension is added to the input.

9) Collecting Demonstrations: In order to incorporate strategies into the automation, these must first of all be identified. Regtuit et al. [4] used k-means clustering to identify strategies in terms of parameters describing the conflict from data acquired during experiments in which a human ATCo resolves conflicts. Even though Regtuit [4] showed how strategy can be incorporated from human-in-the-loop experiments, this research will not use human demonstrations. Instead, the MVP method will be used to generate demonstrations. Whereas ATCos can be inconsistent, the MVP is consistent in how it solves a conflict.

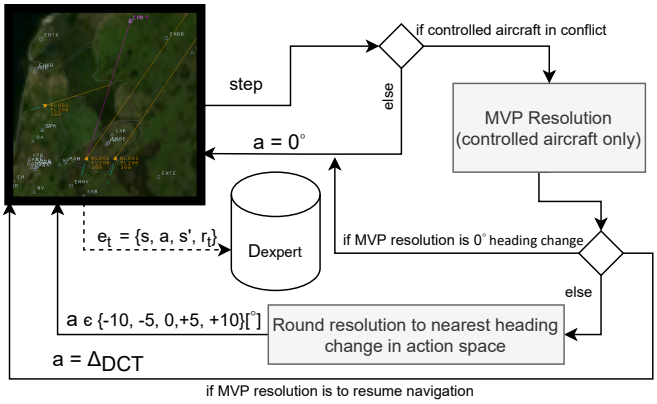


Figure 5. MVP method applied to collect demonstrations.

Throughout the first two case studies, 'imperfect' demonstrations will be presented to the RL agent. Instead of the MVP method being able to take a step of 10 degrees at each simulation step, it is restricted to taking steps of 5 degrees. This is done to stress the added value of RL, instead of having an agent that relies heavily on the pre-trained model.

B. Hyperparameters and Training Loop

In general, large sizes of the memory replay buffer, up to 10M experiences, are implemented in deep RL as this stabilises the learning [47]. For simpler environments, such as the MountainCar Game, capacities ranging between 20,000 and 100,000 already suffice [48]. Since the agent will not

encounter a wide range of different conflicts, it is chosen to use a replay buffer with a size of 50,000 for all experiments. Whenever the capacity of the replay buffer is reached, samples will be removed in the order that these were added to the replay buffer. As in [37], it is ensured that demonstrations are kept in the replay buffer. A hyperparameter selection showed that Dueling DQN for this environment shows the best performance for an initial learning rate of 0.00001 for the Adam optimizer and a batch size of 128. To ensure that the demonstrator's action was always a margin higher than the others, it was chosen to use a $l(a_E, a)$ of 15. Other hyperparameters used during training are listed in Table IV, in which $c_{minimum}$ represents the minimum amount of samples needed in the replay buffer to start updating the network. For other hyperparameters, the values used in the original paper about DQfD are implemented.

Table IV
HYPERPARAMETERS.

Hyperparameter	Value
ϵ_{start}	1.0
ϵ_{end}	0.05
ϵ_{decay}	5,000
n	128
α	0,00001
τ	100
γ	0.99
$c_{minimum}$	10,000
$l(a_E, a)$	15

Contrary to the algorithm shown in Algorithm 1, the network update frequency is lowered from once for every step in the environment to once every four steps. The RL agent thus collects four experiences before updating. It speeds up learning, as network updates are computationally more expensive than forward passes, and ensures that the experience replay buffer more closely represents the state distribution of the current policy which prevents the network from overfitting [49]. Furthermore, to avoid overfitting on experiences seen early on in the training, the experience replay memory has been filled with 10,000 randomly collected experiences prior to the learning phase. These are also the first to be removed if the replay buffer has reached its capacity. For the DQfD agent, the demonstrations make up a part of this initialised replay buffer.

C. Case Study 1: Increasing Data Efficiency & Performance

In the first case study, it is researched how a RL agent can learn to perform CD&R in a two-aircraft traffic scenario. Furthermore, the ability of the agent to speed up the learning by utilising demonstrations is investigated. Two RL agents are trained in the same traffic scenario to research the effect of demonstrations: a Dueling DQN agent and a DQfD agent with a pre-train period of 30,000 epochs.

Most en-route conflicts are crossing path conflicts. The RL agent will be trained to perform CD&R for a traffic scenario with crossing path conflict angles. The different conflict initiations that the RL agent will encounter are summarised below:

- Conflict angles: {45, 55, ..., 90, ..., 125, 135} [°]
- t_{cpa}: 400 [s]
- d_{cpa}: 0 [nm]
- $CAS_{\text{controlled a/c}}$: 230 [kts]
- $CAS_{\text{observed a/c}}$: 250 [kts]
- Type of aircraft: B737

From the possible initial states of the observed aircraft it can be noticed that the agent can encounter 11 different types of conflict. The experiment setup is visualised in Figure 6. It is chosen to keep the amount of conflicts the controlled aircraft can encounter limited as Deep RL is data inefficient and training for a wide range of conflicts would require an extensive training time to find a convergent policy. At the start of each episode, an observed aircraft is randomly initiated at one of the conflict angles. With a large amount of episodes, this ensures that the agent will encounter all conflicts approximately the same amount of times. All conflicts are initiated at a d_{CPA} of 0 nm as this is the most difficult conflict to solve. For this scenario, the velocity vector of the controlled aircraft is exactly in the middle of the FBZ. This ensures that the RL agent will not simply optimise towards taking the minimum amount of actions to get out of the no-go zone, but optimises towards minimising the flight path distance. In a sensitivity analysis, a RL agent will be trained in a learning environment with more diverse conflicts.

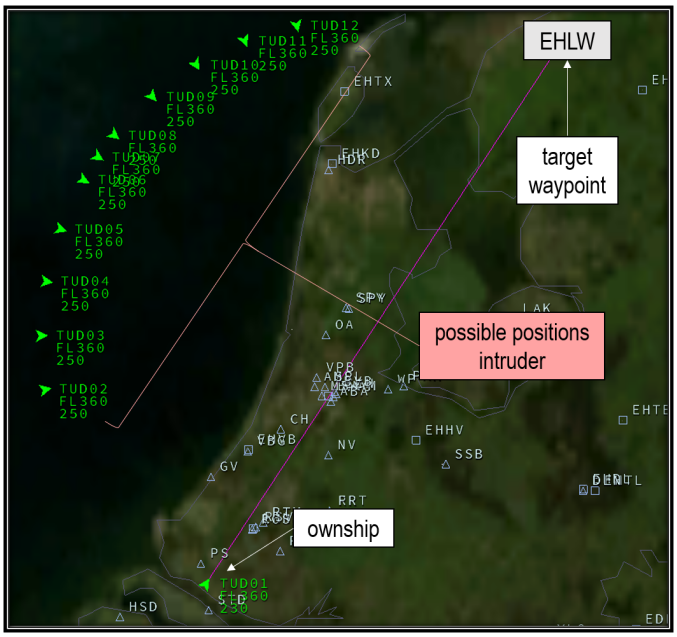


Figure 6. Experiment setup for case study 1. This figure shows the possible positions of the observed aircraft at the various conflict angles.

For this experiment, demonstrations are collected for only a

few of the conflicts that can occur. These are for the conflicts at conflicts angles: 45, 75, 90, 105, 135 degrees. This amounts to a total of 654 demonstrations in the demonstration replay buffer. In this learning environment, both a Dueling DQN and DQfD agent will be trained. The training phase of both agents are compared to see whether the agent is able to speed up the learning by utilising demonstrations.

To validate the performance of the learned policies, both agents will first be tested for a set of traffic scenarios it has been trained with. The reward obtained for these conflict scenarios will be compared as this is a direct measure for the performance of the agent. To research how the resolution method generalises, two experiments are performed. First of all, the trained policy of the DQfD agent is tested in traffic scenarios that it has not encountered during the training phase. The details of these conflicts are summarised below:

- Head-on conflicts: {145, 165, ..., 195, 215} [°]
- Crossing path: {225, 245, ..., 295, 315} [°]
- Velocities observed aircraft: {222, 230, 250} [kts]

Secondly, the trained policy will be tested for 1000 episodes in an environment in which an observed aircraft is initialised at a semi-random conflict angle, velocity and t_{CPA}. The ranges used are detailed below:

- Conflict angles: [45 − 135] [°]
- Velocity: [230 − 250] [kts]
- t_{CPA}: [200 − 400] [kts]

The values of the test set are chosen to ensure the aircraft encounters conflicts that resemble traffic scenarios from the training set (faster observed aircraft, crossing path angles). The *success rate* will be defined as the amount of episodes in which the RL agent reaches the exit waypoint without encountering a LOS.

D. Case Study 2: Incorporating Strategy with Demonstrations

In order to research how this automation can be used to incorporate strategy from ATCos, an experiment is set up in which a RL agent has to cope with a limited amount of conflicts. Only a limited amount of conflicts are used for this case study as training a RL agent is time consuming. Demonstrations are provided for the conflict angles: 45, 90, 135 degrees at an unbiased conflict. This amounts to a total of 458 demonstrations. A pre-training phase of 30,000 epochs was used to ensure a well-performing policy prior to the normal learning phase.

In this experiment setup, imperfect demonstrations are collected and used to pre-train the DQfD agent. This is done to research whether the agent improves on the demonstrations it has learned from. The MVP method is activated after 150 seconds to incorporate a strategy on the timing of the resolution and is restricted to taking steps of either +5 or -5 degrees. The trajectory of the demonstration for the conflict scenario of 45 degrees, provided by the adjusted MVP method, is shown in Figure 7.

An exploration rate of 0.01 is used to ensure the agent can explore the state-space along the trajectory of the pre-trained model and thereby also improve the demonstration.

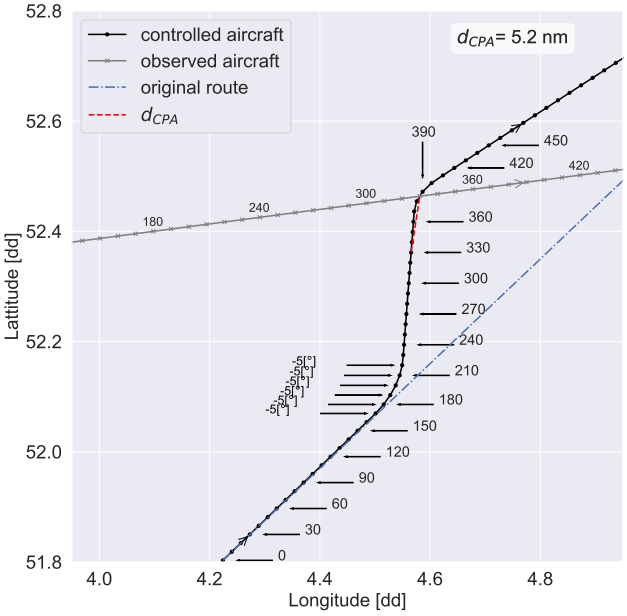


Figure 7. Imperfect demonstration used to pre-train the DQfD agent. The longitude and lattitude are shown in *decimal degrees* (dd).

Since the DQfD agent will start exploring from the policy that it has learned during the pre-training phase, a qualitative analysis on the conformity between the RL agent and the human demonstrator will be performed for the conflict angle at which the pre-trained model is most conformal with the demonstrations.

E. Case Study 3: Increasing Transparency for the Designer and User

To increase the transparency of the RL agent, Reward Decomposition (RDX) is implemented. According to Juozapaitis et al, who developed RDX, RDX can be used to spot anomalies in the automation and to expedite the explanation of action selection [15]. In this case study, a drDuel-DQN will be trained with three reward components. These are the components related to a LOS (SH - Short Term Conflict), taking 'nonzero' actions (NA) and the deviation from the target heading (TH). The model architecture is shown in Figure 8. In case study 1, it is chosen to use a high terminal t_{sim} to minimise its effect on the policy to which the RL agent converges. In this case study, the terminal t_{sim} is lowered to increase the amount of conflicts encountered per time unit. This is done to speed up the learning. The conflict is defined by the following characteristics:

- Conflict angles: {45, 90, 135} [°]
- t_{cpa}: 250 [s]
- d_{cpa}: 0 [nm]
- $CAS_{\text{controlled aircraft}}$: 230 [kts]
- $CAS_{\text{observed aircraft}}$: 250 [kts]
- Type of aircraft: B737
- $t_{sim_{final}}$: 600 [s]

To the best knowledge of the authors of this paper, this is the first implementation of RDX for a complex RL task. The

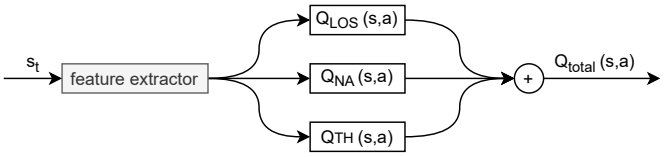


Figure 8. Model structure of the drDuel-DQN implemented.

results of this case study will focus on the ability of the agent to learn a well-performing policy. Furthermore, the added value of RDX will qualitatively be analysed by examining the RDX at various moments in time during conflict resolution.

F. Case Study 4: Incorporating Strategy Through Reward Shaping

In RL, the policy is shaped by the reward function. The goal of a RL agent is to learn a policy that maximise the reward it accumulates over the course of an episode. One of the personal strategies ATCos apply is related to the conflict geometry of the resolution [46]. Putting this research into a broader perspective in which the SSD representation is used in a multi-agent setting or in a hierarchical RL algorithm, one can tune the reward function to incorporate strategies. In this experiment setup, a drDuel-DQN agent will be trained to be conformal to a certain conflict geometry preference (CGP), expressed as either preferring to steer the controlled aircraft in front or behind the observed aircraft. In Figure 9, the various strategies an ATCo can apply to solve a right angle conflict are shown. Although steering the controlled aircraft in front of the observed aircraft, as shown in Figure 9(c), requires extra miles, it is a viable resolution strategy of an ATCo [46].

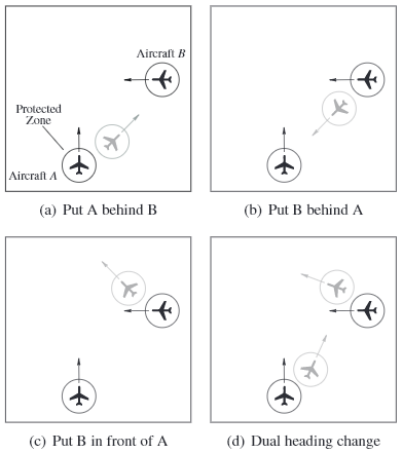


Figure 9. Conflict solution alternatives to a right angle conflict [46].

A component related to the CGP is added to the reward function as shown in Equation 15.

$$r(t) = r_{\text{conflict}}(t) + r_{\text{action}}(t) + r_{\text{Flight Path}}(t) + r_{\text{CGP}}(t) \quad (15)$$

It is decided to positively reward the agent for resolving the conflict in conformance with the CGP. A positive or negative reward will be given once the conflict has been solved, when the aircraft have passed each other. This reward is given if the distance between the controlled, aircraft A, and observed aircraft, aircraft B, increases for the first time, as shown in Equation 16.

$$r_{\text{CGP}}(t) = \begin{cases} +40, & \text{If } |\text{d}_{A,B}(t)| > |\text{d}_{A,B}(t-1)| \\ & \text{and } CG = CGP \\ 0, & \text{else} \end{cases} \quad (16)$$

Whether or not the conflict geometry is met can be determined by considering the initialisation of the conflict and by considering the d_{CPA} components in terms of longitude and latitude. The earlier case studies consider crossing path conflicts between 45 and 135 degrees in which the agent optimises for minimising the flight path. The observed aircrafts all have a higher initial CAS, meaning that the most efficient resolution would be to steer the controlled aircraft behind the observed aircraft. To show that certain strategies can be incorporated by shaping the reward function, a drDuel-DQN agent will be trained to avoid a conflict at a conflict angle of 90 degrees with the CGP set at steering the controlled aircraft in front of the observed aircraft.

IV. Results

A. Case Study 1 - Dueling DQN in Two-Aircraft Conflict

The results of this case study are focused on the ability of the RL agent to avoid a conflict for conflict scenarios it has been trained for and for those it hasn't. The RL agent was trained for 50,000 epochs, which amounts to 3024 episodes for the Duel-DQN agent and 2901 episodes for the DQfD agent. The difference in the amount of episodes is related to the variable episode length. The exploration rate has been lowered to 0.01 during the last 10,000 epochs of training for both RL agents to enhance the convergence of the training algorithm. During training, the Dueling DQN and DQfD agent acquired 200,000 frames and used 6.4 million experiences to update the network weights. The training phase of the Dueling DQN agent took 27 hours on a NVIDIA Tesla K80 GPU, whereas the DQfD agent took 32 hours.

In Figure 10, one can see the accumulated reward of the Duel-DQN and DQfD agent. It is important to note that the DQfD agent was first pre-trained for 30,000 epochs and afterwards trained for another 50,000 epochs. It can be seen that the pre-training phase of only 30,000 epochs on a limited data-set increased the data efficiency of the algorithm as the performance of the pre-trained agent is significantly higher during the first 750 episodes. Both agents did see a lot of new experiences during training as the exploration rate was kept relatively high at 0.05. This means the agent took an exploratory action once every 20 steps. The DQfD agent slightly outperforms the Duel-DQN agent by the time training is stopped.

The reward obtained, when testing the trained policies of the DQfD, Dueling DQN and the purely pre-trained model for conflicts it had encountered during the training phase, is

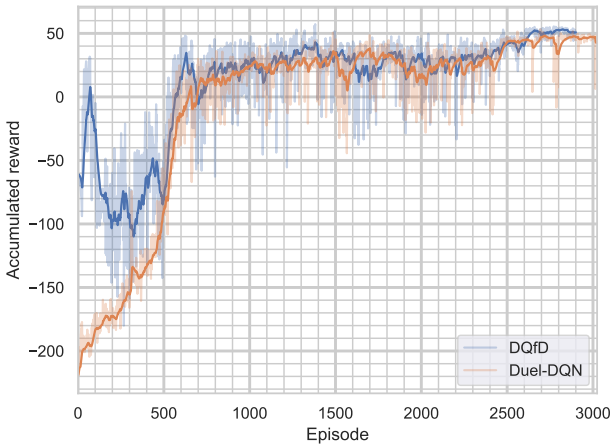


Figure 10. Accumulated reward obtained for the Dueling DQN and DQfD agent.

shown in Figure 11. For all conflict angles, both agents are able to avoid a conflict and reach the target waypoint. It can be seen that the DQfD agent outperforms the Dueling DQN agent for all conflict angles. Furthermore, one can see that the DQfD agent significantly improves its performance when it starts to acquire new experiences, after the pre-train phase. In Figure 12, the flight path distance per conflict angle is shown. One can see that the DQfD agent has learned a better performing policy, both in terms of accumulated reward and flight path, compared to the Dueling DQN agent. The negative correlation between the flight path distance and the reward confirms that the deviation from the target heading can indeed be used in the reward function to penalise for extra flight path distance. The trajectory of the controlled and observed aircraft at a conflict of 135 degrees is shown in Figure 13. The two agents have learned different policies to solve conflicts. This can mostly be explained by the fact that the DQfD agent takes an exploratory step once every 20 seconds, starting from a policy it has learned by pre-training on a demonstrations data set. The Dueling DQN agent, on the other hand, took many exploratory steps during the first 5000 epochs.

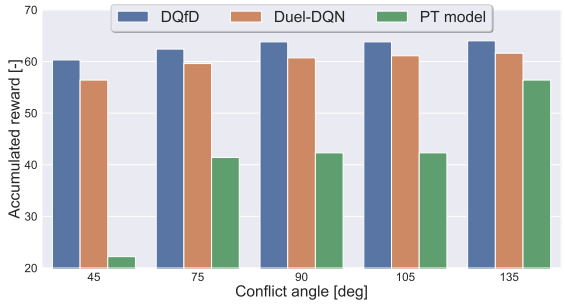


Figure 11. Comparison of the accumulated reward per conflict angle obtained in the testing environment using DQfD and Dueling DQN.

The trained policies were tested for all the possible conflict

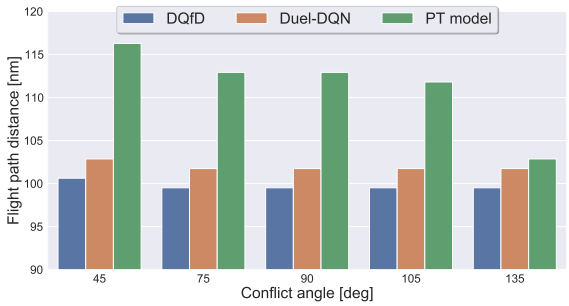


Figure 12. Flight path distance of the resolutions as provided by the DQfD, Dueling DQN and purely pre-trained model.

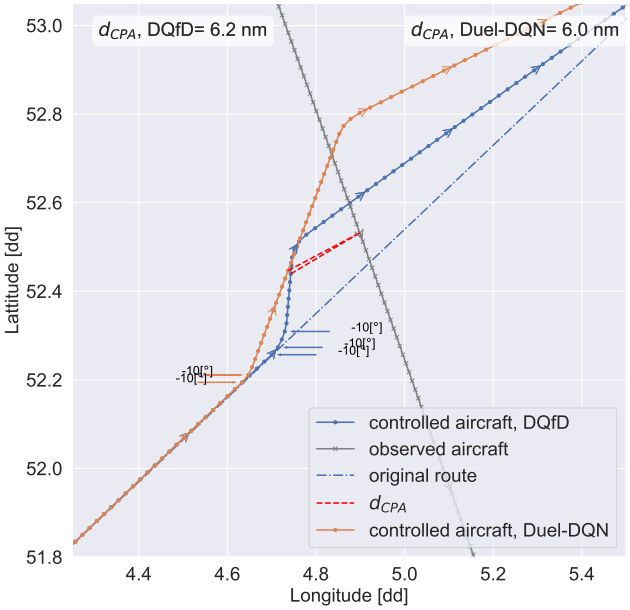


Figure 13. Flight path of controlled aircraft controlled by the DQfD and Dueling DQN model.

scenarios from the training set. For both policies, zero LOS were encountered and all aircraft were able to reach the Exit Waypoint.

The testing results of the DQfD agent for conflicts not in the training set are shown in Figure 14 in terms of flight path distance flown before reaching the exit waypoint. During the testing phase, the controlled aircraft did not reach the target waypoint twice, both at the boundary of the testing scenarios. A LOS occured at a conflict angle of 305 degrees, at a Calibrated Airspeed (CAS) of the observed aircraft of 250 kts. Furthermore, it did not reach the target waypoint within the given episode termination time (20 min) at a conflict angle of 45 degrees and speed of the observed aircraft of 222 kts. For all other conflicts the controlled aircraft reached the Exit Waypoint without a LOS. One can see that the RL agent solves conflicts it was trained for more efficiently in terms of flight path distance. This is because the agent has learned to steer behind the observed aircraft since it encountered only faster observed aircrafts during the learning phase. This 'steering

behind' strategy causes the RL agent to struggle how to efficiently resolve a conflict at a conflict angle of 45 degrees with a slower observed aircraft, and at a conflict angle of 305 degrees with a faster observed aircraft.

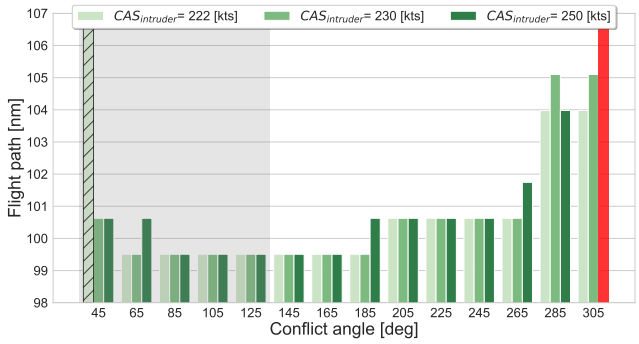


Figure 14. Flight path distance of the controlled aircraft during testing for various velocities and conflict angles of the observed aircraft (intruder). The grey area indicates the range of conflicts the RL agent has been trained for; the red bars indicate that a LOS occured; the striped bars indicate that the controlled aircraft did not reach the exit waypoint within time constraints (1200s).

When the model was put to the test for 1,000 episodes with semi-randomly initialised conflicts, it showed a *success rate* of 100 %. This highlights the robustness of the model for conflicts that share characteristics with the traffic scenarios encountered in the training set.

B. Case Study 2 - Incorporating Strategy with Demonstrations

In case study two, the DQfD agent has been pre-trained for 30,000 and trained for another 40,000 epochs. Experiments showed that after 30,000 pre-train steps, the model has learned a strategically conformal policy for a conflict at a conflict angle of 45 degrees. For the conflict angles of 90 and 135 degrees, the model is able to avoid a conflict, but it is not yet conform with the demonstrator. Therefore, it is chosen to consider the traffic scenario with a conflict at 45 degrees conflict angle to analyse the degree of conformance between the policy learned by the DQfD agent and the demonstrator.

In Figure 15 and Figure 16, the resolution manoeuvres provided by the demonstrator, pre-trained model (PT) and DQfD agent are visualised in terms of the conflict angle and d_{CPA} between the observed aircraft and the controlled aircraft. In this figure, the relation between the time to the closest point of approach and the simulation time, t_{CPA}-t_{SIM}, has been fixed at the initialisation of the conflict. This is done since the t_{CPA} actually can increases up to infinity during the conflict resolution. It can be noticed that all agents take a set of actions to avoid a conflict at a similar t_{CPA} (4:20, 4:00, 3:50 [min]), of which the action selection is shown in Equation 17.

$$\text{AS1} = \begin{cases} [-5°, -5°, -5°, -5°, -5°, -5°], & \text{demonstrator} \\ [-5°, -5°, -5°, -5°, -5°, -5°] & \text{PT} \\ [-10°, -10°, -5°], & \text{DQfD} \end{cases} \tag{17}$$

The purely pre-trained RL agent mimics the demonstrator accurately in terms of timing and action selection. The total reward obtained by the DQfD agent for this episode is 60.5, whereas the demonstrator obtained a reward of 47.8 and the pre-trained model obtained a reward of 44.8. This difference is mostly due to the amount of nonzero actions taken by the demonstrator and the resolution which is slightly less efficient in terms of flight path flown.

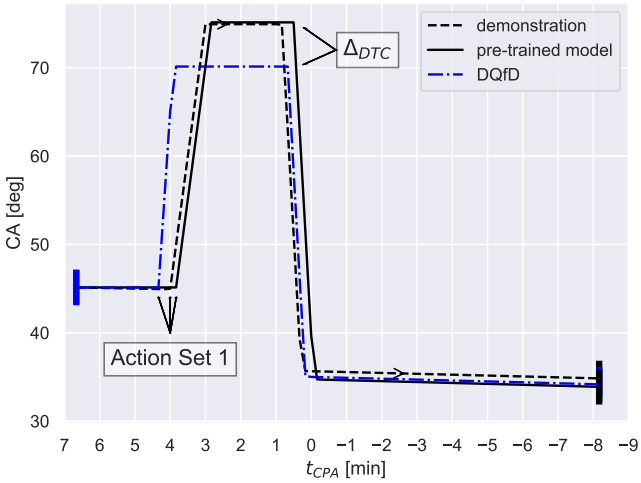


Figure 15. Comparison of conflict resolution by demonstrator, the DQfD agent after pre-training and the DQfD agent trained for another 40,000 epochs after pre-training in terms of conflict angle (CA).

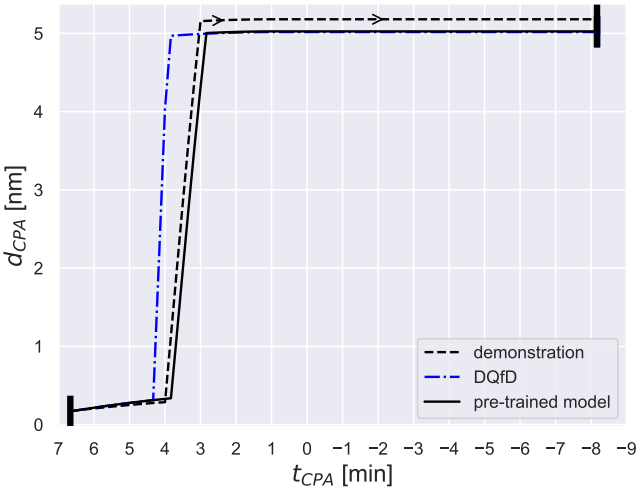


Figure 16. Comparison of conflict resolution by demonstrator, the DQfD agent after pre-training and the DQfD agent trained for another 40,000 epochs after pre-training in terms of d_{CPA}.

Whereas the demonstration data contained multiple actions of -5 degrees, the RL agent has learned to take steps of -10 degrees instead. The RL agent has actually learned to improve upon the demonstrations. The combination between the low exploration rate of 0.01 and a high resolution action space has restricted the agent to explore totally different trajectories. Moreover, the expert loss component forces the RL agent to follow demonstrations as much as possible. The agent seems to maximise the reward it can accumulate by

exploiting information it retrieves from exploratory actions along the policy of the pre-trained model.

C. Case Study 3 - Added Value of Decomposed Rewards

For a convergent deep RL algorithm, the loss decreases steadily over time. As this is the first research known to combine Duel-DQN and reward decomposition, it is of interest to see whether it can actually converge to a well-performing policy. In Figure 17, the loss during the training phase is visualised. As expected, the loss decreases over time and the algorithm seems to converge to a (sub-)optimal policy. The absolute values of the loss function are often difficult to interpret since the designer is not aware which experiences from the memory replay are used during an optimisation step. When the optimiser updates on sparsely encountered experiences, the loss is likely to be higher. To gain more insight into whether the agent learns what is expected to learn, the individual loss components of the decomposed rewards can be considered. The loss components related to getting into a LOS and taking nonzero actions decreased steadily during epoch updates. However, the loss component related to following the target heading, shown in Figure 18, shows unstable behaviour. This highlights how RDX can be utilised to gain more detailed insights into how the RL agent is actually learning. Although the loss seems to be relatively unstable, the RL agent has still learned a policy which is able to avoid conflicts and direct the aircraft to the exit waypoint.

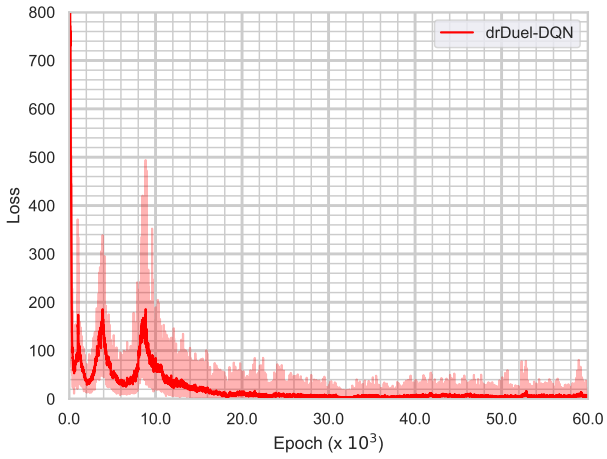


Figure 17. Loss of drDuel-DQN during the training period. The dark line represents the weighted average of the loss.

In Figure 19, one can see the trajectory of the controlled aircraft controlled by the policy learned using a drDuel-DQN for a conflict initiated at a t_{CPA} of 250 seconds and a conflict angle of 45 degrees. To avoid a conflict, the agent takes four consecutive actions of -10 degrees and then awaits the resolution of the conflict before continuing its track towards the exit waypoint.

In Figure 20, one can see the decomposed reward values for each action in the action space at a simulation time of

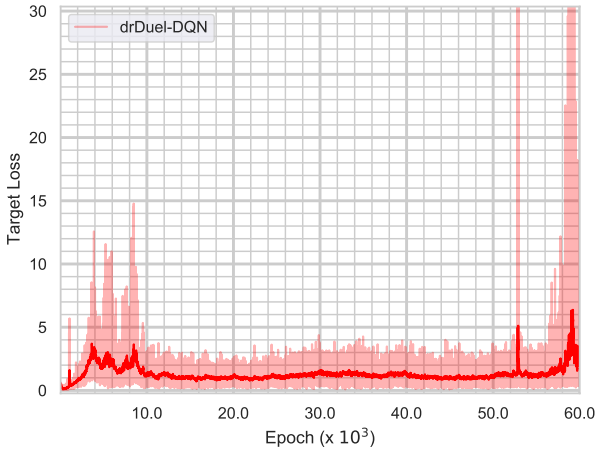


Figure 18. Loss related to following the target heading ('TH').

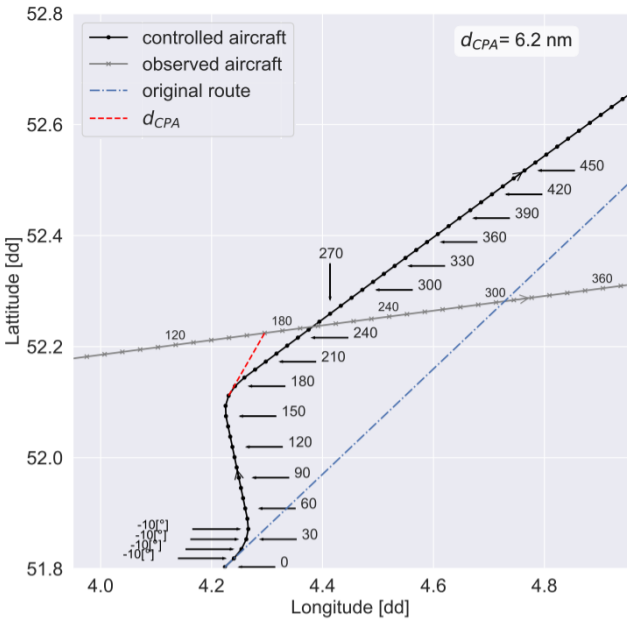


Figure 19. Trajectory of controlled and observed aircraft for a conflict at 45 degrees using the trained policy from Dueling DQN agent with reward decomposition.

10 seconds. Although the loss related to the 'TH' component seems relatively fluctuant, the agent seems to have learned the value component related to the deviation of the target heading. As expected, the actions '0 degree heading change' and 'DCT' have the highest expected reward since the deviation from the target heading will then remain zero. The reward component for the actions steering behind the observed aircraft (-10, -5 degrees) have a higher value for this component than the actions steering the aircraft in front of the observed aircraft. This is also what is expected as the latter results in a larger path deviation than the former. This same relationship can be found when looking at the value component for nonzero actions (NA). Steering the controlled aircraft in front of the observed

aircraft would require more actions to avoid the conflict than steering the controlled aircraft behind, which is reflected in the RDX. The most negative value related to a short term conflict (SC) is for the action '-5 degrees'. This would steer the aircraft into the conflict and require more time to avoid the conflict. Therefore, the agent has learned that taking this action most likely results in a conflict.

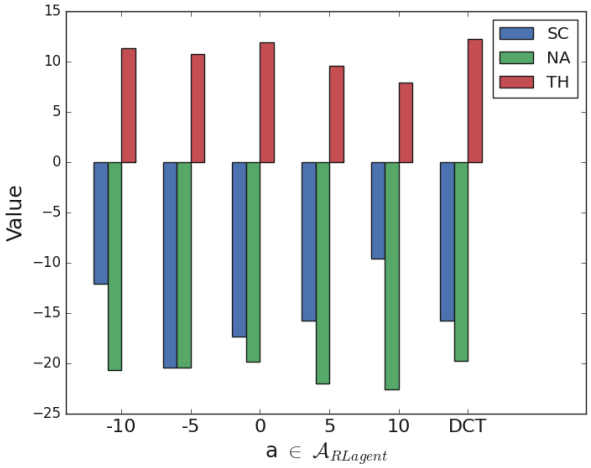


Figure 20. Reward decomposition for the components related to getting into a short term conflict (SC), taking nonzero actions (NA) and following the target heading (TH) at t_{sim} of 10 seconds.

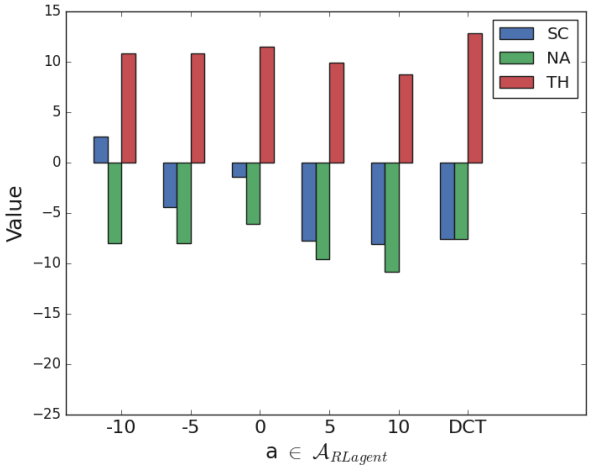


Figure 21. Reward decomposition for the components related to getting into a short term conflict (SC), taking nonzero actions (NA) and following the target heading (TH) at t_{sim} of 40 seconds.

One can see that for all the actions, the agent has learned that the expected return for a short term conflict is negative. In Figure 21, the RDX at t_{sim} = 40 seconds is visualised. What this figure shows is that the agent expects a positive return for the short term conflict. As the agent only acquires negative values for this reward component, this is not what is expected to occur for any state. This seems to be caused by an anomaly in the learning. In [15], Juozapaitis et al. identified a similar anomaly when training a decomposed RL agent in an environment with a relatively limited state and action space. Juozapaitis et al. explored why and concluded that the use of the Adam optimiser, due to its adaptive learning rate, can initially learn small positive values for actions leading to negative rewards. These errors propagate through the entire learning process as these values are used as target values during updates.

D. Case Study 4 - Using Reward Shaping to Mimic Strategy

In the final case study, the RL agent was trained to avoid a conflict and steer in front of the observed aircraft. The RL agent was trained for 30,000 epochs. The RL agent was trained for less epochs as the agent was only trained for a single traffic scenario. In Figure 23, the trajectory of the controlled aircraft controlled by the learned policy is shown. Whereas in case study 1 the agent has learned to steer behind the observed aircraft to minimise the flight path, it now steers in front of the observed aircraft. The RDX at a t_{sim} of 10 seconds, for

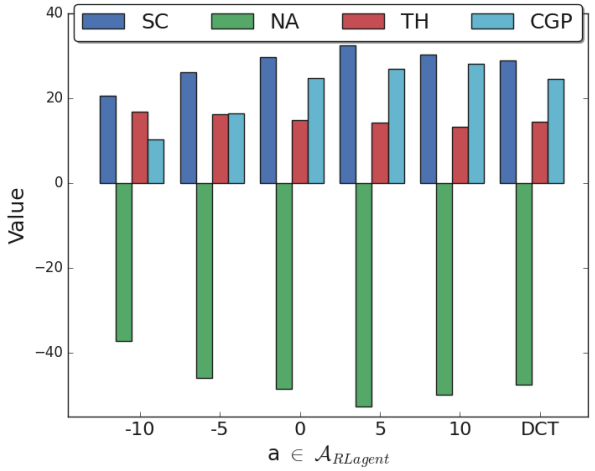


Figure 22. Reward decomposition for the components related to getting into a short term conflict (SC), taking nonzero actions (NA), following the target heading (TH) and being strategically conformal with the conflict geometry preference (CGP) for t_{sim} = 10 seconds in the flight path shown in Figure 23.

the resolution shown in Figure 23, is visualised in Figure 22. Similar to the results presented in subsection IV-C, the value function related to the short term conflict has an unexpected positive value. However, the other components do assist in understanding the reasoning of the agent. The actions to steer the controlled aircraft behind the observed aircraft (-10, -5) have a slightly higher value for the reward component related to the flight path, 'TH'. The value related to being conformal with the conflict geometry preference, 'CGP', is higher for the actions related to steering the aircraft in front of the observed aircraft. This shows that by shaping the reward function, the strategy can be incorporated in the automation. From the RDX, it becomes apparent that the value related to the CGP is much higher than that related to the flight path. In this experiment, a value of 40 was chosen to force strategic conformance. To trade-off conformance and flight path efficiency, one can use

the RDX to tune the reward component related to strategic conformance.

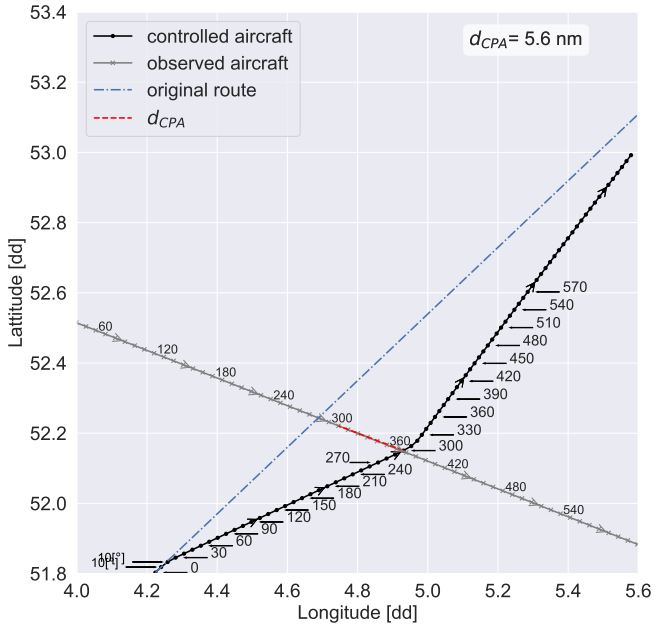


Figure 23. Flight path resolution provided by DQfD agent trained to steer in front of the observed aircraft.

V. Sensitivity Analysis

The case studies showed that the RL agent was able to resolve conflicts efficiently in terms of flight path distance for 'similar' conflicts. During the research, it was noticed that the loss became volatile when training the RL agent in a learning environment with diverse conflicts, e.g. conflicts at which the observed aircraft could have either a higher or lower velocity than the controlled aircraft. Therefore, the sensitivity analysis is focused on examining the stability of the Dueling DQN agent. To evaluate the stability, the performance of the Dueling DQN agent during training is evaluated.

A. Effect of Action Space Resolution

In the case studies, conflicts are encountered in which one set of resolution commands, {-10, -5}[deg] or {+5, +10}[deg], is preferred over the other. This behaviour was found to be optimal w.r.t. the conflicts encountered during training (CA: [45$\longrightarrow$135 deg], $CAS_{\text{observed aircraft}}$: 250 kts). To research the capability of the RL agent to learn a policy that is applicable to a wide range of conflicts, a RL agent is trained to avoid a conflict and steer in front of an aircraft for observed aircraft initiated at crossing path conflict angles coming from both the left and the right of the controlled aircraft. The optimal policy would learn how to consistently steer the controlled aircraft in front of the observed aircraft, and not prefer a set of solution over the other. The conflict angles the RL agent will be trained for are listed below:

$$\text{Conflict angles} : \{80, 90, 100, 260, 270, 280\}[deg] \quad (18)$$

After running an experiment, it was noticed that the loss of the RL agent was unstable in the limit. The RL agent struggled to find a policy in which it could actually learn to avoid a conflict and be conform with the conflict geometry preference (steer in front). It was hypothesised that the high resolution action space (HAS), shown in Equation 13, can cause this behaviour since exploratory actions do not significantly affect the trajectory, especially if an exploratory action is taken once every 20 steps (ϵ = 0.05). To research whether the HAS destabilises the learning in a setting with a large variety of conflicts, a RL agent was trained for the same conflicts, but with a wider action space, thus of lower resolution (LAS). The action space used in this experiment is shown in Equation 19. Both RL agents were trained for 75,000 epochs.

$$\mathcal{A}_{\text{RL agent}_{\text{low reso}}}(s) = \{-30, -15, 0, 15, 30, \Delta_{\text{TH}}\}[^\circ] \quad (19)$$

The accumulated reward during training is visualised in Figure 24. One can see that the agent with the LAS learns to avoid conflicts faster. Furthermore, the rewards accumulated are less volatile for the agent with the LAS.

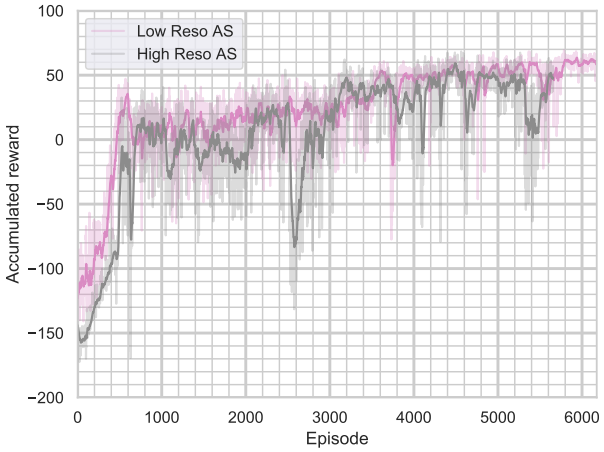


Figure 24. Accumulated reward of the RL agent trained with a high and low resolution action space.

The average loss per episode is shown in Figure 25. The loss of the RL agent with the LAS space converges whereas the RL agent with the HAS space remains unstable.

The conflict geometry of the resolution provided by the RL agent during training with HAS and LAS are shown in Figure 26 and Figure 27. One can see that the LAS RL agent resolves more conflicts, and does so conform with the CGP, compared to the HAS RL agent.

In most RL problems, the action space is chosen based on intuition [50]. In this research the narrow action space was selected to enable ATCo conformal resolutions, but the effect of using a narrow action space was initially not deemed relevant. This analysis stresses the importance of tuning the action space.

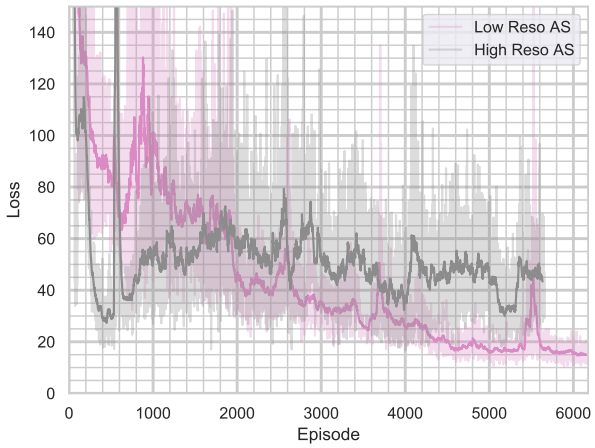


Figure 25. Loss during training of the lower and higher resolution action space.

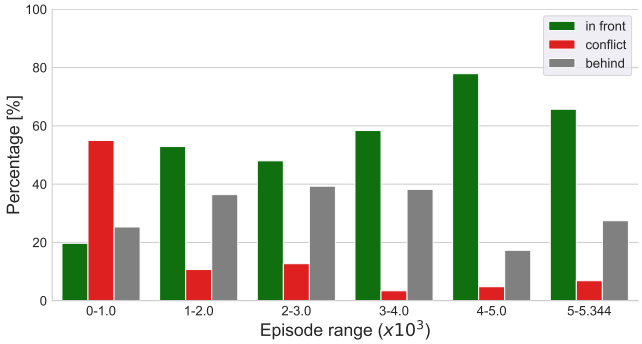


Figure 26. Conflict geometry during episode high resolution action space.

VI. Discussion

The discussion will address five different subjects: subsection VI-A on the method's applicability in the current ATM system, subsection VI-B on the added value of demonstrations, subsection VI-C on strategy replication, subsection VI-D on explainability and, finally, subsection VI-E on the SSD as state representation for an artificial agent.

A. Applicability in ATM System

ATCos perform CD&R by identifying potential conflicts pairwise and solving these sequentially. Results in this research support the hypothesis that a RL agent can perform CD&R for two-aircraft conflicts by having the SSD to represent the state of the conflict pair. Although the RL agent in the first case study was trained to avoid a LOS and reach the exit waypoint for a limited number of different conflicts, it is able to generalise its policy to unseen traffic scenarios.

In realistic air traffic sectors, solving one conflict can disrupt the air sector and lead to multiple other conflicts. This effect has been disregarded in the two-aircraft traffic scenario. ATCos become experts on performing CD&R in a specific air traffic sector. When training a RL agent to solely solve conflicts in one specific air traffic sector, the authors believe

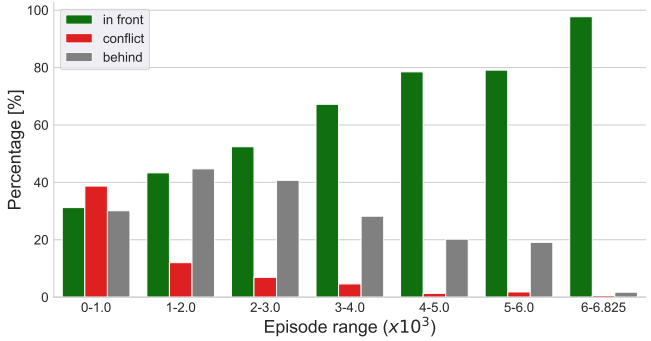


Figure 27. Conflict geometry during episode low resolution action space.

that the RL agent can also learn how to minimise sector disruption. A metric to measure sector disruptiveness is called the Domino Effect Parameter (DEP) [51]. Further research should investigate how this parameter can be incorporated in the reward function.

In particular traffic scenarios, an ATCo can prefer to provide both aircraft in a conflict with a resolution command. Especially for conflicts in which commanding a single aircraft to perform a manoeuvre would lead to a large path deviation or sector disruption, combined manoeuvres are the preferred solution. Transitioning to Future Air Navigation Services (FANS) in which the air sectors will be more crowded, it is expected that these type of manoeuvres are critical to ensure safe and efficient air traffic operations. With the current implementation of the automation, these type of conflict resolutions are not incorporated. A possible solution is to expand the action space of the RL algorithm to incorporate actions commanding both aircraft at the same time. For a successful implementation, the SSD of the observed aircraft must also be incorporated in the state representation for the RL agent. This enables the RL agent to learn in a Markov Decision Process, the underlying state of which is fully visible.

The scope of this research did not include the selection of which aircraft should perform a resolution manoeuvre. This part of the CD&R process can be seen as a centralised coordination mechanism. In sectors with an increased traffic density, this part of CD&R will become more difficult. The entire ATC control task can be approached from a decentralised as well as a centralised point of view. In a decentralised setting, multi-agent RL, in which the state of each aircraft is represented by the SSD, can be an interesting area of research. It is, however, believed that a centralised mechanism allows the ATCos to monitor the automation, since resolution actions are taken sequentially. To research whether the automation can be implemented in a realistic air traffic sector, an additional artificial agent should be trained to perform aircraft selection.

Finally, to apply this method in the current ATM system, altitude changes cannot be disregarded. Since altitude changes are the preferred resolution action [18], a separate RL agent can be trained to initiate an altitude change if deemed necessary. If an altitude change is not possible, the separately trained RL agent for 2D CD&R can resolve the conflict.

B. Added Value of Demonstrations

The results show that pre-training on demonstrations can significantly speed up learning. Especially in environments in which rewards are sparse, pre-training on demonstrations provides a viable alternative to the extensive exploration phase that is often required. For case study 1, only 654 demonstrations were used to pre-train the DQfD agent.

Apart from data-efficiency advantages, demonstrations enable the RL agent to converge to a better performing policy, after having trained for the same number of epochs, compared to a RL agent without pre-training. In case study 1, both agents were trained with the same hyperparameters. Results show that the DQfD agent achieves a more optimal policy with regard to minimising the flight path and the number of resolution commands. This relates to the fact that the RL agent can take exploratory steps from an already well-performing policy. The Dueling DQN agent optimises starting from randomly acquired data points and optimises towards a sub-optimal policy. In theory, the Dueling DQN agent should be able to learn a similar performing policy, but this would require an extensive exploration period. By implementing DQfD as was done in this research, with an increased probability for sampling demonstrations (just as in the original paper on DQfD), the agent will continuously learn from the demonstrations and shape its policy towards these. As this might not be desired in particular learning environments, demonstrations could be removed after a while in favour of having more novel experiences in the replay buffer.

Disadvantages of DQfD are that it adds additional hyperparameters to tune and requires expert demonstrations to be effective, as the algorithm is sensitive to noisy demonstrations [37], [52]. In this research, consistent demonstrations were acquired using an altered MVP method. One of the issues raised in researches into strategically conformal automation for ATCos is that ATCos are inconsistent in the strategies they employ over time [46]. It, therefore, remains questionable whether human generated demonstrations can benefit the RL agent using DQfD. To solve for this issue, Gao et al. and Jing et al. have, respectively, developed *Normalised Actor-Critic* (NAC) and *Reinforcement Learning from Demonstrations* (RLfD) which have both shown to be robust to suboptimal demonstrations [52]. Future research focused on human demonstrations is encouraged to explore implementing either NAC or RLfD.

C. Strategy Replication

Results show that demonstrations can be utilised to increase the strategic conformance between a human operator and the automation. Important parameters for the degree to which the automation is strategically conformal to the demonstrator are the number of demonstrations present in the replay memory, the pre-training phase, the exploration rate and the sampling probability of demonstrations. This research shows that strategic conformity can be increased through DQfD for traffic scenarios in which the pre-trained model is conform with the demonstrations and exploration is kept at a minimum. Whereas supervised learning can learn such a model in a limited amount of epochs, as shown by Van Rooijen [5], DQfD needs an extensive pre-training period as it also has to satisfy the Bellman Equation. Due to limited computational resources, it was not realistic to pre-train a DQfD agent to be conform the demonstrations for a large variety of conflicts. The action space in this research is narrow, which restricts the agent to see totally different trajectories. This, in combination with the low exploration rate used, can be the reason why the final policy of the DQfD agent shows strategic conformance with the demonstrator.

Finally, DQfD enables the RL agent to improve on the demonstrations it has been trained with, but from this research alone it cannot be concluded how the degree of conformance and optimisation can be balanced. Future research into utilising demonstrations to develop conformal automation is encouraged to focus on how the conformance between the pre-trained model and the demonstrations can be increased, and how to trade-off conformance and optimal control by tuning the hyperparameters of this algorithm.

In addition to using demonstrations to increase strategic conformance between an ATCo and the automation, reward shaping can be utilised. Results in this paper show how conflict geometry preference can be forced as resolution strategy of the automation. A next step to explore the potential of reward shaping is to link a performance measure, such as the deviation from the flight path, directly to strategic conformance. Being strategically conformal is desired in conflicts where the deviation from the flight path, as a result of following a certain strategy, is minimal. Future research is encouraged to explore how the reward function can be shaped to trade-off strategic conformance and optimal control.

D. Explainability

Explainability in this research has been focused on increasing the 'what will it do' prediction and improving the transparency of the RL algorithm. Results show that reward decomposition can increase the transparency of the RL algorithm for the designer as it enabled the authors of this paper to spot an anomaly in the learning and enabled real-time evaluation of the decision process of an agent. It is unknown to the authors how the identified anomaly would have been spotted without reward decomposition.

The reward function in this research contained three components, four in case of the conflict geometry preference incorporation. From the decomposed rewards, it could be seen that the value component related to 'nonzero action' in general had a higher magnitude than the reward component related to the flight path. This shows that the agent has optimised towards a policy in which it prefers to minimise the number of resolution actions over the flight path efficiency. For RL applications in which the reward function is composed of many components, the RDX can provide insights as to what the agent has learned to value, in terms of decomposed rewards, in any situation. This information can be used for the designer to tune the reward function and verify the functioning of the RL

agent. Especially for RL applications which implement direct and discounted rewards (rewards given at multiple timesteps), this information can be insightful.

In this research, finding the correct hyperparameters to decrease the loss has shown to be challenging. Although the agent is able to find reasonably well-performing policies, the loss remains volatile. This research shows that RDX can be used to analyse the performance of the RL agent in more detail. The authors encourage research to focus on how RDX can be utilised to link some performance measure to individual reward components. In [53], Clements et al. have developed a method to estimate risk and uncertainty based on the variance over the learned network weights (θ). Using this method on weight parameters related to the decomposed weight parameters can provide the designer with useful information about the certainty of a model w.r.t. the components in the reward function.

Finally, a first step to expedite action selection of a RL agent, trained for CD&R, has been taken in this research. The hierarchy between the decomposed values can assist in constructing an understanding of why an agent prefers a particular action over others. In RL applications that require complex reward functions, Minimal Sufficient Explanations (MSX) can be constructed from the RDX to indicate critical positive and negative reasons, respectively, for the preference of an action over another one [15]. This work did not research to what extend RDX can assist in improving the mental model a user has of the automation. The loss remained volatile, due to which the decomposed value functions do not accurately represent the true value. If these value approximations become more accurate, it is believed that these value functions can directly be linked to parameters interpretable for ATCos. The reward component related to minimising the flight path distance could then be linked to additional flight path flown. This allows an explanation interface in which not values, but human interpretable parameters back-up action selection.

E. SSD State Representation

1) Pre-Processing SSD: In this study, two stacked SSDs were used to represent the state of the conflict pair. Results in this research show that the SSD contains all the information required for an artificial agent to learn a CD&R policy that minimises the flight path. In the SSD, the no-go zone contains the sets of heading and speed combinations of the controlled aircraft that lead to a LOS. It was chosen to enable the RL agent to take actions into the no-go zone as briefly steering into a conflict can increase the efficiency of the resolution in terms of flight path. The RL agent did indeed steer into the conflict as the first action taken to avoid a conflict was not at the start of the episode ($t_{sim} < 50$ seconds) but at a later stage. This implies that the agent is capable of extracting information on the t_{CPA} from the translation, the width and the colouring of the FBZs. The agent has identified the FBZ as feature in the SSD as it generalises well to unseen traffic scenarios.

One of the issues encountered in this research was that the RL agent did not seem to be able to learn a well-performing policy for conflict scenarios in which the observed aircraft could either have a larger or lower velocity than the controlled aircraft. An analysis to compare representing the state of the RL agent by either the full SSD or the upper half of the SSD showed that using the upper half of the SSD increased the data-efficiency, without losing performance. The limitation of this analysis might have been that the traffic scenarios were not diverse enough. Revisiting features contained in the SSD, as shown in Figure 2, one can see that the velocity of the observed aircraft is contained in the relative position of the tail of the FBZ w.r.t. the centre of the SSD. With the current state representation, the FBZ related to the observed aircraft is only shown for the velocities within the flight envelope. Information on the velocity of the observed aircraft is, therefore, lost if the velocity of the observed aircraft is lower or equal to V_{min} of the controlled aircraft. This information is also lost whenever the tail of the FBZ is not present in the upper half of the SSD, with the speed vector pointing upwards, but contained in the lower half. It is hypothesised that this loss of information causes the RL agent to struggle how to learn a well-performing policy for diverse conflicts. Future research should focus on the effect of pre-processing the SSD image to benefit a RL agent.

2) Improving the State Representation: In all case studies, the RL agent was solely trained to avoid conflicts for two-aircraft traffic scenarios. A practical limitation of the SSD for human operators is that in complex traffic scenarios with multiple observed aircraft, the SSD becomes too crowded which reduces the solution space to a limited number of speed and heading combinations. The FBZs of different observed aircraft overlap which causes information on the approaching conflict to be lost. This same negative correlation between the utility of the SSD for ATCos and an overly crowded SSD holds for artificial agents. The parameter affecting the number of FBZs visible in a SSD is the look-ahead time. A possible solution is to include an algorithm, which can be deterministic, which prioritises the upcoming conflicts. Another solution to this problem can possibly be found in more advanced representations of the SSD. Using the theory presented by Velasco et al. [54], one can calculate the time-to-loss of separation per velocity vector in the FBZ and use this to colour the SSD per pixel. This colour coding, of which the construction is visualised in Figure 28, provides more detailed information for the RL agent on whether it can briefly steer into an upcoming conflict whilst FBZs overlap.

VII. Stabilising Training in Deep RL

Deep RL algorithms are notoriously unstable during training [55], [56]. In this research, training a RL agent until it showed convergent behaviour was challenging. The sensitivity analysis performed in this research has shown that the width of the action space significantly impacts the ability of the RL agent to find a convergent policy. However, this is not the only aspect of the RL problem that influences the stability of the algorithm. In [47], Fedus et al. stress the importance of scaling the replay memory buffer size according to how sensitive the

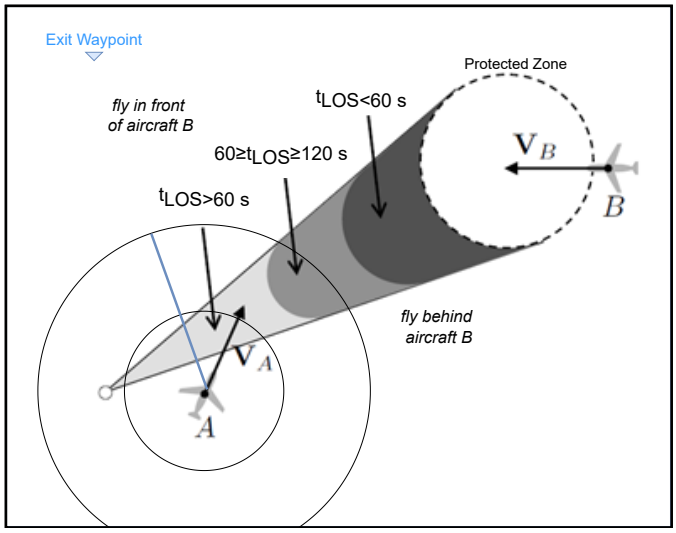


Figure 28. Features in the SSD with colour coding dependent on t_{LOS}.

network is to catastrophic forgetting. In this research, the RL agent consistently learned to avoid a conflict for a certain period of time, but then started to encounter conflicts again. As experiences of getting into a conflict are sparse, the size of the experience replay may be responsible for this behaviour. There remain significant gaps in the community's understanding of fundamental principles on which deep RL methods rely. Therefore, future research is encouraged to research and map the effects of hyperparameters on the stability of deep RL algorithms in detail.

VIII. Conclusions

This work explores how deep reinforcement learning can be applied to Air Traffic Control with an aim to contribute to the explainability of the automation. Specifically, Deep Q-learning from Demonstrations with a Dueling Network architecture and Dueling DQN with decomposed rewards are used to train two different RL agents to perform 2D CD&R. The solution space diagram is used to represent the state of a conflict pair for 2D conflict detection and resolution. Achieved model performance shows that both deep RL agents can learn all relevant features to resolve a conflict, effectively in terms of flight path minimisation, from the SSD. The Deep RL agents identify the no-go areas in the SSD, in terms of velocity and heading, enabling it to generalise to unseen traffic scenarios.

Pre-training on artificially generated demonstrations speeds up the learning. In this research, suboptimal demonstrations are used to pre-train the RL agent. Results show that the RL agent can improve on the demonstrations it has been trained with. Nonetheless, by visual inspection of the learned strategy, this research shows demonstrations can be utilised to increase strategic conformance as the RL agent starts to optimise from a semi-supervised pre-trained model.

Reward decomposition is combined with a Dueling DQN agent to contribute to the explainability of the RL agent for the designer. The CD&R task can naturally be decomposed into meaningful components related to minimising the flight path distance and avoiding a conflict. Learning decomposed values enables unique insights into what the RL agent has actually learned. This provides the opportunity to shape the reward function in detail and identify strange behaviour of the RL agent, highlighting the potential of reward decomposition to contribute to the explainability of the automation for designers.

References

[1] EUROCONTROL, "Model for Task and Job Descriptions of Air Traffic Controllers. European Air Traffic Control Harmonisation and Integration Programme." ECAC, Tech. Rep., 1996.

[2] B. Beernink, C. Borst, J. Ellerbroek, R. Van Paassen, M. Mulder, B. Beernink, and V. Paassen, "Toward an Integrated Ecological Plan View Display for Air Traffic Controllers," in *International Symposium on Aviation Psychology*, 2015, pp. 55–60. [Online]. Available: https://corescholar.libraries.wright.edu/isap_2015https://corescholar.libraries.wright.edu/isap_2015/98

[3] C. Westin, C. Borst, and B. Hilburn, "Strategic Conformance: Overcoming Acceptance Issues of Decision Aiding Automation?" *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 41–52, 2 2015.

[4] R. M. Regtuit, C. Borst, E. J. van Kampen, and M. M. van Paassen, "Building strategic conformal automation for air traffic control using machine learning," in *AIAA Information Systems-AIAA Infotech at Aerospace, 2018*, no. 209989. American Institute of Aeronautics and Astronautics Inc, AIAA, 2018.

[5] S. J. Van Rooijen, "Personalized Automation for Air Traffc Control using Convolutional Neural Networks," Master's thesis, Technical University of Delft, 2019. [Online]. Available: http://repository.tudelft.nl/.

[6] D. Gunning, "Explainable Artificial Intelligence (XAI)," Tech. Rep., 2017. [Online]. Available: https://www.cc.gatech.edu/~alanwags/DLAI2016/(Gunning)%20IJCAI-16%20DLAI%20WS.pdf

[7] M. Ribeiro, J. Ellerbroek, and J. Hoekstra, "Review of conflict resolution methods for manned and unmanned aviation," *Aerospace*, vol. 7, no. 6, 6 2020.

[8] R. Puca, E. J. Van Kampen, C. Borst, M. Tielrooij, and Q. P. Chu, "Experience-based AI methods for ATC decision-making support," Master's thesis, Delft University of Technology, 2014.

[9] M. Brittain and P. Wei, "Autonomous Aircraft Sequencing and Separation with Hierarchical Deep Reinforcement Learning," *ICRAT*, 2018.

[10] Marc Brittain and Peng Wei, "Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach," *CoRR*, vol. abs/1905.01303, 2019. [Online]. Available: http://arxiv.org/abs/1905.01303

[11] M. Brittain, X. Yang, and P. Wei, "A Deep Multi-Agent Reinforcement Learning Approach to Autonomous Separation Assurance," 3 2020. [Online]. Available: http://arxiv.org/abs/2003.08353

[12] D. T. Pham, N. P. Tran, S. K. Goh, S. Alam, and V. Duong, "Reinforcement learning for two-aircraft conflict resolution in the presence of uncertainty," in *RIVF 2019 - Proceedings: 2019 IEEE-RIVF International Conference on Computing and Communication Technologies*. Institute of Electrical and Electronics Engineers Inc., 5 2019.

[13] N. P. Tran, D. T. Pham, S. K. Goh, S. Alam, and V. Duong, "An intelligent interactive conflict solver incorporating air traffic controllers' preferences using reinforcement learning," in *Integrated Communications, Navigation and Surveillance Conference, ICNS*, vol. 2019-April. Institute of Electrical and Electronics Engineers Inc., 4 2019.

[14] L. Pack Kaelbling, M. L. Littman, A. W. Moore, and S. Hall, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[15] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, "Explainable Reinforcement Learning via Reward Decomposition." International Joint Conference on Artificial Intelligence, 2019.

[16] B. Kirwan and M. Flynn, "Investigating Air Traffic Controller Conflict Resolution Strategies," EUROCONTROL, Tech. Rep., 2002.

[17] International Civil Aviation Association, *Doc 4444: Procedures for Air Traffic Management*, 16th ed., 2016. [Online]. Available: www.icao.int.

[18] E. M. Rantanen and A. Nunes, "Hierarchical Conflict Detection in Air Traffic Control," *International Journal of Aviation Psychology*, vol. 15, no. 4, pp. 339–362, 2005.

[19] A. Bisseret, "Application of signal detection theory to decision making in supervisory control: The effect of the operator's experience," *Ergonomics*, vol. 24, no. 2, pp. 81–94, 2 1981. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/00140138108924833

[20] D. J. Law, J. W. Pellegrino, S. R. Mitchell, S. C. Fischer, T. P. McDonald, and E. B. Hunt, "Perceptual and cognitive factors governing performance in comparative arrival-time judgments." *Journal of Experimental Psychology: Human Perception and Performance*, vol. 19, no. 6, pp. 1183–1199, 1993. [Online]. Available: http://doi.apa.org/getdoi.cfm?doi=10.1037/0096-1523.19.6.1183

[21] A. BISSERET, "Analysis of Mental Processes Involved in Air Traffic Control," *Ergonomics*, vol. 14, no. 5, pp. 565–570, 9 1971. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/00140137108931276

[22] U. Metzger and R. Parasuraman, "Effects of automated conflict cuing and traffic density on air traffic controller performance and visual attention in a datalink environment," *International Journal of Aviation Psychology*, vol. 16, no. 4, pp. 343–362, 2006.

[23] X. Xu and E. M. Rantanen, "Effects of air traffic geometry on pilots' conflict detection with cockpit display of traffic information," *Human Factors*, vol. 49, no. 3, pp. 358–375, 6 2007.

[24] C. D. Wickens, J. Hellenberg, and X. Xu, "Pilot Maneuver Choice and Workload in Free Flight," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 44, no. 2, pp. 171–188, 6 2002. [Online]. Available: http://journals.sagepub.com/doi/10.1518/0018720024497943

[25] T. L. Seamster, R. E. Redding, J. R. annon, J. M. Ryder, and J. A. Purcell, "Cognitive Task Analysis of Expertise in Air Traffic Control," *The International Journal of Aviation Psychology*, vol. 3, no. 4, pp. 257–283, 1993.

[26] J. K. Kuchar and L. C. Yang, "A Review of Conflict Detection and Resolution Modeling Methods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, pp. 179–189, 2000.

[27] D. van der Hoff, "A Multi-Agent Reinforcement Learning Approach to Air Traffic Control," Master's thesis, Technical University of Delft, 2020. [Online]. Available: http://repository.tudelft.nl/

[28] S. B. Van Dam, M. Mulder, and M. M. van Paassen, "Ecological interface design of a tactical airborne separation assistance tool," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 38, no. 6, pp. 1221–1233, 2008.

[29] P. Hermes, M. Mulder, M. M. van Paassen, J. H. L. Boering, and H. Huisman, "Solution-Space-Based Complexity Analysis of the Difficulty of Aircraft Merging Tasks," *Journal of Aircraft*, vol. 46, no. 6, pp. 1995–2015, 11 2009. [Online]. Available: https://arc.aiaa.org/doi/10.2514/1.42886

[30] Y. Huo, D. Delahaye, and Y. Wang, "Sensitivity Analysis of Closest Point of Approach," in *ICRAT 2018, 8th International Conference for Research in Air Transportation*, Barcelone, Spain, Jun. 2018. [Online]. Available: https://hal-enac.archives-ouvertes.fr/hal-01823194

[31] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*. IEEE, 8 2017.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2 2015.

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 9 2016.

[34] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," 11 2016. [Online]. Available: http://arxiv.org/abs/1511.06581

[35] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *3rd International Conference for Learning Representations*, 12 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[36] S. Ruder, "An overview of gradient descent optimization algorithms," 9 2017. [Online]. Available: http://arxiv.org/abs/1609.04747

[37] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep Q-learning from Demonstrations," 4 2017. [Online]. Available: http://arxiv.org/abs/1704.03732

[38] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *International Conference on Machine Learning (ICML)*, 2016.

[39] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction*, second edition ed. Cambridge: The MIT Press, 2018.

[40] T. Schaul, J. Quan, I. Antonoglou, D. Silver, and G. Deepmind, "PRIORITIZED EXPERIENCE REPLAY." International Conference on Learning Representations (ICLR), 2016.

[41] J. Hoekstra and J. Ellerbroek, "BlueSky ATC Simulator Project: An Open Data and Open Source Approach," in *7th International Conference for Research in Air Transportation (ICRAT)*. ICRAT, 2016.

[42] J. Sun, J. Ellerbroek, J. Hoekstra, and J. M. Hoekstra, "OpenAP: An open-source aircraft performance model for air transportation studies and simulationstransportation studies and simulations," *Aerospace*, vol. 7, no. 8, 2020.

[43] G. Mercado Velasco, M. Mulder, and M. van Paassen, "Air traffic controller decision-making support using the solution space diagram," in *Proceedings of the 11th IFAC/IFIP/IFORS/IEA Symposium on Design, Analysis and Evaluation of Human-Machine Systems (IFAC-HMS 2010)*, W. Chul Yoon, Ed. IFAC, 2010, pp. 1–6, the 11th IFAC/IFIP/IFORS/IEA Symposium on Design, Analysis and Evaluation of Human-Machine Systems (IFAC-HMS 2010) ; Conference date: 31-08-2010 Through 03-09-2010.

[44] A. Amiranashvili, A. Dosovitskiy, V. Koltun, and T. Brox, "Motion perception in reinforcement learning with dynamic objects," 1 2019. [Online]. Available: http://arxiv.org/abs/1901.03162

[45] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[46] C. Westin, "Strategic Conformance Exploring Acceptance of Individual-Sensitive Automation for Air Traffic Control," Ph.D. dissertation, Technical University of Delft, 2017. [Online]. Available: https://doi.org/10.4233/uuid:49c6fe9d-2d29-420a-91a2-a97e2049e15e

[47] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 3061–3071. [Online]. Available: http://proceedings.mlr.press/v119/fedus20a.html

[48] R. Liu and J. Zou, "The effects of memory replay in reinforcement learning," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2018, pp. 478–485.

[49] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the Deep Q-Network," 11 2017. [Online]. Available: http://arxiv.org/abs/1711.07478

[50] A. Kanervisto, C. Scheller, and V. Hautamäki, "Action Space Shaping in Deep Reinforcement Learning," *IEEE Conference on Games 2020*, pp. 479–486, 4 2020. [Online]. Available: http://arxiv.org/abs/2004.00980

[51] J. Krozel, M. Peters, K. D. Bilimoria, C. Lee, and J. S. Mitchell, "System Performance Characteristics of Centralized and Decentralized Air Traffic Separation Strategies," *Air Traffic Control Quarterly*, vol. 9, no. 4, pp. 311–332, 10 2001.

[52] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, "Reinforcement Learning from Imperfect Demonstrations," 2 2018. [Online]. Available: http://arxiv.org/abs/1802.05313

[53] W. R. Clements, B. Van Delft, B.-M. Robaglia, R. B. Slaoui, and S. Toth, "Estimating Risk and Uncertainty in Deep Reinforcement Learning," 5 2019. [Online]. Available: http://arxiv.org/abs/1905.09638

[54] G. A. Mercado Velasco, C. Borst, J. Ellerbroek, M. M. Van Paassen, and M. Mulder, "The Use of Intent Information in Conflict Detection and Resolution Models Based on Dynamic Velocity Obstacles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 2297–2302, 8 2015.

[55] E. Nikishin, P. Izmailov, B. Athiwaratkun, D. Podoprikhin, T. Garipov, P. Shvechikov, D. Vetrov, and A. G. Wilson, "Improving Stability in Deep Reinforcement Learning with Weight Averaging," 2018.

[56] V. Nguyen, S. Schulze, and M. A. Osborne, "Bayesian Optimization for Iterative Learning," 9 2019. [Online]. Available: http://arxiv.org/abs/1909.09593

2

Air Traffic Control: Control Task Definition and Past Automation Efforts

The purpose of this chapter is to get a thorough understanding of ATC and investigate what traffic scenario's and specific tasks of an ATCo are relevant to analyse. First, the fundamental components of ATC will be elaborated upon in section 2.1. Then, section 2.2 elaborates on the future of ATM to get an indication of what transition the automation developed in this research must facilitate. Having identified Conflict Detection & Resolution as main task of an ATCo, the task is formalised in section 2.3. In this section, the strategies an ATCo applies for conflict detection and resolution are explained. Next to developing autonomous forms of automation, researchers are developing supervisory control support tools for ATCos. These are presented in section 3.1. Finally, a survey into the design considerations for automated ATC and past automation efforts is presented in section 2.4 and section 2.5.

At the end of this chapter, a clear view on how to formalise the task at hand is presented and the relevant traffic scenarios are identified. In the thesis, a RL agent will be trained to analyse these traffic scenarios.

2.1. Introduction to Air Traffic Control

Air Traffic Management (ATM) is responsible for the integrated management of air traffic and airspace (Association 2016). ATM comprises of three main services: Air Traffic Services (ATS), Air Traffic Flow Management (ATFM) and Airspace Mangement (ASM). From these services, ATS has the purpose of ensuring safe and orderly traffic flow. Within ATS, ATC is responsible for maintaining safe and orderly operations. Today, ATCos are in full control of maintaining safe and orderly operations. Apart from lower level cognitive tasks such as visualisation of flight data and tracks, few tasks of an ATCo are automated (C. D. Wickens, Mavor, and R. Parasuraman 1998). ATC is still relying on radar and the use of radio to communicate with the pilots. However, due to recent advances in achieving a digital datalink through the use of ADS-B, ATC interaction is transitioning to digital.

For ATCos, conflict detection and resolution (CD&R) are considered the most important tasks according to (Bekier, Molesworth, and Williamson 2012). Therefore, this task will be the focus of this research. CD&R entails preventing violations of minimum separation standards and it will be discussed in more detail in the next section. The Airspace is divided into four different control zones: a circular area around an airport called the control zone (CTR), controlled by the Aerodrome Control (TWR); the terminal control area (TMA), which is controlled by the approach control (APP); the control area (CTA), controlled by the Area Control Center (ACC); finally the upper control area (UTA), which is controlled by organisations which function across Flight Information Regions (FIRs) e.g. Eurocontrol. The APP is most concerned with incoming and departing flights between the CTR and the CTA and the TWR is concerned with the control of aircraft on the ground. The ACC is responsible for flights which are en-route and will be the focus of this research, due to its limited constraints in terms of deviation paths for aircraft.

2.2. Future of ATM

The ATM system as it is in place today shows a few significant shortcomings which are addressed in the Future Air Navigation System (FANS). Foremost, the en-route system consist of a mixture of direct and organised tracks, and fixed airways. An increasing amount of air traffic without significant changes to the way air transport is currently managed will result in more delays, costing the airlines money (Gurtner et al. 2017). Also, the current airways force aircraft to take indirect routes, increasing the CO2 emissions of aviation industry.

SESAR (Single European Sky) and NextGen (Next Generation Air Transportation System) are the European and American variant of the FANS. Both programmes include Trajectory Based Operations, which essentially enable free-flight and therefore direct routing. The role of controllers would shift from actively controlling aircraft to "management by exception" (Dekker and Woods 1999; C. D. Wickens, Mavor, and R. Parasuraman 1998). One of the goals of these programs is to significantly increase the capacity of the ATM system. However, the controller workload remains to be the single greatest functional limitation on the capacity of the ATM system. The key factor contributing to the controller workload is the traffic complexity, even more so than air traffic density (Hilburn 2004). Removing the current day structure of air traffic control shall therefore highly increase the workload, and therefore the performance, of ATCos. Currently CD&R is still the full responsibility of ATCos, but FANS will incorporate shared CD&R between ATCos, pilots and automated algorithms (Esa M. Rantanen and C. D. Wickens 2012). To facilitate the transition to FANS and to increase air traffic capacity on the short run, implementing high degree automation in future ATC is crucial to maintain safety.

2.3. Formalisation of the Conflict Detection & Resolution Task

In order to perform the CD&R, ATCos use radar to analyse the situation and radio to communicate their commands. The information available to the ATCos of every aircraft is listed below:

- Flight number
- Position: longitude and latitude
- Heading: direction of aircraft (true North = 0° \ 360°)
- Speed

In general, an ATCo works in one certain sector for the entirety of their professional life. Therefore, ATCos become specialists in detecting and resolving potential conflicts just from looking at a radar screen. A sector is composed of beacons, such as a Very High Frequency (VHF) omnidirectional range(VOR) and distance measuring equipment (DME), and waypoints. A waypoint is a fixed point of reference which is set within the range of a beacon and can be used by aircraft equiped with Area Navigation (RNAV) to avoid having to fly directly over beacons. The beacons are connected through airways which at high altitudes are about 14 km wide. A sketch showing how aircraft can navigate is shown in Figure 2.1. In high altitude routing (HAR) zones, pilots are able to fly user-preferred routes if it also is a non-restrictive routing (NRR) zone, although, between specific fixes (intro into and exit of HAR airspace) (Nolan 2010). In Figure 2.2, an official chart of a low-altitude sector is shown in which one can see that beacons are connected by airways.

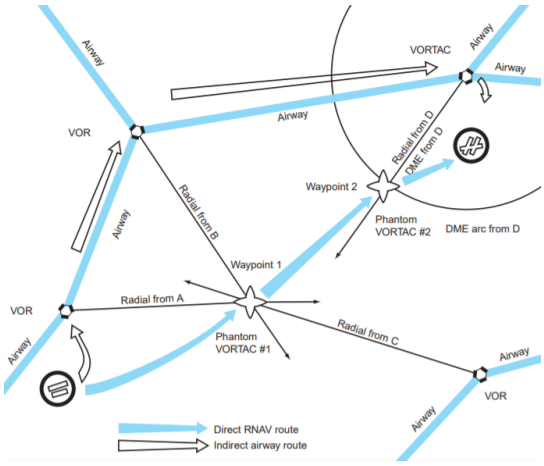


Figure 2.1: Sketch of how an aircraft can either fly an indirect route by following airways going over beacons or set waypoints to fly a more direct route if allowed. (Nolan 2010)

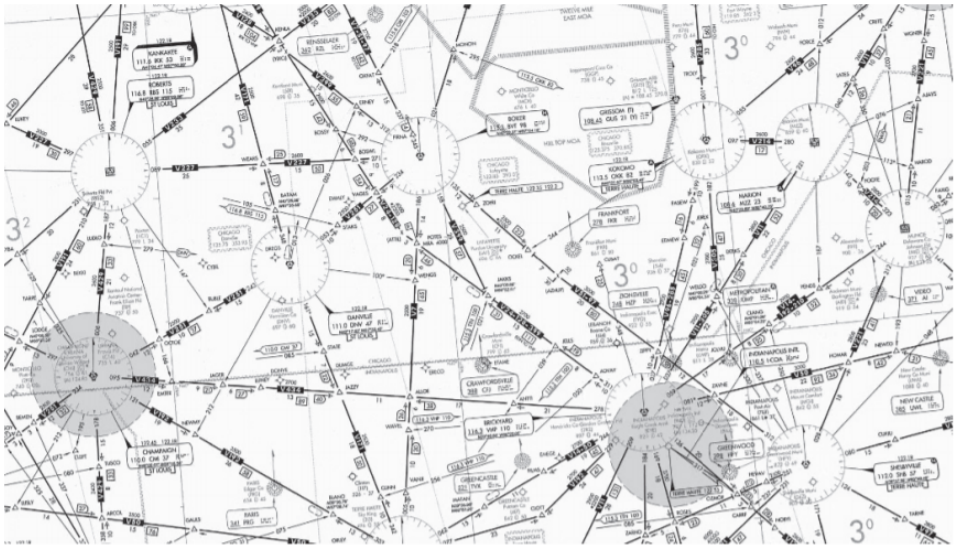


Figure 2.2: Low-altitude chart of an airspace. One can see that it is composed of beacons and airways connecting them. (Nolan 2010)

In order to fully understand the CD&R tasks, both the conflicts detection and resolution strategies ATCos currently apply to solve conflicts have to be elaborated on. In subsection 2.3.1 and subsection 2.3.2, the conflict detection and resolution tasks are analysed.

2.3.1. Conflict Detection

To maintain safe operations, the International Civil Aviation Association (ICAO) has dictated the following minimum separation standards(Association 2016):

- Vertical separation:
  - 2000 ft above FL410.
  - 1000 or 2000 ft between FL290 - FL410, subject to regional air navigation.
  - 1000 ft below FL290.
- Horizontal separation: minimum of 5 nm.

Two aircraft are said to be in conflict in case a vertical or horizontal loss of separation occurs. (Kirwan and Flynn 2002) conducted a study into the strategies of ATCos, in which it was shown that ATCos try to predict future conflicts with a prediction time varying between 5 and 10 minutes.

Three types of conflict that occur in longitudinal separation can be distinguished (Association 2016):

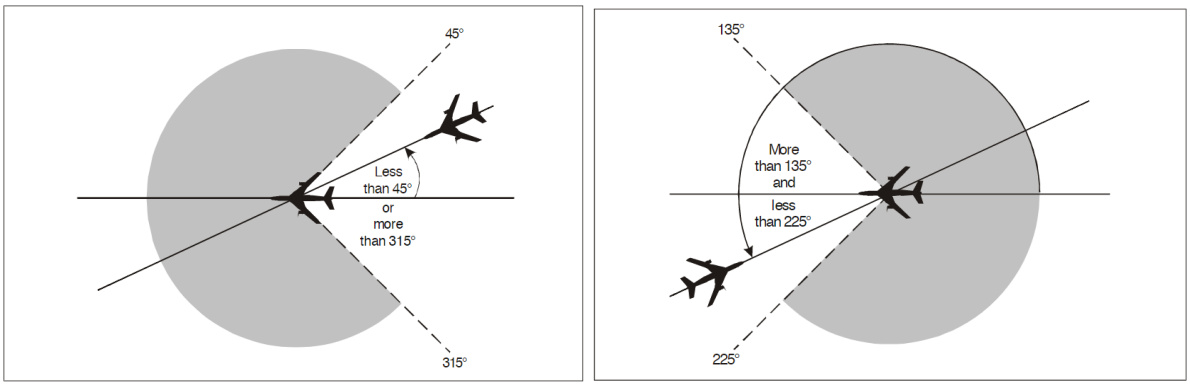
- *Same track*: both aircraft have the same direction tracks and intersecting tracks or portions thereof, the angular difference of which is less than 45 degrees or more than 315 degrees, and whose protected air spaces overlap. This is shown in Figure 2.3a.
- *Reciprocal tracks*: both aircraft have opposite direction tracks and intersecting tracks or portions thereof. The angular difference of which is more than 135 and less than 225 degrees, and whose protected air spaces overlap. This is shown in Figure 2.3b.
- *Crossing tracks*: intersecting tracks or portions thereof other than those specified above. This is shown in Figure 2.4.

ATCos apply different strategies to each of these scenario, this will be explained in more detail in subsection 2.3.2.

To see whether the two aircraft are to be in conflict with each other, ATCos try to estimate the Closest Point of Approach (CPA). Altitude is in all cases checked first by the ATCo, and in case the vertical separation standards are met, a non-conflict decision is made (Esa M Rantanen and Nunes 2005). When vertical separation standards are not met, an ATCo tries to estimate whether a pair of aircraft would be in conflict with each other. (BISSERET 1971) showed that ATCos were more cautious, and therefore more likely to predict two aircraft would be in conflict, in case the angle of conflict increases.

$$\text{Conflict Angle} = Heading_{a/c_1} - Heading_{a/c_2} \tag{2.1}$$

This suggests the accuracy in identifying potential conflicts of ATCos deteriorates with increasing conflict angle. A study performed by (X. Xu and Esa M. Rantanen 2007) showed that ATCos tend to judge potential conflicts more urgent in case the conflict is between two aircraft being at a small spatial distance. Potential conflicts between aircraft at larger spatial distance with an equal or smaller t_{CPA}, due to an increased airspeed, are perceived as less urgent. This phenomenon is called the "distance-over-speed bias". Moreover, (Law et al. 1993) showed that ATCos have more trouble judging which aircraft would arrive first at a potential conflict point. The research results summarised above indicate that an increase in conflict angle and speed difference are negatively correlated with the accuracy of ATCos in judging potential conflicts. Next to these particular conflict situations, air traffic density negatively influences the performance of ATCos to detect potential conflicts significantly (Metzger and Raja Parasuraman 2006).



(a) Aicraft on same track.(Association 2016)

(b) Aircraft on reciprocal tracks.(Association 2016)

Figure 2.3: Aircraft on the same track and on reciprocal tracks.

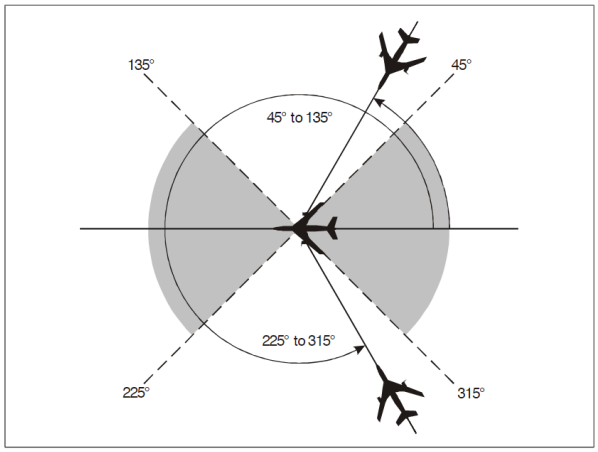


Figure 2.4: Aircraft on crossing tracks (Association 2016).

Important measures for ATCos, which will also be of interest for the artificial agent are the distance and time to the CPA. The distance and time to CPA can be calculated using Equation 2.2 and Equation 2.3 (Huo, Delahaye, and Y. Wang 2018). In this equation, v is the velocity vector and p the initial position of either aircraft i or j.

$$\begin{aligned} d_{CPA}(p_i, p_j, v_i, v_j) &= |t_{CPA} \cdot (v_j - v_i) + p_j - p_i| \\ &= \sqrt{(t_{CPA} \cdot \Delta V_x + \Delta X)^2 + (t_{CPA} \cdot \Delta V_y + \Delta Y)^2} \end{aligned} \tag{2.2}$$

$$t_{CPA}(p_i, p_j, v_i, v_j) = \frac{-(p_j - p_i) \cdot (v_j - v_i)}{|v_j - v_i|^2} = \frac{-\Delta X \Delta V_x - \Delta Y \Delta V_y}{(\Delta V_x)^2 + (\Delta V_y)^2} \tag{2.3}$$

2.3.2. Conflict Resolution

To resolve potential conflicts, ATCos can instruct pilots to perform three types of manoeuvres:

- Altitude change
- Heading change
- Speed change

In conflict resolution, ATCos employ a set of procedures to handle possible conflict scenarios. To favor one manoeuvre over another, (Esa M. Rantanen and C. D. Wickens 2012) refer to the work of (C. D. Wickens, Hellenberg, and X. Xu 2002) and defines the factors influencing the *expected utility* of the controller's decision. The three main influences are:

- *Expediency*: manoeuvres that resolve a conflict faster are preferred over time-consuming ones.
- *Preservation of airspace*: least disruptive manoeuvres to the overall traffic flow are preferred.
- *Visualisation*: manoeuvres that more clearly resolve the conflict are preferred.

An example adhering to these principles is that ATCos prefer vertical manoeuvres exploiting gravity over climbing since the manoeuvre is less time consuming. (Esa M Rantanen and Nunes 2005) showed that the control mechanisms preferred by ATCos are first of all a change in altitude, secondly a heading change and lastly a speed change. Rantanen and Nunes argue that an altitude change is preferred since it is a "non-radar" manoeuvre, and therefore does not need constant monitoring. Speed changes, however, are difficult to mentally represent for the controller and are therefore less often implemented by ATCos (BISSERET 1971; Helbing 1997). This is in-line with the visualisation influence mentioned by (Esa M. Rantanen and C. D. Wickens 2012). With regard to the three types of conflict, which are shown in Figure 2.3a, Figure 2.3b and Figure 2.4. (Kirwan and Flynn 2002) identified resolution strategies ATCos employ:

- *Same track*: align faster aircraft first with its exit point, also referred to as the Change Over Point (COPx).
- *Reciprocal track*: no strong heuristic for what action to take. However, this type of conflict is always solved first.
- *Crossing track*: strong heuristic to put slower a/c behind the faster one.

The aforementioned strategies are summarised in Table 2.1. These will be considered when designing the reward function for the artificial agent.

Table 2.1: Summary of high-level principles ATCos adhere to and strategies employed for CD&R.(Kirwan and Flynn 2002; Esa M Rantanen and Nunes 2005; Esa M. Rantanen and C. D. Wickens 2012; Fothergill and Neal 2008; Van Meeuwen et al. 2014; Kallus, Van Damme, and Dittmann 1999; Seamster et al. 1993)

Category	Principle	Description
High-level Principles	Safety	Highest priority
		Anticipate that things can deteriorate
		Always have a fail-safe plan B
	workload management	Keep it simple
		Look for the one key action that resolves the problem
		Minimise # aircraft to move
		Identify the conflicts pairwise
		Reduce complexity
		Prefer resolutions which require less co-ordination
	Efficiency	Penalise the one that needs something
		Inconvenience least people
		Minimise penalty for a/c
		Change in line with a/c intentions
Conflict Detection	Attention	Focus visual attention on crossing points and sector borders
		Adopt a look-ahead time between 5 and 10 minutes
	Strategy	1. Compare aircraft altitudes
		2. If same altitude, compare flight directions
		3. If same altitude and direction, compare speeds
	Contextual factors	Determine urgency and priority by estimation of relative speeds and distances
		In low workload conditions, wait and see before taking actions
		In high workload conditions, act immediately after detection a potential conflict
Conflict Resolution	Attention	First solve conflicts pairwise and later check for consequences on other aircraft
		Select resolution with least amount of monitoring
		Select resolution that requires least amount of sector disruption
	Strategy	1. Prefer level changes; Try to keep a/c at the same levels
		2. Vector aircraft; lock a/c on headings when using vectors
		3. Speed solutions last since at cruising altitude speed envelope is small
	Contextual factors	Turn slower a/c behind (minimise extra distance flown)
		Better to put a/c behind than trying to go through the middle
		If vectoring, vector both aircraft
		Solve the head-on first
		Turn faster one direct to route so leaves sector before slower one on same route

2.4. Design Considerations of Automation in ATC

For the transition to a high degree of automation in future ATC, the support of ATCos is important. Without the support of ATCos, the gains of automated systems may not be fully realised. This section briefly surveys previous studies into the acceptance of any form of automation by ATCos.

(Bekier, Molesworth, and Williamson 2012) investigated the existence of a 'tipping point' at which ATCos were not longer willing to accept or cooperate with the automation. The research showed that the tipping point of most ATCos, especially the more experienced ones, was reached in case the level of automation shifted the decision-power away from the controller. For an ATCo to accept automation, a high level of trust must be established that at whatever level the automation is implemented, the job is performed successfully.

One of the aspects likely influencing the willingness to accept a shift in decision power is accountability (Bekier, Molesworth, and Williamson 2012). Since air traffic controllers have been charged with criminal offences in the past, it is understandable that ATCos are concerned about a shift in authority whilst being accountable. (Bekier, Molesworth, and Williamson 2012) showed that ATCos were more willing to accept a higher degree of automation to support efficiency of scheduling rather than in safety-related tasks, strengthening this suspect.

Important for ATCos is thus the degree to which certain tasks are automatised. (R. Parasuraman, Sheridan, and C. Wickens 2000) introduced the four classes of functions to which automation can be applied to, equivalent to the four-stage model of human information processing. These are:

- Information acquisition
- Information analysis
- Decision selection
- Action implementation

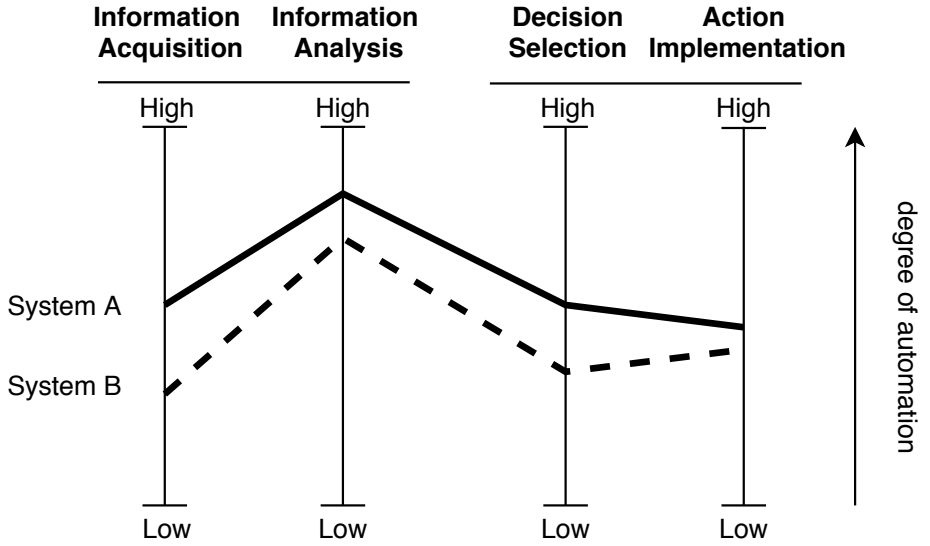


Figure 2.5: Four classes of functions which automation can be applied to. (R. Parasuraman, Sheridan, and C. Wickens 2000)

Within these four broad categories, the degree of automation can vary across a continuum from low to high (R. Parasuraman, Sheridan, and C. Wickens 2000), as shown in Figure 2.5. This framework can be used as guide to determine what types and levels of automation are considered important for a particular system. The variance in the degree of automation across the four classes of functions namely impacts the performance of systems dependent on human-automation interaction.

In fact, changes in the level of automation across the function classes affect human performance areas. These are the mental workload, situation awareness, complacency and skill degradation. These are considered to be primary evaluative criteria for automation design by (R. Parasuraman, Sheridan, and C. Wickens 2000). In most cases, high levels of automation in information acquisition and analysis has showed a significant decrease in mental workload. However, evidence shows that workload can indeed increase when it is difficult for human operators to initiate and engage with the automation. Especially for a high level of automation for decision selection and action implementation, this should be taken into account.

The Air Traffic Controller (ATCo) has two main cognitive tasks: maintaining situational awareness (SA) and decision making for control actions (EUROCONTROL 1996). Increasing automation of information analysis might improve the SA significantly. Take for example the horizontal situation indicator. Alternatively, when having a high level of automation for decision selection, the human operator tends to be less aware of changes in the dynamic environment. This might already occur if the human has a passive decision-making role in which he or she has to check the functioning of the automation (Raja Parasuraman, Molloy, and I. L. Singh 1993). The SESAR and NextGen program both incorporate the concept of free flight. In case the level of automation will go up for ATC, and FANS will be implemented, this might lead to an increase in air traffic complexity and a decrease in situation awareness for the ATCos.

Another phenomenon that has been identified is "over-trust" or complacency of the human operator in the automation. For a higher degree of automation, the operator becomes less likely to identify the occasions on which the automation fails and to act properly to the failure (Raja Parasuraman, Molloy, and I. L. Singh 1993; Bowden, Ren, and Loft 2018). Especially in case the automation is not reliable, this is an important aspect to consider.

Finally, in case decision making is continuously performed by automation, the human controller will become less skilled in performing the tasks of an ATCo (Volz et al. 2016). This might be detrimental to the design of a

system in which humans and automation have to work together. In case the automation fails, the situation is more hazardous whilst the human controller might fail to perform as well. The aforementioned risks can be combined and is formalised as the out-of-the-loop (OOTL) performance problem (Bowden, Ren, and Loft 2018). To successfully design an automated concept for ATC, the risks associated to the OOTL problem must be taken into account.

(R. Parasuraman, Sheridan, and C. Wickens 2000) also note two secondary evaluative criteria: automation reliability and costs of decision/action outcomes. A system should be reliable for benefits in terms of mental workload and situation awareness to hold. This is because reliability is closely related to the trust a human has in an automated system. Furthermore, when implementing automation in decision selection and action implementation, the costs associated with these decisions and actions must be considered. Tasks whose outcomes have low associated risk are good candidates for high automation in decision choices. According to these researchers, high level of decision automation is not suitable in case humans must take over control in case automation fails.

In this research the arguments elaborated upon in this section will be considered during the design of the automation for ATC.

2.5. Automation Efforts

Lower level cognitive tasks such as the visualisation of flight data and tracks have already been implemented (C. D. Wickens, Mavor, and R. Parasuraman 1998). For example, Medium-Term Conflict Detection (MTCD) and Short-Term Conflict Alert (STCA) are implemented in current ATC, providing warning in case a potential conflict seems to happen based on (multiple) predicted trajectories. The MTCD can predict potential conflicts looking up to 20 minutes ahead.

2.5.1. Failed Initiatives

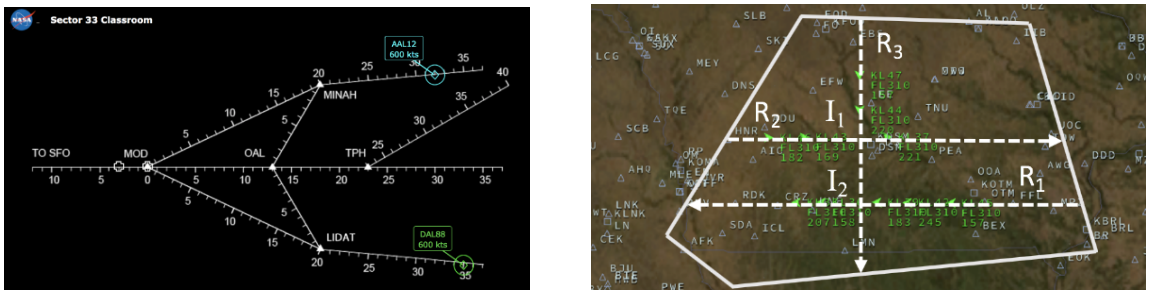
In an effort to increase the degree of automation in ATC, Eurocontrol launched the ARC2000 project. The core idea behind this project was to develop an automated ATC system based on "4D-tubes" for en-route free flight. The strategy was to guarantee conflict free flight for the next 20 minutes by planning ahead using these tubes (Garot and Durand 2005). The project was halted since it was at that time not possible to develop a system that could tackle the problem of global optimisation of flight routes. Another effort by Eurocontrol was the development of CORA (en-rout Conflict Resolution Assistant). This project was human-centered and its purpose was to automate ATC according to controllers' strategy for generating advisory resolutions (De Prins et al. 2008). One of the proposed concepts was to artificially generate solutions and filter these based on rules and heuristics adhering to an ATCos' strategy. This project however never passed the concept phase.

2.5.2. Autonomous ATC

In recent years, researchers have turned to reinforcement learning as possible solution for the CD&R tasks. (Puca et al. 2014) implemented a reinforcement learning method, Q-learning, to solve a gridworld problem in which an agent has to avoid an object which represents the conflict area. It was concluded that RL could provide good solutions for CD&R tasks. However, the lack of flexibility of this method would render unacceptable computation times for problems only differing slightly from each other (Puca et al. 2014). Inspired by the development of *AlphaGo,* (Brittain and Wei 2018) developed an hierarchical Deep Double Q-Network for autonomous aircraft sequencing and separation. The algorithm implemented a parent and child agent, and the web application NASA sector 33, shown in Figure 2.6a, was used as traffic control simulation:

- Parent agent:
  - *state space*: raw pixels of the radar screen.
  - *action space*: route combination (route A or B) for all aircraft. If there are two aircraft, the action space consists of 4 unique route combinations.
- Child agent:
  - *state space*: aircraft positions, aircraft velocities and route combination (from parent agent).
  - *action space*: velocity changes: 300, 360, 420, 480, 540, 600[kts].

The aircraft could have 6 different airspeeds varying between 300 and 600 kts. Furthermore, the airways are fixed and the aircraft could therefore not alter heading continuously, but could take a combination of airways. A numerical analysis of the implementation of the algorithm showed promising results, although it must be noted that the action space was limited. Following the promising results of applying Deep RL to the ATC problem, a Deep Distributed Multi-Agent Reinforcement Learning framework was developed in (Brittain and Wei 2019). In this research, BlueSky was used as simulation environment, which is shown in Figure 2.6b. The goal of the research was to investigate whether a multi-agent RL method utilising an actor-critic model, which incorporated the loss function from Proximate Policy Optimisation to stabilise the learning, could solve conflicts between aircraft in a *dynamic*, *stochastic* and *high-density* en-route sector (Brittain and Wei 2019).



(a) Sector 33 used in (Brittain and Wei 2018).

(b) BlueSky simulation environment used in (Brittain and Wei 2019).

Figure 2.6: Simulation environments used in previous automation efforts.

Since it is a multi-agent scenario, the state and action space of each agent was given by:

- agent i:
  - *state space*: $s_t^o = (I^{(o)}, d^{(1)}, LOS(o,1), d^{(2)}, LOS(o,2), \ldots, d^{(n)}, LOS(o,n), I^{(1)}, \ldots, I^{(n)})$
  - *action space*: $a_t = [v_{min}, v_{t-1}, v_{max}]$

The state of the agent is composed of $I^{(i)}$, which represents the distance to the goal, the aircraft speed, aircraft acceleration, distance to the intersection, route identifier, and half the loss of separation distance of aircraft i. Furthermore, $d^{(i)}$ represents the distance from the controlled aircraft to aircraft i and $LOS(o,i)$ represents the loss of separation distance between aircraft o and i. The action space consists of three speeds, the minimal, the current and maximal cruise speed. The sector used in the simulations contains multiple intersections and merging points. A simplification, critical to the convergence of the policy, was made: aircraft on conflicting route must have not reached the intersection. The AI agent was able to avoid conflicts for 100 % of the time for merging routes scenario and 99.97 % for the intersecting routes scenario. The researches suggest that the stochastic environment might have initiated a traffic scenario in which conflict was not avoidable. Overall, this research shows promising results and confirms that deep multi-agent reinforcement learning methods can enable a safe and efficient automated en-route airspace. However, limitations on the action-space and assumptions were critical in obtaining a converging policy.

Brittain and Wei were not the only researchers implementing Actor-Critic methods to achieve autonomous ATC. (Pham et al. 2019) developed a Single-Agent Actor Critic (Deep Deterministic Policy Gradient - DDGP) framework to automate conflict resolution. An uncertain environment was simulated in which two aircraft were in conflict. Arbitrarily, one of the two aircraft was the controlled aircraft, and the other was seen as the observed aircraft. The goal of the controlled aircraft was to avoid the conflict by changing its heading. The trained model could achieve 87%∓ accuracy for avoiding the conflict by solely altering heading. An important result from the research is that even though the uncertainty of the environment slowed the convergence speed, the convergence value was not influenced (Pham et al. 2019). (Tran et al. 2019) also explored the concept of having a controlled and an observed aircraft, and provided the artificial agent with human solutions to conflicts. An Actor-Critic framework, incorporating DDGP, was used as RL solution method. The controlled aircraft could avoid the conflict by changing its heading, thus changing its route. The model's performance is assessed by the similarity between the agent's suggested trajectory change point (TCP) and that of the human controller in unseen scenarios of the test set. It shows that in 65% of the cases, the artificial agent chooses a route similar to that of the human controller.

2.5.3. Strategic Conformal Automation

To increase trust and acceptance of automation, strategic conformal automation has been proposed in (Hilburn, Westin, and Borst 2014). In line with this proposed method, (Regtuit et al. 2018) developed a strategic conformal automation for ATC using RL. In this research, ATCos were asked to solve conflict scenarios. The artificial agent was presented the same traffic situation and was rewarded for mimicking the actions taken by an ATCo. The results seem to support the hypothesis that RL could be used to identify and replicate human control strategies from logged human control strategies (Regtuit et al. 2018).

Next to using RL for strategic conformal automation, (Van Rooijen 2019) investigated whether human strategy could be extracted from the Solution Space Diagram (SSD) using supervised learning. The research showed that the SSD contains a majority of the parameters influencing ATCos decision making for CD&R. Furthermore, it confirmed the hypothesis on which strategic conformal ATC automation depends, which is that ATCos apply personal strategies to solve CD&R conflicts. This was shown in an experiment in which models trained on personal data had a higher prediction score than generic models for individual ATCos. The main limitation of the methods proposed to achieve strategic conformal automation is that these purely try to mimic strategies, but are not able to optimise the resolutions.

2.6. Concluding Remarks

In the previous automation efforts, two main areas of research were explored. First of all, strategic conformal automation has been studied in the context of reinforcement learning and supervised learning. Strategy could be identified by using reinforcement learning through showing demonstrations to the learning agent. This approach has a few setbacks. To consistently identify an ATCo's strategy, 55 demonstrations had to be used in the same scenario. The generalisation that ATCos have different strategies for crossing, reciprocal and same track conflicts might be inaccurate since ATCos could have different strategies in different parts of the conflict domains. Moreover, different types of aircraft also have an effect on their strategy. To identify all these different strategies, a vast amount of demonstrations are needed for different conflict geometries, which is unlikely. Furthermore, it is difficult to verify whether the agent's policy is both safe and strategic conformal. The recent research into the use of supervised learning is interesting as it is relatively effective in mimicking strategy of ATCos. In the research performed by (Van Rooijen 2019), the SSD was used to learn the strategy of controllers. One of the cons asserted to supervised learning to learn strategy is that the model might learn the strategy of a controller, but will never improve an ATCos strategy. If an ATCo performs inefficiently in complex traffic scenarios, the automation will do that as well.

The second area of research is that of developing autonomous ATC. Recent studies showed promising results for the implementation of Deep RL, although the state and action spaces remained fairly limited. The advantage of using RL is that careful design of the reward function can steer the behaviour of an agent. Previously identified strategies, such as preferably choosing the least disruptive manoeuvres, can be taken into account next to efficiency and safety considerations. For autonomous ATC, using reinforcement learning seems to be the most promising technique.

In previous studies into autonomously automating ATC with RL, adding more aircraft to the traffic scenario would increase the state-space of the agent. For both the hierarchical and multi-agent approach discussed in section 2.5, this was the case. Combining the support tools which were developed to assist ATCos with reinforcement learning might be an interesting solution to this problem. The SSD visualises the solution space of a single aircraft in terms of heading and velocity changes. From the SSD, an ATCo can acquire information on whether an aircraft will be in conflict and when that will happen. It is hypothesised that using raw pixel data of the SSD as input for a reinforcement learning algorithm can enable an artificial agent to learn what action (heading and/or velocity change) to take in a large variety of traffic scenarios. Since all the information is captured in the SSD of which the pixel size is fixed, the dimensions of the state-space remain constant. The SSD is aircraft specific and currently it is displayed in case an ATCo selects an aircraft.

The CD&R task of an ATCo using the SSD consists of two problems: what aircraft to select and what action to perform. In this research, it shall be investigated whether the raw-pixel data of the SSD can be used to learn an agent what action to take. To select an aircraft, either a rule-based algorithm will be designed or an agent will be trained by using supervised or reinforcement learning.

Before diving into state-of-the-art reinforcement learning algorithms, the next chapter elaborates on the fundamentals of reinforcement learning.

3

Representing the State of an Aircraft Using the Solution Space Diagram

This chapter provides a literature review of supervisory control support tools which have been developed for ATC. This part of the literature study aims to answer research question 2b: 'What information can be extracted from the SSD?'.

3.1. Supervisory Control Support Tools

Apart from automating ATC, another area of research is the development of tools to support humans in a supervisory control tasks. Two supervisory control support tools which have the potential to decrease the workload of ATCOs are the separation monitor and Solution Space Diagram (SSD). These are elaborated upon in subsection 3.1.1 and subsection 3.1.2.

3.1.1. Separation Monitor

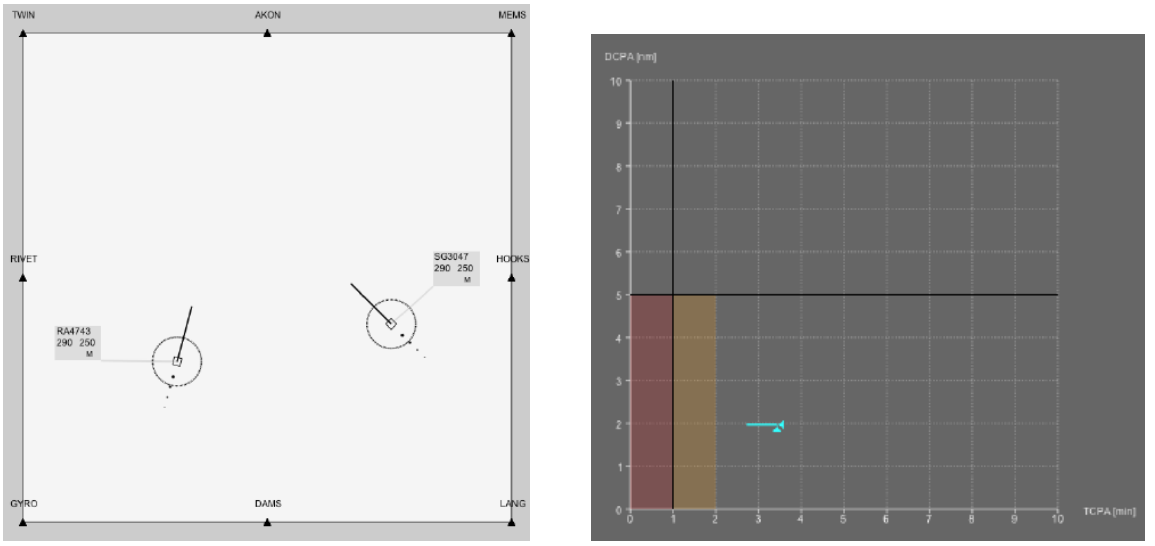
Two parameters which are key for ATCOs to detect potential conflicts are the CPA and t_{CPA}, as explained in subsection 2.3.1. Currently, ATCOs estimate whether the CPA between two aircraft will be lower than 5 nm. In case two aircraft would be in conflict according to their estimation, the t_{CPA} determines when the ATCO orders a manoeuvre for one of the two aircraft in conflict. Based on calculated aircraft trajectories which can be extrapolated into the future, a graph showing the aircraft combinations and their corresponding CPA can be shown as function of the t_{CPA}. An example of a potential conflict portrayed with the separation monitor is shown in Figure 3.1a and Figure 3.1b.

The power of this monitor is that it gives a quick overview of all the upcoming conflicts. The separation monitor alone however does not contain any information on how to solve the upcoming conflicts. To this end, the Solution Space Diagram was developed.

3.1.2. Solution Space Diagram

The concept of the SSD, which was developed in (Hermes et al. 2009), is to use ecological interface design (EID) to facilitate fault detection and generate automated advisories in CD&R for the human controller. The SSD displays the locomotion constraints of the controlled aircraft. The controlled aircraft is the aircraft which an ATCO selects and from which he or she gets to see the SSD. The display shows how surrounding aircraft and/or obstacles affect the solution space of the controlled aircraft in terms of heading and speed.

The construction of the diagram for a two aircraft conflict consists of a step-by-step process. First of all, the projected zone around the observed aircraft has to be constructed, which in our case is 5 nautical miles. Then, the relative velocity vector can be constructed and a triangular area can be identified for which the aircraft would be in conflict in relative space. The velocity of the observed aircraft can then be added to the relative velocity cone to determine the conflict velocity cone for in absolute space (Mercado Velasco et al. 2015). This conflict cone is referred to as the Forbidden Beam Zone (FBZ). Finally, the flight envelope of the controlled aircraft can be added to determine the bounds in which the aircraft can vary its speed, $[V_{min}, V_{max}]$. These steps are visualised in Figure 3.2.



(a) Air traffic scenario.

(b) Combining d_{CPA} and t_{CPA} to see what aircraft pairs will be in conflict.

Figure 3.1: Simulation environment with separation monitor.

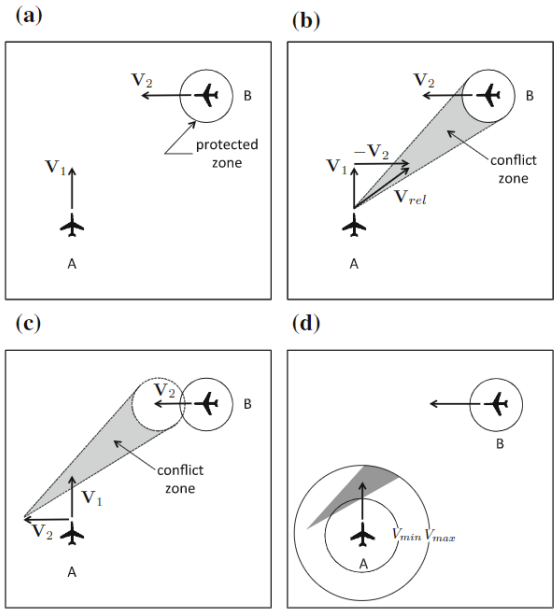


Figure 3.2: Step-by-step construction of the SSD(Borst et al. 2017): (a) one can see the traffic geometry; (b) conflict zone in relative space; (c) conflict zone in absolute space; (d) resulting SSD.

The controlled aircraft shall not be in collision with the observed aircraft in case its velocity vector (incorporating heading and absolute value) is outside the velocity obstacle. A conflict scenario with multiple aircraft can easily be incorporated in the SSD of the controlled aircraft by calculating the triangular velocity obstacles relative to the observed aircraft. The display incorporates all the useful parameters identified for the CD&R task in subsection 2.3.1 and subsection 2.3.2. The features of the SSD representing these parameters are shown in Table 3.1. The power of this presentation is that the fault detection performance, especially in increasing the traffic complexity, is positively affected (Borst et al. 2017). Incorporating this display in a form of automation will thus ensure that the human controller remains able to validate the computer generated advice. Although it does contain all the relevant information for CD&R, the SSD on its own does not contain enough information for the ATCO to identify the most efficient resolution.

Table 3.1: Parameters incorporated in the SSD. Adapted from (Van Rooijen 2019).

Parameter	**Feature in the SSD**
Velocity	Length of the velocity vector.
Velocity envelope	Inner and outer circle on the SSD.
Heading	Direction of the velocity vector.
Exit waypoint	Strikingly colored heading vector within the speed limits.
Conflict in terms of heading and speed	FBZ.
t_{CPA}	Width of the FBZ (smaller t_{CPA}, wider FBZ); Color coding of FBZ.
d_{CPA}	Reflected by a translating FBZ.
Conflict angle	Inclination of the FBZ.
Velocity other a/c	Length FBZ.
Traffic Density	Amount of FBZs represented in the SSD.
Traffic Complexity	With a higher complexity, the FBZs of individual observed aircraft overlap less and solution space becomes smaller.
Objects	Represented as a FBZ covering entire range of velocities in a certain direction.

An extension to the SSD is to incorporate intent routes of the observed aircraft. This will increase the detection time horizon and thereby enhance more efficient strategic solutions. In (Mercado Velasco et al. 2015) a proposed solution was developed. This shows that the SSD can be extended to include more relevant information for the ATCO. For now, the focus shall be on the SSD solely taking into account the triangular velocity obstacle.

The SSD essentially thus contains all the relevant information for the CD&R task. One can also notice that all of the relevant information is contained in the upper half of the SSD, since an aircraft will generally not change its heading with more than 90 degrees at once.

4

Reinforcement Learning Fundamentals

Reinforcement Learning is the problem faced by an artificial agent that must learn behaviour through interacting with its environment (Pack Kaelbling et al. 1996; Francois-Lavet et al. 2018). It is a computational approach to learn from interaction, mimicking the way that biological agents, such as humans, learn. In fact, many of the basic algorithms in RL were inspired by biological learning systems (R. S. Sutton and A. G. Barto 2018). The promise of RL is that it is a way of programming agents based on reward and punishment without having to specify how the agents needs to achieve a task (Pack Kaelbling et al. 1996). In recent years, RL methods have been developed which exceed human performance in games such as Atari to Go and no-limit poker (Botvinick et al. 2019). Breakthroughs in the development of deep learning methods have been major contributors to these super-human performances. Before diving into these breakthroughs in RL, the basic principles of RL should be well-understood. The goal of this chapter is to provide a high-level overview of the working principles and different solution methods of RL.

RL applies in principle to any sequential decision-making problem that relies on past experiences (Francois-Lavet et al. 2018) and can be formalised as the optimal control of incompletely-known Markov decision processes (MDP) (R. S. Sutton and A. G. Barto 2018). The RL problem consists of 4 main elements: a policy, a reward signal, a value function and optionally a model that mimics the behaviour of the environment. The policy is a mapping from the state to the action. The agent uses its policy to decide what action to take in a particular state. The performance of an action is measured by the reward signal and the value function indicates the expected reward signals to accumulate over the future. Essentially, one wants to optimise the policy to maximise the accumulation of reward signals obtained from interacting with the environment. The artificial agent thus learns from experiences retrieved from interacting with the environment. This introduces an important challenge in RL, namely the trade-off between exploiting what has already been experienced and exploring the unknown. Another feature that needs to be accounted for is delayed reward. Although a certain action might give rise to a large immediate reward, it also affects the future state of the environment and thereby the actions and opportunities available to the agent. Since the goal of the agent is to maximise the accumulative reward, an effective RL algorithm requires foresight or planning (R. S. Sutton and A. G. Barto 2018).

Similar to other machine learning techniques, RL has to deal with the bias-variance trade-off. This essentially entails that a model with high bias has failed to find a pattern in the data whilst a model which over-fits on the data shows high variance. Due to the over-fitting, the model will then fail to generalise. One therefore aims to find a model that balances these two considerations. Another factor influencing the design choice is the curse of dimensionality. Whenever the state and action spaces increase, the amount of possible state-action pairs often grows exponentially (Pack Kaelbling et al. 1996). Therefore, the agent has to deal with a large amount of options which can endanger the convergence of a RL algorithm.

Briefly said, there are two types of RL solution methods. First of all, there are Tabular Solution Methods, in which the methods can often find exact solutions. These are well suited for problems in which the state and action spaces are limited. Secondly, there are Approximate Solution Methods, in which the goal is to find a good approximation of the optimal policy as the state and action spaces are too large to find the exact optimum.

Each of the solution methods has to deal with the following challenges:

- Exploration vs exploitation
- Bias-variance trade-off
- Curse of dimensionality
- Design of the reward function

First, in section 4.1, the finite Markov decision processes are explained in more detail. Then, in section 4.2 and section 4.3, the tabular and approximate solution methods for RL are elaborated upon.

4.1. Finite Markov Decision Processes

MDPs are a mathematical formalisation of sequential decision making. In MDPs, actions not only influence immediate reward, but also subsequent states through future rewards (R. S. Sutton and A. G. Barto 2018). Naturally, MDPs involve the trade-off between immediate and delayed reward.

The learner and decision maker in an MDP is called an agent. The agent interacts with its environment by performing an action. The environment responds by presenting new situations, states of the agent, and giving rise to new rewards. RL is therefore essentially a mapping from a situation or state to an action to maximise the reward. This interaction is visualised in Figure 4.1. At every discrete time step, the agent chooses a certain action, $a_t \in A(s)$, based on the state, $s_t \in S$, and reward, $r_t \in R$, retrieved by the previous interaction with the environment.

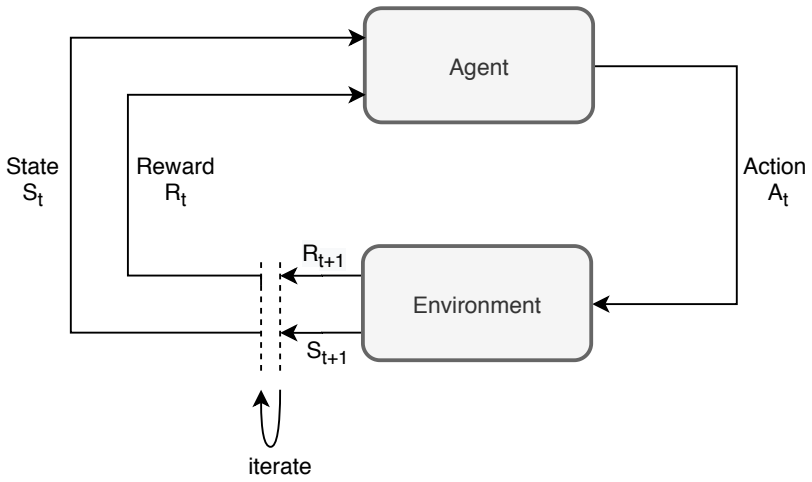


Figure 4.1: The agent-environment interaction in a Markov decision process. Adapted from (R. S. Sutton and A. G. Barto 2018).

The state an agent is in can be deterministic or stochastic. For a deterministic state, taking the same action in a certain state always results in the same new state whilst for a stochastic state, the next state is a random variable (Lucian Bus¸oniu Bus¸oniu et al. 2019). ATC is essentially a problem with stochastic state transitions since wind influences the path of aircraft significantly. The problem can be simplified and modelled with deterministic state transitions by excluding wind and other uncertainties. Since most of the theoretical fundamentals of RL are based on the deterministic environment, RL algorithms perform better in deterministic environments (Jaakkola, S. Singh, and Jordan 1999). For the sake of this research, deterministic state transitions shall be used to simulate ATC.

The formalisation of a MDP can be defined with a state space S, an action space A and a transition function f which maps the state changes as a result of control actions as shown in Equation 4.1 for a discrete time step (Lucian Bus¸oniu Bus¸oniu et al. 2019). f thus has dimension $S \times A$.

$$S_{i+1} = f(s_i, a_i) \tag{4.1}$$

The agent also receives a scalar reward signal, according to a function $R : S \times A$:

$$r_{i+1} = R(s_i, a_i) \tag{4.2}$$

In a finite MDP, the random variables S_t and R_t have well defined discrete probability distributions dependent on the previous state and action. This probability, $p : S \times R \times S \times A \longrightarrow [0,1]$, denotes the dynamics of the MDP and is shown in Equation 4.3 in which s' represents the next state.

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{4.3}$$

Finally, as mentioned above, the agent chooses the action using a certain policy mapping states to actions. This policy is defined by the function $\pi : S \Longrightarrow A$. In a MDP, having the current state and action, in combination with the transition and reward function, always allows the agent to reach the next state and the reward. In some RL problems, the agent can reach a terminal state, e.g. the finish of a racetrack. In this case, the agent restarts from the initial state and performs another *episode*. All of the non-terminal states are in then denoted by $S^+ \in S$.

4.1.1. Return

The goal of the learning agent is to find a policy which optimises the cumulative reward retrieved along a trajectory starting from the initial state. Actions taken might not only effect the immediate reward, but also the subsequent situations the agent encounters and its accompanied rewards. This introduces the so-called challenge of delayed rewards (R. S. Sutton and A. G. Barto 2018): actions taken in the present effect the potential to achieve good rewards in the future, whilst the immediate reward provides no information about these future rewards. In case of RL problems with a terminal state, it makes sense to calculate the cumulative reward by taking the sum of rewards retrieved when reaching the terminal state (R. S. Sutton and A. G. Barto 2018). However, there are also RL problems in which the agent-environment interaction does not naturally break into episodes, but goes on without a limit. These tasks are referred to as *continuing tasks*. Theoretically, the final step in such tasks are at $T = \infty$, and the return could also be infinite. To overcome this issue, use can be made of the concept *discounting*. Instead of maximising the sum of rewards, the agent tries to maximise the sum of discounted rewards it receives over the future. The return, G_t, using discounting is shown in Equation 4.4 (R. S. Sutton and A. G. Barto 2018). In this equation, γ is the discount rate, $0 \leq \gamma \leq 1$.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1} \tag{4.4}$$

Alternatively, Equation 4.4 can be written as shown in Equation 4.5, which includes the possibility of $T = \infty$ or $\gamma = 1$ (R. S. Sutton and A. G. Barto 2018).

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{4.5}$$

From this equation, it can be seen that if $\gamma < 1$, the return is bounded; if $\gamma = 0$, the agent only takes immediate reward into account. Essentially, a reward that is received k time steps into the future is worth γ^{k-1} of what it would be valued in case it would have been immediately (R. S. Sutton and A. G. Barto 2018).

4.1.2. Value Functions

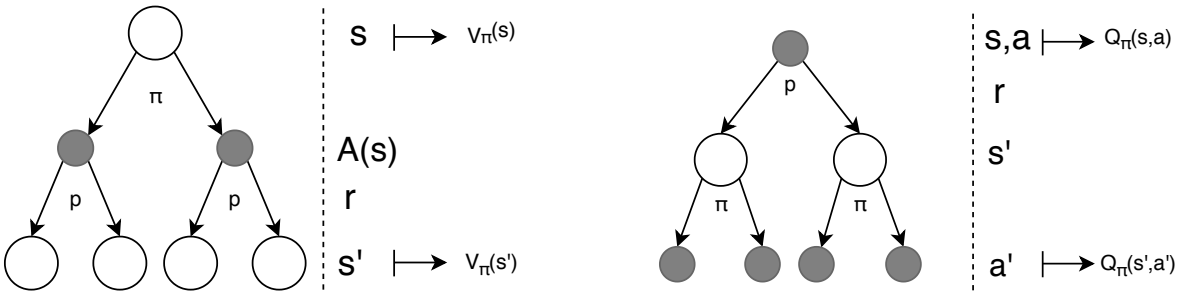
Since maximising the accumulated sum of returns is the goal of an RL agent, RL agents make use of *value functions*. The value function is an estimation of expected future rewards from the state the agent is in or the action it takes. The value function of a state $s \in \mathcal{S}$ under policy π is given by Equation 4.6 and is named the *state-value function*. In this equation, $E_\pi[\cdot]$ denotes the expected value of a random variable following a policy π (R. S. Sutton and A. G. Barto 2018).

$$v_\pi(s) \doteq E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], \text{for all s} \in \mathbb{S}. \tag{4.6}$$

Moreover, the state-action value, or *action-value*, denotes the expected reward expected from taking an action $a \in \mathbb{A}(s)$ in state $s \in \mathbb{S}$ under policy π and is given by Equation 4.7.

$$q_\pi(s, a) \doteq E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a], \text{for all s} \in \mathbb{S}. \tag{4.7}$$

The backup diagrams for the state and action-value functions are shown in Figure 4.2b.



(a) State-value function.

(b) Action-value function.

Figure 4.2: Backup diagram of the state-value and action-value function. In this diagram, $s' = s_{t+1}$ and $a' = a_{t+1}$.

Similarly as for the return, one can define a recursive relationship for the value function (R. S. Sutton and A. G. Barto 2018):

$$\begin{aligned} v_\pi(s) &\doteq E_\pi[G_t|S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \text{for all s} \in \mathbb{S}, \end{aligned} \tag{4.8}$$

in which the next states are denoted by $s' \in S$ and p represents the dynamics of the MDP. This equation is called the Bellman equation for v_π. The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way (R. S. Sutton and A. G. Barto 2018).

In theory, there is a precise optimum policy to be found for finite MDPs. This optimum policy is denoted by $\pi_\star$ and holds if and only if $v_\pi \geq v_{\pi'}$, meaning that the expected future rewards under the policy π is larger than that under policy $\pi^\star$. There might in fact be multiple optimal policies, which will all be denoted as $\pi_\star$. These optimal policies all share the same state-value function, which can be calculated using Equation 4.9 and is denoted by $v_\star$(R. S. Sutton and A. G. Barto 2018).

$$v_\star(s) \doteq \max_\pi v_\pi(s), \text{for all s} \in S \tag{4.9}$$

Next to sharing the same value function, they also share the same action-value function. This is shown in Equation 4.10 and is denoted by $q_\star(s, a)$.

$$q_\star(s, a) \doteq \max_\pi (s, a) \text{ for all s } \in \mathbb{S} \text{ and a} \in \mathbb{A}(s) \tag{4.10}$$

For state-action pair (s,a), the action-value denotes the expected return for taking an action a from a state s and thereafter following an optimal policy(R. S. Sutton and A. G. Barto 2018). Therefore, the action-value function can be rewritten in terms of the optimal state-value function:

$$q_\star(s, a) = E[R_{t+1} + \gamma v_\star(S_{t+1})|S_t = s, A_t = a]. \tag{4.11}$$

$v_\star$ is a value function for a policy and must therefore satisfy the Bellman equation shown in Equation 4.8. However, since it is the optimum value function, one can rewrite the Bellman in equation into the so-called Bellman optimality equation. The Bellman optimality equation for $v_\star$ and $q_\star$ is shown in Equation 4.12 and Equation 4.13.

$$\begin{aligned} v_\star(s) &= \max_{a \in A(s)} q_{\pi\star}(s,a) \\ &= \max_a E[R_{t+1} + \gamma v_\star(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\star(s')] \end{aligned} \tag{4.12}$$

$$\begin{aligned} q_\star(s,a) &= E[R_{t+1} + \gamma \max_a q_\star(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_a q_\star(s', a')] \end{aligned} \tag{4.13}$$

The Bellman equations also generalises to the continuous state and action spaces. It is relevant to note that even though there is a precise model of the dynamics of an environment available, it is usually impossible to compute an optimal policy by solving the Bellman's equation. Critical to solving the problem facing the agent is the amount of computations that the agent can perform per single time step. Another limiting factor in retrieving the optimal policy is the available memory to store approximations of value functions, models and policies. In case state and action spaces are limited, the RL problem is a *tabular case* and use can be made of the memory to store the approximations. If the required memory capacity is too large, some sort of parameterised function representation must be used to approximate the value functions, which is called the *approximate case*.

4.2. Tabular Solution Methods

In the tabular case, the approximate value functions can be represented in arrays or tables since the state and action spaces are small enough for the available memory capacity. Although the tabular solution methods are not suited for the ATC problem, a brief explanation of the tabular solution methods is given since it provides essential insights for the approximate solution methods.

The goal of a reinforcement learning agent is to find the optimum policy, which is referred to as the *control problem*. To do so, an optimal method for estimating the value function must be found first. This is referred to as the *prediction problem*. There are two categories of tabular solution methods. First of all there is dynamic programming (DP), which refers to the collection of methods to compute optimal policies given a perfect model of the environment as a MDP(R. S. Sutton and A. G. Barto 2018). Secondly, there are model-free methods such as temporal difference learning and Monte-Carlo learning. Each of them have a unique approach to solving the prediction and control problem. For DP methods, the complete probability distributions of all possible transitions is required. To complete a single iteration, the algorithm sweeps through the entire state space, $\mathbb{S}$. The iterative process of evaluating the policy and acting greedily upon the value function is guaranteed to converge to the optimal policy(R. S. Sutton and A. G. Barto 2018). This concept or process is referred to as General Policy Iteration (GPI), and is visualised in Figure 4.3. GPI is used in the other RL algorithms as well which shall be elaborated upon.

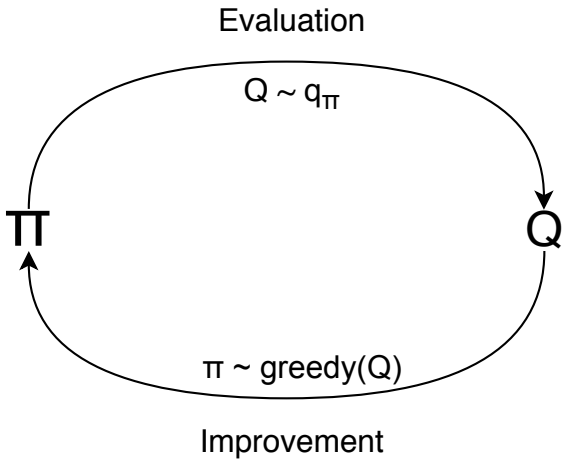


Figure 4.3: Overall idea of generalised policy iteration (GPI). Adapted from (R. S. Sutton and A. G. Barto 2018)

Model-free methods that are suited are any **temporal-difference method** (TD) or a **Monte Carlo method** (MC). Both types of methods make use of the experience acquired in sample episodes. Therefore, the agent interacts with the environment in both methods. The biggest difference between these two methods is that Monte Carlo methods have to wait until the end of the episode before the return is known, and an update can be performed, whilst TD can learn *online* after every step within the episode. TD methods namely bootstrap. They base the update in part on an existing estimate and can therefore also learn from incomplete sequences whilst MC methods cannot. TD methods are generally more practical than MC methods and widely used in the reinforcement learning. However, as TD methods bootstrap, their target will have a certain bias as it updates based on an estimate of the value function(R. S. Sutton and A. G. Barto 2018).

The idea of general policy iteration (GPI) used for DP is adapted for a model-free solution method to solve for the control problem. Whereas in DP, the value functions are computed from knowledge, the value functions are now learned from sample returns with the MDP(R. S. Sutton and A. G. Barto 2018). The return retrieved from an experience can be calculated as discounted sum, which is shown in Equation 4.14.

$$G_t = \sum_{k=0}^{T-t} \gamma^k R_{t+k+1} \tag{4.14}$$

For Monte Carlo methods, the return is used to update the action values. This is shown in Equation 4.15.

$$\begin{aligned} v_\pi^{n+1}(s_t) &= \frac{1}{n}\sum_{i=1}^{n} G_t \\ &= v_\pi^n(s_t) + \frac{1}{n}[G_t - v_\pi^{j+1}(s_t)] \\ &= v_\pi^n(s_t) + \alpha[G_t - v_\pi^{j+1}(s_t)] \end{aligned} \tag{4.15}$$

Next to Monte Carlo, there are TD methods which do not need the episode to wait until the end of an episode to learn and update the policy. The simplest form of TD methods is called TD(0), one-step TD, and entails updating the value function as follows:

$$v_\pi^{j+1}(s_t) \longleftarrow v_\pi^j(s_t) + \alpha[R_{t+1} + \gamma v_\pi^j(s_{t+1}) - v_\pi^{j+1}(s_t)] \tag{4.16}$$

Essentially, the target of MC update is G_t, obtained after an entire episode, whilst the target of an one-step TD update is $R_{t+1} + \gamma v_\pi(s_{t+1})$, which can be computed after a single step (R. S. Sutton and A. G. Barto 2018). TD methods base its update on an estimate, they namely sample the expected values and use the current estimate of v_π.

Neither Monte Carlo nor one-step Temporal Difference methods are always the best(R. S. Sutton and A. G. Barto 2018). The ideologies of both methods can be combined in so called n-step TD methods, which generalise both methods so one can shift smoothly between the two. An important limiting factor of the one-step TD methods is that the time step determines both how often an action is taken as well as the interval over which bootstrapping is performed. N-step TD methods decouples these two, enabling bootstrapping to occur over multiple steps. Whereas one-step TD methods considers one next reward and one next state, 2-step TD methods consider the next two rewards and the estimated value of the two steps ahead. This is visualised in Figure 4.4. In case n is increased to the number of samples in an episode, the n-step TD method is a Monte Carlo method.

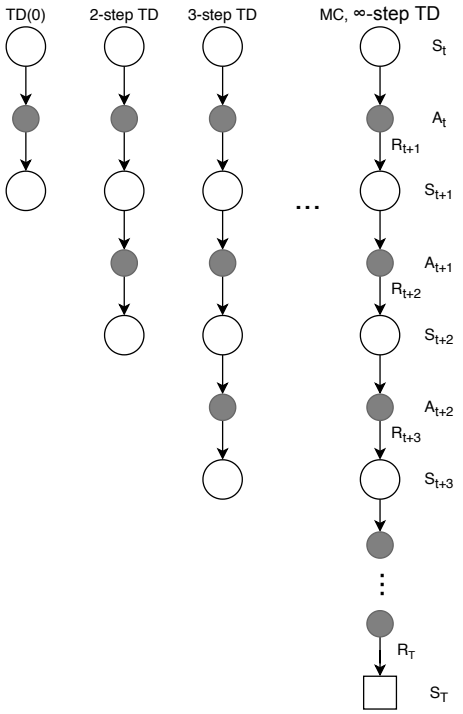


Figure 4.4: Backup diagrams of n-step methods. Adapted from (R. S. Sutton and A. G. Barto 2018).

One of the problems that needs to be addressed by all the learning control methods is **the conflict between exploration and exploitation**. An artificial agents seek to learn action values conditional on subsequent optimal behaviour, but they also need to behave non-optimally in order to explore all actions and possibly find the optimal solution(R. S. Sutton and A. G. Barto 2018). In general, there are two types of approaches to this challenge, the on-policy and off-policy approach.

On-Policy Methods

On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data. The policy being learned is called the *target policy*, whilst the policy used to make decisions is called the *behaviour policy*. Off-policy methods often show more variance and therefore converge more slowly than on-policy methods. However, off-policies can be more powerful and general (R. S. Sutton and A. G. Barto 2018). First, the on-policy model-free control methods are discussed.

To implement model-free methods in control, the same ideology as in DP, generalised policy iteration, can be used. In GPI, both an approximate policy and approximate value function are maintained. There are two types of on-policy control methods that use Monte Carlo methods, either first-visit MC or every-visit MC. Policy evaluation is performed according to Equation 4.15, using the action-value function instead of the state-value function. Policy improvement is performed by making the policy greedy w.r.t. the current value function(R. S. Sutton and A. G. Barto 2018), which means that it tries to maximise this value function. For MC methods it makes sense to alternate between evaluation and improvement on an episode-by-episode basis, since MC requires the episode to be finished before being able to evaluate.

To ensure the agent continues to select exploratory actions, several approaches exist. In RL, the most common approach to ensure exploration as well as exploitation is by implementing ϵ-greedy policies. This means that the agent takes an exploratory action, thus selects a random action, with a probability of ϵ. This means that the greedy action has a probability to be selected of $1-\epsilon+\frac{\epsilon}{|\mathscr{A}(s)|}$. The greedy selection is given by Equation 7.5, which essentially selects the action a for which the action value is maximised.

$$a_t \doteq \underset{a}{argmax}\, Q_t(s_t, \mathbb{A}(s_t)) \tag{4.17}$$

The algorithm used for on-policy TD control is called *SARSA*(State-Action-Reward-State-Action). For on-policy methods, first the state-action value function must be estimated to update the current policy π. This is

shown in Equation 4.18. Similar to the state-value function, the action-value update is based on an existing estimate, $R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})$.

$$Q_\pi(s,a) \longleftarrow Q_\pi(s,a) + \alpha[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)]. \tag{4.18}$$

As in all on-policy methods, the action-value function for the behaviour policy π is continuously updated, whilst at the same time changing the policy towards greediness w.r.t q_π(R. S. Sutton and A. G. Barto 2018). Lastly, the on-policy control method which implements n-step TD prediction is called n-step SARSA.

Off-Policy Methods

As with on-policy methods, there exists both off-policy MC as well as TD methods. The most well known off-policy TD methods is Q-learning, which was a major breakthrough in RL. Q-learning is defined by:

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, A(s_{t+1})) - Q(s_t, a_t)] \tag{4.19}$$

In this case, the learned action-value function Q, directly approximates $q_\star$(R. S. Sutton and A. G. Barto 2018). For correct convergence to the optimal policy, the only requirement is that all state-action pairs continue to be updated, which is a minimal requirement since all methods finding the optimal policy must require it. Q-learning is considered an off-policy method since it can learn from experience obtained from any policy.

Another method using the scheme of Q-learning is called Expected SARSA. In Expected SARSA, the algorithm does not take the maximum over next state-action pairs but uses the expected value, taking into account the likeliness of a certain action under the current policy(R. S. Sutton and A. G. Barto 2018). The action-value update now becomes:

$$Q_\pi(s_t, a_t) \longleftarrow Q_\pi(s_t, a_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q_\pi(s_{t+1}, a) - Q_\pi(s_t, a_t)] \tag{4.20}$$

Expected SARSA is computationally more demanding, but eliminates the variance obtained from the random selection of A_{t+1}. Given the same experience, Expected SARSA therefore performs slightly better than the regular SARSA algorithm(R. S. Sutton and A. G. Barto 2018).

A pitfall for Q-learning and SARSA is that these algorithms both involve maximisation over estimated values. This can induce a positive bias, called the *maximisation bias*. To visualise this, imagine being in a state from which one can choose three actions, of which the true action-value function equals zero. The agent, however, sees the estimated values, which are uncertain and therefore distributed around zero. Although the true maximum action-value for all actions is zero, the maximum will now be a positive value (R. S. Sutton and A. G. Barto 2018).

A concept used to overcome this maximisation bias is *double learning*. In double learning, the episodes are divided into two sets. Two independent estimates, $Q_1(a)$ and $Q_2(a)$, each try to estimate the true value $q(a)$. One of these can than be used to determine the maximisation action, $A^\star = \underset{a}{argmax} Q_1(a)$, and the other can be used to estimate the value, $Q_2(A^\star = \underset{a}{argmax} Q_1(a))$. The estimate is unbiased since using this concept resolves in the following: $\mathbb{E}[Q_2(A^\star)] = q(A^\star)$. The role of the independent estimates can be reversed to get a second unbiased estimate.

Having explored the possibilities within model-based methods and model-free methods, these two sets of solution methods can be unified. In DP, the distribution model of the environment must be known. This entails that the probabilities of next states and rewards for possible actions should be available. Such a model is difficult to obtain. Alternatively, a *sample model* produces single transitions and rewards, similar to those used by most of the model-free RL methods elaborated upon in this chapter. In DP methods, the optimal behaviour is planned, whereas in model-free methods, it is learned. It is straightforward to implement planning methods which are based on an estimate of the model. To learn the model, (state, action) are mapped to (next state, reward) (R. S. Sutton 1991). The RL agent then fully trusts its estimate of the model to plan the optimal behaviour. One of these methods that uses model-learning is called Dyna, of which the working principles is shown in Figure 4.5. The ideology of planning and learning of optimal behaviour can be brought together using this iterative process.

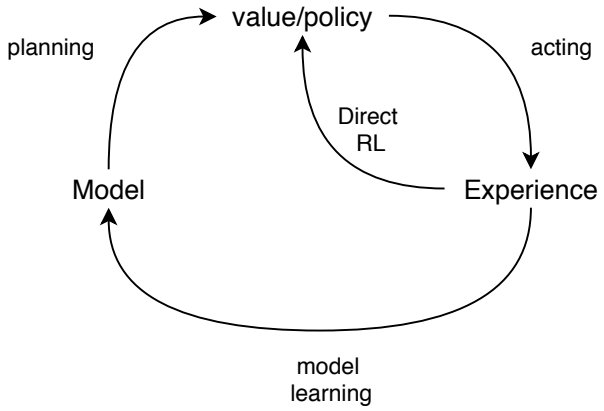


Figure 4.5: Dyna.

4.3. Approximate Solution Methods

Tabular solution methods are well-suited for problems with limited state and action spaces. Whenever the state and action space becomes too large, approximate solution methods need to be implemented. In most of the problems which need to be solved using approximate learning methods, the states encountered will probably have never been seen by the agent before. Therefore, one of the big challenges in these solution methods is *generalisation*. Since generalisation has been studied extensively, the RL methods can be extended with existing generalisation techniques. One of these techniques that RL often requires is function approximation(R. S. Sutton and A. G. Barto 2018). Function approximation can be classified as supervised learning since it expects to retrieve correct input-output behaviour of the function it tries to estimate. Broadly speaking there are two types of RL solution methods for the approximate learning case which both incorporate function approximators. First of all, there are value-based methods and secondly there are policy-based methods.

4.3.1. Value-Based Methods

Instead of representing the value function as a table, it is represented as a parameterised function with a weight vector $\mathbf{w} \in R^D$. Thus, $\hat{v}(s,w) \approx v_\pi(s)$, in which s is an approximate state value. $\hat{v}$ might be a linear function approximator such as a polynomial, but also an artificial multi-layer neural network in which w is a vector representing the connection weights. In approximate learning methods, the weight vector most often has a dimension which is much smaller than the number of states. In that case, a change of a single weight influences multiple states.

In the tabular case, an explicit objective did not have to be specified for the prediction problem since the value function could equal the true value function. An objective often used is the mean squared value error (VE), which is shown in Equation 4.21. The VE gives an indication to what degree the approximation values differ from the true values. As can be seen, the equation for the VE includes μ_s, which is the state distribution representing relative importance of the error in each state s. $\mu(s)$ is often chosen as fraction of the time spent in a certain state s, also referred to as the *on-policy distribution*.

$$\bar{VE}(w) \doteq \sum_{s \in \mathbb{S}} \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2 \tag{4.21}$$

One of the most widely used function approximators are stochastic gradient descent methods (SGD). SGD methods are especially well-suited for online reinforcement learning. In gradient descent methods, the weight vector has a fixed number of real-valued components. Each step, this weight vector is adjusted by a small amount in the direction that would reduce the error on the sample. In fact, the weight is updated proportional to the negative gradient of the sample's squared error(R. S. Sutton and A. G. Barto 2018). This is shown in Equation 4.22. If α decreases over time in a way to satisfy the standard stochastic approximation conditions, then the SGD method is guaranteed to converge to a local optimum(R. S. Sutton and A. G. Barto 2018).

$$w_{t+1} \doteq w_t - \frac{1}{2}\alpha\Delta[v_\pi(s_t) - \hat{v}(s_t,w_t)]^2 = w_t + \alpha[v_\pi(s_t) - \hat{v}(s_t,w)]\Delta\hat{v}(s_t,w_t), \tag{4.22}$$

in which the gradient of f with respect to w is defined as follows:

$$\Delta f(\mathbf{w}) \doteq (\frac{\delta f(w)}{\delta w_1}, \frac{\delta f(\mathbf{w})}{\delta w_2}, \ldots, \frac{\delta f(\mathbf{w})}{\delta w_d})^T. \tag{4.23}$$

In case the precise value of a state is unknown, an unbiased estimate for the value of a state can be used, U_t. There are several different RL methods which use gradient-descent such as the Gradient Monte Carlo Algorithm. These algorithms differ in the way the estimate of the value function is derived and updated. For a Gradient Monte Carlo Algorithm, the weight update would use the return of an episode, G_t, as unbiased estimate of $v_\pi(S_t)$:

$$w \longleftarrow w + \alpha[G_t - \hat{v}(s_t, w)]\Delta\hat{v}(s_t, w) \tag{4.24}$$

Whenever a bootstrapping estimate is used for the target U_t, the convergence guarantees do not hold. These methods are semi-gradient methods. One of the advantages of semi-gradient methods is that these enable significant faster learning, and they allow for continuous and online learning without having to wait for the end of an episode. A typical semi-gradient method is semi-gradient TD(0), in which the value estimate $U_t \doteq R_{t+1} + \gamma\hat{v}(s_{t+1}, w)$. To improve the generalisation in function approximation, *state aggregation* can be used. This technique groups states together, with one estimated value for each group.

An approximate function which is a linear function of the weight vector is called a linear method. In linear methods, there is a real-valued vector corresponding to every state, $x(s)$, with the same amount of components as the weight vector. This vector is called the *feature vector*. The state-value function approximate is then composed by the inner product of the feature vector and the weight vector:

$$\hat{v}(s, w) \doteq w^T x(s) \doteq \sum_{i=1}^{d} w_i x_i(s). \tag{4.25}$$

Since $\Delta\hat{v}(s, w) = x(s)$, SGD updates can be used in linear function approximation. These methods prove to converge to or near a global optimum since it is only one optimum(R. S. Sutton and A. G. Barto 2018). An example of a linear method which incorporates SGD as a learning method is n-step semi-gradient TD. The details of this algorithm can be found in (R. S. Sutton and A. G. Barto 2018). There are several methods to effectively represent the feature vector for linear methods. These are summarised below:

- **Polynomials**: for some tasks, separate state representations alone do not effectively represent the state. One might be more interested in the interaction of these states and therefore these interactions can be added as separate states. The state vector for a k-dimensional state-space can then be represented as a polynomial: $x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}}$, in which $c_{i,j}$ is an integer in the set $0, 1, ..., n$ for an integer $n \geq 0$. Higher-order polynomials generally allow for more accurate state representations. However, the number of features grows exponentially in a polynomial. Therefore, one must choose a subset of these based on prior beliefs and knowledge. The curse of dimensionality should be taken into account in choosing a subset of the polynomials. In general, polynomials are not recommended for online learning.

- **Fourier Basis**: the feature vector is here represented as a sum of sines and cosines. The advantage of using Fourier series is that the approximation can filter out high-frequency components which are considered as noise. However, this also results in the fact that local properties are difficult to represent.

- **Coarse Coding, Tile Coding**: in coarse coding, the state-space is composed of circles representing features. In case the state is inside a circle, that feature is said to be present and retrieves the value 1. The other features are then 0 and called *absent*. In tile coding, *hashing* can be used to reduce memory. Hashing is a method to reduce features with a potentially infinite feature vector to a table with fixed-size.

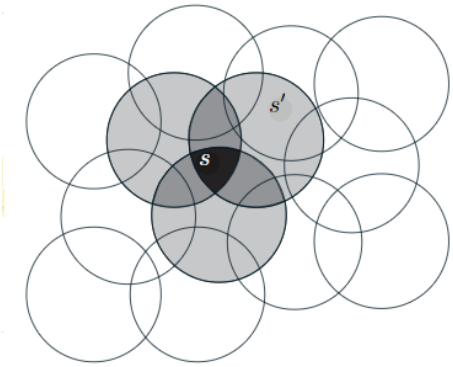


Figure 4.6: Coarse coding. It is a generalisation technique in which state s is transformed to s'. The receptive fields, in this case circles, have one feature in common (the overlapping area). Therefore there will be slight generalisation between them.

- **Radial Basis Functions**: natural generalization of coarse coding to continuous-valued features.

Apart from the linear feature representations, there are also nonlinear function approximators. Among these, the most widely used are the Artificial Neural Networks (ANNs). A detailed explanation on neural networks is given in chapter 6.

4.3.2. Policy Gradient Methods

Whereas for value-based methods the algorithms learn the values and select actions based on these learned values, policy gradient methods instead learn a parameterised policy that can select actions without consulting a value function. These methods learn the policy parameter based on the gradient of a scalar performance measure $J(\theta)$ w.r.t. the policy parameter. The algorithm seeks to optimise the performance, and therefore updates using stochastic gradient ascent in J, as shown in Equation 4.26. If J is a loss function, one wants to minimise this objective. To do so, updates are performed using *gradient descent*. Methods employing stochastic gradient descent are well-suited for online learning. Online learning means that the agent can learn learn from data that becomes available in a sequential order, instead of having to update on the entire training set.

$$\theta_{t+1} = \theta_t + \alpha \Delta \hat{J}(\theta_t), \tag{4.26}$$

In this equation $\Delta \hat{J}(\theta_t) \in \mathbb{R}^{d'}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure w.r.t. its arguments θ_t (R. S. Sutton and A. G. Barto 2018).

Methods which learn the value function next to the policy, are called Actor-Critic Methods (AC). In these methods, the 'actor' and 'critic' refer to the learned policy and the learned value function respectively. The policy can be parameterised in any way, as long as $\pi(a|s,\theta)$ is differentiable w.r.t. its parameters. That is as long as the gradient of the policy, $\Delta\pi(a|s,\theta)$, exists and is finite for all $s \in S$, $a \in A(s)$, and $\theta \in \mathbb{R}^{d'}$. A parameterisation suitable for environments with limited action spaces is called *soft-max in action preferences*. The actions with the highest preference, defined by $h(s,a,\theta) \in \mathbb{R}$, are given the highest probabilities of being selected. The actions themselves can be parameterised arbitrarily by e.g. a deep artificial neural network or a linear function.

In action value methods, an ϵ-greedy action selection is performed. Approximating a policy has three main advantages over ϵ-greedy action selection:

- Approximate policy can approach a deterministic policy whereas for ϵ-greedy selection there is always a probability, ϵ, of selecting a random action.
- Enables the selection of actions with arbitrary probabilities which might be useful for card games in which incomplete information is available to the agent. ϵ-greedy have no natural way to produce stochastic policies.
- For a specific problem, the policy might be a simpler function to approximate than the action value function, for which a PD method will learn faster and yield a superior policy (R. S. Sutton and A. G. Barto 2018).

In PG methods, one wants to know the performance gradient w.r.t. the policy parameter. However, this gradient is dependent on both the action selections and the distribution of the states in which these selections are made. The latter effect is often unknown since it is a function of the environment. In the episodic scenario, the *Policy Gradient Theorem* provides a solution to this problem. This theorem derives an expression for the performance w.r.t. the policy parameter which is not dependent on the unknown distribution of the states, shown in Equation 4.27.

$$\Delta J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \Delta\pi(a|s,\theta), \tag{4.27}$$

in which $\mu(s)$ is the on-policy distribution under π and often chosen as the fraction of the time spent in state s. One of the first policy gradient methods to be developed was REINFORCE. REINFORCE is a Monte Carlo policy gradient algorithm and therefore uses the return retrieved through an entire episode to update its policy. The derivation from the policy gradient theorem to an expression in which the return can be used is shown in Equation 4.28

$$\begin{aligned}
\Delta J(\theta) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \Delta\pi(a|s,\theta) \\
&= E_\pi[\sum_a q_\pi(s_t,a) \Delta\pi(a|s_t,\theta)] \\
&= E_\pi[G_t \frac{\Delta\pi(a|s_t,\theta)}{\pi(a|s_t,\theta)}] \qquad \text{;Because } q_\pi(s,a) = \mathbb{E}_\pi[G_t|s,a]
\end{aligned} \tag{4.28}$$

This derivation results in the REINFORCE update according to the generic stochastic gradient ascent algorithm(Equation 4.26):

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \frac{\Delta\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)} \tag{4.29}$$

An often implemented extension to this algorithm is *Reinforce with Baseline*. In this method a baseline value is subtracted from the return, G_t, to reduce the variance of the gradient estimation whilst not influencing the bias. A more detailed explanation can be found in (R. S. Sutton and A. G. Barto 2018). Like all Monte Carlo methods, REINFORCE tends to learn slow and is inconvenient for online implementations or continuing problems. As seen in the tabular case and for value approximation methods, temporal difference methods provide a solution since these methods bootstrap. The policy gradient methods suitable for continuing problems are actor-critic methods with a bootstrapping critic. Another advantage is that the bias introduced through bootstrapping is often beneficial to the learning speed as it reduces variance. The actor and critic are defined as follows:

- **Critic**: the critic updates the weights of either the action-value function or the state-value function, $\hat{q}(s,a,w)$ or $\hat{v}(s,w)$ depending on the algorithm.
- **Actor**: the actor updates the policy parameters θ of $\pi(s|a,\theta)$, in the direction indicated by the critic.

For the episodic one-step actor-critic method, the full return as used in REINFORCE is replaced by the one-step return.

$$\begin{aligned}
\theta_{t+1} &\doteq \theta_t + \alpha(R_{t+1} + \gamma\hat{v}(s_{t+1},w) - \hat{v}(s_t,w)) \frac{\Delta\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)} \\
&= \theta_t + \alpha\delta_t \frac{\Delta\pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}
\end{aligned} \tag{4.30}$$

This ideology can easily be extended with an n-step return or the addition of eligibility traces. For Policy gradient methods used for continuing problems, the performance should be defined in terms of the average rate of reward per time step (R. S. Sutton and A. G. Barto 2018).

$$\begin{aligned}
J(\theta) \doteq r(\pi) &\doteq \lim_{h\longrightarrow\infty} \frac{1}{h} \sum_{t+1}^{h} \mathbb{E}[R_t|S_0, A_{0:t-1} \sim \pi] \\
&= \lim_{t\longrightarrow\infty} \mathbb{E}[R_t|S_0, A_{0:t-1} \sim \pi]
\end{aligned} \tag{4.31}$$

The pseudo code for an on-policy actor-critic method is given in 1. In this algorithm one can clearly see how the TD error is influenced by the critic and how this influences the update of the policy parameter.

Algorithm 1 Actor-Critic with Eligibility Traces (continuing), for estimation $\pi_\theta \approx \pi_\star$. Adapted from (R. S. Sutton and A. G. Barto 2018).

Require: A differentiable policy parameterisation $\pi(a|s,\theta)$ A differentialbe state-value function parameterisation $\hat{v}(s,w)$

Parameters: : trace-decay rates $\lambda^\theta \in [0,1]$, $\lambda^w \in [0,1]$; step sizes $\alpha^\theta > 0, \alpha^w > 0, \alpha^R > 0$

initialise: Policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g. to 0)
for episode n in episodes **do**
  Initialise s_0
  $z^\theta \longleftarrow 0$
  (d'-component eligibility trace vector)
  $z^w \longleftarrow 0$ (d-component eligibility trace vector)
  I $\longleftarrow 1$
  while s_t is not terminal **do**
    $a_t \sim \pi(\cdot|s_t,\theta)$ Take action a_t, observe s_{t+1}, R_t
    $\delta \longleftarrow R_t - \bar{R}_t + \hat{v}(s_{t+1},w) - \hat{v}(s_t,w)$
    $\bar{R} \longleftarrow \bar{R} + \alpha^{\bar{R}}\delta$
    $z^w \longleftarrow \gamma\lambda^w z^w + \Delta\hat{v}(s_t,w)$
    $z^\theta \longleftarrow \gamma\lambda^\theta z^\theta + I\Delta ln\pi(a_t|s_t,\theta)$
    $w \longleftarrow w + \alpha^w \delta z^w$
    $\theta \longleftarrow \theta + \alpha^\theta \delta z^\theta$
    $I \longleftarrow \gamma I$
    $s_t \longleftarrow s_{t+1}$
  end while
end for

1em

Up until now, only on-policy gradient descent and ascend methods have been covered. In other words, the training samples are gathered according to the target policy. However, off-policy actor-critic methods provide some additional advantages:

- Experience replay can be implemented, increasing the data efficiency
- Since the samples are gathered with a behaviour policy, these algorithms have better exploration properties.

The off-policy gradient descent methods shall be discussed in more detail in the next chapter. An overview of the fundamental algorithms discussed in this chapter is shown in Figure 4.7 and Figure 4.8.

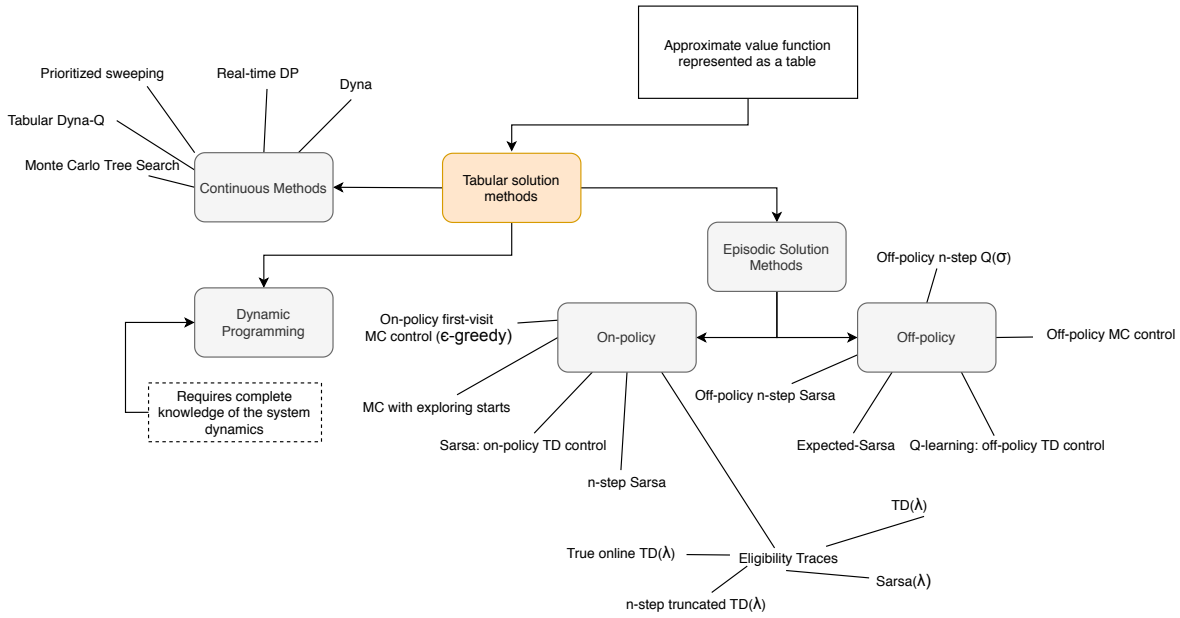


Figure 4.7: Overview of tabular RL solution methods. In this overview, a clear distinction is made between continuous and episodic methods, and between on- and off-policy methods.

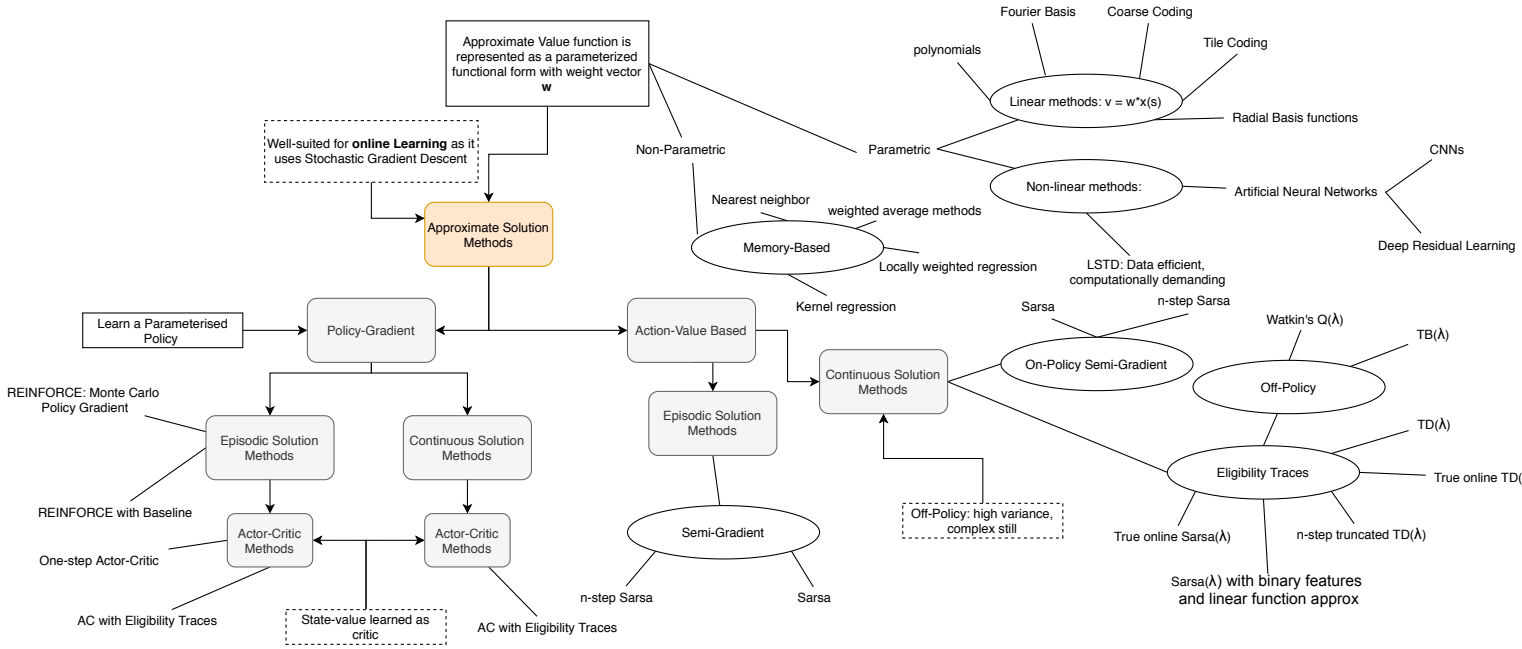


Figure 4.8: Overview of approximate RL solution methods. In this overview, a clear distinction is made between continuous and episodic methods, and between on- and off-policy methods. Methods employing stochastic gradient descent are well-suited for online RL.

4.4. Concluding Remarks

In this chapter, an introduction to MDPs and the associated tabular and approximate solution methods for RL is given. The CD&R task of ATC can be extremely simplified in order to be suited for tabular solution methods. Although applying tabular solution methods might prove that RL can be used to optimise traffic scenarios, the solution cannot be extended to realistic traffic situations due to the increment in state and action spaces. Past automation efforts of the CD&R task, which were discussed in section 2.5 have achieved promising results in upholding safe operations by using deep reinforcement learning. In deep RL, the state-

action value function and/or policy is approximated using deep neural networks. This category of methods belongs to the approximate solution methods.

Before surveying state-of-the-art deep RL algorithms, the next chapter introduces the main concepts of deep learning.

5

Deep Learning: Extracting Information from Visual Imagery

In 2012, a great milestone was achieved when an agent trained with deep learning methods had matched human performance on image recognition tasks (Cireşan, Meier, and Schmidhuber 2012). From that moment on, deep learning techniques have been applied to a wide variety of tasks involving visual imagery. The raw pixel-date of the SSD will be used as input for the RL algorithm. Therefore, deep neural networks shall be used to approximate a value function or a policy directly. In this chapter the working principles of Deep Learning are set out.

First, section 5.1 introduces two commonly used neural networks: Artificial (ANN) and Convolutional Neural Networks (CNNs). Then, section 5.2 explains the activation functions required in deep neural networks. Having explained some of the fundamentals of RL, an example of an ANN is setup in section 5.3 which is used throughout the remainder of this chapter. Section 5.4 elaborates on the objective function of deep learning algorithms. section 5.5 explores the different optimisation algorithms available to meet the objective. Finally, the regularisation techniques used to avoid overfitting on the training data are explained in section 5.6

5.1. Artificial and Convolutional Neural Networks

Networks with deep architectures can automatically create learning features instead of fully relying on "hand-crafted" features (Sutton and Barto 2018). In deep learning, there are three general types of neural networks: ANNs, CNNs and Recurrent Neural Networks (RNNs). The focus of this chapter is only on ANNs and CNNs. The reason for this is that RNNs are useful for sequence prediction problems such as analysing language, which is not applicable to this research.

An ANN, also known as Multilayer Perceptrons (MLPs), is useful for regression and classification prediction given (tabular) input data. In this research, an ANN can be useful if the ATC environment is represented by a state-space consisting of multiple parameters such as the coordinates of the aircraft in the sector. An ANN is composed of an input, output and a set of "hidden" layers. The amount of hidden layers represent the ‘deepness’ of the neural network. An example of an ANN is given in Figure 5.1. This example has three inputs, two outputs and two hidden layers. During training of a RL agent, the network weights are updated in order for the network to accurately estimate the outputs, given the inputs.

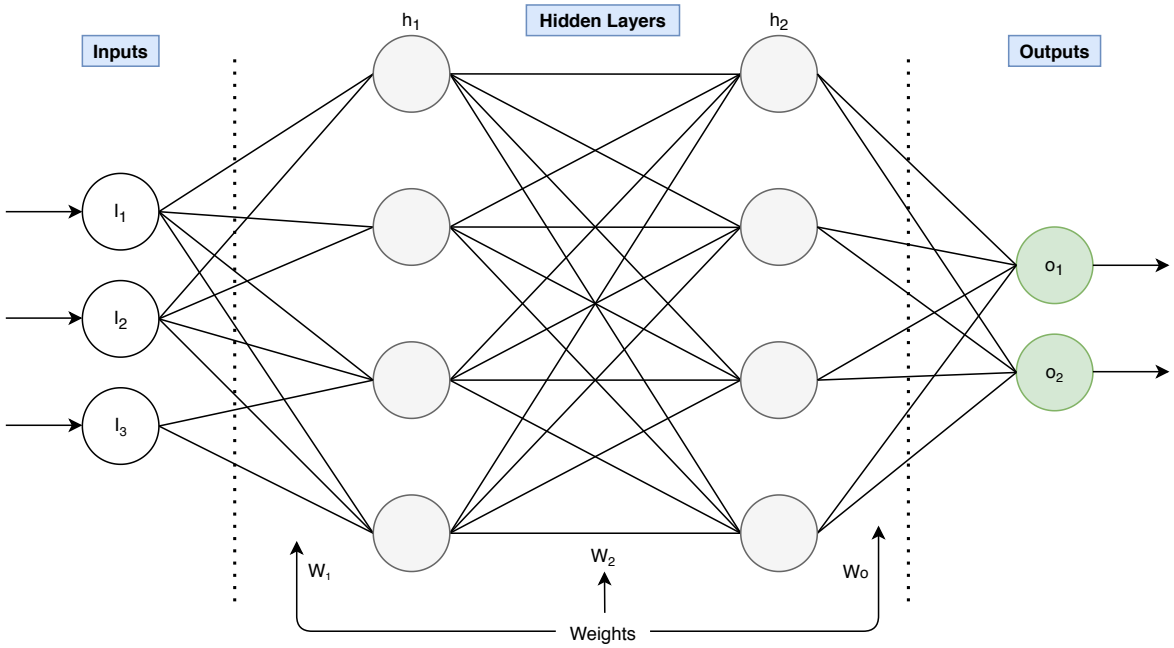


Figure 5.1: Example of an ANN with two hidden layers.

A class of deep learning networks which is generally applied to visual imagery, or any other grid-like topology, is called Convolutional Neural Networks (Goodfellow, Bengio, and Courville 2016). The definition as given in (Goodfellow, Bengio, and Courville 2016) is the following: "*Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.*" By applying multiple convolutions, or filters, visual features such as edges and lines can be extracted from an image. A convolution is denoted by an astrix and the standard convolution operation applied in CNNs is shown in Equation 5.1. In this equation k is referred to as the **kernel function**, which is a probability density function. The output is essentially a weighted average and is often referred to as the **feature map**.

$$s(t) = (x \star k)(t) = \int_{-\infty}^{\infty} x(\tau)k(t-\tau)d\tau \tag{5.1}$$

When convolutions are applied to multi-dimensional input arrays, the kernel is also a multi-dimensional array of which the parameters are adjusted during training. An example of this is shown in Figure 5.2. During training, the parameters of the kernel, w, x, y and z, are adapted. One can also see that the kernel is essentially slided over the input image to generate the values of the new 2x3 output map.

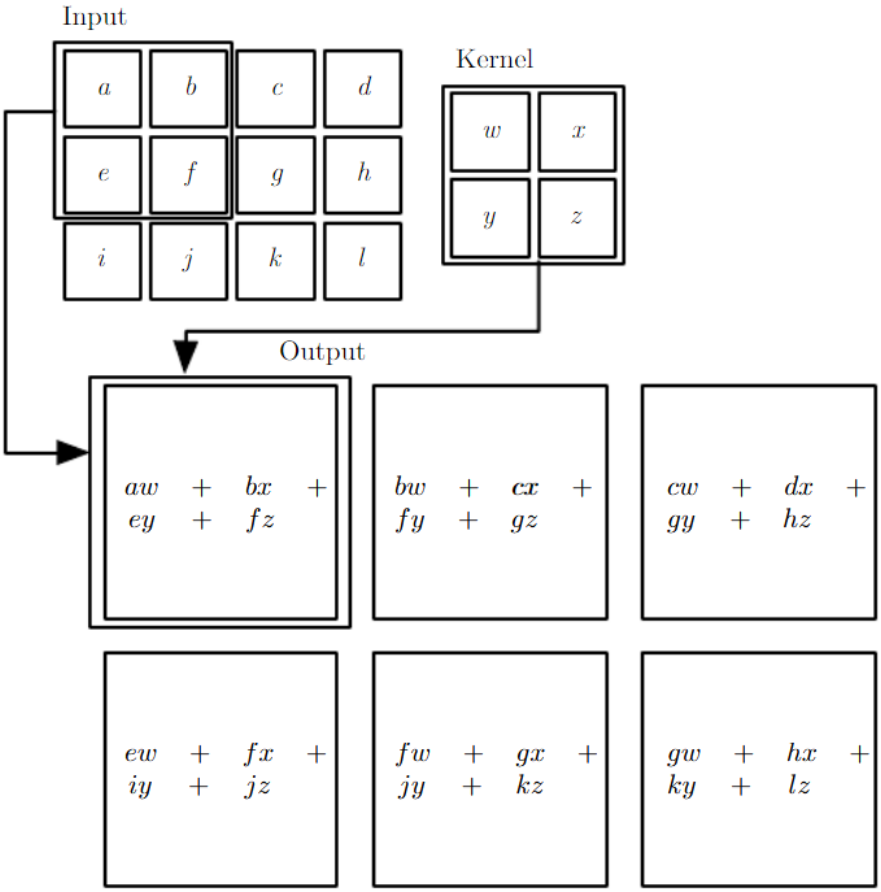


Figure 5.2: Example of a 2D convolution operation. In this image, the kernel matrix has a width of 2 and height of 2. (Goodfellow, Bengio, and Courville 2016)

One of the advantages of a CNN with a kernel size smaller than the input is that it reduces the amount of parameters that need to be stored and improves its statistical efficiency (Goodfellow, Bengio, and Courville 2016). In a fully connected neural network, each weight is used only once when computing the output. If you have a neural network with m inputs and n outputs, the network requires mxn parameters and O(mxn) run time[1]. However, if the convolutional layer has a kernel width of three (1D example), $k = 3$, the input only affects three outputs instead of all outputs. This means that you have to store kxn parameters and O(kxn) run time. As one can see, this can significantly lower the computational complexity and memory requirements. Additionally, **parameter sharing** can be applied to further reduce the storage requirement of kxn weights down to k weights. Instead of learning weights of the kernel for every separate location in the multi-dimensional array, a single set of weights is learned.

A convolutional layer is typically composed of three stages. In the first stage, the convolution is applied. After this, a nonlinear activation function is applied to introduce nonlinearity. More details on this are described in the next section. Finally, a **pooling** layer is applied to convert the output of this nonlinear activation function to some summary statistic. The goal of this pooling layer is to ensure that the network is invariant to small local translations of the input. In Figure 5.3, an example of a *max pooling* operation is shown. A node in the final output is affected by three nodes of the input of the pooling layer, taking the maximum value of these nodes. When applying a pooling layer to a 2D array, applying a max pooling operation ensures that the output is invariant to small translations and rotations of that image. The degree to which it is invariant is dependent on the size of the pooling operation.

[1] Big O notation: indicates how the run time or space requirements grow as the input size grows (Avigad and Donnelly 2004)

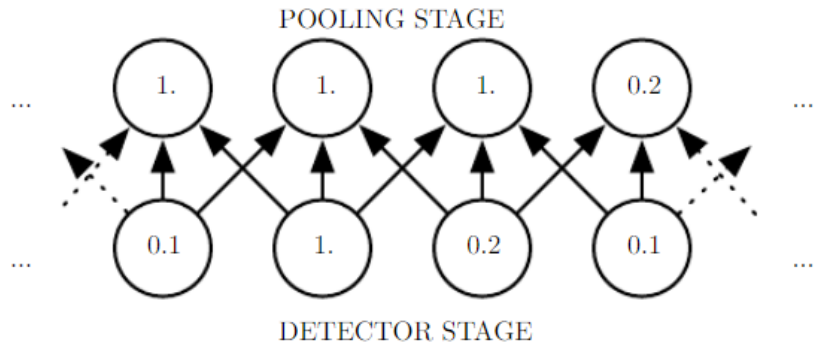


Figure 5.3: Example of a max pooling operation. The final output nodes (upper layer) are each affected by three nodes from the input of the pooling layer (lower layer). (Goodfellow, Bengio, and Courville 2016)

Next to applying a kernel, there are other variants on the basic convolution. The most popular ones are summarised below:

- **Padding**: when a kernel matrix is applied to the input image, the size of the output is dependent on the size of the input and kernel. In case the kernel size is larger than 1, the image 'shrinks', as illustrated in Figure 5.2. This causes the NN to lose information at the borders. To prevent that from happening, zero-padding can be applied. Zero padding extends the input array is extended with zeros to ensure that the image does not shrink.

- **Striding**: this defines the number of pixels the kernel shifts at a time. If the stride is 1, the kernel is shifted one pixel to the right/left/down/up at a time. If the stride value becomes larger, the image is essentially downsampled.

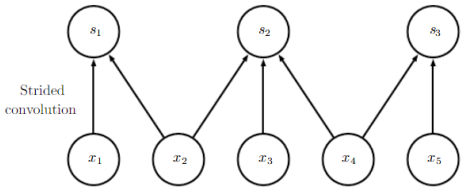


Figure 5.4: Stride operation visualised.(Goodfellow, Bengio, and Courville 2016)

In Figure 5.5, one can see an example of a CNN. In this CNN, an input image of size 32x32 is mapped to an output layer of size 10. In the first convolution layer, six different kernels of size 5x5 are applied to get an activation layer consisting of 6 feature maps (C1). The total size of the output of this operation is 28x28x6. Pixels are lost at the border as the kernel has a size of 5x5. Subsequently, an average pooling operation that takes the average of a 2x2 pixel area is applied, converting the 2x2 pixel area to a single pixel value. It essentially subsamples the array to a size of 14x14x6 (S2). In the second convolutional layer (C3), a kernel with size 5x5 is applied with 16 feature maps. Then, the same pooling layer is applied to subsample the activation layer to the size of 5x5x16 (S4). Finally, a convolutional layer is applied with 120 feature maps and a kernel of size 5x5, which flattens the activation volume into a vector (C5). The last two layers are a hidden layer with 84 (F6) neurons and an output layer of 10 neurons (OUTPUT). Finally, the softmax layer is applied to convert the predicted output classes to a probability density function.

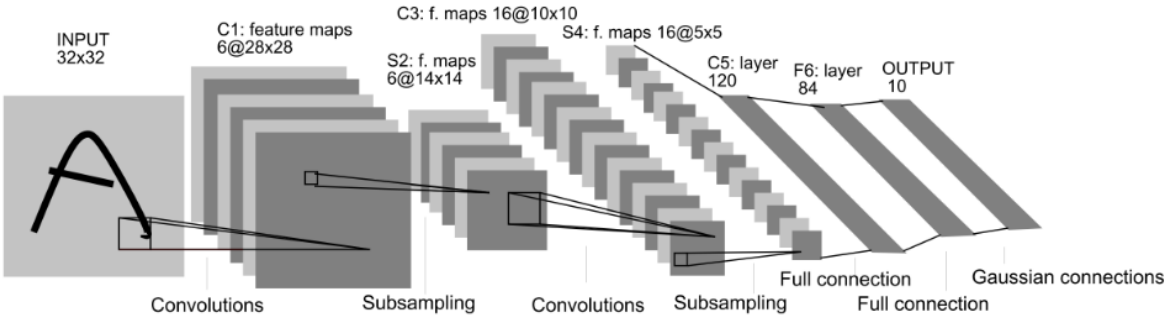


Figure 5.5: Convolutional Neural Network structure (LeNet5) for character recognition.(LeCun et al. 1998)

5.2. Activation Function

To describe the features present in a state, a nonlinear function is needed (Goodfellow, Bengio, and Courville 2016). To do so, hidden layers have activation functions which determine the hidden layer values. If no activation would be applied, the output would simply be a linear function of the input. The most commonly used activation function is the **rectified linear unit** (ReLU), which is described using Equation 5.2. As one can see in Figure 5.6, the function takes a value of zero for $z \leq 0$ and becomes a ramp function after zero.

$$g(z) = max\{0, z\} \tag{5.2}$$

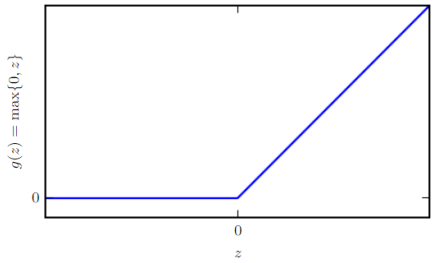


Figure 5.6: The ReLU activation function. This activation function is the default activation function recommended for use with all neural networks. Applying this function to the output of a linear function results in a nonlinear output. (Goodfellow, Bengio, and Courville 2016)

Other activation functions are sigmoid and the hyperbolic tangent functions. These however suffer from vanishing gradients, especially in deep neural networks.

5.3. Example of an ANN for Classification

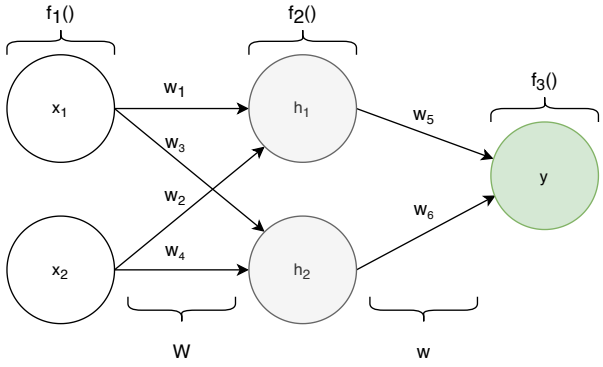


Figure 5.7: Example neural network used throughout this chapter.

Since an ANN is more intuitive as a CNN, an ANN will be used as running example for the next sections to elaborate on the optimisation phase of the learning process. The ANN, shown in Figure 5.7, has two inputs and one output. The ANN has a simple structure and is fully connected.

The goal of this ANN is to classify something based on the two inputs. So, e.g. to classify whether an aircraft is in the climbing phase based on its vertical speed and altitude. The forward propagation through this ANN is shown in Equation 5.3. In this equation, $g()$ represents any **activation function** that can be applied in the hidden layers, e.g. a maximisation. Furthermore, c^T and b represent bias nodes which are added to increase the flexibility of the model.

$$f(x;W) = g(x_1 w_1 + x_2 w_2 + c1)\, w_5 + g(x_1 w_3 + x_2 w_4 + c2)\, w_6 + b = g(W x^T + c^T) + b \tag{5.3}$$

When viewing this at a higher level, by just looking at the layers, the forward pass becomes the following:

$$f(x;W) = f^3(f^2(f^1(x))), \tag{5.4}$$

in which f_1 is the input layer, f_2 the hidden layer and f_3 the output layer. The classifier in this case thus is $y = f(x;W)$ and one wants to optimise this function to accurately classify y, $f(x;W) \longrightarrow f(x;W^\star)$.

5.4. Loss Function

In this classification example, one wants the network to classify the input correctly. To update the weights in the network, these must be updated with regard to some objective. A logical loss function, also referred to as cost function, for this example would be the mean squared error of the classification and the value in the training data set. This is shown in Equation 5.5. When updating the parameters using batches, the total loss function is a sum of these individual classification losses, as shown in Equation 5.6.

$$L(x, y; W) = (f(x;W) - y)^2 \tag{5.5}$$

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i; W) \tag{5.6}$$

The goal of the optimisation process is to minimise this loss function. Having a loss function of zero would mean that all classification in the training batch are done correctly. If a small change is made to W, the loss function changes according to:

$$L(x, y; W + \epsilon) \approx L(x, y; W) + \epsilon L'(x, y; W) \tag{5.7}$$

Whenever the gradient of the loss function is negative, a step in the positive direction needs to be taken to minimise the loss function. To minimise the loss function, a step in the opposite sign of the gradient should thus always be taken. This is called **stochastic gradient descent**. The gradient of the loss function is composed of a vector containing all the partial derivatives of the loss function with respect to the weight parameters, $\frac{dL}{dW}$. An efficient technique to compute the gradient is called **backpropagation**.

5.5. Optimisation

Having set an objective, one wants to update the weight parameters to improve the value of the objective function. A widely known algorithm to do is called **stochastic gradient descent**. In this algorithm, a step is taken in the negative direction of the gradient of the loss function. This is shown in Equation 5.8. In this equation, ϵ is the learning rate. This is an important hyper parameter to tune since a learning rate which is too high might lead to the neural network never being able to learn since it overshoots the optimum constantly. On the flip side, a learning rate which is too small will take very long to find a good solution.

$$\theta' = \theta - \epsilon \Delta_\theta J(W) \tag{5.8}$$

Since the introduction of SGD, few improvements on this algorithm have been made. One of the disadvantages of SGD is that all the weights are updated with the same step size. To improve on this, **Root Mean Square Propagation (RMSProp)** was developed. In this algorithm, parameter specific learning rates are maintained which are updated based on the exponentially weighted moving average of the mean magnitude of the gradients in the previous updates. Another gradient descent algorithm which tackles this problem is the **Adaptive**

Gradient algorithm (AdaGrad). This algorithm scales the parameter specific learning rate inversely proportional to the square root of the sum of all the previously obtained values of the gradient and is especially useful for environments in which the gradients are sparse(Goodfellow, Bengio, and Courville 2016). More details about the exact working principles of this algorithm can be found in (Duchi, Hazan, and Singer 2011).

Lastly, an algorithm which combines RMSProp with momentum was developed in (Kingma and Ba 2014) and is called the **Adam (adaptive moments) Optimisation Algorithm**. Momentum is incorporated as an estimate of the first-order moment (with exponential weighting) of the gradient. Furthermore, an estimation of the second-order momentum term is included. These estimates are corrected with a bias term to account for their initialisation at the origin(Goodfellow, Bengio, and Courville 2016). This is currently the most widely used optimiser since it slightly outperforms RMSprop in the end of the optimisation when gradients become sparser due to the incorporated bias correction terms(Ruder 2017). In Figure 5.8, a comparison between optimisation algorithms, used to train an algorithm on the MNIST data set is shown. It can be seen that the Adam algorithm slightly outperforms the other algorithms.

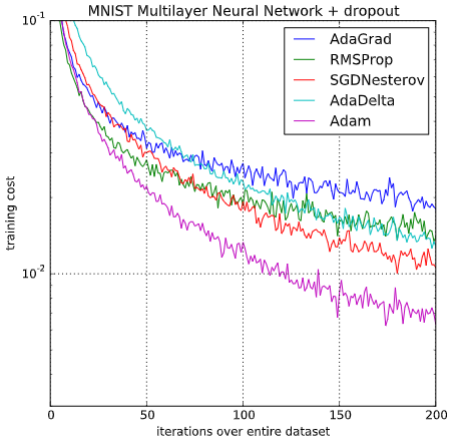


Figure 5.8: Comparison of optimisers on the MNIST data set. It can be seen that Adam slightly outperforms other algorithms. (Kingma and Ba 2014)

5.5.1. Back Propagation

Computing the gradient needed for the optimisation is computationally demanding due to the large amount of weights present in a NN. Backpropagation is an efficient method to compute gradients. It does this through re-using values. For the loss function in Equation 5.5, the gradient is composed of the partial derivatives $\frac{dL}{dW}, \frac{dL}{dc}, \frac{dL}{db}$. Let's consider the first two partial derivatives. Using the chain rule, these can be computed as shown in Equation 5.9 and Equation 5.10.

$$\frac{dL}{dW} = \frac{dL}{df}\frac{df}{dz}\frac{dg}{dW} \tag{5.9}$$

$$\frac{dL}{dc} = \frac{dL}{df}\frac{df}{dz}\frac{dg}{dc} \tag{5.10}$$

One can see that for the computation of both these partial derivatives, $\frac{dL}{df}\frac{df}{dz}$ needs to be calculated. Backpropagation uses this commonality to reuse these values once they have been calculated to efficiently compute the partial derivatives(Goodfellow, Bengio, and Courville 2016). In Equation 5.11 - Equation 5.14 it can be seen that if calculated in the correct order, values can be reused when calculating partial derivatives. In these equations, $\bar{x} = \frac{dL}{dx}$.

$$\bar{f} = \frac{dL}{df} \tag{5.11}$$

$$\bar{z} = \bar{f}\frac{df}{dz} \tag{5.12}$$

$$\bar{W} = \bar{z}\frac{dz}{dW} \tag{5.13}$$

$$\bar{c} = \bar{z}\frac{dz}{dc} \tag{5.14}$$

The topological order of computations need to be known to effectively reuse values. To get the topological order, first a computational graph is setup. The computational graph for the example is shown in Figure 5.9.

In a computational graph, nodes can be a scalar, matrix or tensor whilst the lines connecting them represent operations. From the computational graph it can be seen that the topological order of the running example is (c,x,w,z,b,y',y,L). To calculate the loss, the forward pass is performed. In the forward pass, n_i is evaluated using its function $f^{(i)}(n_i)$.

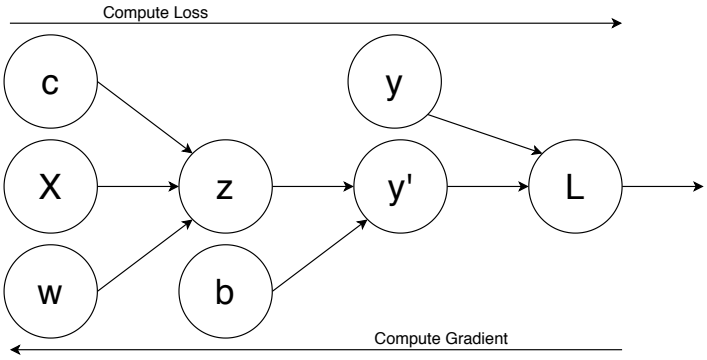


Figure 5.9: Computational graph of the example shown in Figure 5.7.

During the backward pass, the gradient is computed. To do so, only the gradient with respect to the children of that node need to be computed. This is shown in Equation 5.15. Using this equation, the gradient of bias term b would be: $\bar{b} = \bar{y'}\frac{dy'}{db}$. $\bar{n_N} = 1$ since the gradient with respect to itself is 1. Using this approach, gradients can be calculated efficiently. Each node aggregates the error signal of its children and passes on a message from its parents.

$$\bar{n_i} = \sum_{n_j \in Children(n_i)} \bar{n_j} \frac{dn_j}{dn_i} \tag{5.15}$$

5.6. Regularisation

In deep learning, one wants to find a balance between fitting the training data and having a generalised model. When fitting a function on training data, the function can underfit, fit the data just right or overfit the data.

To avoid the neural network from overfitting on the training samples, regularisation techniques can be applied to the neural network. Regularisation techniques discourage overly complex models. Below, the most widely applied regularisation techniques are explained.

- *L1 and L2 regularisation*: these are parameter norm regularisers. Essentially, an extra term is added to the loss function to construct the regularised loss. For L2 regularisation, also referred to as weight decay, this is shown in Equation 5.16. The goal of this regularisation technique is to drive the weights closer to the origin. In L1 regularisation, weights are allowed to decrease all the way to zero, essentially removing nodes and thus selecting certain features from the network. For L1 regularisation the added term is $\alpha||w||_1$.
$$\tilde{J}(w;x,y) = \frac{\alpha}{2} w^T w + J(w;x,y) \tag{5.16}$$
- *Early stopping*: prematurely stop the optimisation process to limit the overfitting on the training set. Instead of optimising towards $w^\star$, updates try to reach the regularised optimum $\tilde{w}$. This is only useful if the weights are initialised near zero.
- *Noise robustness*: adding noise the network weights and outputs encourages stability of the model. Adding noise to the input data would actually be the same as training set augmentation. (Graves 2011)
- *Dropout*: randomly remove nodes/units from the neural network during training. This essentially removes more and more complex features. More details can be found in (Geoffrey E. Hinton et al. 2012).

5.7. Concluding Remarks

In this chapter, the main concepts of Deep Learning have been explained. When representing the state of an environment using variables, one should use an ANN. For visual imagery, CNNs are used as they can

extract visual features from the image by applying multiple convolutions. Activation functions are critical for learning complex relations since these add non-linearity. In the learning phase, Deep learning algorithms try to minimise a loss function using an optimiser. In general, the best performing optimiser is the Adam optimiser. This optimiser will therefore also be used in this thesis. To optimise the loss function, optimisers update the weights in the negative direction of the gradient of the loss function. Backpropagation is applied to efficiently calculate the gradients. Lastly, one wants to avoid overfitting on the training data. To do so, one can apply different regularisation techniques such as dropout and parameter norm regularisation.

In deep reinforcement learning these concepts are used to train a network to estimate either an actor policy or a value function. Whenever the input of a network is an image, it is common to use a CNN to represent the policy or value function.

6

State-of-the-Art Reinforcement Learning Algorithms

In this chapter, deep reinforcement learning techniques are explored which can be used to develop autonomous ATC by using the SSD as input. Following the chapters introducing RL and Deep learning, this chapter aims to answer research question 2c: 'Which deep RL algorithms are useful for a multiple-aircraft traffic scenario?'. First of all, the trade-off between single agent and multi-agent RL is made in section 6.1. Since the ATC CD&R task has a certain hierarchy to it, *hierarchical reinforcement learning* is elaborated on in section 6.2. Then, in section 6.3, the state-of-the-art reinforcement learning algorithms are explained.

6.1. Single Agent vs. Multi-Agent RL (MARL)

The multi-agent environment is fundamentally more complex than the single-agent environment since multiple agents interact with the environment at the same time. Research in multi-agent RL is mainly focused on two focal points (L Bus¸oniu Bus¸oniu et al. 2008):

- Stability of the agents' learning dynamics. A
- adaption to the changing behaviour of the other agents.

MARL comes with quite a few challenges. First of all, the curse of dimensionality has an effect on the learning performance of the agents. The introduction of a new agent will add dimensions to the joint state-action space, which leads to an exponential increase of computational complexity. Secondly, non-stationarity arises since all agents are learning simultaneously. The agents therefore face a moving-target learning problem since the optimal policy varies dependent on the changing policies of the other agents. Thirdly, just as with single agent RL, the exploration-exploitation trade-off must be addressed in MARL. In the MARL scenario, the agents do not only have to explore the environment, but they should also explore other agents. Consider an agent which tries to explore the behaviour of other agents, which simultaneously are also exploring, then this might destabilise the learning of the agent (L Bus¸oniu Bus¸oniu et al. 2008). To deal with these problems, coordination between agents is required.

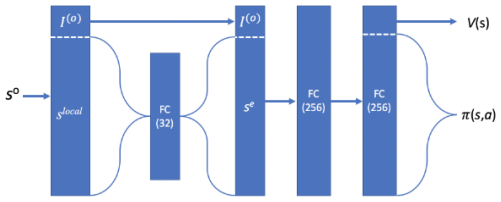
There are two main approaches to MARL, namely the *decentralised* and *centralised* approach. In the decentralised approach, independent learners directly use the algorithms which are valid for the single-agent case in the multi-agent setting. This however violates the Markov property which states that the future dynamics and reward only depends on the current state (Hernandez-Leal, Kartal, and Taylor 2019). This method can therefore fail in settings in which an opponent of the agent learns based on the past history of interactions. However, in terms of scalability, this method has proven to be advantageous. An example of such an algorithm is Dec-HDQRN (Omidshafiei et al. 2017). In centralised methods, agents have a shared critic that is provided with the policy of all agents during training and a decentralised actor to optimise the policy. These learning methods do not suffer from the non-stationarity from which decentralised methods suffer.

According to (Hernandez-Leal, Kartal, and Taylor 2019), modern MARL algorithms can further be subdivided into four categories:

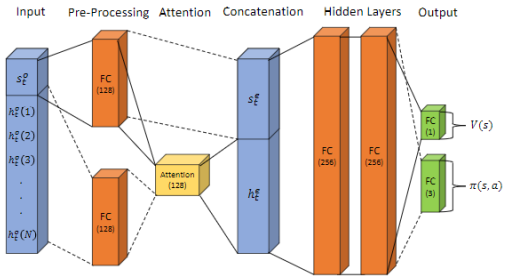
- Emergent behaviours: these algorithms explore the use of classical RL algorithms for a single agent setting in the multi agent setting.

- Learning communication: in these works, agents share information with each other through certain communication protocols.
- Learning cooperation: most research in this category is focused on fostering cooperation in learning agents.
- Agents modelling agents: these works are focused on agents which try to construct models of the other agents to predict their future behaviour in the environment.

Successful implementations of multi-agent RL to automate ATC were developed in (Brittain and Wei 2019; Brittain, Yang, and Wei 2020). In (Brittain and Wei 2019), a Deep Distributed Multi-Agent Reinforcement Learning framework (DD-MARL) was implemented. In this framework, a centralised learning with decentralised execution framework is utilised. The learning architecture consists of a single neural network in which the actor and critic share layers of the same neural network, reducing the number of parameters to be trained. This is shown in Figure 6.1a.



(a) Learner neural network architecture used in (Brittain and Wei 2019)



(b) Learner neural network architecture used in (Brittain, Yang, and Wei 2020)

Figure 6.1

A common disadvantage of many multi-agent RL problems is that an increase in agents, also increases the shared state dimensions. The effect of this relates to the curse of dimensionality, but is also a limitation in the sense that general feedforward neural networks typically require a fixed-size input which prohibits a decrement or increment in the initial number of agents. This problem was also addressed in (Brittain and Wei 2019) by letting each agent have access to the state information of N-closest agents. The hyper-parameter N had to be tuned through experimentation in a single traffic scenario, limiting the transferability to other environments. (Brittain, Yang, and Wei 2020) addressed this problem by implementing an attention layer, shown in Figure 6.1b, which can have a variable input size. In the research, it is shown that the performance is invariant to the number of aircraft and therefore poses a promising solution for ATC applications. Furthermore, the attention variant also showed to converge in a fewer number of training episodes than the neural network architecture incorporating the information of N-nearest agents.

Although these are promising results, the multi-agent setting nevertheless increases computational complexity. Non-stationarity in the learning process is encountered since coordination between agents is needed, which might lead to divergence. (Hoff 2020) used a decentralised multi-agent reinforcement learning algorithm to automate ATC. It was found that in more complex and high-density air traffic scenarios, this method failed to find a solution due to a lack of coordination between the agents. Furthermore, it is hypothesised that automating ATC in a multi-agent RL setting will increase the workload of ATCOs to monitor the automation. This is because agents can perform actions simultaneously. Since one of the goals of this research is to develop an explainable form of autonomous ATC, it is chosen to automate the CD&R task using a single agent setting. This type of automation is more conformal with how ATCOs currently approach the CD&R task.

6.2. Hierarchical Reinforcement Learning

In this section, an introduction to hierarchical reinforcement learning is given. Whereas deep learning relies on a hierarchy of features, hierarchical learning relies on a hierarchy of policies. The main concept is that apart from primitive actions, policies should also be able to run sub-policies which are suitable for fine control (Arulkumaran, Deisenroth, et al. 2017). Many real-world problems contain a certain hierarchy. This hierarchy can be translated to the world of reinforcement learning and create a sample efficient solution method. To translate the hierarchy found in real-life to reinforcement learning, the MDP setting defined for RL must

be adapted. Whereas in an MDP, the state transitions occur at a discrete time step, the state transitions for hierarchical RL can occur at irregular time intervals. This is because sub-actions in a hierarchy might take an irregular amount of time. A setting that does so is called a Semi-Markov Decision Process(SMDP). The state transition now does not only depend on the state and action, but also on the time elapsed since the action has been taken, $p(s'|s,a) \longleftarrow p(s',\tau|s,a)$.

There are three main techniques which implement hierarchy in reinforcement learning:

1. **Feudal Networks**: in this method, a managerial hierarchy is learned. At the highest level, the agent will learn how to assign tasks to its sub-agents. The framework employs a manager and worker module. The input to the manager module is a state and it outputs a goal. The worker module then uses this goal as input, together with its own state, and tries to form a policy to meet the goal set by the manager module. The power of Feudal networks is that each level of hierarchy can be provided with a different state resolution, providing enough information to be able to make a choice at that level. The reduced state-space from which the sub-agents learn significantly speeds up the learning process. This type of HRL is however not guaranteed to converge (Vezhnevets et al. 2017). Consider a maze with a U-shaped barrier. The maze can be split up at successively finer grains, as shown in Figure 6.2. Multiple managers can then be assigned to separate parts of the maze. At every timestep, an action is taken at all levels. However, the highest manager sets the task for the subsequent second level, which on its turn sets a goal for the third level. At the lowest level, the geographical moves are considered which lay within the confines of the larger state defined by the third level manager.

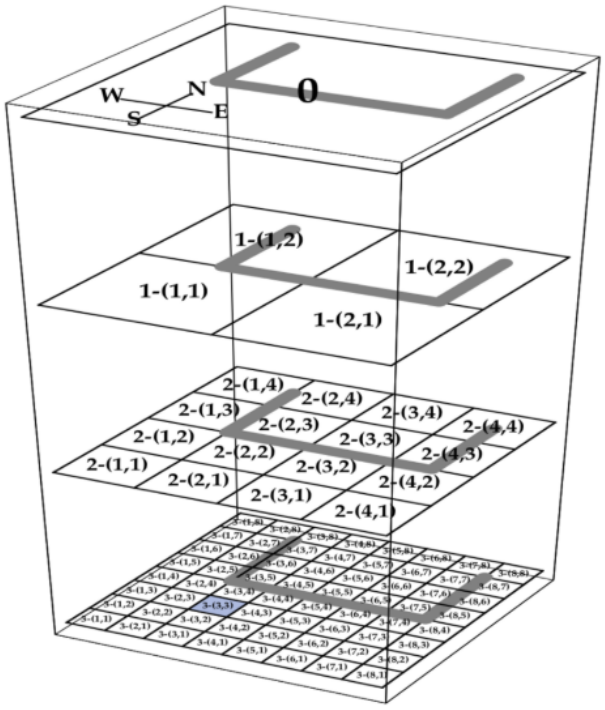


Figure 6.2: Feudal breakdown of a RL problem (Dayan and Geoffrey E Hinton 1993).

2. **Options** (R. S. Sutton, Precup, and S. Singh 1999): This is the most well-known hierarchical reinforcement learning framework. This framework is based on Markov options – closed loop policies for taking actions over a certain period of time – which are defined as a triple given by $o =< I_o, \pi_o, \beta_o >$. $I_o \subseteq S$ is the initiation set, $\pi_o : S \times A \longrightarrow [0,1]$ is the policy of the option and $\beta_o : S \longrightarrow [0,1]$ is the termination condition. This framework essentially captures the idea that actions are composed of other sub-actions. *Options* can be defined at different abstraction levels. Examples of an *option* are picking up an object, opening a door, but also twitching a muscle or applying a joint torque. The concept of an *option* is visualised against an MDP and SMDP in Figure 6.3. The agent trains a *policy over-options* which can initialise an option. In case an option is terminated, the agent can select a new option. The

main advantage of options is that it has proven to significantly speed up learning. Opposed to Feudal networks, options have theoretical convergence proofs towards an optimal policy in case the the action space consists of primitive actions and options. In (Arulkumaran, Dilokthanakul, et al. 2016), a method is developed which combines this option framework with deep Q-networks by augmenting DQN with "option heads". In this paper, the researchers show that using options indeed lowers the sample complexity.

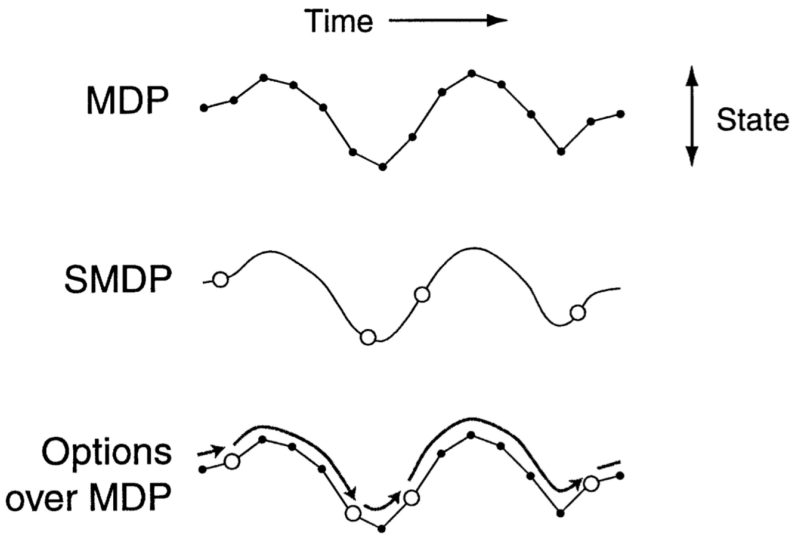


Figure 6.3: Options framework as opposed to the SMDP and MDP (R. S. Sutton, Precup, and S. Singh 1999).

3. **Hierarchical Abstract Machines** (HAM): a HAM consists of multiple non-deterministic finite state machines which can invoke lower-level machines (Parr and Russell 1998). Essentially, HAMs provide a hierarchical means of expressing constraints at varying levels of specificity and detail. Machine states can be represented by: *action states, call states, choice states* and *stop states*. Action states execute an action in the environment, call states deterministically execute another machine as subroutine, choice states non-deterministically select a next machine state and stop states halt execution of the machine and return control to the previous call state (Parr and Russell 1998). After an action or call state, the transition function determines the next machine state which is dependent on the current machine states and features of the environment state. Typically, machines are presented with a partial description of the environment. Learning in this framework only occurs in the choice states. This allows learning to occur only on a part of the entire state space, which speeds up the learning. In Figure 6.5, a non-deterministic finite-state controller for negotiating obstacles, shown in Figure 6.4, is given. In case the agent encounters an obstacle, a *choice state* is created to choose between the two next possible machines, 'Follow Wall' and 'Back Off'. Just as options, HAMs have an optimality guarantee. The disadvantage of the method is that HAMs are generally very complex to define and implement. An example of a reinforcement learning algorithm incorporating this ideology is called HAMQ-learning.

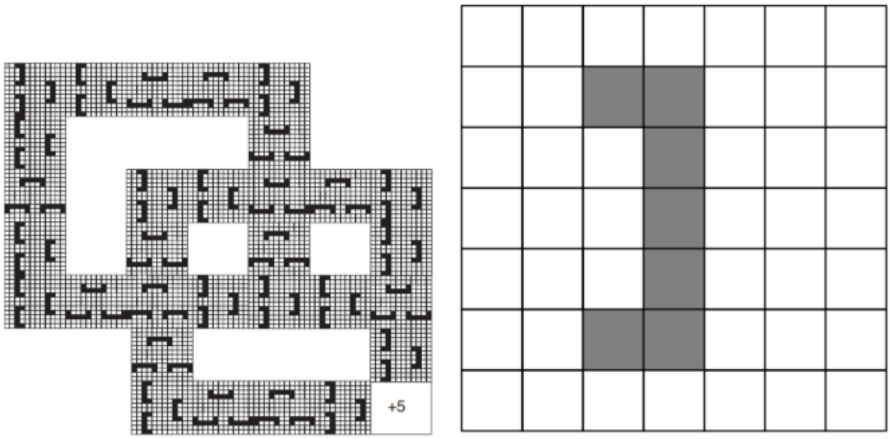


Figure 6.4: Environment with obstacles which an agent must avoid. Initial state is in the top left of the figure. Next to the gridworld a closeup of one of the obstacles is given (Parr and Russell 1998).

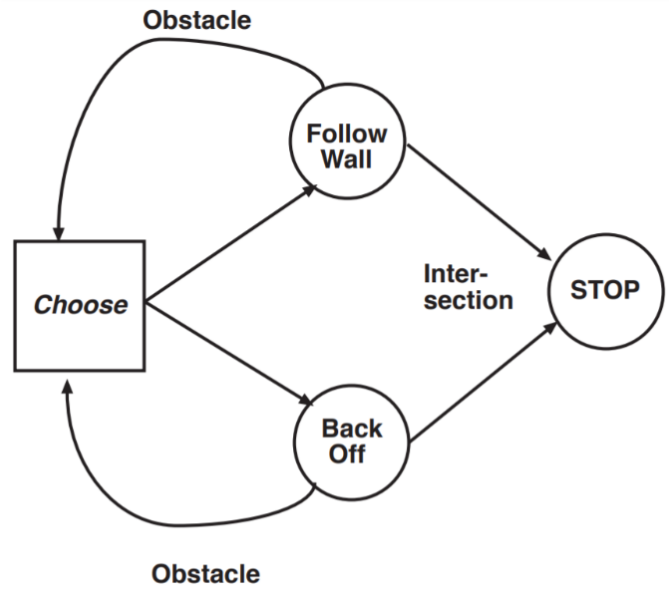


Figure 6.5: Nondeterministic finite-state controller for negotiating obstacles (Parr and Russell 1998).

In the next sections, deep reinforcement learning shall be elaborated upon. The hierarchical concepts introduced in this section do not directly translate to the domain of deep RL. There are however hierarchical deep reinforcement learning algorithms developed which incorporate the ideologies of Feudal networks, options and HAMs.

6.3. State of the Art Deep Reinforcement Learning Algorithms

The introduction of deep reinforcement learning teaches agents to make decisions in high-dimensional state and action spaces in an end-to-end framework in which features are learned in the layers of a deep neural network. It has significantly improved the generalisation property of RL (Shao et al. 2019). Below, one can find some remarks on deep RL.

- Exploration-Exploitation: this remains a struggle in RL, as well for DRL. Solutions proposed are to add parametric noise to the network (Arulkumaran, Deisenroth, et al. 2017) and to incorporate randomised value functions.

- Sample efficiency: hierarchical reinforcement learning and demonstration have posed significant improvements.
- Generalisation and transfer: just as for tabular methods, deep reinforcement learning methods suffer from *overfitting* to a certain learning environment. The degree to which a model is generalised therefore remains of interest (Zhang et al. 2018).
- The action space can be both discrete as continuous.

On-policy methods in deep RL have higher variance and will therefore require more samples to converge to an optimal solution. In the next sections the focus shall be on advanced deep learning methods. The environment to which these algorithms are applied, have discrete action spaces. Therefore, algorithms based on continuous action spaces such as Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC) are excluded from the analysis.

In this section a comparison shall be made between state-of-the art algorithms which are compatible with the SSD as input. The criteria used to evaluate a deep RL algorithm are:

- Generalisation properties.
- Data efficiency.
- Convergence properties.

6.3.1. State-of-the-Art Value-Based Methods

Deep Q-Network (DQN) (Mnih, Kavukcuoglu, et al. 2015) is an online value based approximate solution method in which a deep neural network is used to approximate the optimal action value function. The optimal value function is determined by solving the Bellman equations as shown in Equation 4.13. The steps taken in a DQN are visualised in Figure 6.6. For a given state and action, the deep neural network outputs an estimate of the action value Q(s,a;θ).

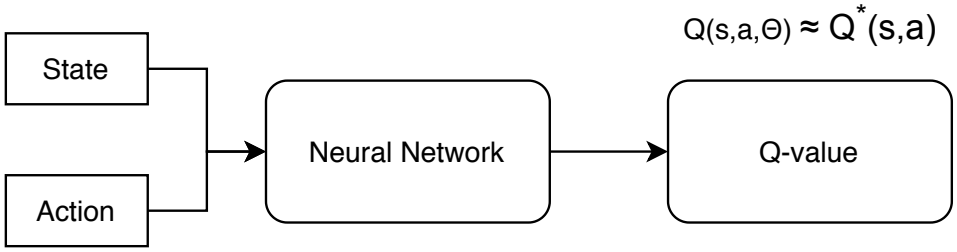


Figure 6.6: Visualisation of DQN.

The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2], \tag{6.1}$$

in which θ_i^- is the target network and θ_i are the parameters of the Q-network, also called the prediction network.

Essential to the functioning of this architecture are the introduction of Experience Replay and having a fixed target network:

- **Experience Replay**: experiences (actions, states, transitions and rewards) are stored from which mini-batches are created to perform updates on the network. Experience replay removes correlation in the observation data and smooths changes in the data distribution (Mnih, Kavukcuoglu, et al. 2015). More formally put, the agent's experiences are stored at each time step t in the data set $D_t = e_1, ..., e_t$, in which $e_t = (s_t, a_t, r_t, s_{t+1})$. The Q-learning updates are performed on randomly drawn samples, $(s, a, r, s') \sim U(D)$.
- **Fixed Target Network**: the target network, θ_i^- is updated every X steps and the parameters are held fixed between individual updates. This increases the stability of the algorithms.

In 2015, (Mnih, Kavukcuoglu, et al. 2015) showed that Deep-Q Networks could outperform all the previous algorithms, receiving only pixels and scores as input, on the Atari 2600 games[2]. This was the first algorithm to show that policies could be learned not only from handcrafted features, but from high-dimensional sensory inputs using end-to-end learning as well.

In addition to Deep Q-Networks, double learning can be added to create Double Deep Q-Networks (DDQN). (Hasselt, Guez, and Silver 2015) showed that the max operator in Q-learning which is used to select the action resulting in maximal future rewards causes the Q-values to be overestimated. Just as explained in the tabular case, a second independent value function is being trained. This value function is now yet another estimate using a deep network with weights θ'. During each iteration, one of the networks is used to estimate the maximisation action, and the other is used to estimate the value. (Brittain and Wei 2018) used a DDQN in a deep hierarchical agent algorithm to sequence and separate aircraft in a simplified ATC setting which showed promising results.

(Schaul, Quan, Antonoglou, Silver, and Deepmind 2016) improved the DQN and DDQN in terms of performance on the Atari 2600 games by adding **prioritised replay** instead of experience replay. The basic idea of prioritised replay is to draw important samples more often instead of randomly drawing samples. During optimisation, the algorithms tries to minimise the temporal-difference (TD) error. The basic idea of prioritised sweeping is to increase the replay probability for samples in the set of experiences which have a high expected learning progress, as measured by the magnitude of the TD error. The TD error for the DQN and DDQN architectures are given in Equation 6.2 and Equation 6.3.

$$\delta_i = r_i + \gamma \max_{a \in \mathbb{A}} Q_{\theta^-}(s_{t+1}, a) - Q_\theta(s_t, a_t) \tag{6.2}$$

$$\delta_i = r_i + \gamma \max_{a \in \mathbb{A}} Q_{\theta^-}(s_{t+1}, \operatorname{argmax}_{a \in \mathbb{A}} Q_\theta(s_{t+1}, a)) - Q_\theta(s_t, a_t) \tag{6.3}$$

For some environments, it is not necessary to know the action value at every time step. For these types of problems, (Z. Wang et al. 2016) introduced the Dueling Deep Q-Network architecture. In a Dueling DQN, the action value calculations are decomposed into two streams:

- A(s,a): the advantage of taking action a in state s, against all the other possible actions in that state.
- V(s): value function of being in that state.

This is shown in Equation 6.4 and visualised in Figure 6.7.

$$Q(s, a) = A(s, a) + V(s) \tag{6.4}$$

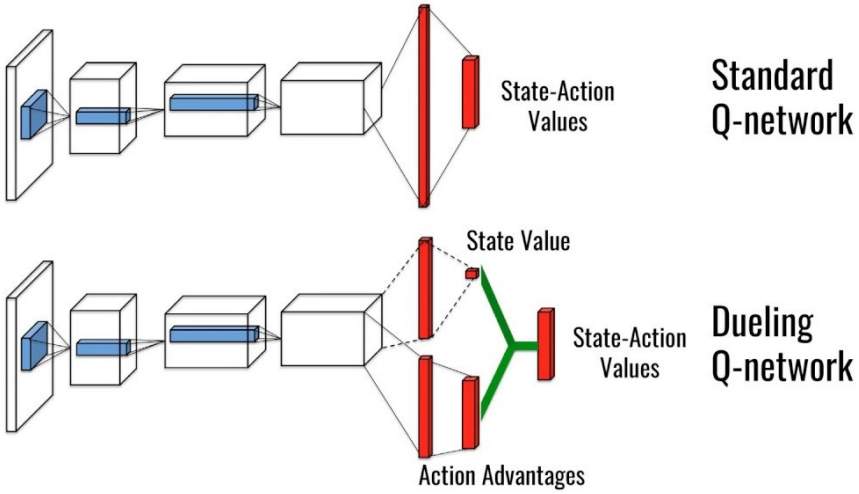


Figure 6.7: Network architecture of a Dueling Deep Q-Network. Adapted from (Z. Wang et al. 2016).

[2]Atari 2600 is a home video game console on which simple and low-resolution games as Pong and Pac-Man could be played. In deep RL, the performance of an algorithm is commonly evaluated on multiple Atari games. This enables a fair comparison between the performance of different RL algorithms.

In the paper, the dueling architecture obtains state-of-the-art results on the Atari 2600 domain, improving the performance of DQN and DDQN with prioritised replay. According to the researchers, the advantage of this architecture mostly lies in its ability to learn the state-value function more efficiently. Instead of updating the value of a single action in a certain state is updated, the value function is updated every iteration. It is shown that this advantage is more prominent if the amount of actions grow.

In (Hessel et al. 2017), the authors develop an RL architecture called RAINBOW which combines the successes of the extensions to DQN, which are double learning, prioritized replay, dueling networks and multi-step learning (A3C, described in subsection 6.3.2), distributional RL and the use of noisy nets to deal with the exploration/exploitation trade-off. The details of the network can be found in more detail in (Hessel et al. 2017). Over 57 Atari games, RAINBOW is able to match the best performance of any DQN after having trained on 44M frames whilst only having seen 7M frames. Continuing the training, RAINBOW is able to reach significantly better performance than any DQN network. The new architecture thus is more data efficient and achieves better performance in the limit.

Distributed learning

In deep learning algorithms, the most time is spent on generating experience and only a small fraction of that is spent on updating the neural network. The limiting factor in update speed is thus generation of experience and not the actual updating of the network. Decoupling of generating experience and network updating can allow for an increase in generating experience and hence a higher update speed. This ideology was used to develop a distributed variant of DQN, called Ape-X DQN (Horgan et al. 2018). In distributed learning, there are multiple workers which gather experience whilst the policy is being updated centrally from experience replay. Each worker functions on a different CPU core and the learning speed of this algorithm is thus dependant on how many CPU cores there are available. This is visually represented in Figure 6.8

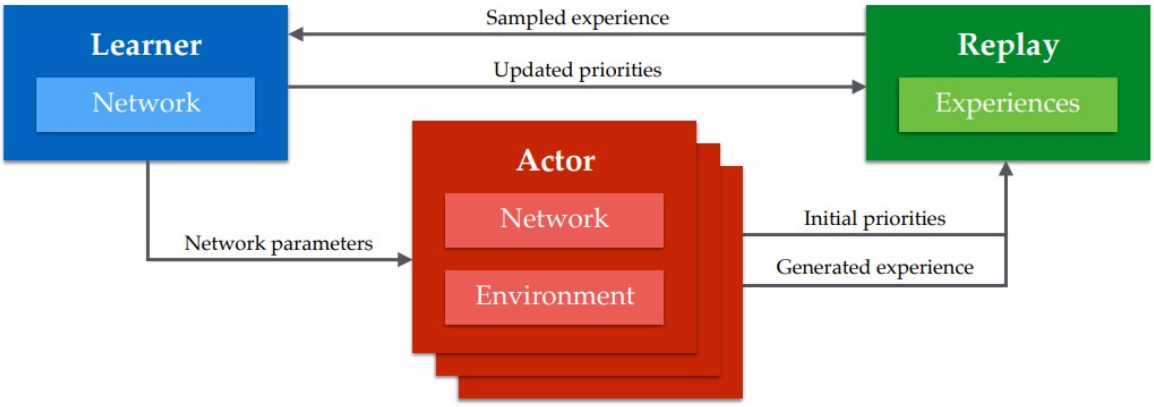


Figure 6.8: Decoupling of obtaining experience and learning used in Ape-X.

DQN with Hierarchy

A variant to DQN which utilises the computational advantages enabled by hierarchical reinforcement learning is called h-DQN. In h-DQN, the option framework discussed in section 6.2 is implemented. There is thus a meta-controller which sets a goal based on the given state and action. Given this the goal, the controller then acts to reach that goal. Separate DQNs are used inside the meta-controller and the controller.

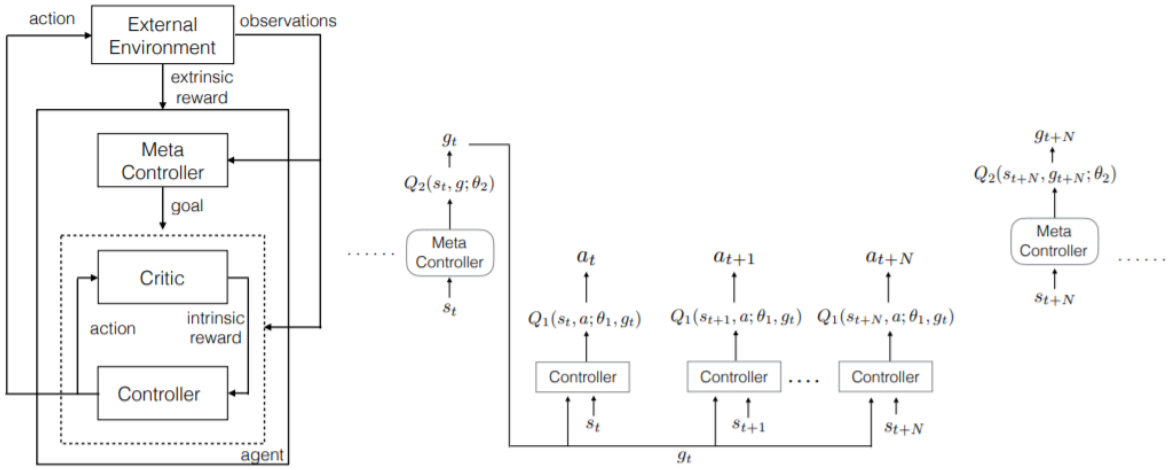


Figure 6.9: Overview of h-DQN (Kulkarni et al. 2016)

Especially for environments with sparse rewards and thus sparse feedback, on which DQN fails to learn a stable well-performing policy, this method has proven to be powerful (Kulkarni et al. 2016). Considering the ATC environment, rewards for certain actions will also be sparse. Think for example of rewards retrieved for acting conform to how an ATCO would resolve certain traffic scenarios.

6.3.2. State-of-the-Art Policy Gradient Methods

Recently, (Mnih, Puigdomènech Badia, et al. 2016) developed a multi-threaded asynchronous variant of the on-policy advantage actor-critic method and called it **A3C**. The aim of their research was to develop a method which was computationally more efficient than previous architectures. In this proposed method, multiple actors-learners run in parallel to explore different parts of the environment. To ensure different parts of the environment are explored, each actor-learner can have its own exploration policy. The most prominent benefits of this algorithm are that the reduction in training time is roughly linear with the amount of actors-learners. Furthermore, for most deep learning methods experience replay is needed to stabilise the learning, which can only be implemented for off-policy methods. However,the introduction of multiple parallel actors-learners stabilise the learning as well and therefore allow on-policy methods to be used as well. In this paper, the researchers show that in five Atari 2600 games, A3C outperforms DQN in terms of training speed but also obtains a higher score in the limit. The method was also compared with other state-of-the-art algorithms on 57 Atari games. These other algorithms were trained for 8-10 days on a Graphics Processing Unit (GPU), which is computationally much more powerful than a CPU, whilst A3C was trained in 4 days on 16 CPU cores. A3C showed to significantly improve the state-of-the-art average on these games in half the training time. Furthermore, after 1 day of training it already matched the scores obtained by the Dueling Deep Q-Networks.

One of the disadvantages of this asynchronous implementation of the multi-threaded actor-critic method is that the different agents communicate with the global network parameters separately. This causes the agents to sometimes work with an outdated version of the network parameters. A natural solution which researchers found was the construction of a synchronous and deterministic implementation, which was called A2C. In this implementation, the network averages over all the agents in its updates and does so if all the agents have finished exploring a sector. Research indicates that A2C makes use of the GPU more effectively and achieves similar or improved performance compared to A3C.

Finally, there are more advanced actor-critic methods achieving state-of-the-art performance. (Gruslys et al. 2017) propose a method which is more sample efficient based on a multi-step return off-policy actor-critic architecture called Reactor. A new policy evaluation algorithm called Distributional Retrace is implemented. Furthermore, a new $\beta-leave-one-out$ policy gradient algorithm is used. More details on the exact working principles can be found in (Gruslys et al. 2017). Reactor was compared to DQN, Double DQN, DQN with prioritised experience replay, Dueling DQN, prioritised Dueling DQN, ACER, A3C and Rainbow across 57 Atari games. It showed that Reactor exceeded the overall performance of the other algorithms under random human starts of the games.

Trust Region Methods
In (Schulman, Levine, et al. 2015), the researchers developed Trust Region Policy Optimisation (TRPO) which has guaranteed monotonic improvement to the optimisation of control policies. This algorithm has strong theoretical foundations, but it is sample inefficient and therefore impractical. Proximal policy optimisation (PPO) (Schulman, Wolski, et al. 2017) is an improvement to this algorithm and similar to TRPO does not use a line search method such as gradient descent but makes use of trust regions. In line search methods, the steepest direction is determined first in order to move forward in that direction by a step size. In trust regions, this process is reversed. First the maximum step size is determined, using KL-divergence in PPO, after which the optimal point to continue exploring within that maximum step size is chosen. KL-divergence is the difference between the distributions of the two policies after each iteration, π_θ and $\pi_{\theta_{\text{old}}}$. The advantages of PPO are that it is much easier to implement than its predecessor Trust Region Policy Optimisation (TRPO), it generalises better and has a better sample-complexity. PPO samples data by interacting with the environment and implements stochastic gradient ascent to optimise the objective:

$$L(\theta) = \hat{\mathbb{E}}_t[\min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t, clip(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon))\hat{A}_t]. \tag{6.5}$$

Not only to its direct predecessor, but it also achieves significantly better results on Atari games over the entire training period compared to A2C and ACER - another trust region policy optimisation method - which indicates it learns much faster. However, in the paper ACER did score higher over the last 100 episodes. Since ACER and PPO achieve similar performance and PPO is simpler to implement, it is chosen to exclude ACER from the analysis.

6.3.3. Imitation Learning
Although DRL techniques have achieved state-of-the-art performance on a number of tasks, data efficiency remains to be a challenge for RL. In theory, data efficiency is not a major concern in simulations since enough data can be generated. However, for practical use-cases, data-efficiency is essential. In (Hester et al. 2017), the researchers constructed a novel method called Deep Q-learning from demonstrations (DQfD). In the method, supervised learning is used to train an initial policy and value function based on the demonstrations. In the next phase, a DQN is used to improve this value function. Important to note is that a regularised loss function is used to minimise overfitting on the demonstration data set. The research shows that the DQfD indeed has better initial performance than Prioritised Dueling Double Deep Q-Networks on 41 of the 42 Atari games the methods were trained on. Furthermore, state-of-the-art results were obtained on 11 of the 42 Atari games with human demonstration data, proving the potential of this method being both data-efficient and high performing.

In (Pohlen et al. 2018), the DQfD was extended using a transformed Bellman operator and a temporal consistency (TC) loss to improve the stability of the algorithm. This new algorithm is called Ape-X DQfD, which also utilises distributed learning, and was able to consistently perform well on the 42 Atari 2600 games, exceeding average human performance on 40 of the 42 games. It must be noted that in order to train the paper, 128 workers were used to gather experience.

6.3.4. Inverse Reinforcement Learning
In the previous section, the concept that an agent can learn an initial policy from demonstrations, i.e. learn a mapping between states and actions, was introduced. There is another research area based on Learning from Demonstrations which is called inverse RL. This category of RL relies on the assumption that the policy of the expert creating demonstrations is optimal and consistent with regard to an unknown reward function. This reward function is learned using direct RL and subsequently learns an optimal policy with regard to the reward function. An advantage that IRL has over pure imitation learning is that it allows constant learning over time, online, through real interactions with the environment.

Generally, IRLs have proven exceptionally difficult to use for high-dimensional problems (Fu, Luo, and Levine 2017). Furthermore, they have shown to be less data efficient compared to imitation learning. Causes for this are that a large variety of optimal policies and many different reward functions can fitted to the demonstrations provided. IRLs are therefore an ill-defined problem. To deal with these cons, (Fu, Luo, and Levine 2017) developed an inverse reinforcement learning algorithm based on adversarial networks. This method seems to be especially successful in environments in which there is a significant variability in the environment compared to the demonstrations.

By learning the reward function through IRL, the artificial agent could possibly learn the strategy of an individual ATC controller. Implementing such an algorithm is therefore of interest for strategic conformal automation. On the flip side, these methods of RL will never improve the performance obtained by ATCOs since the agent ideally would mimic the strategies applied by the ATCO.

6.3.5. Hierarchy

In (Frans et al. 2017), a method is developed which incorporates a meta-learning approach to learning hierarchical policies by using shared primitives. These are policies that are executed for large numbers of timesteps. The method is called Meta Learning Shared Hierarchies (MLSH) and can be incorporated with an arbitrary reinforcement learning algorithm such as DQN, A2C and PPO. The main concept is that the policy is divided into several sub-policies which are each responsible for reaching a certain sub-goal. A master policy on its turn chooses the sub-policy. To fully exploit the gains of implementing this hierarchical architecture, the amount of sub-policies should be equal to all the sub-goals to a task. An overview of MLSH is given in Figure 6.10. Especially in an environment with sparse rewards, this type of hierarchical learning is useful. Normally, naive PPO cannot learn in such a setting as exploration over the action space does not result in a reward signal. Instead of exploring over the action space, the agent can explore over the different sub-policies, which probably results in a sequence in which a reward signal is retrieved. Overall, this method is able to significantly increase the sample efficiency in case the amount of sub-policies represent the amount of sub-goals.

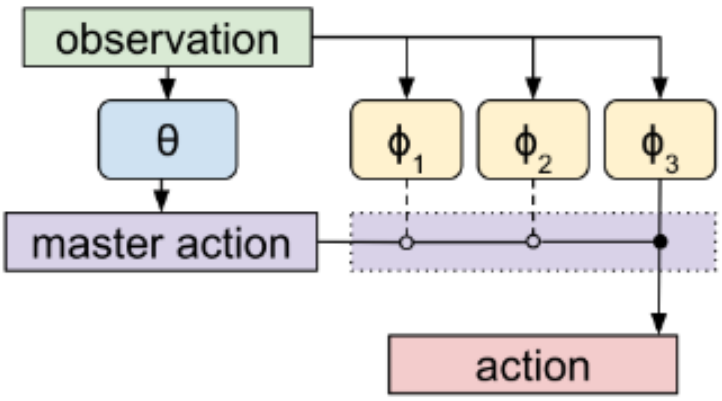


Figure 6.10: Overview of Meta Learning Shared Hierarchies (MLSH) (Frans et al. 2017).

The difference of this method compared to h-DQN is that it has a controller for every sub-goal instead of a single controller trying to achieve the goal set by the meta-controller. ATCOs in a sense try to resolve traffic scenarios similar.

6.3.6. Concluding Remarks

In this section, state-of-the-art deep RL methods are discussed. In the preliminary analysis, a trade-off between algorithms well-suited for this research is performed. Based on literature, it can be concluded that some of the methods elaborated upon in this chapter are not interesting to include in the trade-off.

First of all, multi-agent RL algorithms were discussed. Implementing a multi-agent RL method increases the computational complexity significantly. Furthermore, since coordination is needed in both a centralised and decentralised, non-stationarity in the learning process is a common challenge faced during implementation which can lead to divergence. More importantly, since every agent performs an action simultaneously, it is hypothesised that such automation of ATC will be difficult for the ATCO to supervise and fully understand. Therefore, the multi-agent RL algorithms will be excluded from the trade-off.

In the CD&R task of an ATCO a clear hierarchy is present. An ATCO must first select an aircraft, then decide what type of action is needed and finally decide on the magnitude of that action. Implementing hierarchical RL will significantly increase the sample efficiency. However, the aforementioned tasks in CD&R can be

decoupled and a separate agent can be trained for both tasks. In the task determining what action to take, a hierarchy is present though. Considering a simplified ATC environment in which an ATCO cannot alter the altitude, an ATCO can choose to do three things: only alter the heading, alter the speed or alter both the heading and speed. For hierarchical RL to function properly, concrete subgoals must be set with regards to these tasks. These might be reaching a highlighted pixel in the SSD which is not in the FBZ. The amount of subgoals will however be just as many as the action space of a RL agent which does not apply hierarchical learning. Therefore, the data efficiency advantages will not be achieved when applying hierarchical learning in this environment. For this reason, hierarchical RL methods will be excluded from the trade-off as well.

In terms of data-efficiency, value based methods perform well. Extensions to one of the first deep learning methods available, DQN, have continuously been made. Especially prioritised experience replay, double learning and a duelling architecture are seen as major contributors to improving the performance of DQN. RAINBOW combines the most valuable extensions to DQN which lead to a significant improvement on the performance of the agent in the limit. It must be noted that this algorithm also uses distributed learning. Due to the good performance in terms of data efficiency, a DQN method with the aforementioned extensions is still interesting to take into account in the trade-off.

For ATCOs, their understanding of the automation is of importance for accepting of it. One of the aspects contributing to their understanding is strategic conflormal automation (Van Rooijen 2019). DQfD learns from demonstrations and can contribute to reaching the goal of this research by enabling the agent to mimic certain strategies applied by the expert and thereafter improve on these. Whereas inverse RL methods are also able to mimic strategies, these methods will never be able to improve the performance achieved by an ATCO. Therefore inverse RL methods are excluded from further analyses as well.

From literature, it can be concluded that methods which use distributed learning have significantly improved the performance of RL algorithms. Distributed learning stabilises the learning and increases the efficiency in terms of computation time. The effectiveness of these algorithms depend significantly on the computing resources available. The computing resources available for this research are limited and therefore it is chosen that for now, the algorithms which implement distributed learning are not considered when trading-off algorithms. Most of the distributed learning algorithms are extensions to value and policy based methods, such as Ape-X DQN and distributed PPO. These extensions towards distributed learning can therefore still be made at a later stage in the research.

Summarising, deep reinforcement learning algorithms which are well-suited for this research are:

- Duelling DQN with prioritised replay.
- PPO.
- DQfD.

The distributed extensions to these algorithms are:

- Ape-X and Rainbow.
- Ape-X DQfD.
- Distributed PPO.

7

Making AI Explainable with the Solution Space Diagram

In subsection 3.1.2, the solution space diagram has been introduced. This diagram will be used as input for the RL agent and as supervisory control support tool for monitoring the automation. Although it can show whether the actions taken by the agent are safe or not, it does not visualise why an agent takes certain actions. When a human operator must supervise a form of automation that the operator does not fully understand, a higher workload is experienced. Therefore, explainability of the automation is one of the focus areas of this research. In this chapter, literature available on making AI solutions explainable is reviewed to answer research question 2c: 'What techniques are there available to contribute to the explainability of the automation?'. Section 7.1 introduces a framework for designing explainable AI solutions. Then, section 7.2 and section 7.3 elaborate on two methods which can contribute to the explainability of Deep RL.

7.1. Explainable AI

A new era has arrived in which machine learning implementations will drive dramatic transformations in a large variety of sectors. The introduction of these machine learning solutions is even referred to as the fourth industrial revolution (M. Xu, David, and Kim 2018). However, the effectiveness of state-of-the-art AI solutions is still limited due to their inability to explain decisions. To address this hurdle of machine learning, Defence Advanced Research Projects Agency (DARPA) has launched a program dedicated to developing Explainable Artificial Intelligence (XAI) (Gunning 2017). The basic premise of what XAI is trying to achieve is shown in Figure 7.1. In addition to developing a model which just outputs a solution to a problem, an explanation interface is added. This interface should be able to extract the reasons for a solution from the explainable model.

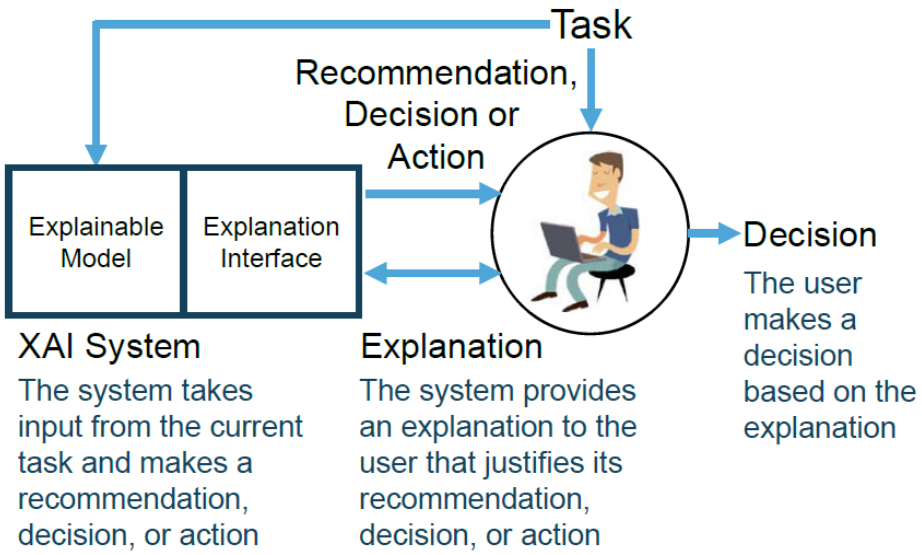


Figure 7.1: Framework for developing Explainable AI (Gunning 2017).

Ideally, the explanation is presented to the user, who still has to approve the proposed resolution based on the explanation backing up this proposition. According to (Gunning 2017), measures of the effectiveness of this framework are:

- User Satisfaction:
  - Clarity of the explanation (user rating).
  - Utility of the explanation (user rating).
- Mental Model:
  - Understanding of individual decisions.
  - Understanding of overall model.
  - Strength/weakness assessment.
  - 'What will it do' prediction.
  - 'How do I intervene' prediction.
- Task Performance:
  - Does the explanation improve the user's decision, task performance?
  - Artificial decision tasks introduced to diagnose the user's understanding.
- Trust Assessment: Appropriate future use and trust.
- Correctability (Extra Credit):
  - Identifying errors.
  - Correcting errors.
  - Continuous training.

This is merely one framework for developing explainable AI. In this research, the factors contributing to the explainability of the automation of CD&R using RL are investigated. Therefore, explainability must first be clearly defined in the context of this research. Improving the explainability will in this research be focused on improving the mental model and increasing the trust the human operator has in the automation. The human operator takes a supervisory role. When identifying factors contributing to the explainability of the automation, one must take the considerations listed underneath "Mental Model" into account and see whether trust in the automation can be increased. To evaluate to what degree a model is explainable, human-in-the-loop experiments should be performed to retrieve subjective performance metrics. Currently, there are namely no objective performance metrics to evaluate explainability. Evaluating to which degree the automation is explainable exceeds the scope of this research and is left for future research.

Currently, AI methods achieving high performance, such as neural networks, are not transparent. For example, a Convolution Neural Network trained to identify cats from photos will output a probability of an image containing a cat. It however does not show 'why' this image contains a cat. Reasons could be that the following features were identified: fur, cat ears and claws.

7.2. Visualising Features Exciting Neurons in Neural Networks

In (Tjoa and Guan 2019), a survey was performed on techniques to make convolutional neural networks more explainable. According to the researchers there are two main types of methods: *perceptive* and *mathematical* interpretability methods.

The first category of methods is focused on interpretabilities that can be perceived by humans. These interpretabilities are often considered to be obvious. An example method is the construction of *saliency maps* which show the importance of input components relative to the decision/resolution. These can be in the form of probabilities or pixels which form heatmaps. This, however, only indicates what parts of an image excite the neurons in the network the most.

Another method is Layered-wise Relevance Propagation (LRP) which was introduced in (Bach et al. 2015). In this method, a backpropagation occurs in which the output can be mapped to the input. Essentially, the pixels which contribute to the decision are excited. The strength of this method is that any output can be retraced to the input to see what part of the input would contribute to reaching a certain output. An example of this is shown in Figure 7.2. In this example, one can see LRP can be used to see what features in an image contribute to classifying an image to e.g. 'cat'. Extending this ideology to the problem at hand, the LRP constructed heatmap can be superimposed on the SSD to show which features contribute to taking a certain action. In RL, the agent tries to predict what action to take to maximise the reward it will receive in the future. With this method, it will thus be possible to indicate which parts of an image contributed in making a certain decision. In Figure 7.2 an example of this is shown for classifying numbers from an image. The image shows that the randomly drawn image contains the most features of being classified as '6'. However, it still does not really show 'why'.

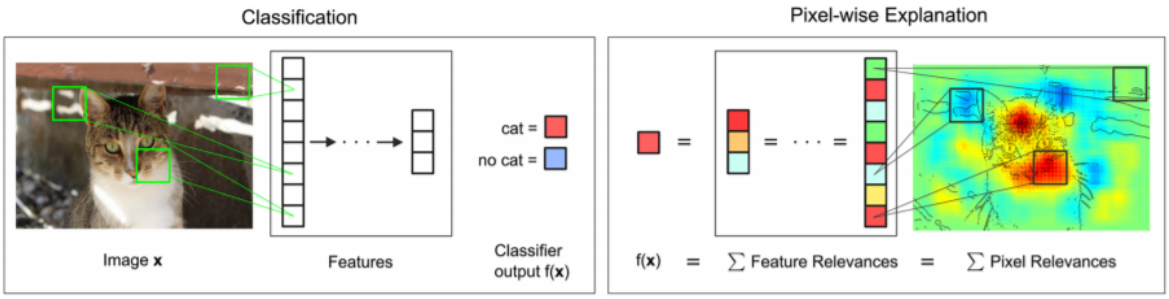


Figure 7.2: Visualisation of the pixel-wise decomposition process. (Bach et al. 2015)

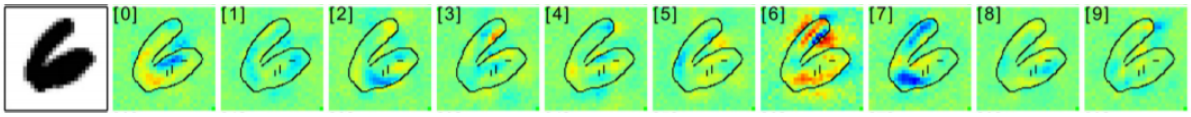


Figure 7.3: Pixel-wise decomposition for all classes for a randomly drawn image from the MNIST test set. (Bach et al. 2015)

7.3. Reinforcement Learning Specific Explanations

The aforementioned methods were not reinforcement learning specific, but can also be applied to supervised learning. Reinforcement learning is intuitive in the sense that the designer can steer the behaviour of the automation by carefully designing the reward function. The sole goal of the agent is namely to maximise the reward it receives over the entire episode.

For problems with a clear causal structure, (Madumal et al. 2020) performed research into the possibility of explaining sample-free reinforcement learning through a causal lens. In this setting, an action influence

model is constructed which depicts how states are influenced by other states in case a certain action is taken. From this action influence model, a description model - with prior understanding of the causal structure - can be learned for the questions 'Why' and 'Why not'. A test was performed to see whether the understanding of actions taken by an agent would improve after having seen example actions with the corresponding explanations for 'Why' and 'Why not' an agent took certain actions. Empirical results on an experiment with 120 participants showed that the humans got a significantly better understanding of the agent's strategy. This was tested by showing the participants a scenario of gameplay of Starcraft II after which the participant had to select one of three explanations, which included the option 'I don't know'. This shows that humans can indeed obtain a better understanding of artificial agents through making AI explainable. The CD& R task unfortunately does not seem to be suited to generate these explanations since there is not such a clear causal structure present.

For RL, (Juozapaitis et al. 2019) developed a RL algorithm which uses the reward differences to generate explanations for actions taken by the agent. In this method, the reward function is decomposed into different *reward types*. For the ATC problem one can think of safety, efficiency and airspace disruptiveness as different reward types. To gain insight into the reasoning for an agent to prefer an action over another, the decomposed Q-vectors for these actions can be compared to see what action would e.g. result in the highest expected future rewards corresponding to 'safety'. This is referred to as Reward Difference Explanations (RDX). To do so, decomposed Q-functions must be learned, for which a new algorithm was developed called drQ. Since the pixel data of the SSD shall be used as input to the algorithm, a deep RL algorithm shall be used. This concept can also be applied to DQN. Let the reward function be composed of a set of reward types C, then the reward function becomes vector-valued, $R \longrightarrow \vec{R} : S \times A$. This is formalised in Equation 7.1. Naturally, the value function, which is represented by a function approximator (e.g. neural network), can be decomposed as well, as shown in Equation 7.2.

$$\vec{R}(s,a) = \sum_{c \in C} R_c(s,a) \tag{7.1}$$

$$\vec{Q}^{\pi}(s,a|\theta) = \sum_{c} Q_c^{\pi}(s,a|\theta_c) \tag{7.2}$$

In the paper, the researchers present a theorem for drQ which implies that drQ's overall Q-function $Q^t(s,a)$ converges to the optimal Q-function $Q^{\star}(s,a) = \sum_{c \in C} Q_c^{\pi^{\star}}(s,a)$ and hence an optimal policy $\pi^{\star}$ (Juozapaitis et al. 2019). This theorem can directly be extended to drDQN. Accordingly, the update rules for the decomposed Q-value functions can be defined. DrDQN operates similar to DQN, except for the fact that each component Q-function is updated based on the current greedy action of the target network. The network obtains experiences and saves these in the replay memory. Following this, a mini-batch of experience samples is retrieved from the replay memory. This mini-batch is then used to update the parameters, θ_c according to Equation 7.3 - Equation 7.5. Periodically, the target network is updated with current learning parameters.

$$L(\theta_c) = \sum_{i=1}^{k} (y_{c,i} - Q_c(s_i, a_i; \theta_c))^2) \tag{7.3}$$

$$y_{c,i} = \begin{cases} r_c & : \text{for terminal } s_i' \\ r_c + \gamma Q_c(s_i', a_i^+; \theta_c') & \text{for non-terminal } s_i' \end{cases} \tag{7.4}$$

$$a_i^+ = argmax_{a'} \sum_{c \in C} Q_c(s_i', a', \theta_c') \tag{7.5}$$

In a gridworld in which an agent can move UP, DOWN, LEFT or RIGHT, the RDX is shown in Figure 7.4. One can see that by having trained individual Q-values for the different rewards the agent can receive, it can make a decision on what action would give it the largest reward.

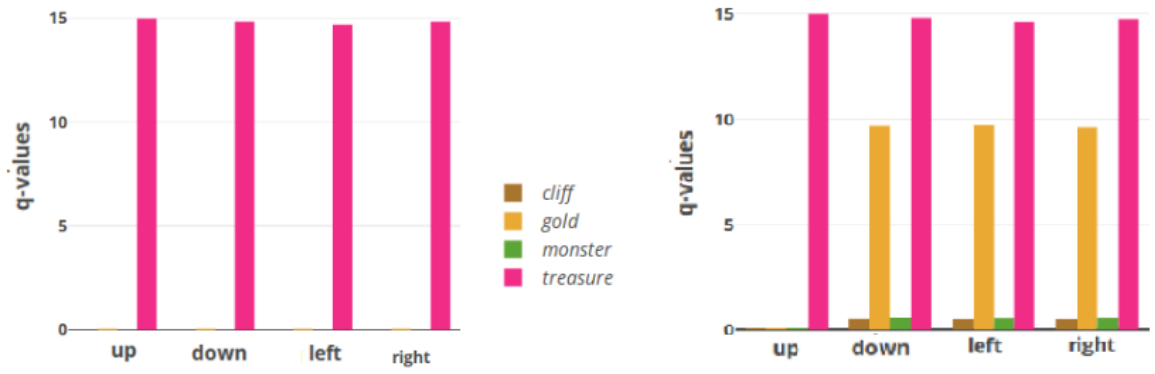


Figure 7.4: RDX of a gridworld in which reward can be obtained through *treasure*, *monster*, *gold* and *cliff* (Juozapaitis et al. 2019).

This method is focused on decomposing the reward function and the state-action value to real-time be able to see why an agent chooses a certain action. Another option would be to analyse the types of rewards an agent receives in hindsight. Although this would not enable real-time explainations, it is hypothesised that re-experiencing traffic scenarios, in which the human operator did not understand why the automation choose to perform a particular action, could improve the mental model of the automation. Especially since ATCos have to control a single sector in which similar traffic scenarios are encountered frequently, this might be a powerful strategy.

There are two types of explanations the authors focus on. First of all the MSX, secondly the RDX.

7.4. Concluding Remarks

Currently, the research performed into constructing explainable forms of automation is limited. Explainability in this research refers to the mental model a human operator has of the automation and the degree to which he or she trusts the automation. This research focuses on finding factors which contribute to the explainability and incorporate these in the design of the automation. Evaluating the degree to which the automation is explainable is left for future research.

There are two appropriate methods to contribute to the explainability of a deep reinforcement learning method. First of all one can visualise what features in an image drive the decision made by an agent. This can be achieved by constructing saliency maps, but also by LRP. Secondly, one can focus on reward decomposition to see what rewards an agent expects to retrieve when taking a particular action. It is hypothesised that the last method will be most insightful for ATCos since it is able to show the reasoning of the agent. Furthermore, decomposing the reward function can assist the designer of the automation to monitor learning and identify unwanted behaviour of the reinforcement learning agent.

8

Reward Shaping and Performance Evaluation

This chapter aims to answer research question 4a: 'What factors should be incorporated in the reward function?'. With this goal in mind, the design considerations for the two agents, one for choosing which aircraft should perform an action and one for choosing what action to perform, are discussed in this chapter. The final thesis focuses on automation for conflict resolution. However, as implementing automation for conflict resolution is dependent on the ability to develop automation to choose which aircraft should perform an action, design considerations for the latter are also reviewed in this chapter. First, in section 8.1, the possible types of automation for choosing which aircraft should perform an action are explored. This is done by surveying available literature and analysing which strategies employed by ATCOs should be taken into account. Then, section 8.2 elaborates on the design of the reward function for the agent choosing what action the aircraft should take. Finally, section 8.3 discusses how the automation can be evaluated in terms of performance.

8.1. Aircraft Choice

In this section, the possible types of automation for selecting which aircraft should perform a manoeuvre are set out.

8.1.1. Design Considerations

As presented in Table 2.1, ATCOs apply certain strategies. The ones relevant for choosing which aircraft should perform an action are:

- Identify conflicts pairwise.
- Prefer resolutions which require less co-ordination.
- Look for the one key action that resolves the problem.
- Focus visual attention on crossing points and sector borders.
- Adopt a look-ahead time between 5 and 10 minutes.
- Turn slower aircraft behind faster aircraft.
- Solve the head-on conflicts first.

Other factors that the agent should account for in this situation are:

- If possible, all aircraft should have their heading on the target heading.
- Optimally, aircraft should minimise deviation.

When choosing a type of automation for selecting which aircraft should perform a manoeuvre, the conformity with these strategies should be taken into account.

8.1.2. Possible Types of Automation

There are three realistic options for automating the selection process of which aircraft should perform an action: using supervised learning, design a rule-based algorithm and reinforcement learning. First of all, the

aircraft selection can be performed using a **rule-based algorithm**. In that case a model of the environment is required. Relevant variables for this environment are shown in Table 8.1.

Table 8.1: Variables needed for an agent to choose which aircraft should perform an action.

Variables	Reason	Size
t_{CPA} [sec]	identify conflicts	$\frac{\#a/c!}{(2\cdot(\#a/c-2)!)}$
d_{CPA} [nm]	identify conflicts	$\frac{\#a/c!}{(2\cdot(\#a/c-2)!)}$
V_{TAS} [km/h]	steer slower a/c behind faster	# a/c
$\delta_{\text{target heading}}$	identify which a/c is not on target heading	# a/c

By including the t_{CPA}, the agent will also be able to solve conflicts between aircraft on reciprocal tracks earlier whilst they have a large spatial distance since these will have a smaller t_{CPA} compared to conflicts between aircraft on the same track or crossing track.

An example of a rule-based algorithm would be to solve the potential conflicts with a $t_{CPA} < 5$ min first. If all these potential conflicts have been solved, the aircraft of which the $\delta_{\text{target heading}}$ is not equal to zero is selected. This is just an example, but to create a well-performing algorithm, these rules and the hierarchies between them should be carefully determined. One of the disadvantages of this type of automation is that such an algorithm will not be able to be ATCO conform neither will it learn to perform optimally. Furthermore, it is difficult to incorporate strategies as "prefer resolutions which require less co-ordination" in the rule based algorithm.

A second option is to use **supervised learning** to train an agent to select which aircraft should perform an action. In (Van Rooijen 2019), experiments were conducted in which participants had to select an aircraft which either did not fly on its target heading or was part of a potential conflict. In the experiment, the strategic conformal automation then selected what action the aircraft had to take. The data available (in terms of t_{CPA}, d_{CPA} etc.) on when the participant selected what aircraft can be used to train a supervised learning. The participants were however not experts in CD&R due to which the data might contain a lot of inconsistencies. (Westin, Borst, and Hilburn 2015) performed an experiment to research the degree of consistency for repeated conflicts over time. The researchers concluded that "controllers are consistent, but disagreed on how to solve conflicts". This means that controllers individually are consistent but all apply slightly different strategies. Therefore, it is hypothesised that a supervised agent can be trained from a large data set containing data from an ATCO on when to select which aircraft to perform a manoeuvre. One of the advantages of this method is that this type of automation would be conform with the conflict detection strategies applied by the ATCOs. This might however not be the most suitable method since a lot of data needs to be acquired.

A third option is to design a **reinforcement learning** agent for this task as well. The state for this reinforcement learning agent can be composed of either radar screen pixel data or a model of the environment in terms of relevant parameters for conflict detection. The advantage of using the radar screen as input is that the state space is constant. A well trained agent that can decide which aircraft should perform a manoeuvre can therefore more easily be transferred to a new environment with a different amount of aircraft. Applying reinforcement learning in this environment is however difficult since the reward the agent receives is also dependent on the action selection.

Both for the underlying data for supervised learning and the model for reinforcement learning, it can be seen from Table 8.1 that in case there are 9 aircraft in the sector, the agent should already be trained with a state of size $36+36+9+9=90$. To limit the size of the state space, (Brittain and Wei 2019) gave the agent access to the information of the N-closest aircraft. This is a solution, but the hyperparameter N needed extensive tuning for each traffic scenario. Another option is to include an attention layer, of which the input size can be variable.

8.2. Reward Function Design for Manoeuvre Choice

In this section, the reward function for choosing what action to take is defined. The sole objective of the learning agent is to maximise the reward it retrieves in the long run. The reward function influences both the time to converge and the convergence point (Matignon, Laurent, and Le Fort-Piat 2006). Therefore, the reward function should be designed carefully in order to reward 'good' actions and punish unwanted actions. Furthermore, one of the goals of this research is to investigate whether a more explainable automation can be

developed. To do so, the preferences of human controllers should be accounted for in the reward function. The raw pixel data of the SSD will be the input for the agent. The ultimate goal for the agent is to optimise the manoeuvre in terms of:

- Safety
- Efficiency
- ATCO conformance

In previous research into automating ATC with RL, different kinds of reward functions have been defined. The events that were rewarded with a positive or negative reward are summarised in Table 8.2 per research. These can be used as inspiration for the design of the reward function in this research.

Table 8.2: Reward function design in papers using RL to automate CD&R for ATC.

Research	Positive	Negative
(Brittain and Wei 2018)	higher cruise speeds, amount of aircraft reaching their goal.	conflict, being out of time (threshold).
(Brittain and Wei 2019; Brittain, Yang, and Wei 2020)	[-]	conflict (-1), $minimal\ separation < d_{o,i} < 10\ nm$ (function of distance, $-1 \leq r \leq 0$)
(Pham et al. 2019)	[-]	conflict, larger deviation
(Regtuit et al. 2018)	[-]	Los of separation, deviating from reference signal (demonstration) and every pilot request (to minimise the amount)

In terms of safety, the agent should be penalised for the amount of conflicts. Furthermore, to increase safety margin, a negative reward can be awarded to aircraft which near a conflict situation. These are aircraft with a $2 \leq t_{CPA} \leq 5[min]$. In terms of efficiency, the deviation of the resolution can be taken into account. This can both be done in a continuous way by considering the heading deviation from the target or at the end of an episode by considering the flown path relative to the direct path. Finally, some might argue that an airspace in which most aircraft have a high cruise speed increases the throughput of the airspace and therefore increases its efficiency. However, since air traffic controllers value a less disruptive manoeuvre which upholds the safety requirements more than the throughput of a sector, this will not be included.

In Table 2.1, several strategies of ATCOs regarding CD&R are identified. The most important strategies for conflict resolution are summarised below:

- Solve conflicts pairwise and later check for consequences on other aircraft.
- Look for the one key action that resolves the potential conflict.
- Prefer resolutions which require less co-ordination.
- Select resolution that requires least amount of sector disruption.
- First try to solve conflict by vectoring aircraft, only if necessary perform a speed resolution.
- Turn slower a/c behind (minimises extra distance flown).
- Manoeuvres that more clearly resolve the conflict are preferred.

8.3. Performance Evaluation

To evaluate the performance of the automation, one needs metrics for evaluating the automation in a two-aircraft traffic scenario and for the multi-aircraft traffic scenario. The most important metrics for CD&R are listed below:

- # LOS.
- # conflicts solved last-minute (2<t_{CPA}<5 min).
- # actions that need to be taken.
- % of aircraft reaching its exit waypoint.

Next to these metrics, one can define a scoring function on the efficiency. Furthermore, test scenarios will be set up to see whether the RL agent shows expected behaviour, conform with the design of the reward function. One of these tests could be to see whether the RL agent steers the slower aircraft behind a faster one. These metrics shall be evaluated for both the two-aircraft and multi-aircraft traffic scenario.

Next to evaluating the performance with absolute metrics, it is relevant to compare the automation with a baseline. In (Van Rooijen 2019), the Multi-Voltage Potential (MVP) method, which was developed in (J.M Hoekstra, van Gent, and Ruigrok 2002), was used as baseline. The MVP is a decentralised approach to automating CD&R. To make a fair comparison between the MVP method and the RL agent in the two-aircraft traffic scenario, the MVP method should be altered slightly. Instead of being able to provide all aircraft with resolutions, the MVP is limited to providing resolutions to the controlled aircraft only, just as the RL agent is. When setting a baseline with the MVP method in a multi-aircraft traffic scenario, the MVP method is enabled to provide resolutions to all the aircraft.

8.4. Concluding Remarks

In this chapter, the design options for choosing what aircraft should perform an action are discussed. In the final thesis a decision shall be made on whether to use a rule-based algorithm to automate this process or to train an agent using supervised or reinforcement learning to do so. Furthermore, considerations when designing the reward function for choosing what action an agent should take are elaborated upon. In the final thesis, it will be decided which considerations will be included in the reward function and to what degree they are valued.

9

Preliminary Analysis

In this chapter, a trade-off is made between two different deep RL architectures. These are PPO and DQfD, of which some of the basic principles were elaborated on in chapter 6. First, in section 9.1, the experiment for the trade-off is setup. Then, in section 9.2, the algorithms of PPO and DQfD are explained in detail. Furthermore, to be able to make a fair trade-off, the hyperparameters of both algorithms are carefully selected or tuned. The trade-off is performed in section 9.3, in which one of the two algorithms is selected to be used in the final thesis. Finally, section 9.4 explores the effect of using a different network architecture on the performance of the chosen algorithm.

9.1. Experiment Setup

In (Van Rooijen 2019), the visual value of the SSD for deep neural networks was investigated. The author showed that the SSD could be used to train an agent with supervised learning. This research will, however, be the first to explore the value of the SSD for a reinforcement learning agent. The details on the environment are given in subsection 9.1.3. To be able to make a well-considered trade-off, the algorithms are both quantitatively or qualitatively evaluated along five trade-off criteria.

9.1.1. Selection of Trade-Off Criteria

For this research there are several criteria that the algorithms should meet. During the trade-off, the algorithms will be evaluated for each of the following criteria:

1. **Generalisation**: since the agent should be able to make a decision in situations it has not encountered, the degree to which it is not over-fitting to the training samples is of importance. To evaluate the generalisation property the performance of the agent during unseen traffic scenarios is evaluated.
2. **Sample-efficiency**: it is beneficial for the agent to learn efficiently since the range of traffic scenarios it can encounter and actions it can take are relatively high compared to most Atari games RL is being used for. To evaluate this, the amount of frames it takes the learning algorithms to meet a certain level of training performance are compared.
3. **Convergence-properties**: deep reinforcement learning can become unstable due to sparse rewards in the environment. Considering the options for designing the reward function, as presented in chapter 8, it is expected that the agent shall also deal with sparse rewards, e.g. rewards assigned for the disruptiveness of an action. Convergence is also influenced by hyperparameter selection. The convergence property of an algorithm is compared both theoretically as through analysing the training curve.
4. **Explainability of the algorithm**: since this research investigates the development of an algorithm which contributes to the explainability of the automationt for ATC, this is an important criteria in the selection of the RL algorithm. However, this is difficult to measure quantitatively and it will therefore be evaluated qualitatively according to the principles of Explainable AI.
5. **Controller conformity**: (Van Rooijen 2019) showed that controllers have personal strategies regarding the CD&R task. To increase acceptance of automation in ATC, tailoring automation to an individual's strategy is proposed as possible solution in (Westin, Borst, and Hilburn 2015). To evaluate this criteria a qualitative analysis w.r.t. controller conformance shall be performed on all the algorithms.

9.1.2. Methodology

To be able to perform a trade-off between different RL algorithms, hyperparameters of the algorithms should be tuned and multiple tests need to be done. Performing these tests using advanced ATC simulation environments such as BlueSky is unnecessary since the trade-off is based on criteria which can be evaluated using a simplification of the ATC environment as well. In the preliminary analysis, an agent is trained with PPO and DQfD in a simplified learning environment. A manual hyperparameter tuning will be performed after which each agent is trained until it reaches stable performance. Then, the two algorithms are compared according to the criteria listed in subsection 9.1.1. Based on this comparison, one of the two algorithms is selected to be used in the final thesis. This methodology is visualised in Figure 9.1.

For DQfD, expert demonstrations are needed. An agent which was trained with a publicly available PPO implementation was used to generate these demonstrations. Since the 'expert' does not perform perfectly, it was chosen to filter the demonstrations for well-performing examples.

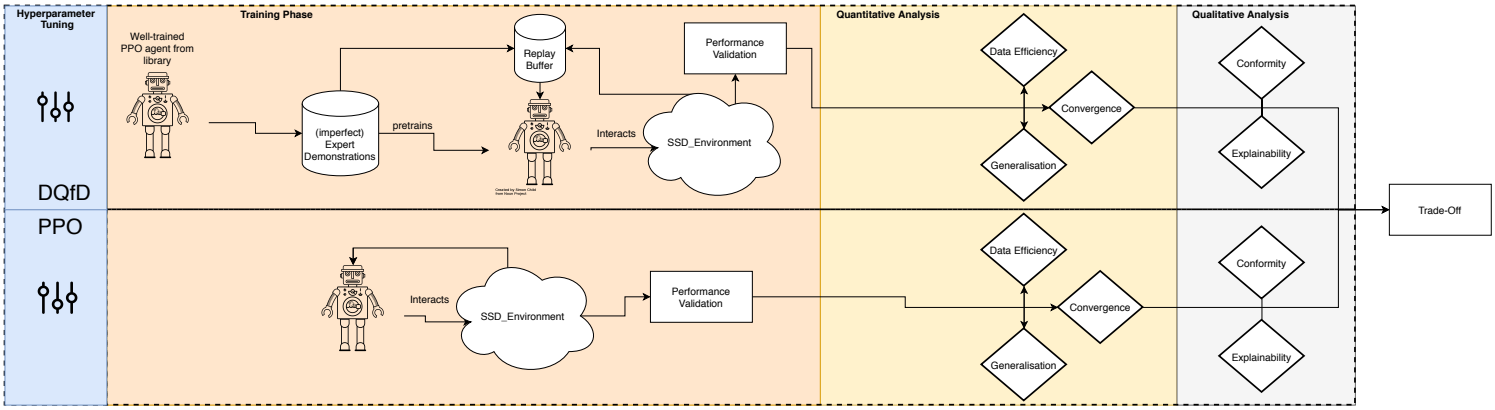


Figure 9.1: Methodology used for preliminary analysis.

9.1.3. Environment

For the preliminary analysis, an environment with limited action and observation space has been developed using *the OpenAI Gym library*. The SSD in this environment contains one randomly initiated Forbidden Beam Zone. This environment will be referred to as the *SSD environment*.

As explained in subsection 3.1.2, the velocity vector is composed of the velocity and the heading of the aircraft. The velocity is indicated by the length of the arrow and the heading is represented by the angle the arrow makes with the centre line. The velocity vector has a green colour in the SSD environment. The FBZ is indicated by a red polygon spanning between the inner circle and outer circle, which represent the flight envelope in terms of speed. The target heading, for ATCOs this will be the bearing to the exit waypoint of that aircraft, is represented by a blue line.

State and Action Space

The state-space of the environment is composed of raw pixel data of the SSD. Since RGB (Red, Green, Blue) is applied, a pixel can have a value between 0 and 255 at each of the three layers. The image is cropped to remove the extra white space leaving an image height of 202 pixels and width of 201 pixels, each consisting of three channels. The agent can only alter its heading. Its action space is shown in Equation 9.1.

$$A_{agent} = \{-90°, -67.5°, -45°, -22.5°, 0°, 22.5°, 45°, 67.5°, 90°\} \tag{9.1}$$

The lines drawn in the SSD to indicate the current and target heading are relatively large to be clearly distinguishable for the agent. This is possible because the headings in the action space lie far apart. In an action space spanning significantly more possible headings, the lines must be made thinner. Thicker lines are preferred since the image can then be downscaled in order to decrease the state space whilst the lines still remain distinguishable.

Reward Function

The reward function is composed of five components, which are listed below.

- $R_t = +10$ in case the heading (green) is exactly the same as the target heading and the arrow head is not in the FBZ.
- $R_t = 1$ in case the new heading is closer to the target heading and not in the FBZ.
- $R_t = -0.5$ in case the new heading did not change compared to the previous heading.
- $R_t = -2$ in case the new heading is further away from the target heading and not in the FBZ.
- $R_t = -30$ in case the new heading is in the FBZ.

The agent is thus punished for having a velocity vector which is in the FBZ and rewarded if the heading is close or exactly on the target heading. In theory, the maximum reward the agent can receive is 13. It can do so if the target heading is at 0° whilst the initial heading is either −90° or 90° and there is no FBZ separating the target and initial heading. It can then take three steps into the direction of the target heading, each time receiving a reward of 1, and finally take the action leading to the target heading, receiving a reward of 10.

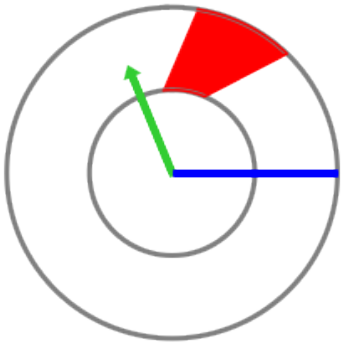


Figure 9.2: Example of SSD environment on which algorithms are trained. This gives a reward of -2 since the previous heading was closer to the target heading.

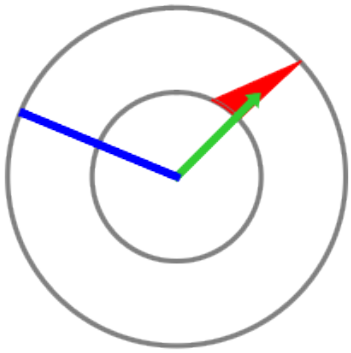


Figure 9.3: Example of arrow head being in the FBZ. This gives the agent a reward of -30.

9.1.4. Image Preprocessing

The images provided to the RL agents should be pre-processed using the same techniques. For both the PPO and DQfD agent, the image is pre-processed using the following steps:

1. Image is cropped to remove the white spaces.
2. Image is downsized from 3x202x201 ⟶ 3x128x128.
3. Image is converted to greyscale: 3x128x128 ⟶ 1x128x128
4. Image is normalised using a mean value and standard deviation retrieved from 10.000 images.

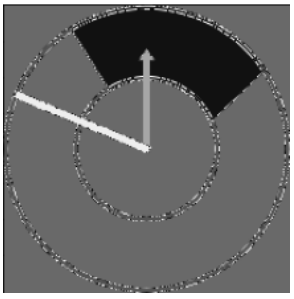


Figure 9.4: 202x201 pixels.

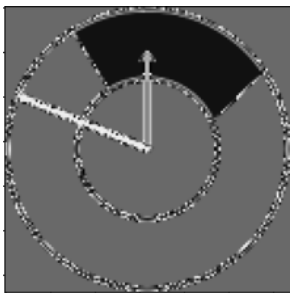


Figure 9.5: 128x128 pixels.

Figure 9.6: 64x64 pixels.

Figure 9.7: 32x32 pixels.

It was decided to downscale to 128x128 instead of all the way to 64x64 since it was hypothesised that the agent could not accurately determine what heading would be inside or just outside the FBZ. Furthermore, the image is converted to greyscale in order to decrease the dimensions by a factor 3.

For any convolution neural net, normalisation is useful to ensure that each input pixel has a similar data distribution. This causes the training process to speed up significantly (Buduma 2017). To do so, the mean

value of the pixel input and standard deviation must be calculated. Prior to training the algorithms, 10,000 runs of the environment are done to determine these parameters.

In the final thesis, BlueSky will be used as simulation environment. Since the construction of the SSD slightly differs in that environment, a comparison between downscaling the image to 1x128x128 and 1x64x64 will be made only in the final thesis. Furthermore, to decrease the state-space, an analysis will be performed to see whether using only the upper half of the SSD (in which the current heading always points upwards) can speed up the learning process.

9.2. Algorithms

In this section, the details of the algorithms which are included in the comparisons are explained in detail. First, in subsection 9.2.1, the working principles behind DQfD are elaborated on.

9.2.1. DQfD

In (Hester et al. 2017), the DQfD algorithm is developed. The core idea of this algorithm is to speed up learning by leveraging small sets of demonstrations. The original algorithm presented in (Hester et al. 2017) is reproduced and used to train an agent in the SSD environment in order to make a comparison with PPO. The core idea of the algorithm is to first train the agent on demonstrations, which are stored in a demonstrations replay buffer. The agent imitates the demonstration data by updating the network with a supervised large margin classification loss and a temporal difference loss. The temporal difference loss is needed to learn a self-consistent value function to continue learning after the pre-training phase. Having pre-trained on the demonstrations, the agent obtains new experiences and continues training on both demonstrations as well as the new experiences. To do so, the researchers have implemented prioritised replay. The algorithm is shown in pseudo code in Algorithm 2.

Algorithm 2 The pseudo-code of Deep Q-Learning from Demonstrations (DQfD) (Hester et al. 2017). The behaviour policy $\pi^{\epsilon Q_\theta}$ is ϵ-greedy with respect to Q_θ.

Require: $\mathbb{D}^{replay}$: initialised with demonstration data set; θ: weights for initial behaviour network (random); θ': weights for target network (random); τ: frequency at which to update target net; k: number of pre-training gradient updates;

for steps t $\in \{1,2,...k\}$ **do**
  Sample a mini-batch of n transitions from D_{replay} with prioritisation
  Calculate loss J(Q) using target network
  Perform a gradient descent step to update θ
  if t **mod** $\tau = 0$ **then**
    $\theta' \longleftarrow \theta$
  end if
end for
for steps t $\in \{1,2,...\}$ **do**
  Sample action from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
  Play action a and observe (s', r)
  Store (s, a, r, s') into $\mathbb{D}$, overwriting oldest self-generated transition if over capacity occurs
  Sample a mini-batch of n transitions from D^{replay} with prioritisation
  Calculate loss J(Q) using target network
  Perform a gradient descent step to update θ (Adam optimiser)
  if t **mod** $\tau = 0$ **then**
    $\theta' \longleftarrow \theta$
  end if
  $s \longleftarrow s'$
end for

Network Architecture

There are multiple different ways to parameterise the value function using a neural network. In (Van Rooijen 2019), it was concluded that a simple architecture is preferred for supervised learning since it decreases the number of trainable network parameters and because it mitigates overfitting. Furthermore, performance in

the limit did not seem to be affected. In RL, the most time-consuming part is acquiring new data, much more than updating the weight parameters. For this trade-off, a comparison between different architectures will therefore not be done. It is decided that both DQfD and PPO will be trained using the same network architecture. Only after deciding which algorithm to use for the final thesis, a comparison between commonly used networks will be done.

The network architecture used in (Hester et al. 2017) is a Dueling DQN architecture (Z. Wang et al. 2016). In the dueling architecture, a state value and action advantage is computed to approximate the state-action value. The dueling network thus has two heads which share same the feature learning module, one for estimating the state value and one for estimating the action advantage.

The feature learning module is composed of the same convolutional neural network as used in (Mnih, Kavukcuoglu, et al. 2015; Hasselt, Guez, and Silver 2015). It has three convolutional layers and is described per layer in Table 9.1.

Table 9.1: Setup of the feature extractor used for DQfD. The first and second convolutional layer have a stride of 4 and 2.

Layer #	Type of Layer	Input Size	# Feature Maps	Kernel Size
1	CONV(s4)	1	32	8x8
3	ReLU	32	32	-
4	CONV(s2)	32	64	4x4
6	ReLU	64	64	-
7	CONV(s1)	64	64	3x3
9	ReLU	64	64	-

The feature learning module is extended with a stream for the value and advantage function, which is visualised in Figure 9.8. The details of these layers can be found in Table 9.2. As one can see, both the value stream and advantage stream are composed of a fully-connected layer with 512 units. Remember that the input of the network is an image composed of 128x128 pixels. Due to the different convolutions applied to the image, the dimensions are altered. With Equation 9.2, one can calculate the dimension of the output of a convolutional layer. Doing this for all the layers in the feature learning module, one can calculate that it has an output size of 5408. This is the input size of the two streams.

$$\text{size of the output} = \left(\left\lfloor \frac{size - (\text{kernel size} - 1) - 1}{\text{stride size}} \right\rfloor + 1\right) \cdot \#\text{ feature maps} \tag{9.2}$$

Table 9.2: Head of value and advantage approximator in the Dueling DQN.

	Type	Input Size	Output Size
value stream layer #1	Linear	5408	512
value stream head	Linear	512	1
advantage stream layer #1	Linear	5408	512
advantage head	Linear	512	# actions

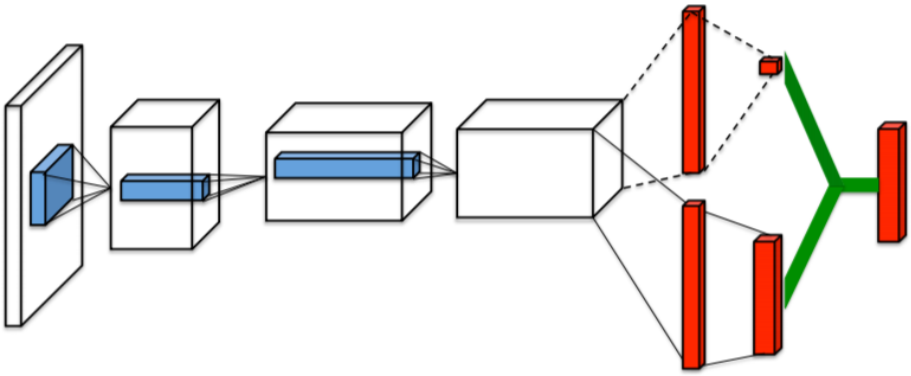


Figure 9.8: Dueling network architecture used in DQfD (Z. Wang et al. 2016). The top stream represents the state-value head, which has a single output. The bottom stream represents the action advantage head, which has as many outputs as possible action.

To calculate the state-action value, the state-value and action advantage are added. This is shown in Equation 9.3. In this equation, θ represents the weights of the feature learning module whilst α and β represent the weights of the action advantage and value stream of fully-connected layers. The advantage is corrected by an average value over all the advantages over the next possible actions in order to address the issue of identifiability. Without including such a term, the values of V and A cannot be recovered from Q.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|\mathscr{A}|}\sum_{a'} A(s,a';\theta,\alpha)) \tag{9.3}$$

Prioritised Replay

Experience replay has posed the opportunity to re-experience rare experiences and to break temporal correlations by using both experiences obtained in the past with recent experiences acquired during updates. This has stabilised the training process of the value function (Schaul, Quan, Antonoglou, and Silver 2016). In (Schaul, Quan, Antonoglou, and Silver 2016), a method to prioritise experiences is constructed. This is done to replay important transitions more frequently to speed up learning. Implementing prioritised replay with DQN outperformed an implementation with uniform replay in 41 out of 49 Atari games. In DQfD, prioritised replay is used to balance the amount of demonstration data contained in a mini-batch during training with new experiences as well. To define the priority a sample has, the authors suggest two variants. First of all proportional prioritisation, which is shown in Equation 9.4. In this equation, δ_i is the temporal difference (TD) error. This error indicates how far the value is from the next-step estimate: it indicates how surprising or expected the transition is (Andre, Friedman, and Parr 1998).

$$p_i = |\delta_i| + \epsilon \tag{9.4}$$

The second variant is called rank-based priority, which is shown in Equation 9.5. In this equation $rank(i)$ is the rank of transition i when the replay memory is sorted according to $|\delta_i|$.

$$p_i = \frac{1}{rank(i)} \tag{9.5}$$

Rank-based priority is more robust as it is insensitive to outliers (Schaul, Quan, Antonoglou, and Silver 2016). In the paper (Hester et al. 2017) in which DQfD is first described, the authors however implement the proportional priority, due to which this variant of determining the priority is used for this implementation of DQfD as well. In the paper, an additional demonstration priority bonus, ϵ_d, is added to p_i of the demonstration data to boost the frequency at which they are selected. In that case, $p_i = |\delta_i| + \epsilon_a + \epsilon_d$. The probability of sampling a transition is given in Equation 9.6. In this equation, α determines how much prioritisation is used. Setting α to zero corresponds with the uniform case.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{9.6}$$

To have an accurate estimation of the expected value with stochastic updates, these updates must have the same distribution as its expectation. However, the probabilities are non-uniform due to the prioritisation.

This induces a bias in the solution of the expected value estimate. To correct for this, importance-sampling (IS) weights can be used, which are shown in Equation 9.7. In this equation, N is the size of the replay buffer and β controls the amount of IS with $\beta = 0$ corresponding to no IS and $\beta = 1$ to full IS (Schaul, Quan, Antonoglou, and Silver 2016). In most applications, the bias is annealed by defining a schedule on β in which β is linearly increased over time to the value of 1. In the Q-learning update, $w_i\delta_i$ must be used instead of δ_i.

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^{\beta} \tag{9.7}$$

Implementing these weights in the Q-learning update is called *Weighted Importance Sampling*.

Loss Function

The loss function is setup of four components. It contains a double Q-learning loss for every update, both during pre-training as well as during the normal training phase. Additionally, for updates using demonstration data, a n-step double Q-learning loss, a supervised large margin classification loss and an L2 regularisation for generalisation purposes. These components are explained below:

- The double Q-learning loss is implemented to overcome the maximisation bias present in regular DQN updates. A policy network is used to calculate the $argmax$ over the next state values whilst a target network is used to calculate the value over the next state-action pair.

$$a_{t+1}^{\max} = \underset{a}{\text{argmax}} Q(s_{t+1}, a; \theta) \tag{9.8}$$

  The double Q-learning loss is then the difference between the state-action value function as given by the target and current network, as shown in Equation 9.9. In this equation, θ' indicate the weights of the target network.

$$J_{DQ}(Q) = (R(s,a) + \gamma Q(s_{t+1}, a_{t+1}^{max}; \theta') - Q(s,a;\theta))^2 \tag{9.9}$$

- The n-step double Q-learning loss is included during updates on demonstration data to propagate the values of the demonstrator's trajectory through all the previous states (Hester et al. 2017).

$$J_{n-step}(Q) = (r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \underset{a}{\text{argmax}} \gamma^n Q(s_{t+n}, a; \theta') - Q(s,a;\theta))^2 \tag{9.10}$$

- To ensure that the agent imitates the demonstration data, a large margin classification loss is included. It does this by making sure that the values of the other actions are at least a margin lower than the one taken by the demonstrator. This is shown in Equation 9.11, in which $l(a_E, a) = 0$ in case $a = a_E$. For all the other actions, $l(a_E, a)$ takes a positive value. Essentially, by minimising this loss function, a margin between the value of the expert demonstration and the second best action is added (Piot, Geist, and Pietquin 2014).

$$J_E(Q) = \underset{a \in A}{max}[Q(s,a;\theta) + l(a_E, a)] - Q(s, a_E; \theta) \tag{9.11}$$

- The last term is the L2 regularisation term, which is included to prevent over-fitting. In the algorithm it is included by adding weight decay to the Adam optimiser.

The total loss function then becomes:

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q) \tag{9.12}$$

In this loss function, λ are the weighting coefficients of the loss components. λ_3 is zero in this implementation since the L2 regularisation loss is added by using weight decay in the Adam optimiser. To fully understand how this algorithm functions, the computational graph of DQfD is constructed in Figure 9.9. One can see that the target net does not require a gradient since its network parameters are not updated at each epoch.

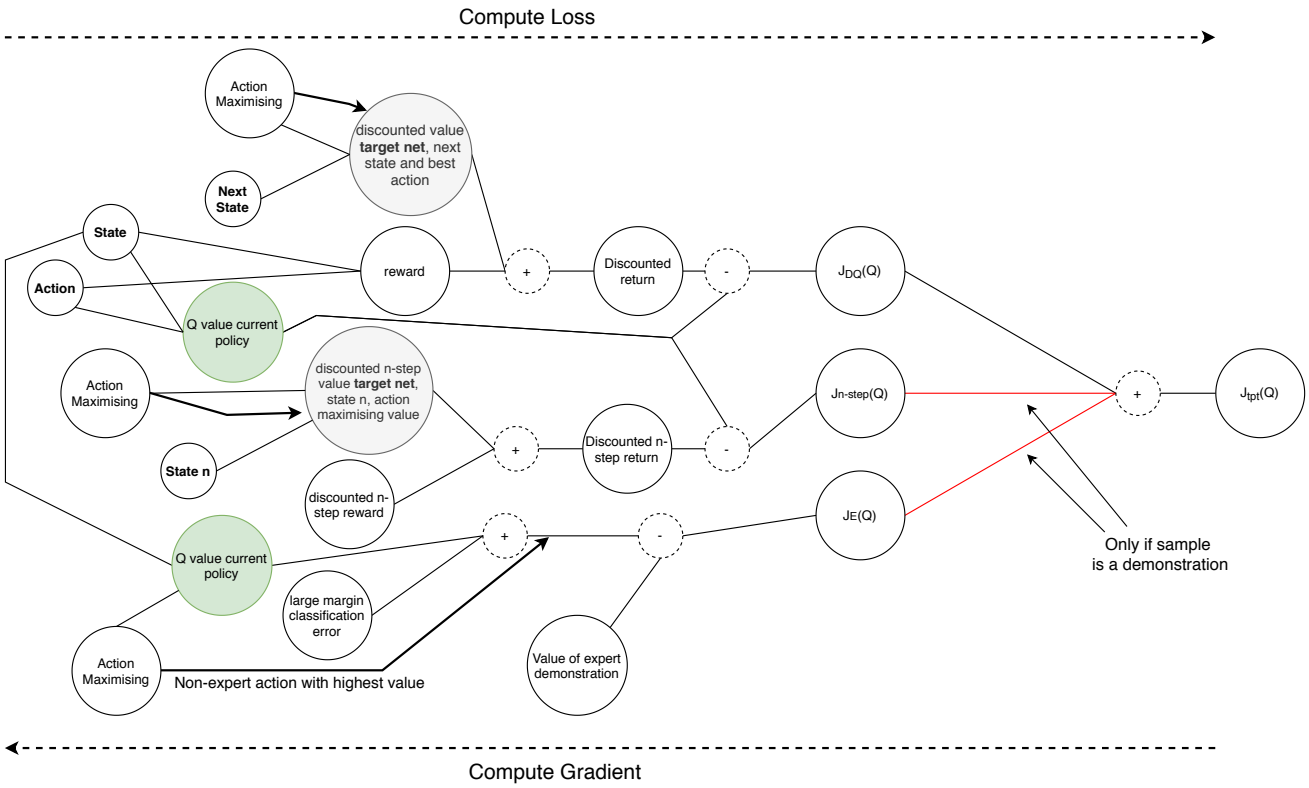


Figure 9.9: Computational graph of DQfD. Nodes which are green require a gradient (the weights of these networks are updated) and the grey nodes indicate that the target network is used to calculate the value. Note that the n-step loss and large phase margin loss are only applied when the sample is a demonstration.

Hyperparameters

Prior to comparing DQfD with PPO, a hyperparameter selection and tuning must be performed. The selection of hyperparameters is important stage in the RL process that affects the performance (R.S. Sutton and A.G. Barto 1998). In Table 9.3, the hyperparameters of DQfD are listed. The only hyperparameter of which the value is not stated in the paper is the learning rate. Since the problems faced in the paper, the Atari games, are more complex than the SSD environment it is chosen to decrease τ from 10.000 to 10. This should only be increased in case the learning becomes unstable. The hyperparameters which are often most important to tune are the discount factor, the learning rate and exploration rate. For computational reasons, it is decided to only tune the discount factor and the learning rate for the trade-off. In case this algorithm is selected in the trade-off, a more extensive hyperparameter tuning will be performed in the final thesis.

There are three methods which can be used for hyperparameter selection: manual tuning, tuning through a grid search and tuning through Bayesian optimisation. In the final thesis, a grid search or Bayesian optimisation will be performed. For computational reasons it is chosen to perform a manual grid search for the trade-off.

Table 9.3: Value of the hyper parameters used in DQfD which are not tuned for the trade-off.

Hyper Parameter	Value Paper	Value Used	Explanation
Discount rate	0.99	*tuning*	importance of future states
Batch size	128	128	size of batch used to update network
ϵ	0.01	0.01	value of ϵ
τ	10,000	10	frequency at which to update target network
k	750,000	6,400	number of pre-training steps
$l(a_E, a)$	0.8	0.8	expert margin when $a != a_E$
α	0.4	0.4	prioritized replay exponent
ϵ_a, ϵ_d	0.001, 1.0	0.001, 1.0	prioritised replay constants
β_0	0.6	0.6	prioritized replay importance sampling exponent
n	10	3	length of N-step returns
lr	[-]	*tuning*	learning rate optimiser
λ_1, λ_2	1.0, 1.0	1.0, 1.0	weighting coefficients in the loss function

First, the discount rate is tuned. The values of the discount rate which are compared are: 0.2, 0.4, 0.6, 0.8, 0.99. In Figure 9.10, the validation results of training the agent for 600 epochs are shown. The first 50 epochs, the agent pre-trains on 2000 demonstrations. After that, new samples are generated and these are also used for training. One can see that the agent trained with a discount factor of 0.2 continuously has the highest validation reward on average for the first 600 epochs. Therefore, a discount factor of 0.2 will be used to train the agent in the final trade-off and to tune the learning rate.

Figure 9.11 shows the training results of agents having a discount factor of 0.2 and a learning rate of: 0.01, 0.001, 0.0001 and $1.0 \cdot 10^{-5}$. One can notice that the reward received during validation does not vary as much as it did when tuning the discount factor. Initially, the agent with a learning rate of $1.0 \cdot 10^{-5}$ achieves the highest validation reward. At epoch 600, the validation reward received is more similar. Overall, the agent with $lr = 1.0 \cdot 10^{-5}$ achieves the highest validation reward and is therefore chosen to be used for the trade-off.

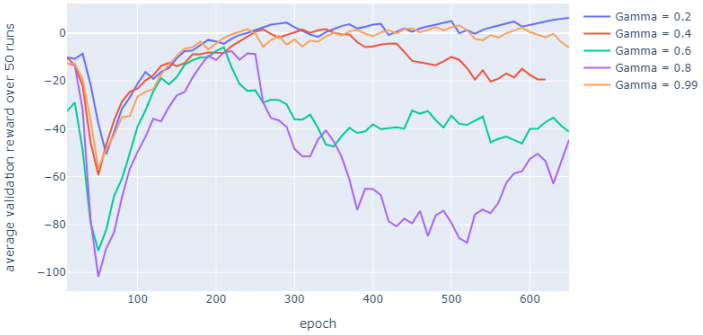


Figure 9.10: Average validation rewards received over 50 episodes using the current policy of the DQfD agent.

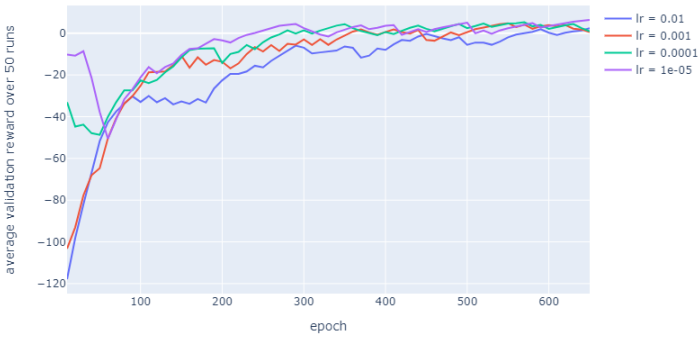


Figure 9.11: Tuning of the learning rate with $\gamma = 0.2$.

DQfD vs. Dueling DQN

Having selected the hyperparameters for DQfD, it is interesting to see how DQfD performs relative to Dueling DQN in terms of data-efficiency. In Figure 9.12, the validation curve for Dueling DQN and DQfD is shown for the first 600 epochs. DQfD indeed is more data-efficient as it achieves a much higher validation reward on average in the first 600 epochs.

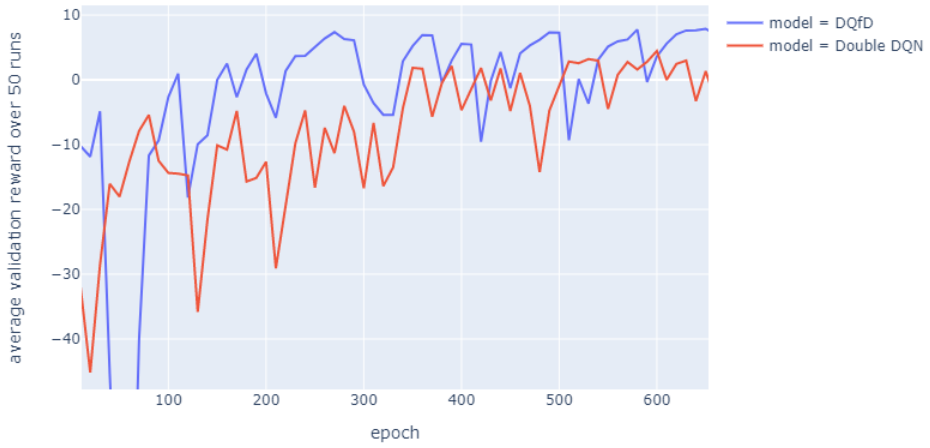


Figure 9.12: Average validation reward retrieved during the first 600 epochs of training. One can see that DQfD is more data efficient.

9.2.2. PPO algorithm

Unlike Double DQN, the Proximal Policy Optimisation algorithm is a policy gradient method. This means that instead of mapping the state-action values, it updates its policy based on the reward retrieved at a certain step. PPO is an advanced actor-critic methods, and therefore learns both a policy as well as a value function. A stochastic policy is implemented to allow the agent to explore the state space without always taking the same action. As explained in subsection 6.3.2, PPO is a trust region method which aims at taking the largest possible step to improve its policy with the data available without risking a sudden collapse in performance. There are two variants to the PPO algorithm presented in (Schulman, Wolski, et al. 2017). The first variant is one that implements a clipped surrogate objective. The second variant is a KL-penalised objective, which can be used instead or in addition to the clipped surrogate objective. It adapts the KL-penalty to achieve some target value of the KL divergence d_{targ} at each policy update. KL divergence is a measure for the difference between distributions. Results in (Schulman, Wolski, et al. 2017) show that the clipped surrogate objective performs better than the KL-penalised objective. Therefore, the latter variant shall not be used in this implementation of PPO.

The general structure of the PPO algorithm is shown in Algorithm 3. One can see that each iteration, the agent acquires a fixed amount of experience and optimises for K epochs on that acquired experience.

Algorithm 3 The pseudo-code of Proximal Policy Optimisation (PPO) (Schulman, Wolski, et al. 2017).

Require: θ: weights for initial behaviour network (random); N: experience horizon; K: number of epochs to perform at each iteration; M: mini-batch size;

1: **for** iteration=1,2,... **do**
2: **for** actor = 1,2,...,N **do**
3: Run policy $\pi_{\theta_{old}}$ in environment for T time steps
4: Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
5: **end for**
6: Optimise surrogate L w.r.t. θ, with K epochs and mini-batch size $M \leq NT$
7: $\theta_{old} \longleftarrow \theta$
8: **end for**

The algorithm uses the current policy to obtain new experiences for N steps. At each step, the difference between the state-action value and state-value, also called the advantage function, is determined. Since an update on the network parameters is performed every N steps, the estimator for the advantage function must not look further ahead then N steps. The estimator used in (Schulman, Wolski, et al. 2017) originates from (Mnih, Puigdomènech Badia, et al. 2016) and is calculated by using the following equation:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{r+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{9.13}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. Having obtained the experiences and computed the advantages, the policy parameters are updated by optimising a loss function with the Adam optimiser for K epochs.

During training, a categorical policy is implemented. This is a stochastic policy which can be used for a discrete action space.

Network Architecture

To make a fair comparison between PPO and DQfD, the same feature extractor will be implemented for PPO. This network architecture was also used to evaluate the performance of the algorithm in the original paper (Schulman, Wolski, et al. 2017). Details about this network can be found in subsection 9.2.1. Instead of having a single network with two heads to estimate the state-value and action advantage, an actor and critic network co-exist. The actor network outputs a probability density over the next possible actions. To do so, a **softmax** layer is added to this network. The critic network outputs a single value. The details on the heads of the two networks are shown in Table 9.4.

Table 9.4: Critic and actor head for PPO algorithm.

	Layer #	Type	Input Size	Output Size
critic head	1	Linear	5408	1
actor head	1	Linear	5408	# actions
	2	SoftMax	#actions	# actions

Loss Function

The loss function is composed of three components when a neural network architecture is used which has shared parameters between the policy (actor) and value (critic) function: the clipped surrogate loss, a squared-error loss between the policy surrogate and value function, and an entropy loss. These components are explained below:

- Clipped surrogate loss:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{9.14}$$

  In this equation, $r_t(\theta)$ is the probability ratio: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. This means that $r_t(\theta_{old}) = 1$. This objective can be decomposed into three parts. The first component, $r_t(\theta)\hat{A}_t$ represents the conservative policy iteration (Kakade and Langford 2002). The second component clips the ratio between $1-\epsilon$ and $1+\epsilon$, which removes the incentive for $r_t(\theta)$ to move outside these bounds. Finally, a minimum operator is applied on the clipped and unclipped objective (Schulman, Wolski, et al. 2017). This part of the loss function ensures that the probability ratio is only included if it would make the objective worse. More details can be found in (Schulman, Wolski, et al. 2017).

- Squared-error loss:

$$L_t^{VF}(\theta) = (V_\theta(s_t) - V_{\theta_{old}}(s_t))^2 \tag{9.15}$$

  This is the critic loss component, which is also added in the DQfD algorithm.

- Entropy bonus term: $S[\pi_\theta](s_t)$. This is added to ensure sufficient exploration.

The total objective is then given by Equation 9.16. In this equation, c_1 and c_2 are coefficients for the critic loss and entropy bonus. In the paper, $c_1 = 1$ and $c_2 = 0.01$.

$$L^{\text{PPO}}(\theta) = L^{\text{CLIP}}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \tag{9.16}$$

The computational graph is shown in Figure 9.13.

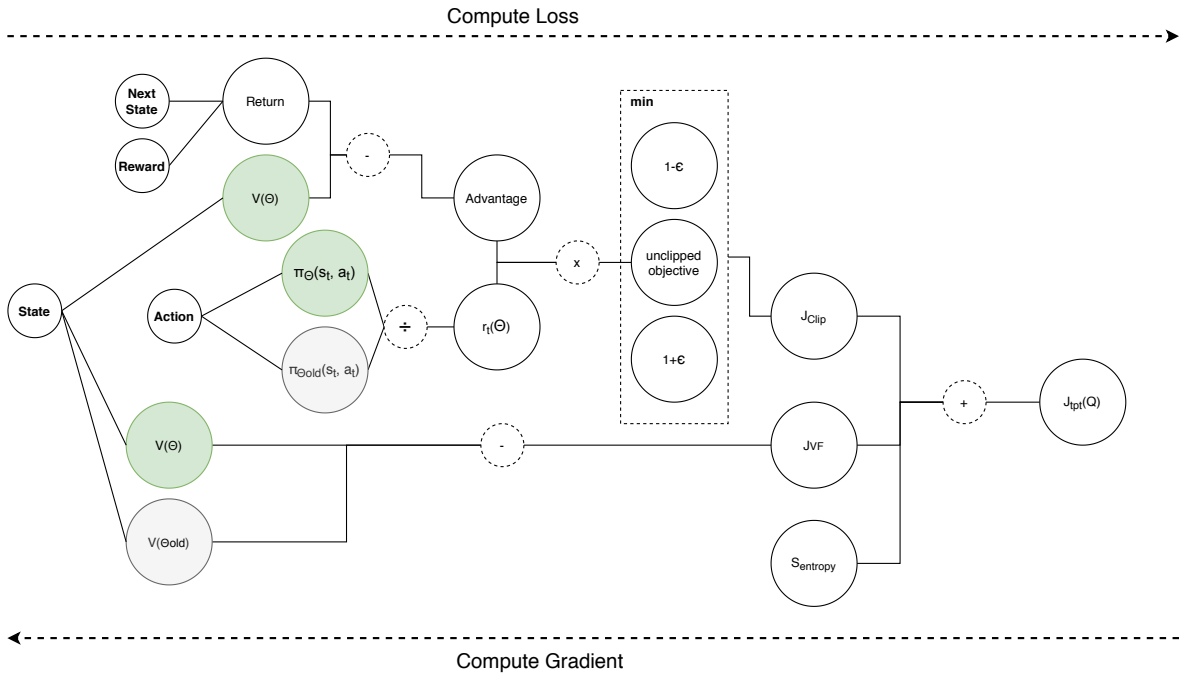


Figure 9.13: Computational graph of the PPO algorithm. Nodes which are green require a gradient (the weights of these networks are updated) and the grey nodes indicate that the old network weights are used to calculate the value.

Hyper Parameter Tuning

PPO is a policy gradient method, the algorithm therefore consists of two phases: acquiring experience and updating its policy. Accordingly, there are two questions to be answered. Namely, how much experience the agent should gather before updating and how the policy should be updated based on the acquired experience. These questions are answered based on the hyperparameters chosen. In Table 9.5, the hyperparameters of PPO are listed.

Just as for DQfD, it is chosen to only tune the discount and learning rate for computational reasons. First, agents with different discount factors are compared using a learning rate of 0.00001. Then, using the chosen discount rate, agents are trained with different learning rates.

Table 9.5: Hyperparameters present in PPO (Schulman, Wolski, et al. 2017).

Hyper Parameter	Acquiring Experience	Updating Policy	Values Paper[3]	Value Used
Horizon	x		2048	2048
Mini-batch range	x		64	64
Epoch range	x		10	10
γ		x	0.99	*tuned*
λ_{GAE}		x	0.95	0.95
Learning rate		x	[-]	*tuned*

In Figure 9.14, the validation rewards retrieved when training the agent with a discount factor of 0.2, 0.4, 0.6, 0.8, 0.99 are shown. It can be seen that initially, the agent with a discount factor of 0.6 outperforms the other agents and also has the steepest validation reward curve at 500 epochs. Therefore, it is chosen to perform the hyperparameter tuning of the learning rate with $\gamma = 0.6$. Using a discount rate of 0.6, the learning rate is

tuned. The results of running PPO agents with a discount factor of 0.1, 0001, 0.0001 and $1.0 \cdot 10^{-5}$ are shown in Figure 9.17. One can see that the agent with a learning rate of $1.0 \cdot 10^{-5}$ continuously achieves a higher validation reward than the other agents. Therefore, it is chosen to use that agent for the trade-off.

Figure 9.14: Average validation rewards received over 50 episodes using the current policy of the PPO agent. Smoothing weight vector of 0.8.

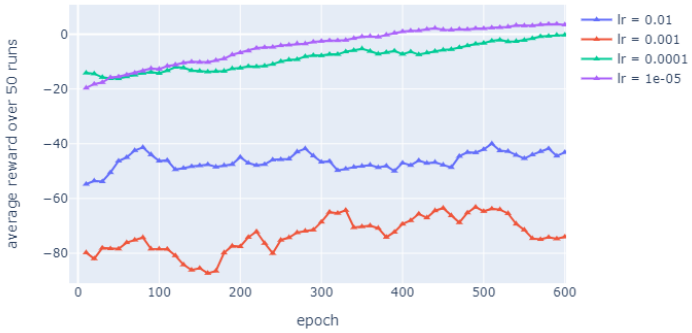


Figure 9.15: Tuning of the learning rate for the PPO agent. One can see that a learning rate of 1e-05 performs best during training. Smoothing weight vector of 0.8.

9.3. Trade-Off

In this section, the DQfD agent and PPO agent are compared in terms of generalisation, sample-efficiency, stability properties, their potential to contribute to the explainability and controller conformity. The two agents were trained until reaching a stable performance on the validation data.

DQfD was trained until epoch 1650 to achieve stable performance whilst PPO was trained until epoch 1050. Stable performance was achieved when the validation curve flattened. The validation curves for the PPO and DQfD agent are shown in Figure 9.16 and Figure 9.17.

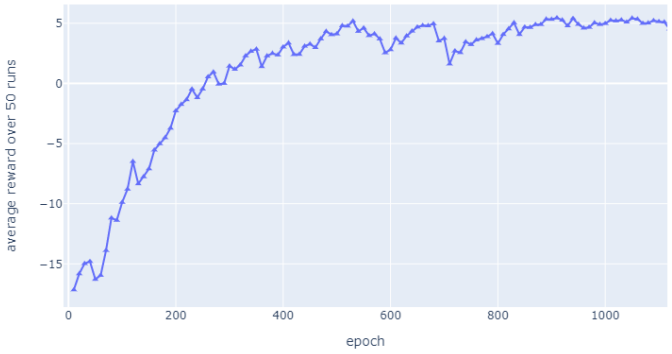


Figure 9.16: Average validation rewards received over 50 episodes during training of the PPO agent.

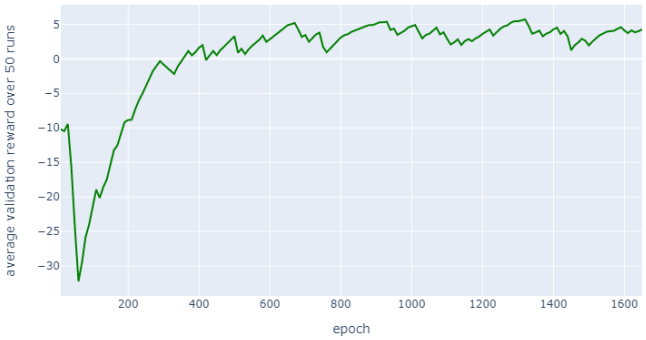


Figure 9.17: Average validation rewards received over 50 episodes during training of the DQfD agent.

9.3.1. Generalisation

To compare the performance of the DQfD and PPO agent, both were used to select actions in 200 simulations. At each of these simulations, the agents were presented with the same initial state.

In Figure 9.18, a box plot of the received reward during the simulations is shown for the DQfD and PPO agent. One can see that the median of the reward received of the DQfD agent is 10, and that the lower and upper bound are also 10. The DQfD agent thus shows a stable performance. This can be explained by the fact that the demonstration data solely contained examples in which the agent selected the action that would turn the aircraft to the target heading, retrieving a reward of 10. Due to the large phase margin loss, the trained agent mimics these actions when encountering similar states. This shows that DQfD can be used to force the automation to take certain actions in a particular situation. By including demonstrations from ATCOs, the automation can become more conform with the strategies applied by ATCOs.

The PPO agent shows more variance in its performance. The median of the received reward during the 200 simulations is 8. The agent did however learn to first take a step in the direction of the target heading before selecting the target heading immediately. This explains the few episode rewards > 10. Although the DQfD agent shows a steady performance, it does not find the optimal solution to the problem. This is caused by the

fact that the demonstrations were imperfect and implied that the agent should always take the action which resulted in a reward of 10. This stresses the importance of good demonstrations when using DQfD.

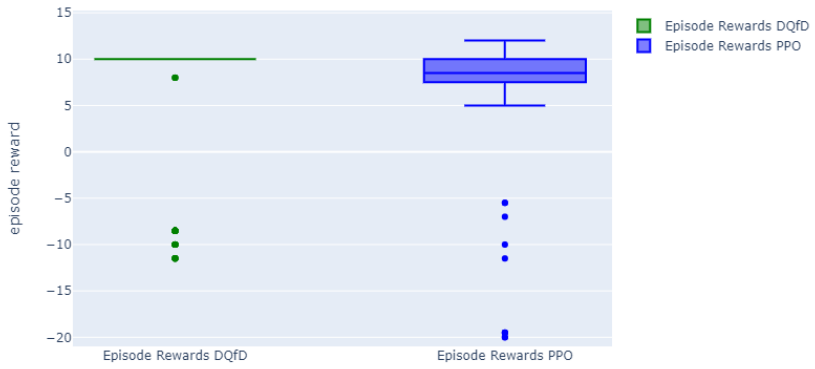


Figure 9.18: Boxplots of received reward on 200 episodes for DQfD and PPO agents. Both agents werre presented with the same initial states.

To get more insight in the performance of the agents, Figure 9.19 shows the percentage of simulations in which the agent received a reward larger than 8. It can be seen that the DQfD agent performs well in 88% of the simulations whilst the PPO agent performs well in only 57% of the simulations. Arguably more important is how often the agent took an action into the FBZ. Figure 9.20 shows that the DQfD agent did not once select an action into the FBZ during the 200 simulations whilst the PPO agent selected an action into the FBZ 17 times.

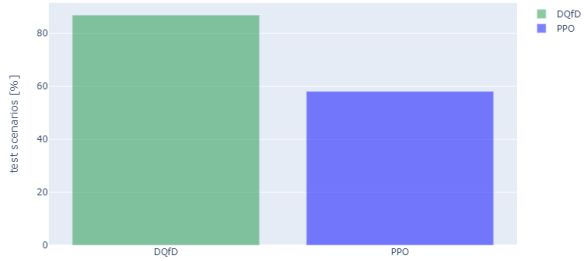


Figure 9.19: Percentage of test scenarios in which the algorithm showed good performance (>8). The percentage is taken from the total number of episodes in which the target heading was outside the FBZ.

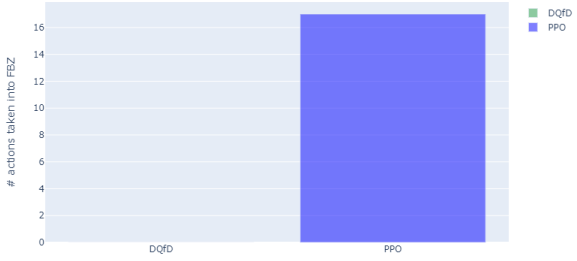


Figure 9.20: Number of times the agent selected an action into the FBZ. One can see that the PPO algorithm selected such an action signficantly more often than DQfD.

9.3.2. Efficiency

There are three components of interest to the efficiency of RL algorithms: the number of frames acquired, the number of frames used to update the network weights and the computation time. The algorithms are compared when achieving a minimal validation reward of 2.

Most important for this thesis is the sample-efficiency, which refers to how many experiences should be acquired for the learning process. The amount of frames acquired are plotted against the validation reward received during training in Figure 9.21 and Figure 9.22. One can see that DQfD needed to acquire much less data (~3,500 frames) than the PPO agent (~88,000 frames) to achieve an average validation reward of 2. It must be noted that in addition to the new frames the DQfD agent needed to acquire, it had 2000 demonstration frames to learn from.

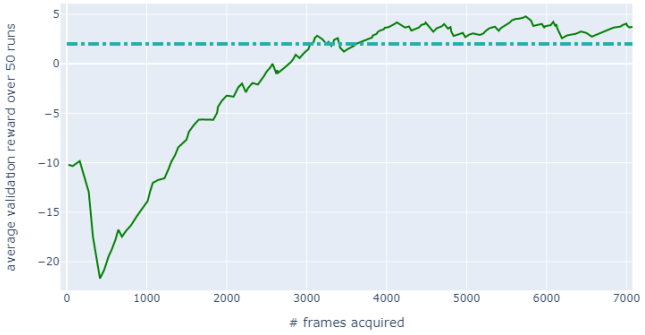


Figure 9.21: Validation curve plotted against the number of new experiences acquired by the DQfD agent.

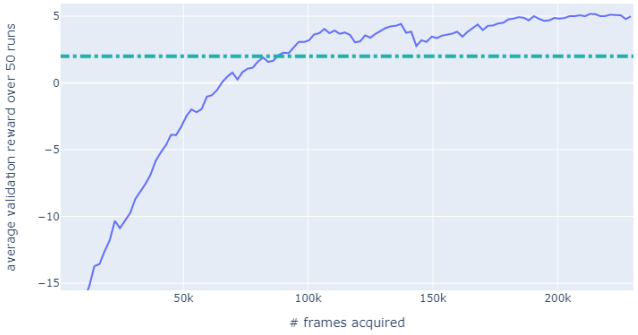


Figure 9.22: Validation curve plotted against the number of new experiences acquired by the PPO agent. One can see that the PPO agent needs to acquire more experiences than the DQfD agent to reach the same validation reward.

This large difference in experiences which the agents need to acquire can be seen in the computation time of the agents. The PPO agent performed updates on ~39,000 frames in ~16,000 seconds to achieve an average validation reward of 2. This is shown in Figure 9.13 and Figure 9.24. On the other hand, the DQfD agent needed to perform updates on ~440,000 frames, which it retrieved from its replay memory, in ~5,600 seconds to achieve the same validation reward. This can be seen in Figure 9.25 and Figure 9.26. The difference in computation time is caused by the amount of time that the PPO agent spends on acquiring data instead of learning from its acquired data. From literature it was already seen that many implementations of PPO use distributed learning to speed up this process. DQfD is much more data-efficient as it uses memory replay.

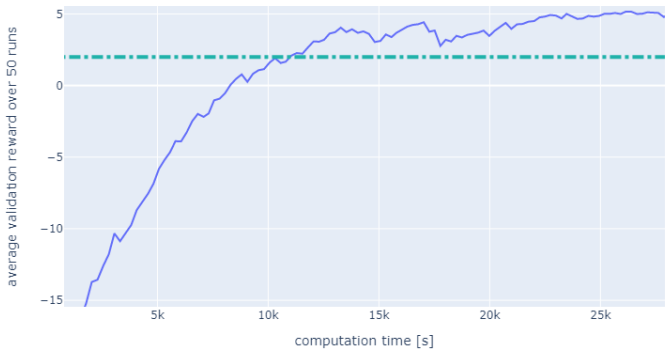


Figure 9.23: Average validation rewards received over 50 episodes using the current policy of the PPO agent plotted against the computation time in seconds.

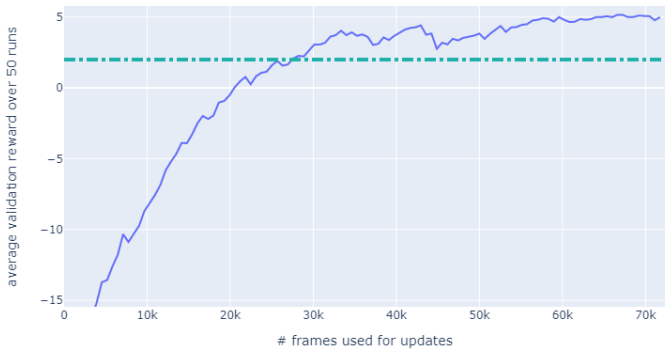


Figure 9.24: Average validation rewards received over 50 episodes using the current policy of the PPO agent plotted against the frames needed for the updates.

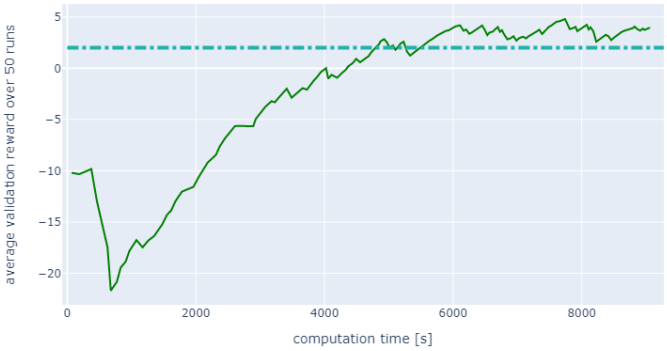


Figure 9.25: Average validation rewards received over 50 episodes using the current policy of the DQfD agent plotted against the computation time in seconds.

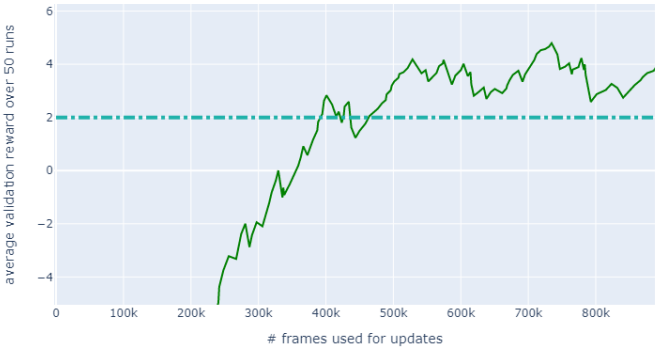


Figure 9.26: Average validation rewards received over 50 episodes using the current policy of the DQfD agent plotted against the frames needed for the updates.

9.3.3. Stability Properties

To analyse the stability properties of the reinforcement learning algorithm, one can look at the loss function. For a stable reinforcement learning algorithm, the loss function should steadily decrease and converge. In Figure 9.27 and Figure 9.28, the loss function for the PPO and DQfD agent are shown. Considering the loss function of the PPO agent, it is apparent that it initially steadily declines but keeps on fluctuating in the limit. The agent has thus not yet converged to a stable policy. The training loss function of the DQfD agent shows much less variance and seems to converge to a steady value.

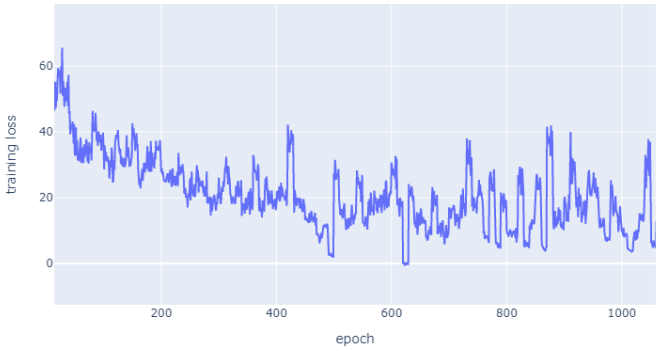


Figure 9.27: Training loss PPO.

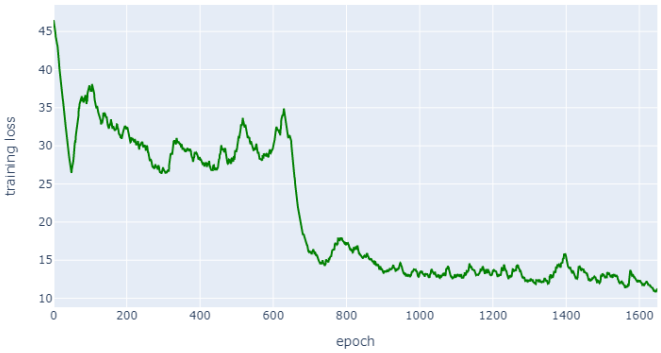


Figure 9.28: Training loss DQfD.

Looking at literature available, Dueling DQN and PPO have shown to learn a well-performing policy on different Atari games. Dueling DQN has shown to achieve high performance only on games with discrete action spaces whilst PPO has also achieves high performance on continuous action spaces (Schulman, Wolski, et al. 2017). Since the action space of the agent in the final thesis will be discrete, it is expected that both methods will converge to a well-performing policy.

9.3.4. Explainability

In this trade-off, explainability will be evaluated qualitatively instead of quantitatively. Since there are yet hardly any implementations of reinforcement learning algorithms which are focused on being explainable, the algorithms shall be evaluated for the potential they have to contribute to the explainability of the automation. Both agents can be designed such that they show wanted behaviour by altering the reward function. For

deep RL agents in specific there are two methods which can contribute to the explainability of the model: visualising features and reward decomposition.

As explained in section 7.3, reward decomposition can only be applied to value based methods. For these methods theoretical guarantees exist that the overall Q-function still converges to the optimal Q-function. This allows the model to continuously show why the agent makes a certain decision. Since PPO is not a value based method, DQfD has the most potential to contribute to the explainability of the model. Next to reward decomposition, saliency maps and LRP can be applied to help human operators understand what features in a SSD have led the agent to make a certain decision. These can be applied on models trained by either PPO or DQfD. To conclude this section, DQfD has higher potential to become more explainable than PPO since reward decomposition can be implemented alongside DQfD.

9.3.5. Controller Conformity

Controller conformity can be achieved in two ways. First of all, careful design of the reward function can cause the agent to mimic strategies employed by ATCOs. This holds for all the algorithms. However, an algorithm with a significant better performance in the limit is more likely to accurately mimick strategies incorporated in the rewards function. Experiments showed that the PPO agent was able to identify scenarios in which the optimal reward was obtained whilst DQfD failed to. Experiments are inconclusive as to whether DQfD or PPO shows better performance in the limit. It can, however, be assumed that both algorithms can show wanted behaviour as they both managed to learn a policy which maximised the retrieved reward.

The second possibility is to have an agent learn on demonstration data (DQfD). Since a large margin classification loss is incorporated, the agent will mimick the demonstrations shown in the expert demonstration data. The performance comparison shown in Figure 9.18 shows that the DQfD agent indeed mimicks the demonstration data.

9.3.6. Concluding Remarks

For the final thesis, it is chosen to use the DQfD algorithm. DQfD shows a significantly better performance compared to PPO after acquiring significantly less experience. PPO requires the acquisition of more new experiences compared to DQfD, which is the most time consuming phase in RL. Literature shows that (distributed) PPO in general outperforms most of the value based methods when trained for a large amount of epochs. In this research, agents should be trained for every sector separately. Training a PPO agent for these sectors would therefore computationally be too demanding for the resources attributed to this research.

Moreover, DQfD has two algorithm specific advantages over PPO: it learns from demonstrations and has the potential to contribute more to the explainability of the automation. Demonstrations in DQfD enable the agent to learn ATCO specific preferences, contributing to the conformity of the automation. Furthermore, as DQfD is a value-based method, reward decomposition can be implemented.

9.4. Finalising the Network Architecture

Increasingly deep CNNs are achieving a high model accuracy on image classification tasks, such as the VGG16 network developed in (Simonyan and Zisserman 2014). However, with deeper NNs, more weights have to be updated at each iteration. Furthermore, as the model is made more complex, the model will overfit the training data. Since training efficiency and generalisation is important to this research, it is decided to keep the neural network simple.

In (Van Rooijen 2019), numerous network architectures were compared to see whether it affected the performance of the automation. For supervised learning, it did not significantly affect the model accuracy, but it did have an effect on the training time. In this section, a comparison between three network architectures is made. The baseline architecture is the one used throughout this chapter, a NN which is widely used in DQN implementations. The second neural network is the neural network which was used by (Van Rooijen 2019) to train the supervised learning algorithms. Finally, the third neural network has three convolutional layers, just as the baseline, but also has two pooling layers. It was chosen to include this network to see the effect of including a third convolutional layer on top of the second network architecture.

Table 9.6: Neural Network Architectures for DQfD. *k* indicates the kernel size and *s* the size of the stride.

Architecture #	Composition
Baseline	CONV(k=(8x8),s4) ⟶ CONV(k=(4x4),s2) ⟶ CONV(k=(3x3),s1)
2	CONV(k=(5x5),s2) ⟶POOL(4) ⟶ CONV(k=(5x5), s1)⟶ POOL(2)
3	CONV(k=(5x5),s2) ⟶ POOL(4) ⟶ CONV(k=(5x5),s1) ⟶ POOL(2) ⟶ CONV(k=(2x2),s1)

The results of training a DQfD network with the three different NN architectures are shown in Figure 9.29. All three networks were trained with a discount rate of 0.2 and a learning rate of $1.0 \cdot 10^{-5}$. The results show that the baseline architecture is more data-efficient and achieves a higher validation reward throughout the first 600 epochs of the training phase. Therefore, the baseline network architecture will be used in the final thesis.

Figure 9.29: Comparison of network architectures. One can see that the baseline network achieves the highest validation reward on average during the first 600 epochs of training.

III

Conclusions and Recommendations

10

Conclusions and Recommendations

10.1. Addressing the Original Research Questions

This work aimed to develop a form of automation for ATC that contributes to being explainable for a human ATCo. In order to contribute to the explainability, two approaches were used. The first aimed to increase the conformance between the automation and the ATCo, whilst the second was aimed at increasing the mental model of the automation for an ATCo. This research aimed to answer the main RQ:

"How can reinforcement learning be applied to the ATC task of conflict detection & resolution by exploiting features from the Solution Space Diagram and contribute to the explainability of the automation?"

There were six subquestions to this main research question. These could be answered through a literature review and by running various experiments.

1. What traffic scenarios and ATC tasks are relevant to analyse?

   A: In the literature review, the control task of 2D en-route conflict detection & resolution was identified as the focus of this research due to its importance in an increasingly crowded airspace. Similar traffic scenarios are encountered frequently in a sector since it is composed of fixed airways. Just as an ATCo specialises in an air traffic sector, an agent should be trained for each sector separately.

2. What type of reinforcement learning agent can be used to automate CD&R for a single controlled aircraft in a multiple-aircraft traffic scenario by using the SSD as state-input?

   (a) What states and actions are used by an ATCo to perform their tasks?

   A: ATCos identify and resolve upcoming conflicts pairwise. ATCos primarily resort to an altitude change in the en-route airspace, secondly a heading change, and lastly, a velocity change. The latter instruction is avoided as the velocity window of aircraft in the en-route airspace is limited by the flight envelope. The time to the closest point of approach, the distance of closest approach, conflict angle, and speed difference are primary conflict characteristics that an ATCo uses to determine what action to take.

   (b) What information can be extracted from the SSD?

   A: A single SSD does not enable an ATCo to determine the most optimal resolution for an upcoming conflict as certain features in the SSD are not distinguishable for a human. All of the essential conflict characteristics identified in *RQ 2a* are contained in a single SSD. It was hypothesised that an artificial agent could learn certain features in the SSD that are hard to interpret for an ATCo. Using the pixel-data of the SSD as input for the reinforcement learning agent ensures that the state has a fixed size, limiting the observation space to specific bounds.

   (c) Which deep RL algorithms are suitable for a two-aircraft traffic scenario?

   A: A literature survey showed that multi-agent RL algorithms still show too much instability when the task becomes too complex and require too many computational resources. Proximal Policy

Optimisation and Deep Q-learning from Demonstrations were identified as potential RL algorithms for this research due to their learning characteristics and potential to contribute to the explainability of the automation.

A: In the preliminary analysis, a trade-off between PPO and DQfD was performed using a simplified ATC environment in which the SSD was provided as input for the agent. It was concluded that DQfD would be used for this research as it was more data-efficient and uses demonstrations in the pre-training phase. It was hypothesised that demonstrations could increase the strategic conformity between the ATCo and the automation. Furthermore, since DQfD is a value-based algorithm, it can be combined with reward decomposition (RQ 4a).

3. What metrics can be used to evaluate the performance of the automation?

   A: A literature review showed that the most significant parameters to analyse the performance of the automation are whether a loss of separation occurred, the flight path distance flown, and whether or not the controlled aircraft could reach the exit waypoint.

4. What factors can contribute to the explainability?

   (a) What techniques are there available?

   A: The literature review concluded that research on explainable reinforcement learning remains limited. Reward decomposition was shown to have the most considerable potential to contribute to the explainability of the reinforcement learning algorithm. By learning decomposed value functions, the RL agent can show in real-time which part of the reward function contributes the most to the action selection. Furthermore, it enables the designer to monitor the RL agent in more detail.

   (b) How can strategy be incorporated?

   A: The final thesis explored two different methods to answer this research question. First of all, by utilising demonstrations and secondly through shaping the reward function. Demonstrations can be utilised to pre-train a RL agent to be strategically conformal to the demonstrator. This research supports the hypothesis that demonstrations can increase the conformance of the automation since results show that the RL agent starts optimising from the pre-trained model. For all conflicts, the RL agent did improve on the demonstrations it was provided with, at the cost of conformance.

   Carefully designing the reward function enables the incorporation of ATCo strategies. This research focused on strategies related to the conflict geometry of the resolution. Putting this concept into a broader perspective, strategies can be incorporated to take into account any of the parameters present in the state representation of the RL agent.

   A: This research question was answered in the final thesis by implementing **decomposed rewards** and increasing strategic conformance by using demonstrations and shaping the reward function. Reward decomposition enables the visualisation of what the RL agent has learned in terms of the various components of the reward function. Reward decomposition takes a first step to increase the understanding of the action selection for the user, and can also assist the designer in analysing what the RL agent has learned in more detail. Results show that reward decomposition can assist in tuning the reward function in environments that require complex learning functions to be used.

5. How can the automation be implemented in a two-aircraft traffic scenario with a controlled and observed aircraft?

   (a) What factors should be incorporated in the reward function?

   A: The literature review highlighted the importance of automation to be conformal with an ATCo. To resolve a conflict, an ATCo, in general, provides the controlled aircraft with a single instruction. After the conflict is solved, the ATCo can instruct the controlled aircraft to continue its navigation towards the exit waypoint. Three components were identified to be included in the reward function. The RL agent would be negatively rewarded for getting into a loss of separation, negatively rewarded for instructing the aircraft to perform a heading change to avoid a conflict, and

negatively rewarded proportionally to the flight path deviation. To enforce that the agent would instruct the controlled aircraft to continue its intended route as soon as the conflict was resolved, it was not negatively rewarded for instructing the controlled aircraft to continue its flight path to the exit waypoint.

A: In the final thesis, this research question was answered by implementing Deep Q-learning from Demonstrations and Decomposed Dueling DQN which use the Solution Space Diagram to represent the state of the controlled aircraft. It was shown that the RL agents were able to extract all significant conflict characteristics from the state representation to resolve conflicts efficiently in terms of flight path deviation. By pre-training the RL agent on a limited training set of demonstrations, the initial performance of the RL agent was improved significantly.

10.2. Concluding Remarks

Results show that the SSD is a promising state representation for a RL agent to solve conflicts pairwise for 2D en-route CD&R. The author does believe that reconsidering the pre-processing of the SSD for a RL agent is essential for future implementations. Furthermore, this research shows that training the RL agent with demonstrations increases the data-efficiency of the algorithm and strategic conformance of the automation.

Whereas this research started with a focus on increasing the explainability of the automation for a human ATCo, it actually has taken a step to increase the explainability for the designer. Reward decomposition provides a designer with useful information on the hierarchies between the various reward components, which can be used to tune the reward function. Also, it enables the designer to spot anomalies in the learning.

10.3. Future Recommendations

Although this research has shown how various concepts can be combined to automate CD&R whilst contributing to the explainability, there are still some challenges to overcome. Future recommendations are focused on five different areas of research: increasing the applicability in the current ATM system; the effect of hyperparameters of the DQfD algorithm on strategic conformance; the development of reward functions that trade-off strategic conformance and optimal control; the potential of reward decomposition to serve as the basis for a user-interface; expanding the resolution method to solve for conflicts in a realistic air traffic sector.

This work only considered two-aircraft traffic scenarios. To test whether this method could work in a realistic traffic scenario, a RL agent should be trained to solve conflicts pairwise in multiple-aircraft traffic scenarios. In multi-aircraft traffic scenarios, the RL agent should be able to solve a wide variety of conflicts. To enable this, the challenges concerning the stability of the algorithm addressed in this research must be overcome first. Moreover, a coordination algorithm should be developed to choose which aircraft must perform a conflict resolution.

In pursuit of strategic conformance through the use of demonstrations, future research is recommended to focus specifically on how hyperparameters of the DQfD algorithm can be tuned to trade-off conformance and optimality. Results show that the RL agent starts to optimise from the pre-trained model but is inconclusive about how demonstrations can precisely be utilised to make up this balance. Parameters that significantly affect the conformance are the number of demonstrations present in the replay buffer, the sampling probability of the demonstrations, the exploration rate, and the pre-train period.

The second approach taken in this research to increase strategic conformance is to shape the reward function to incorporate the preferences of ATCos. Future research is encouraged to map all personalised strategies that ATCos apply and research how these can be incorporated when designing the reward function. An advantage of this approach is that the designer can trade-off optimal control and strategic heterogeneity by tuning the weights related to the various reward components.

To increase the mental model of the automation, this research has implemented reward decomposition. The reward function of the CD&R task can naturally be decomposed into meaningful components. To enable reward decomposition to be useful for an user-interface, research should first focus on decreasing the loss of the individual reward components. When the loss decreases to an acceptable level, one can relate the decomposed values to performance measures. The RL agent could then expedite why it prefers a particular

action in terms of flight path distance to the exit waypoint or in terms of how many new conflicts the RL agent expects a resolution maneuver will cause (sector disruptiveness).

Finally, stability of the learning algorithm remains a challenge for deep RL methods. To enable training a RL agent to solve conflicts for a wide range of traffic scenarios, improving the stability is deemed necessary. More advanced RL algorithms achieve a higher performance in the limit and have better stability characteristics. Especially distributed learning algorithms improve the stability of these algorithms. The effectiveness relies heavily on the computational resources available. In case the availability of these resources improves, the author urges the implementation of state-of-the-art distributed learning algorithms.

IV

Appendices

A

Training Methodology

In this chapter, the training methodology used to train the RL agents in the various case studies is elaborated on. The details of the experiment setup for the various case studies in terms of air traffic scenarios are detailed in the scientific paper. This chapter specifies more of the actual implementation in Python to train the various RL agents.

A.1. Simulation Environment

As simulation environment, it was chosen to use BlueSky (Hoekstra and Ellerbroek 2016). BlueSky is a high-fidelity ATC simulation environment. The user can interact with the simultion environment through an interface or by running BlueSky in 'detached' simulation mode. This research focuses on the en-route airspace. Specific details on the experiment implementation in BlueSky are listed below:

- All traffic scenarios contain aircraft that are initialised at FL (Flight Level) 360. During all the experiments, the aircraft do not change altitude.
- All aircraft used in the simulation are B737's.
- To calculate performance metrics of the aircraft, *T*he *OpenAP* (Sun et al. 2020) was used as performance model.

A.2. State Calculations and the Solution Space Diagram

Consider the controlled aircraft, A, and observed aircraft, B. The future position of either aircraft, dependent on the current velocity vector v, can be calculated using Equation A.1. The distance between the controlled and observed aircraft at future positions can then be calculated using Equation A.2, in which p_A is the position of the controlled aircraft whilst p_B is the position of the observed aircraft.

$$p(t) = p_0 + v \cdot t \tag{A.1}$$

$$d_{A,B}(t) = p_A(t) - p_B(t) \tag{A.2}$$

To calculate the time to closest point of approach, t_{CPA}, the absolute distance between aircraft A and B, $D_{A,B}(t)$, is of interest. The calculations for this variable are shown in Equation A.3.

$$\begin{aligned} D_{A,B}(t) &= ||d_{A,B}(t)|| \\ &= d_{A,B}(t) \cdot d_{A,B}(t) \\ &= ((p_{0_A} + v_A \cdot t) - (p_{0_B} + v_B \cdot t))^2 \\ &= ((p_{0_A} - p_{0_B}) + (v_A - v_B) \cdot t)^2 \\ &= (p_{0_A} - p_{0_B}) \cdot (p_{0_A} - p_{0_B}) + (v_A - v_B) \cdot (v_A - v_B) \cdot t^2 + 2(p_{0_A} - p_{0_B}) \cdot (v_A - v_B) \cdot t \end{aligned} \tag{A.3}$$

Hence, the t_{CPA} can be found by calculating the time which minimises $D_{A,B}(t)$. To find this, the derivative of the absolute distance between the controlled and observed aircraft should equal zero. The full derivation of t_{CPA} is shown in Equation A.4 (Huo, Delahaye, and Y. Wang 2018).

$$\begin{aligned} \frac{d}{dt} D_{A,B}(t) &= 0 \\ 2 \cdot (v_A - v_B) \cdot (v_A - v_B) \cdot t + 2(p_{0_A} - p_{0_B}) \cdot (v_A - v_B) &= 0 \\ t_{CPA} &= -\frac{(p_{0_A} - p_{0_B}) \cdot (v_A - v_B)}{||(v_A - v_B)||^2} \end{aligned} \tag{A.4}$$

Having calculated the time to the closest point of approach, the distance of closest approach can be calculated using Equation A.5 (Huo, Delahaye, and Y. Wang 2018).

$$d_{CPA}(t) = ||p_A(t_{CPA}) - p_B(t_{CPA})|| \tag{A.5}$$

To compose the solution space diagram, use was made of pre-defined functions in BlueSky to calculate the forbidden beam zones. These were plotted using *matplotlib*. On top of that, the following steps were taken:

1. Determine V_{min} and V_{max} of the controlled aircraft using the performance model. Plot these as the inner and outer circle in the SSD.
2. Calculate the FBZ per observed aircraft.
3. Calculate t_{CPA} and d_{CPA} to the observed aircraft.
4. Use t_{CPA} to determine the colour of the FBZ.
5. Plot FBZ between V_{min} and V_{max}.
6. Plot velocity vector of the controlled aircraft, plot *exit bearing*.
7. Rotate the SSD to have the velocity vector of the controlled aircraft pointed upwards.
8. Crop image to contain only the 'upper half of the SSD'.

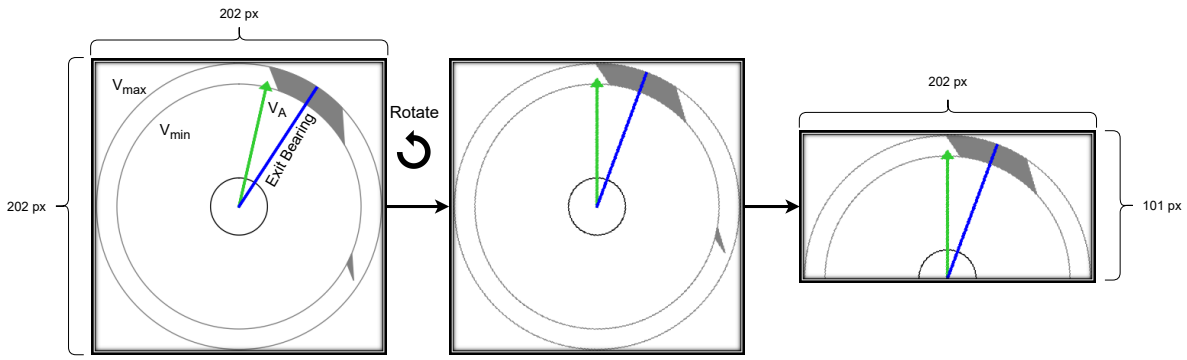


Figure A.1: Construction of the SSD. Details on further pre-processing can be found in the scientific paper.

A.3. Transforming BlueSky

In this thesis, a deep RL method is used. Due to the large number of weights present in the convolutional neural network used in this research, backpropagation is time consuming. Performing backpropagation on a GPU is faster than on a CPU as a GPU can run some of the calculations related to backpropagation in parallel (Navarro, Hitschfeld-Kahler, and Mateu 2014). Computational resources were limited in this research, therefore use was made of Google's Colaboratory. Google Colaboratory allows the user to run code on GPU's for free (for a limited amount of time per day).

BlueSky's back-end was transformed to an *OpenAi's gym environment*. *Gym* was created to serve as standardised format for reinforcement learning implementations. It requires an environment that can give back observations and rewards, whilst requiring an action as input. The *Gym* environment requires the user to define the *observation space*, *action space*, initial conditions of the episode (defined in the *reset* function) and

a step in the RL environment (defined in the *step* function). In this research, the following step and reset functions were used:

- Step(*action*): returns the next state, reward and whether or not the agent has reached the terminal state (*done*). During each step of the RL environment, BlueSky's simulation is run until the *action* is completed, in steps of 10 seconds. For the actions {-10, -5, 0, 5, 10}[deg], the heading change for a B737 will always be completed within 10 seconds. However, for the Δ_{DCT} action, this can take multiple steps of 10 seconds for the action to be completed.

- Reset: resets the environment and initialises an episode.

The experiments were defined in a *Google Colaboratory notebook*. The experiment conditions are used to setup the correct OpenAi gym environment. This ensures that *reset()* of the OpenAI gym environment initialises the episode with the correct initial states of the conflict. The data transfers between the various components needed to enable training are shown in Figure A.2. In this figure, d_{minRT}, represents the minimal distance from the controlled aircraft to the exit waypoint for the episode to be finished.

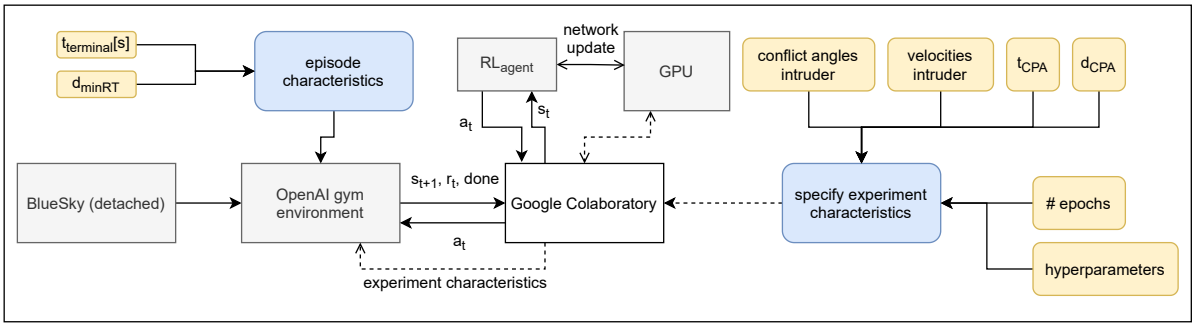


Figure A.2: Software architecture used to train the Dueling DQN, DQfD and drDuel-DQN RL agents.

To monitor the learning, *Tensorflow's Tensorboard* was used. From the notebook in Google Colaboratory, real-time information on training statistics such as the loss per episode and accumulated reward were written to Tensorboard logs. For most complex RL algorithms, publicly available *Python libraries* are available for use. However, the DQfD and decomposed Dueling DQN algorithm is not included in any of these libraries yet. To build these RL agents, the library *Pytorch* (Paszke et al. 2019) was used.

A.4. Training Loop

In this section, the training loop for the Dueling DQN and normal training phase of the DQfD agent is detailed. First of all, the policy network parameters are initialised using Xavier Weight Initialisation (Glorot and Bengio 2010). It was chosen to use this weight initialiser as it has been used in other Dueling DQN implementation (Tavakoli, Pardo, and Kormushev 2017). After the weights of the policy network are initialised, a copy of the network is made and used as target network for the first τ-steps. Further details on the training loop can be found in Figure A.3. It is important to note that samples are removed in the order that they entered the memory replay buffer once its capacity is surpassed.

For the network updates, the Adam optimizer is used (Kingma and Ba 2014). This optimizer maintains a per parameter learning rate. The state of this optimizer should therefore always be loaded before continuing the learning.

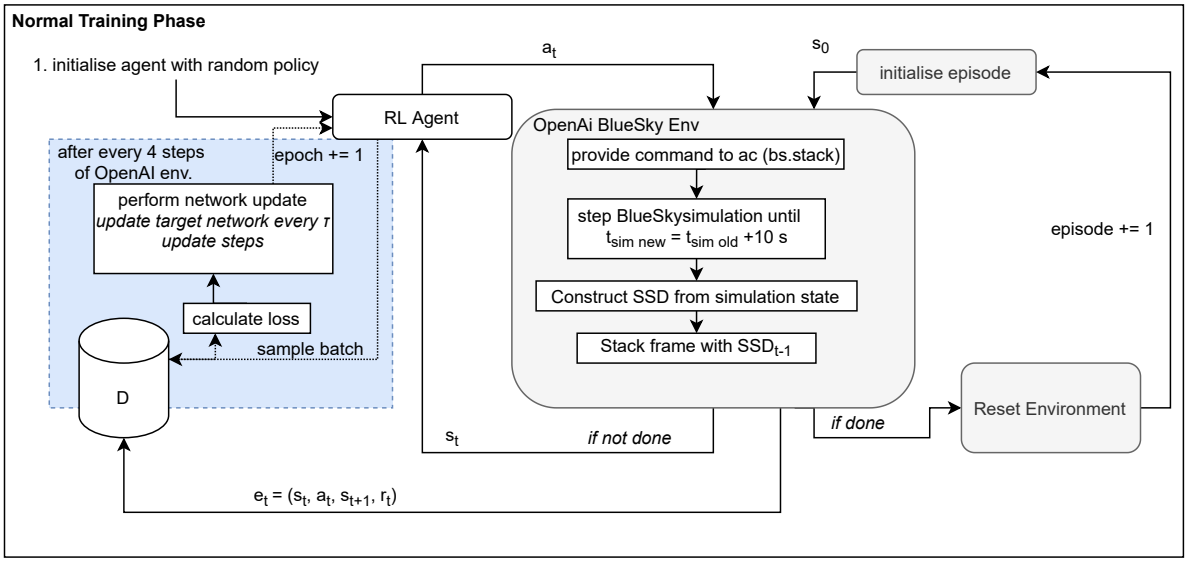


Figure A.3: Training loop used for experiments. Note that the network parameters are updated once every 4 steps of the simulation.

A.5. Pre-Training Phase DQfD

During the pre-training phase of the DQfD agent, which is visualised in Figure A.4, the agent does not interact with the simulation environment. Each iteration, it samples n samples from the *expert memory replay* and performs updates on these using the Adam optimizer. When collecting demonstrations from BlueSky, episode information is collected in the *episode replay buffer*, which is used to calculate the n-step loss.

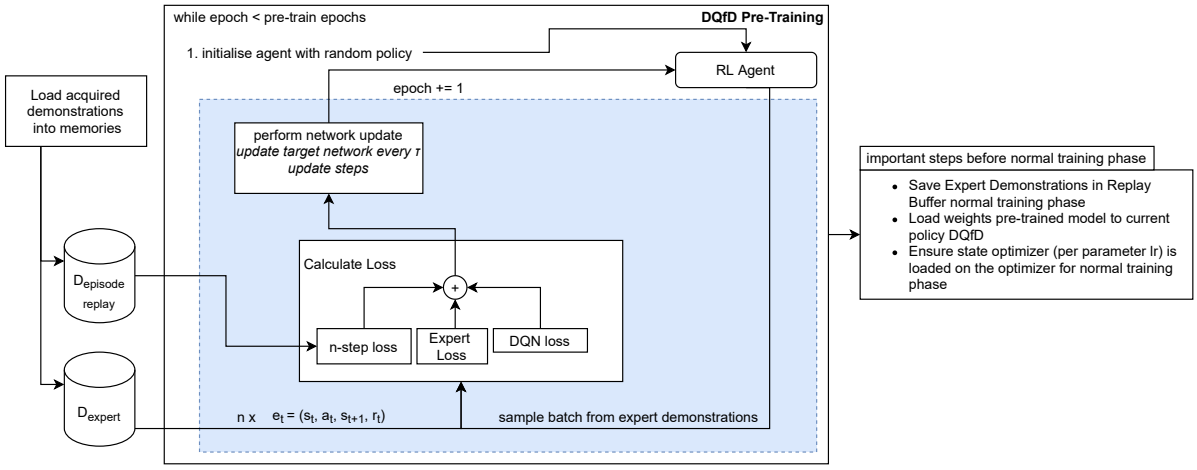


Figure A.4: Pre-training phase of DQfD agent.

A.6. Reward Decomposition

The training loop of the drDuel-DQN agent is similar to that of the Dueling DQN agent. The only difference is that each sample that is stored in the memory replay contains contains the separate reward components, instead of the full reward retrieved at t_{sim}. The experience saved in the memory replay for case study 3 and case study 4 are shown in Equation A.6 and Equation A.7.

$$e_t = (s_t, a_t, s_{t+1}, r_{NA_t}, r_{TH_t}, r_{LOS_t}) \tag{A.6}$$

$$e_t = (s_t, a_t, s_{t+1}, r_{NA_t}, r_{TH_t}, r_{LOS_t}, r_{CGP_t}) \tag{A.7}$$

A.7. Detailed Implementation of Dueling DQN Algorithm

In Algorithm 4, the algorithm used to train a Dueling DQN agent is detailed in pseudo-code. During training, the RL agent first collects random samples to fill up the memory replay for e_{min} steps. After this phase, the RL agent starts to perform updates on the network weights once every 4 steps in the simulation environment. To incorporate the importance sampling weights, the loss on each experience in the batch is multiplied by the related importance sampling weight, w_i. The calculations for the importance sampling weight are detailed in the scientific paper.

Algorithm 4 The pseudo-code of Dueling DQN (Z. Wang et al. 2016). The behaviour policy $\pi^{\epsilon Q_\theta}$ is ϵ-greedy with respect to Q_θ.

Require: $\mathbb{D}^{replay}$: initialised with demonstration data set; θ: weights for initial behaviour network (random); θ': weights for target network (random); τ: frequency at which to update target net; e_{min}: minimum amount of experiences required in replay memory prior to gradient updates; n: batch size; α_0: initial learning rate Adam optimizer; *OpenAI_environment*

$s \longleftarrow$ *Initialise_OpenAI_environment*(t_{CPA}, d_{CPA}, $CAS_{intruder}$, $CA_{intruder}$)
for steps t $\in \{1, 2, ..., e_{min}\}$ {acquiring experiences} **do**
  Sample action a from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
  (s', r) = *Step_OpenAI_environment*()
  Store (s, a, r, s') into $\mathbb{D}$
  $s \longleftarrow s'$
  if s' = done **then**
    Reset_OpenAI_environment()
  end if
end for
for steps t $\in \{1, 2, ..., N_{\text{training epochs}}\}$ {*normal training phase*} **do**
  for steps t $\in \{1, 2, 3, 4\}$ **do**
    Sample action from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
    Play action a and observe (s', r)
    Store (s, a, r, s') into $\mathbb{D}$, overwriting oldest self-generated transition if over capacity occurs
  end for
  Sample a mini-batch of n transitions from D^{replay} with prioritisation {prioritised replay}
  Calculate loss L(Q) using target network {incorporate importance sampling weights, w_i}
  Perform a gradient descent step to update θ (Adam optimiser)
  if t **mod** $\tau = 0$ **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
  epoch += 1
  if s' = done **then**
    Reset_OpenAI_environment()
  end if
end for

A.8. Detailed Implementation of DQfD Algorithm

In Algorithm 5, the implementation of the DQfD algorithm is detailed in pseudo-code. The training phase consists of three phases: pre-training on collected demonstrations; filling the memory replay; the normal training phase. The loss is not only calculated with a DQN loss, as for the Dueling DQN agent, but also includes an Expert and n-step loss component for samples from the demonstrator.

Algorithm 5 The pseudo-code of the DQfD algorithm (Hester et al. 2017). The behaviour policy $\pi^{\epsilon Q_\theta}$ is ϵ-greedy with respect to Q_θ.

Require: $\mathbb{D}^{replay}$: initialised with demonstration data set; θ: weights for initial behaviour network (random); θ': weights for target network (random); τ: frequency at which to update target net; e_{min}: minimum amount of experiences required in replay memory prior to gradient updates; n: batch size; α_0: initial learning rate Adam optimizer; *OpenAI_environment*

load_demonstrations_into_Expert_Replay_memory()
load_demonstrations_into_Episode_Replay_memory()
for steps t $\in \{1,2,...k\}$ {pre-training phase} **do**
  Sample a mini-batch of n transitions from D_{replay} with prioritisation
  Calculate Expert Loss {incorporate importance sampling weights, w_i}
  Calculate n-step Loss using target network {incorporate importance sampling weights, w_i}
  Calculate DQN loss (1-step TD) using target network {incorporate importance sampling weights, w_i}
  Determine total loss by adding individual components
  Perform a gradient descent step to update θ
  if t **mod** $\tau = 0$ **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
end for
$s \longleftarrow$ *Initialise_OpenAI_environment*(t_{CPA}, d_{CPA}, $CAS_{intruder}$, $CA_{intruder}$)
for steps t $\in \{1,2,...,e_{min}\}$ {acquiring experiences} **do**
  Sample action a from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$ {prioritised replay}
  (s', r) = *Step_OpenAI_environment*()
  Store (s, a, r, s') into $\mathbb{D}$
  $s \longleftarrow s'$
  if s' = done **then**
    Reset_OpenAI_environment()
  end if
end for
for steps t $\in \{1,2,..., N_{\text{training epochs}}\}$ {*normal training phase*} **do**
  for steps t $\in \{1,2,3,4\}$ **do**
    Sample action from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
    Play action a and observe (s', r)
    Store (s, a, r, s') into $\mathbb{D}$, overwriting oldest self-generated transition if over capacity occurs
  end for
  Sample a mini-batch of n transitions from D^{replay} with prioritisation {prioritised replay}
  Calculate Expert and n-step Loss on sampled demonstrations {incorporate importance sampling weights, w_i}
  Calculate DQN loss (1-step TD) on all samples using target network {incorporate importance sampling weights, w_i}
  Perform a gradient descent step to update θ (Adam optimiser)
  if t **mod** $\tau = 0$ **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
  epoch += 1
  if s' = done **then**
    Reset_OpenAI_environment()
  end if
end for

A.9. Detailed implementation of decomposed Dueling DQN algorithm

Algorithm 6 details the algorithm used for case study 3 and 4 in pseudo-code. For case study 3, the reward function was decomposed into three different reward components: getting into a short term conflict (SC),

taking nonzero actions (NA) and following the target heading (TH). In case study 4, an additional component was included related to being strategically conformal to a certain conflict geometry preference (CGP).

Algorithm 6 The pseudo-code of the decomposed Dueling DQN. The behaviour policy $\pi^{\epsilon Q_\theta}$ is ϵ-greedy with respect to Q_θ.

Require: $\mathbb{D}^{replay}$: initialised with demonstration data set; θ: weights for initial behaviour network (random); θ': weights for target network (random); τ: frequency at which to update target net; e_{min}: minimum amount of experiences required in replay memory prior to gradient updates; n: batch size; α_0: initial learning rate Adam optimizer; *OpenAI_environment*

$s \longleftarrow$ *Initialise_OpenAI_environment*(t_{CPA}, d_{CPA}, $CAS_{intruder}$, $CA_{intruder}$)
for steps t $\in \{1,2,...,e_{min}\}$ {acquiring experiences} **do**
  Sample action a from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
  $(s', r_{SC}, r_{NA}, r_{TH}, (r_{CGP}))$ = *Step_OpenAI_environment*()
  Store $(s, a, r_{SC}, r_{NA}, r_{TH}, (r_{CGP}), s')$ into $\mathbb{D}$
  $s \longleftarrow s'$
  if s' = done **then**
    Reset_OpenAI_environment()
  end if
end for
for steps t $\in \{1,2,...,N_{\text{training epochs}}\}$ {*normal training phase*} **do**
  for steps t $\in \{1,2,3,4\}$ **do**
    Sample action from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
    Play action a and observe $(s', r_{SC}, r_{NA}, r_{TH}, (r_{CGP}))$
    Store $(s, a, r_{SC}, r_{NA}, r_{TH}, (r_{CGP}), s')$ into $\mathbb{D}$, overwriting oldest self-generated transition if over capacity occurs
  end for
  Sample a mini-batch of n transitions from D^{replay} with prioritisation {prioritised replay}
  Determine greedy action, a_i^+, with respect to the complete target network (all reward components included)
  Calculate loss L_{SC} using target network of the particular reward component, $Q_{SC}(s', a_i^+; \theta_{SC})$
  Calculate loss L_{NA} using target network of the particular reward component, $Q_{NA}(s', a_i^+; \theta_{NA})$
  Calculate loss L_{TH} using target network of the particular reward component, $Q_{TH}(s', a_i^+; \theta_{TH})$
  Calculate loss L_{CGP} using target network of the particular reward component, $Q_{CGP}(s', a_i^+; \theta_{CGP})$ {for case study 4}
  Calculate total loss L(Q) by summing individual loss components {incorporate importance sampling weights, w_i}
  Perform a gradient descent step to update network weights (Adam optimiser)
  if t **mod** τ = 0 **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
  epoch += 1
  if s' = done **then**
    Reset_OpenAI_environment()
  end if
end for

B

Pre-Processing of the SSD and Hyperparameter Selection

In this chapter, the pre-processing of the solution space diagram is analysed. First, section B.1 explains how the SSD should be altered to create a well-defined MDP to train the agent in. Then, section B.2 defines the action space of the controlled aircraft. In section B.3, the first set of experiments are performed to trade-off how the SSD should be pre-processed and determine what batch size and learning rate will be used for this research. In this analysis, a different action space was used than in the scientific article. The reason for this is that the resolution of the action space was increased at a later stage of the thesis to enable a wider range of conflict resolutions for two-aircraft traffic scenarios. Furthermore, a single SSD instead of a stack of two SSDs was used in this analysis as the state representation was altered at a later stage in the thesis. However, the conclusions of this analysis remain significant for the final thesis.

B.1. Altering SSD

To ensure that the environment is a well-defined MDP, the final state should be visible to the agent. When the terminal state is not visible to the agent, the agent has to act in a partially observable MDP (MOMDP). This can lead to instability in the learning and to converging to a sub-optimal policy due to the inability to correctly determine the value of a state (Pardo et al. 2017). At a later stage in the thesis, the components of the reward function are selected. Rewards can either be given continuously, from now on defined as *direct rewards* or at the end of the episode, from now on referred to as *indirect rewards*. Altering the SSD is especially important when incorporating indirect rewards.

To include information on when the RL agent has reached its terminal state, a dot representing the distance from the controlled aircraft to the exit waypoint is plotted along the blue exit bearing indicator. This is shown in Figure B.1.

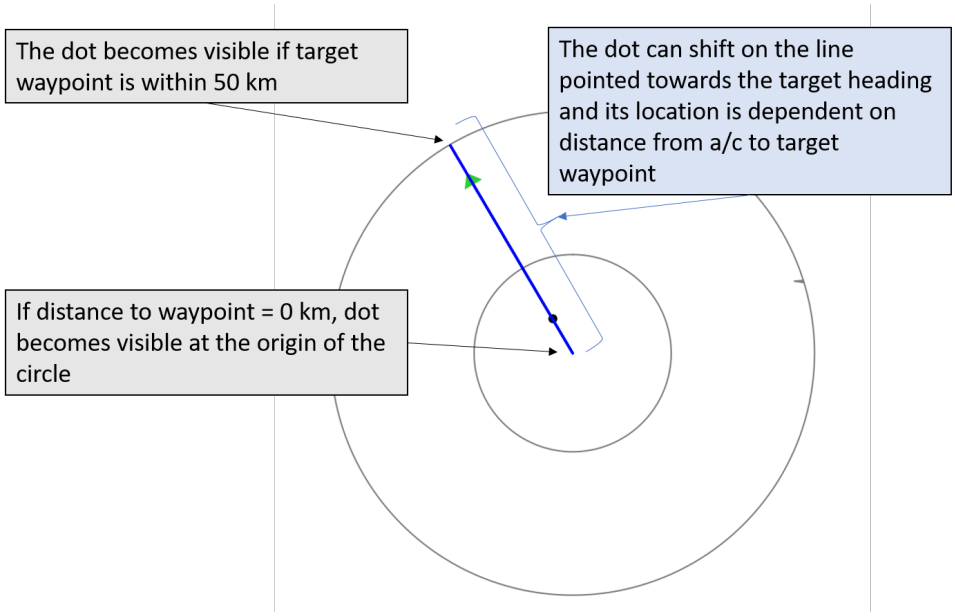


Figure B.1: Adjusted SSD to ensure that the RL agent can observe the terminal state.

B.2. Action-Space

Since one of the goals of this research is to develop a form of automation which is conform with strategies implemented by ATCOs, the action space for the RL agent is analogous to that of a human operator. Human operators most often give commands to pilots in steps of 5 degrees. Furthermore, ATCOs commonly give a single command to solve a potential conflict and once the CPA has passed, they inform the pilot to continue its planned route.

The action space consists of heading changes. A heading change is defined as relative angle from the current heading:

$$\psi_{t+1} = \psi_t + a_t \tag{B.1}$$

Resolutions to upcoming conflicts will generally not involve a more radical action than ± 30 degrees. Therefore, a realistic action space, conform an ATCOs solution space, only spans a range of ±30 degrees. This is shown in Equation B.2, in which taking the action Δ_{TH} instructs the aircraft to follow the target heading.

$$\mathscr{A}_{\text{RL agent}}(s) = \{-30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, \Delta_{\text{DCT}}\}[^\circ] \tag{B.2}$$

The agent will have the opportunity to select an action once every 10 seconds, which is conform with the radar update frequency (Association 2016). The heading change an aircraft can implement depends on the turning rate of the aircraft. A waiting buffer is implemented to ensure that the agent cannot provide an aircraft with a new resolution before the aircraft has reached the previously given command. A larger action space increases the amount of weights in the network. To limit the number of epochs needed for an agent to learn a well-performing policy, the resolution of the action space is lowered to 15 degrees for the first three experiments. The new action space is shown in Equation B.3.

$$\mathscr{A}_{\text{RL agent}}(s) = \{-30, -15, 0, 15, 30, \Delta_{\text{DCT}}\}[^\circ] \tag{B.3}$$

B.3. Learning to Avoid a Conflict

In order to test whether the agent is capable of learning how to avoid a conflict, an agent is trained with the sole purpose of avoiding a loss of separation. In this part of the thesis, the performance models, and thereby realistic flight envelopes, are not yet included in the simulation. The minimum and maximum CAS, independent of the flight phase, of the B737 is taken as minimum and maximum velocity in the SSD. The reward function is composed of two components. The first component is a negative reward for being in conflict with another aircraft. Additionally, a negative reward is added for every non-desired action:

- r_{conflict} = -100 if a loss of separation occurs.
- r_{action} = -5 for all actions except Δ_{DCT} and 0 degrees.

The training environment is defined by a controlled aircraft and a semi-randomly initialised observed aircraft. The observed aircraft is initialised with a slightly higher calibrated airspeed, 340 kts compared to 300 kts of the controlled aircraft. Furthermore, the observed aircraft is initialised at a conflict angle between 5 and 265 degrees, which is visualised in Figure B.2. Dueling DQN is suited for episodic tasks, therefore clear initial and terminal states must be defined. The initial and terminal states of an episode in this learning environment are defined as follows:

- Initial state controlled aircraft:
  - position: (N 52° 12' 10.8" N, E 5° 13' 26.4") [Decimal Degrees]
  - heading: 17 [°]
  - CAS: 300 [kts]
- Initial state observed aircraft:
  - Conflict Angles: {5, 25, . . . , 245, 265 } [°]
  - t_{cpa}: 300 [s]
  - CAS: 340 [kts]
- Terminal state:
  1. Loss of separation between controlled and observed aircraft.
  2. Controlled aircraft reaching target waypoint.
  3. Episode which runs out of time ($t_{sim} > 1200[s]$).

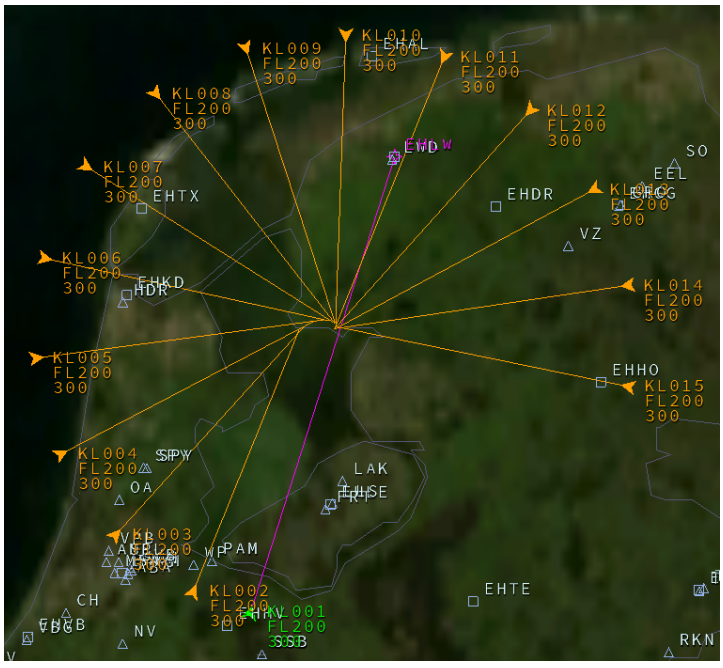


Figure B.2: Possible initial coordinates of observed aircraft. Only one observed aircraft is initialised at the beginning of an episode.[4]

B.3.1. Pre-Processing of SSD

It is hypothesised that all the relevant information to avoid a conflict is present in the upper half of the SSD. This learning environment will be used to investigate how the SSD can best be pre-processed to be used for RL. The four training runs which will be performed are shown in Table B.1. By downscaling the SSD, the size of

[4] In the figure it appears that KL002 and KL003 are in conflict. It is however the case that only one observed aircraft is initiated thus these conflicts will not occur during training.

the state-space, and thereby the amount of weights in the network, is decreased. To perform this experiment, a batch size of 64 was used in combination with a learning rate of 0.00001. These hyperparameters follow from the preliminary analysis.

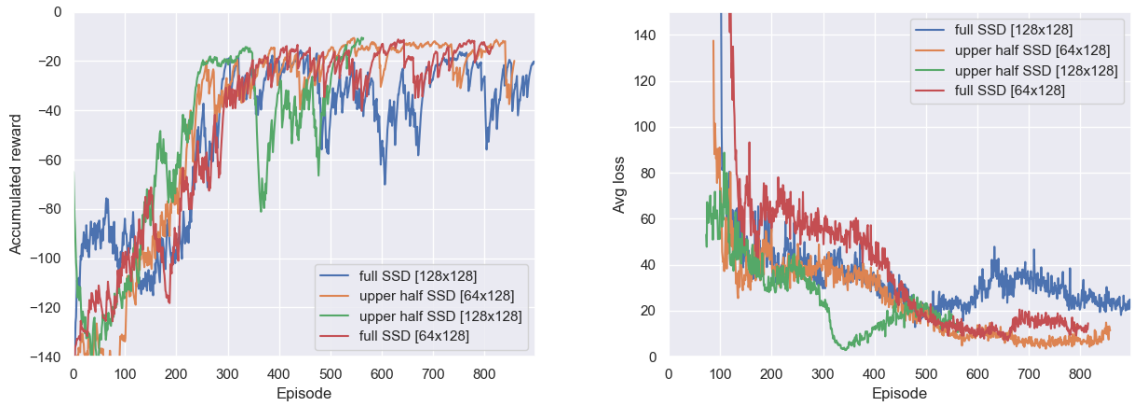
Table B.1: Pre-processing of SSD for the training runs learning to avoid a conflict.

Run	Amount of Information	Pixels	# Weights network
1	Full SSD	128x128	9,513,639
2	Upper Half SSD	64x128	3,222,183
3	Full SSD	64x128	3,222,183
4	Upper Half SSD	128x128	9,513,639

In Figure B.3a and Figure B.3b, the accumulated reward and loss per episode during training the RL agents for 30,000 epochs, with states represented by pre-processing the SSD according to Table B.1, are visualised. Each RL agent has performed a different number of episodes during the training phase due to an inconsistent episode length. Figure B.4a and Figure B.4b visualise the number of conflicts and number of resolution commands given during the training phase. From the graphs, the following conclusions can be drawn:

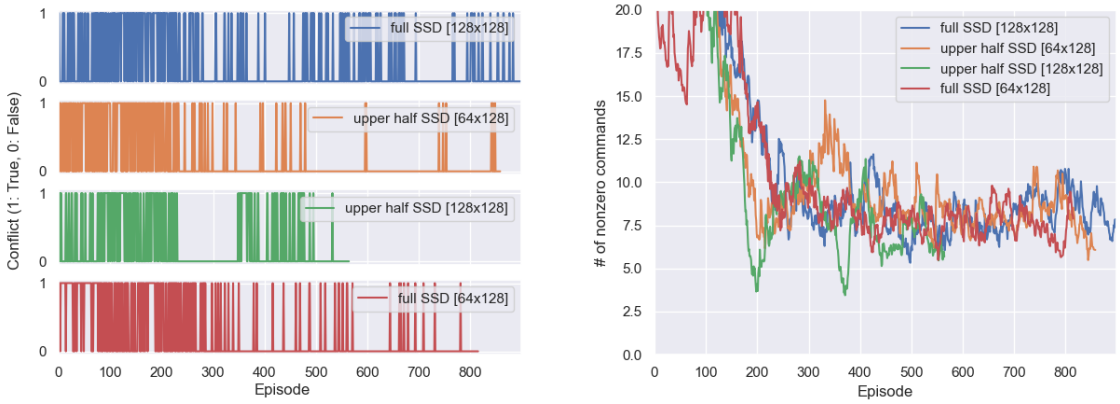
- Both agents with a smaller state space (64x128 pixels), having less weights in the network, outperform the agents with larger state-space (128x128 pixels) when trained for 30,000 epochs. The loss function of these two agents decreases steadily during the training, effectively showing that the agent improves its estimate of the state-action values.
- Cropping of the full SSD to 64 x 128 pixels seems to cause the state to lose information. This is noticeable as the agent which has the same size of the state-space and only uses the upper half of the SSD has a more stable reward curve.
- The 'upper half SSD [64x128]' agent has encountered significantly less conflicts towards the end of the training phase compared to the other agents.
- The amount of resolution actions are similar for all agents.

It is chosen to use the upper half of the SSD to represent the state and to downsize that image from 100x201 pixels to 64x128 pixels in this research. This is due to the slight advantages when considering the learning curves, but also due to the decrease in memory demand compared to the agents which have an input size of 128x128 pixels.



(a) (b)

Figure B.3: Training curves per episode for the Dueling DQN agent being trained for 30,000 epochs learning to avoid a conflict: the left figure displays the reward obtained during the training process; the right curve shows the average loss per episode.



(a) (b)

Figure B.4: Training curves per episode for the Dueling DQN agent being trained for 30,000 epochs learning to avoid a conflict. The left figure visualises whether the episode ended in a conflict and the right figure shows the amount of resolution commands given during an episode.

B.3.2. Hyperparameter Tuning

First of all, it is important to note that these results were obtained prior to altering the training loop for the Dueling and DQfD agent. In this training environment, a single experience was acquired at each iteration instead of four as implemented in the scientific paper. However, the author still recognises the added value of incorporating this section in the appendix as certain relations between hyperparameters and the performance of the algorithm still hold.

The performance of reinforcement learning algorithms is highly dependent on pre-processing of the input and on the hyperparameter selection (Nguyen, Schulze, and Osborne 2019). Especially the learning rate, batch size and exploration parameters have shown to influence the performance of Double DQN significantly. As performance is important for the experiments which are still to be performed, the learning rate, batch size and ϵ_{decay} will be tuned.

First, the effect of changing the learning rate and batch size will be analysed. The best combination will then be used to train three RL agents, each with a different value of ϵ_{decay} to evaluate the effect of altering this hyperparameter. For this hyperparameter selection, the values shown in Table B.2 will be used. According to (Henderson, Romoff, and Pineau 2018), a learning rate between 1e-03 and 1e-05 would show the best performance when using the Adam optimiser. Therefore, these will be analysed. For this hyperparameter selection, the agent starts with a high exploration rate of ($\epsilon_{start} = 0.6$). The exploration rate is decreased to 0.001 in ϵ_{decay} steps to ensure convergence of the loss function.

Table B.2: Hyperparameters compared for tuning.

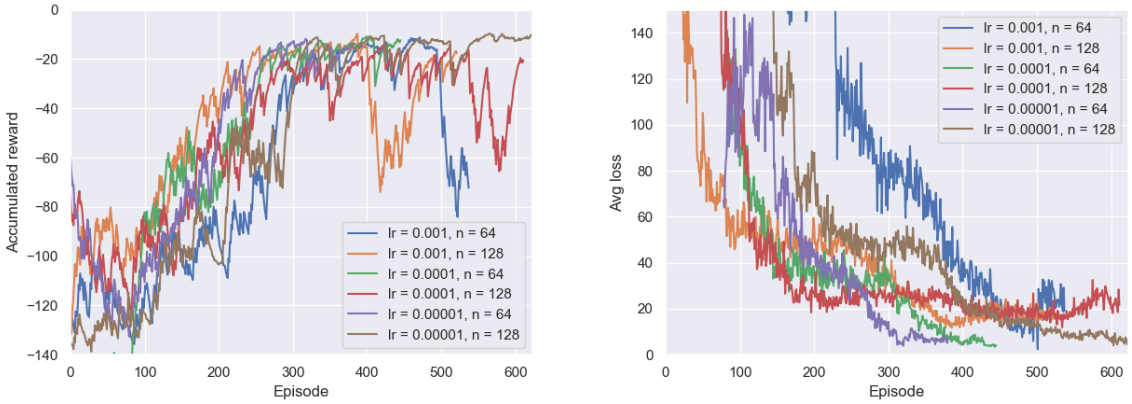
Hyperparameter	Values Tested
Learning rate	0.001, 0.0001, 0.00001
Batch size	64, 128
ϵ_{decay}	3000
ϵ_{start}	0.6
ϵ_{end}	0.001

In Figure B.5a and Figure B.5b, the reward and loss function of the agents trained with the hyperparameters of Table B.2 are shown. All of these agents have been trained with an ϵ_{decay} of 3000. The following observations can be made:

- The RL agents trained with a learning rate of 0.001 perform worse or comparable to the agents trained with lower learning rates.
- Three agents manage to decrease the loss function to a value lower than 5 after 20,000 epochs of training. These are the agents trained with a learning rate of 1e-05 and the agent trained with a learning rate

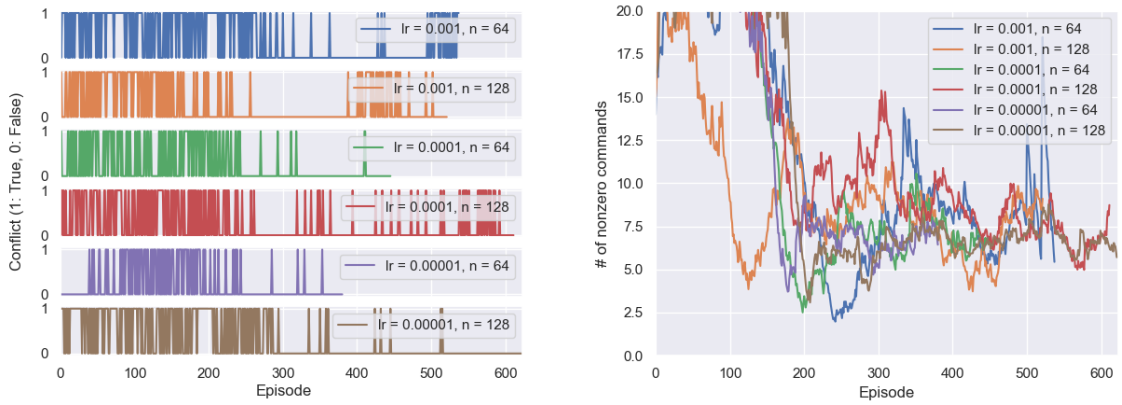
of 1e-04 and batch size of 64. However, it can be noted that the agents with a batch size of 64 manage to achieve a higher reward faster than the one trained with a batch size of 128.

- The RL agent with a batch size of 128 and a lr of 1e-05 seems to have learned a more stable performing policy. Initially, the RL agents gets into many conflicts after which it steadily learns to avoid conflicts. The reward obtained after 20,000 epochs is also slightly higher compared to the RL agent trained with a lr of 1e-05 and batch size of 64.



(a) (b)

Figure B.5: Training curves per episode for the Dueling DQN agent being trained for 20,000 epochs learning to avoid a conflict: the left figure displays the accumulated reward during the training process; the right curve shows the average loss per episode.



(a) (b)

Figure B.6: Training curves per episode for the Dueling DQN agent being trained for 20,000 epochs learning to avoid a conflict. The left figure visualises whether the episode ended in a conflict and the right figure shows the amount of resolution commands given during an episode.

B.4. Concluding remarks

For the final thesis it is chosen to only use the upper half of the SSD to represent the state of the RL agent. Furthermore, the SSD is downsampled from 100x201 pixels to 64x128 pixels. From the hyperparameter comparison, it is chosen to use a learning rate of $1.0x10^{-5}$ in combination with a batch size of 128. Although the training loop and reward function are changed for the experiments performed in the final thesis, it is believed that effect of the analysed hyperparameters do not change. Other decisions on the final selection of hyperparameters are supported in the scientific paper.

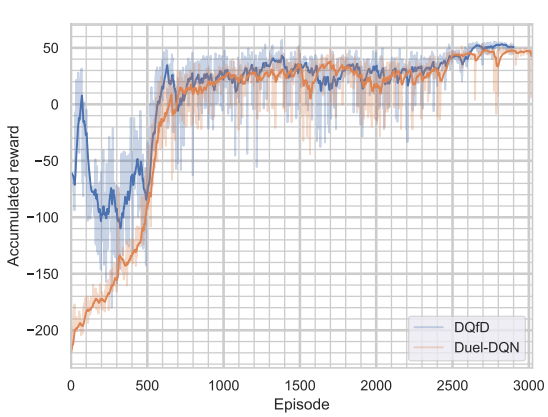
C

Additional Results Case Study 1

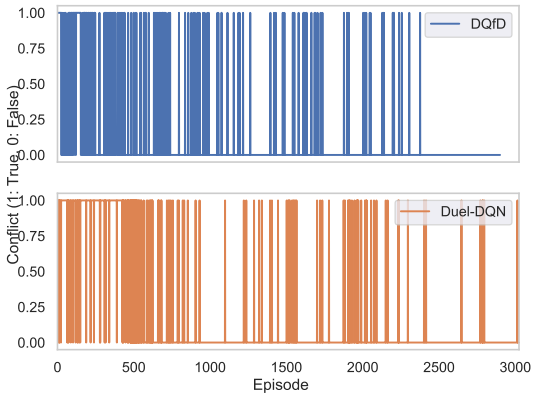
This chapter details the training phase of the RL algorithms trained for Case Study 1. In section C.1, the details of the training phase of the Dueling and DQfD agent are presented.

C.1. Training Curves per Episode

The accumulated reward during training is shown in Figure C.1a. One can see that near the end of the training phase, the agent seems to have converged to a local optimum. In Figure C.1b, whether or not the episode ended in a loss of separation is visualised during the training phase. One can see that both agents initially encounter a lot of loss of separations, whilst near the end of the training phase both have learned to avoid a conflict.



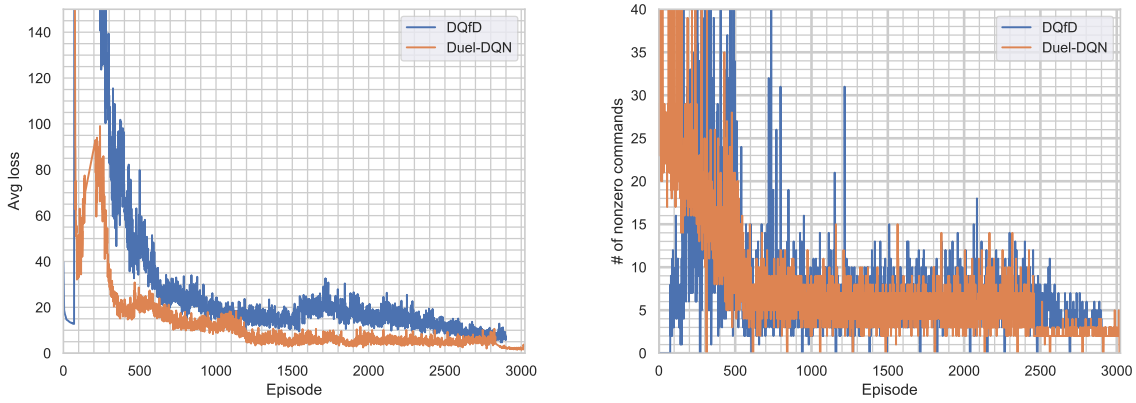
(a) Accumulated reward per episode during training.



(b) Whether or not the episode during training ended in a conflict (1 = True, 0 = False).

Figure C.1

In Figure C.2a and Figure C.2b, the loss and amount of nonzero actions of the Dueling and DQfD agent is visualised per episode. For both agents, the loss clearly converges. Corresponding to this convergent behaviour, it can be seen that the agent also learns to minimise the amount of nonzero actions it takes.



(a) Loss per episode.

(b) Amount of nonzero actions taken during training.

Figure C.2

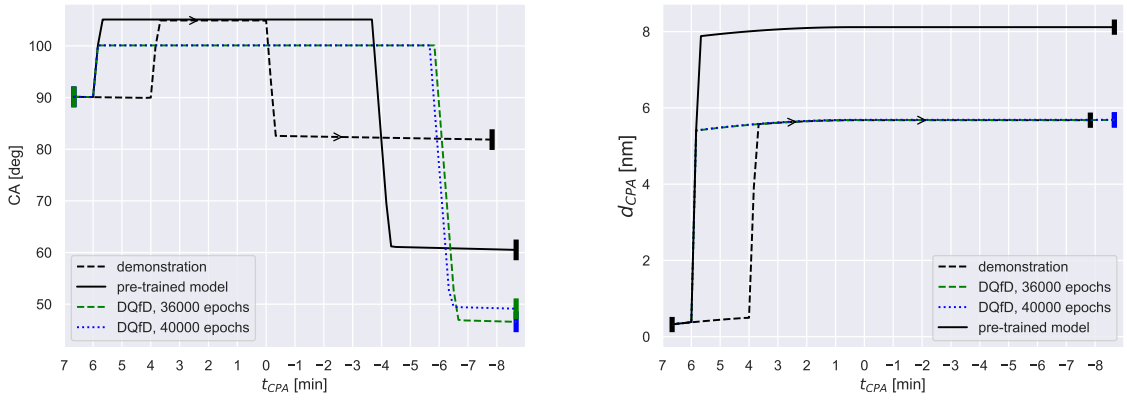
D

Additional Results Case Study 2

In this chapter, additional results for Case Study 2 are presented. First, the resolutions the DQfD agent has learned for the conflict angles 90 and 135 degrees are presented in section D.1. Then, section D.2 details learning curves that show the effect of the pre-training phase. Finally, learning curves for the normal training phase are shown in section D.3.

D.1. Resolutions Conflict Angle 90 and 135 Degrees

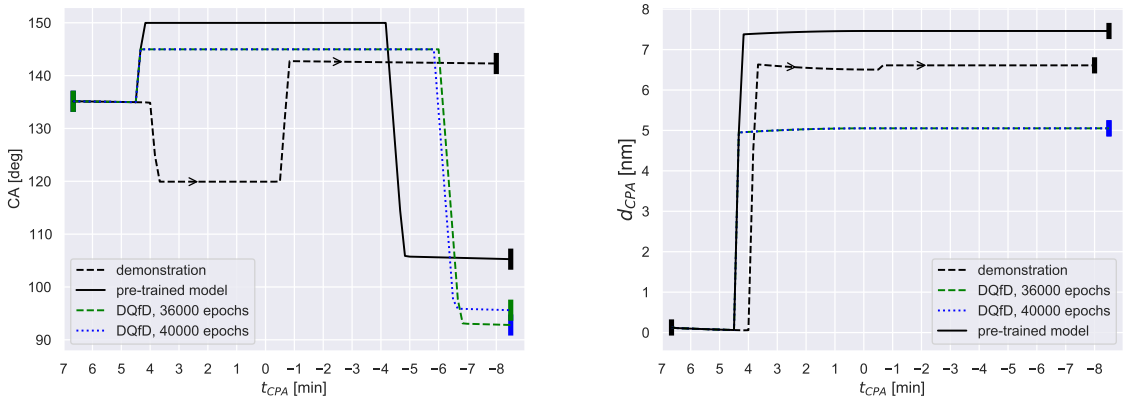
In Figure D.1, the resolution as provided by the DQfD agent is shown at a conflict angle of 90 degrees. The DQfD agent takes its first action at the same time as the pre-trained model did. Figure D.1b shows the d_{CPA} of the resolution. The pre-trained model has clearly not yet learned an efficient resolution in terms of flight path as the d_{CPA} is around 8 nm, whereas the resolution provided by the demonstration has a d_{CPA} of 5.4 nm. After the normal training phase, the DQfD agent has learned a more optimal resolution compared to the pre-trained model. It exactly matches the resolution provided by the demonstration in terms of d_{CPA}.



(a) (b)

Figure D.1: Resolution for the conflict at 90 degrees conflict angle.

Figure D.2a shows the resolution in terms of the conflict angle for the traffic scenario with an observed aircraft at a CA of 135 degrees. The pre-trained model for this conflict is not conform with the demonstration trajectory. Instead of steering the controlled aircraft in front of the observed aircraft, the pre-trained model steers the aircraft behind the observed aircraft. After an additional 40,000 epochs at an exploration rate of 0.01 the DQfD agent still steers the aircraft behind the observed aircraft. Figure D.2b shows the d_{CPA} of the resolution. It shows that the pre-trained model has not yet learned an optimal resolution. Just as for the conflict encountered at a conflict angle of 90 degrees, the fully trained DQfD agent has learned to solve the conflict at a $dcpa \approx 5$ [nm].



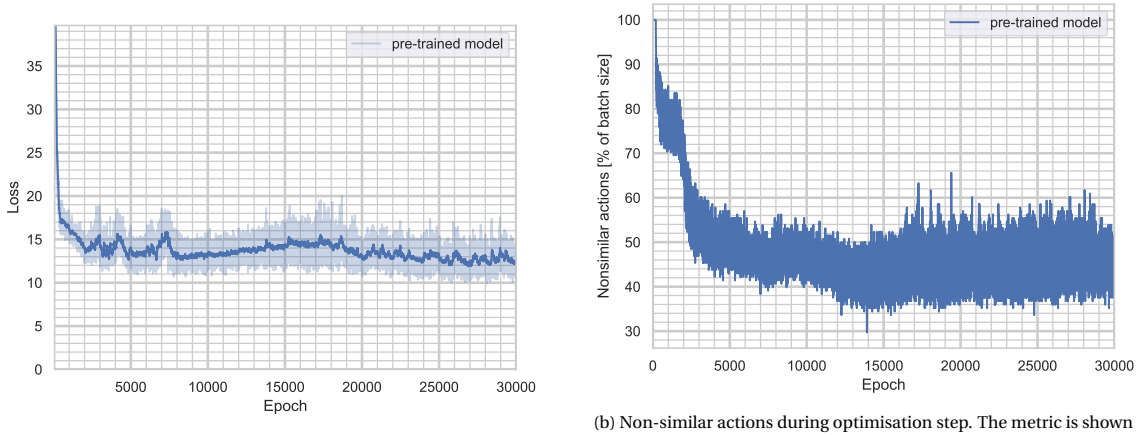
(a) (b)

Figure D.2: Resolution for the conflict at 135 degrees conflict angle.

Results in this section support the conclusion drawn in the paper that the DQfD agent optimises the reward function starting from the policy learned during the pre-train period. To develop strategic conformal automation, research should be focused on maximising the conformance between the pre-trained model and demonstrations.

D.2. Learning Curves Pre-Training Phase

To develop a thorough understanding of the algorithm, this section specifies learning curves that show how the algorithm optimises. In Figure D.3a, the loss is shown during the pre-training phase. One can see that the loss decreases significantly during the first few epochs. Figure D.3b visualises the amount of experiences sampled from the experience replay at which the greedy action does not resemble the action of the demonstrator. It can be noticed that for the first 15,000 epochs, the RL agent learns to learn how to update its parameter weights towards the policy of the demonstrator. However, after epoch 15,000, it does not seem to improve its policy w.r.t. the demonstrator anymore.



(a) Loss during training.

(b) Non-similar actions during optimisation step. The metric is shown as percentage of the batch size (128).

Figure D.3

D.3. Learning Curves of Interest Normal Training Phase

Figure D.4 visualises the ratio of demonstrations used during an optimisation step of the network. One can see that the amount of demonstrations decreases to ≈ 0.02 after 15,000 epochs in the normal training phase (epoch 45,000). It can thus be seen that the RL agent kept on learning from demonstrations, but very limited

after 15,000 epochs into the normal training phase. This causes the network to update more towards the new experiences it has acquired.

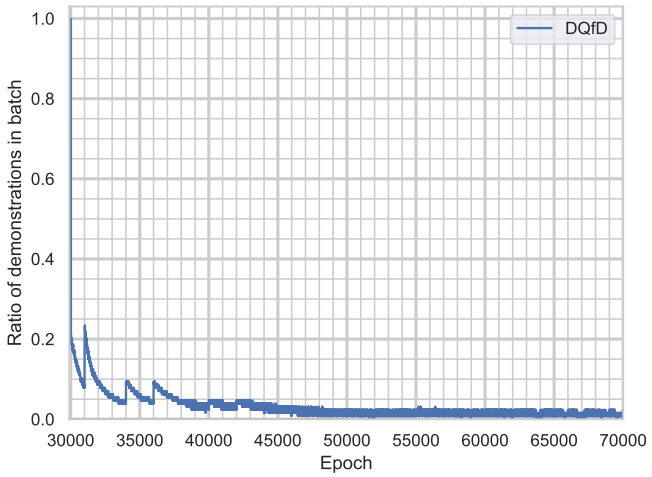


Figure D.4: Ratio of demonstrations used during a batch update.

D.4. Conclusions Drawn from Additional Results

Results support the conclusion that the DQfD algorithm improves on the demonstrations starting from the policy as learned in the pre-training phase. For the traffic scenarios at an initial conflict angle of 90 and 135 degrees, the RL agent performs its first action to avoid the conflict at the same time as the pre-trained model. However, for both resolutions, it seems to have learned how to optimise the resolution in terms of minimising the deviation from the flight path.

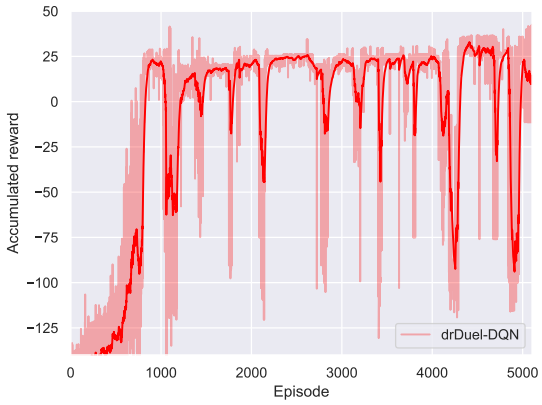
E

Additional Results Case Study 3 & 4

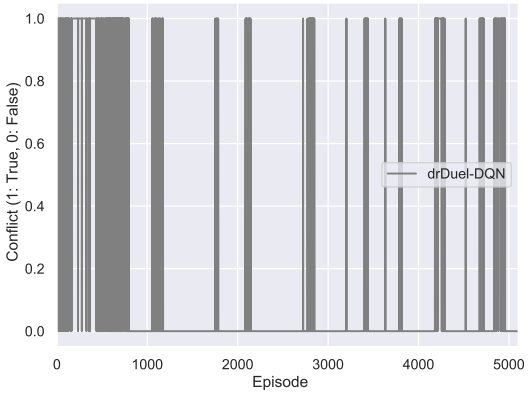
This chapter details additional results obtained for case study 3 and 4. In case study 3, a drDuel-DQN agent was trained for the conflict angles of 45, 90 and 135 degrees. It was chosen to lower the t_{CPA} and $t_{sim_{terminal}}$ to increase the amount of conflicts the agent would encounter per unit time. In section E.1, relevant training curves are visualised for case study 3. In case study 4, a RL agent was trained to avoid an observed aircraft coming from a conflict angle of 90 degrees. Relevant training results for this case study are presented in section E.2.

E.1. Training Curves Case Study 3

In Figure E.1b, the accumulated reward per episode is visualised. The figure shows that the performance of the agent remains fluctuant during training. This can also be deducted from Figure E.1b, which visualises whether the episode ended in a conflict or not. The RL agent seems to continue getting into a conflict. However, one can also note that after getting into a conflicts, the agent seems to improve its performance.



(a) Accumulated reward per episode during training.



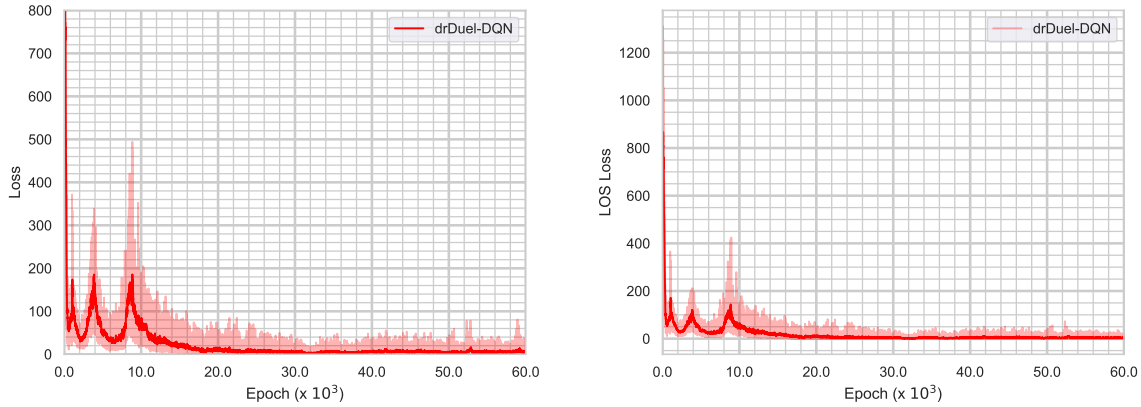
(b) Whether or not the episode during training ended in a conflict (1 = True, 0 = False).

Figure E.1

The total loss per epoch can be seen in Figure E.2a. The individual loss components can be found in Figure E.2b, Figure E.4a and Figure E.4b. Note that these various loss components each have a different scale for the loss on the y-axis.

The peaks in the total loss can be explained by examining the decomposed loss values. Showing the decomposed losses provides insights to the designer as to what components of the reward function the RL agent has been able to learn well. The reward component related to taking nonzero actions seems to be learned well by the agent (low and steadily decreasing loss), whereas the reward for following the target heading seems to

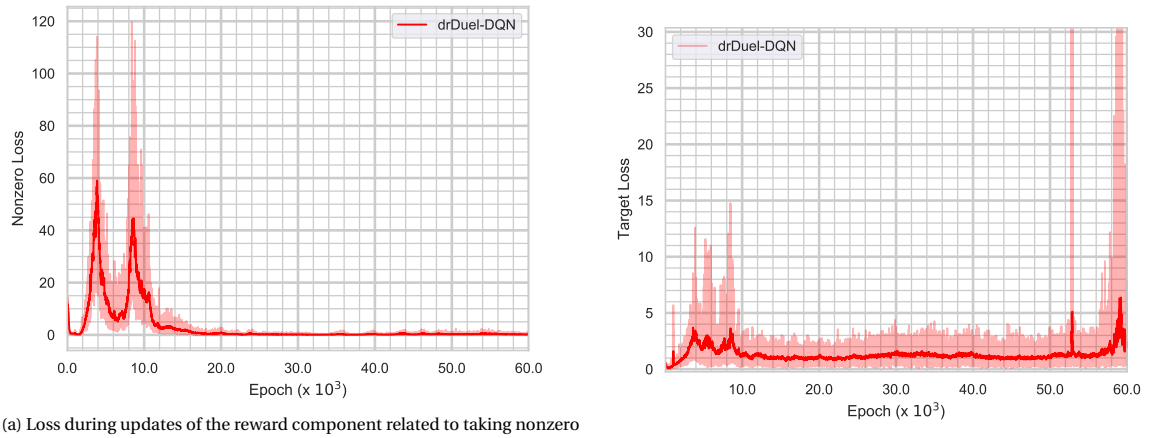
be difficult to learn. Without decomposing the reward function, these insights would not be available to the designer.



(a) Loss per epoch of training.

(b) Loss related to getting into a Loss of Separation per epoch of training.

Figure E.2



(a) Loss during updates of the reward component related to taking nonzero actions per epoch of training.

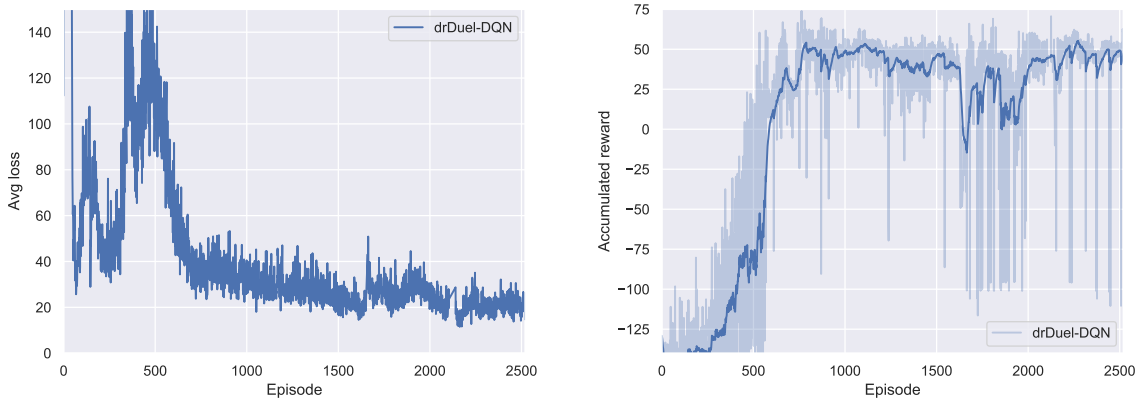
(b) Loss related to the deviation of the target heading per epoch of training.

Figure E.3

The purpose of case study 3 was to show how decomposed rewards can be implemented and show how it can be utilised to increase the explainability of the algorithm. Due to a lack of computational resources and time constraints, the RL agent was not trained up to full convergence. The policy at epoch 60,000 showed to be able to avoid all conflicts from the training set. Therefore, it was chosen to use that policy to perform the experiment for case study 3.

E.2. Training Curves Case Study 4

For case study 4, a RL agent was trained to avoid a conflict for an observed aircraft at a conflict angle of 90 degrees. The agent was trained for 30,000 epochs. As the goal of this case study is not to avoid a conflict by minimising the flight path, the agent used for this case study was not trained until it showed convergent behaviour. As the goal was to research whether the agent could learn decomposed values, the policy at epoch 30,000 was used to research the decomposed values throughout the trajectory.



(a) Average loss per episode during updates.

(b) Accumulated reward during training.

Figure E.4

The conflicts encountered during the training phase by the drDuel-DQN agent are shown in Figure E.5. The agent has has learned to steadily avoid all conflicts between episode 600 and 1600. After episode 1600, it starts to experience conflicts again.

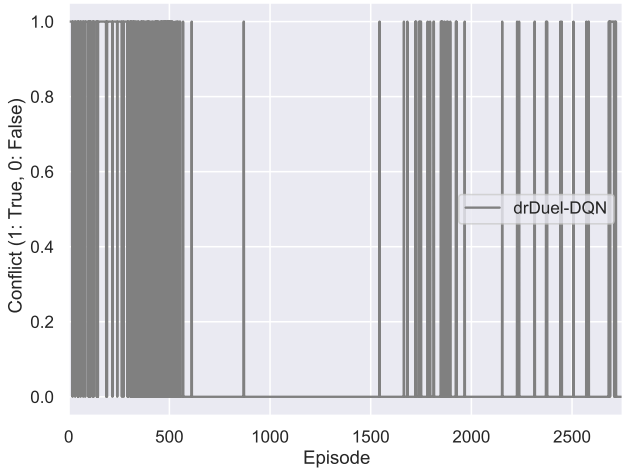


Figure E.5: Conflicts encountered during the learning phase.

Bibliography

Andre, David, Nir Friedman, and Ronald Parr (1998). "Generalized Prioritized Sweeping". In: *Advances in Neural Information Processing Systems*. Ed. by M. Jordan, M. Kearns, and S. Solla. Vol. 10. MIT Press. URL: https://proceedings.neurips.cc/paper/1997/file/7b5b23f4aadf9513306bcd59afb6e4c9-Paper.pdf.

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath (Nov. 2017). *Deep reinforcement learning: A brief survey*. DOI: 10.1109/MSP.2017.2743240.

Arulkumaran, Kai, Nat Dilokthanakul, Murray Shanahan, and Anil Anthony Bharath (Apr. 2016). "Classifying Options for Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1604.08153.

Association, International Civil Aviation (2016). *Doc 4444: Procedures for Air Traffic Management*. 16th, pp. 79–79. ISBN: 978-92-9258-081-0. URL: www.icao.int.

Avigad, J. and K. Donnelly (2004). "Formalizing O Notation in Isabelle/HOL". In: *International Joint Conference on all aspects of Automated Reasoning*.

Bach, Sebastian, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus Robert Müller, and Wojciech Samek (July 2015). "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation". In: *PLoS ONE* 10.7. ISSN: 19326203. DOI: 10.1371/journal.pone.0130140.

Bekier, Marek, Brett R.C. Molesworth, and Ann Williamson (Feb. 2012). "Tipping point: The narrow path between automation acceptance and rejection in air traffic management". In: *Safety Science* 50.2, pp. 259–265. ISSN: 09257535. DOI: 10.1016/j.ssci.2011.08.059.

BISSERET, A. (Sept. 1971). "Analysis of Mental Processes Involved in Air Traffic Control". In: *Ergonomics* 14.5, pp. 565–570. ISSN: 0014-0139. DOI: 10.1080/00140137108931276. URL: https://www.tandfonline.com/doi/full/10.1080/00140137108931276.

Borst, Clark, Vincent A. Bijsterbosch, M. M. van Paassen, and Max Mulder (Nov. 2017). "Ecological interface design: supporting fault diagnosis of automated advice in a supervisory air traffic control task". In: *Cognition, Technology and Work* 19.4, pp. 545–560. ISSN: 14355566. DOI: 10.1007/s10111-017-0438-y.

Botvinick, Matthew, Sam Ritter, Jane X. Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis (May 2019). *Reinforcement Learning, Fast and Slow*. DOI: 10.1016/j.tics.2019.02.006.

Bowden, Vanessa, Luke Ren, and Shayne Loft (Sept. 2018). "Supervising High Degree Automation in Simulated Air Traffic Control". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 62.1, pp. 86–86. ISSN: 1541-9312. DOI: 10.1177/1541931218621019. URL: http://journals.sagepub.com/doi/10.1177/1541931218621019.

Brittain, Marc and Peng Wei (2018). "Autonomous Aircraft Sequencing and Separation with Hierarchical Deep Reinforcement Learning". In: *International Conference for Research in Air Transportation*.

— (2019). "Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach". In: *Computing Resources Repository* abs/1905.01303. URL: http://arxiv.org/abs/1905.01303.

Brittain, Marc, Xuxi Yang, and Peng Wei (Mar. 2020). "A Deep Multi-Agent Reinforcement Learning Approach to Autonomous Separation Assurance". In: URL: http://arxiv.org/abs/2003.08353.

Buduma, Nikhil (2017). *Fundamentals of Deep Learning*. 1st. Sebastopol: O'Reilly Media, p. 105.

Bus¸oniu, L Bus¸oniu, R Babuška, B De Schutter, Lucian Bus¸oniu Bus¸oniu, Robert Babuška, and Bart De Schutter (2008). "A comprehensive survey of multi-agent reinforcement learning". In: 38.2, pp. 156–172.

Bus¸oniu, Lucian Bus¸oniu, Robert Babuška, Bart De Schutter, and Damien Ernst (Nov. 2019). *Reinforcement learning and dynamic programming using function approximators*.

Cireşan, Dan, Ueli Meier, and Juergen Schmidhuber (Feb. 2012). "Multi-column Deep Neural Networks for Image Classification". In: URL: http://arxiv.org/abs/1202.2745.

Dayan, Peter and Geoffrey E Hinton (1993). "Feudal Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Hanson, J. Cowan, and C. Giles. Vol. 5. Morgan-Kaufmann. URL: https://proceedings.neurips.cc/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

De Prins, Johan, Ramon Gomez Ledesma, Max Mulder, and M.M. van Paassen (2008). "Literature review of air traffic controller modeling for traffic simulations". In: *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pp. 2.C.3-1-2.C.3–11. DOI: 10.1109/DASC.2008.4702782.

Dekker, Sidney and David Woods (Sept. 1999). "To Intervene or not to Intervene: The Dilemma of Management by Exception". In: *Cognition, Technology Work* 1, pp. 86–96. DOI: 10.1007/s101110050035.

Djokic, Jelena, Bernd Lorenz, and Hartmut Fricke (2010). "Air traffic control complexity as workload driver". In: *Transportation Research Part C: Emerging Technologies* 18.6, pp. 930–936. ISSN: 0968090X. DOI: 10.1016/j.trc.2010.03.005.

Duchi, John, Elad Hazan, and Yoram Singer (July 2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null, pp. 2121–2159. ISSN: 1532-4435.

EUROCONTROL (1996). *Model for Task and Job Descriptions of Air Traffic Controllers. European Air Traffic Control Harmonisation and Integration Programme*. Tech. rep. ECAC.

European Commission (2009). *European Air Traffic Management Master Plan*. Tech. rep.

Fothergill, S and A Neal (2008). "The Effect of Workload on Conflict Decision Making Strategies in Air Traffic Control". In: DOI: 10.1518/107118108X353101.

Francois-Lavet, Vincent, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau (Nov. 2018). "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3-4, pp. 219–354. DOI: 10.1561/2200000071. URL: http://arxiv.org/abs/1811.12560%20http://dx.doi.org/10.1561/2200000071.

Frans, Kevin, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman (Oct. 2017). "Meta Learning Shared Hierarchies". In: URL: http://arxiv.org/abs/1710.09767.

Fu, Justin, Katie Luo, and Sergey Levine (Oct. 2017). "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: URL: http://arxiv.org/abs/1710.11248.

Garot, Jean-Marc and Nicolas Durand (Nov. 2005). "Failures in the automation of air traffic control ". In: *Colloque de l'AAA 2005*. Toulouse, France. URL: https://hal-enac.archives-ouvertes.fr/hal-01291420.

Glorot, Xavier and Yoshua Bengio (13–15 May 2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. The MIT Press. ISBN: 0262035618.

Graves, Alex (2011). "Practical Variational Inference for Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger. Vol. 24. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e%208ebfab08eba5f49632-Paper.pdf.

Gruslys, Audrunas, Will Dabney, Mohammad Gheshlaghi Azar, Bilal Piot, Marc Bellemare, and Remi Munos (Apr. 2017). "The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning". In: URL: http://arxiv.org/abs/1704.04651.

Gunning, David (2017). *Explainable Artificial Intelligence (XAI)*. Tech. rep. URL: https://www.cc.gatech.edu/~alanwags/DLAI2016/(Gunning)%20IJCAI-16%20DLAI%20WS.pdf.

Gurtner, G., C. Bongiorno, M. Ducci, and S. Miccichè (Mar. 2017). "An empirically grounded agent based simulator for the air traffic management in the SESAR scenario". In: *Journal of Air Transport Management* 59, pp. 26–43. ISSN: 09696997. DOI: 10.1016/j.jairtraman.2016.11.004.

Hasselt, Hado van, Arthur Guez, and David Silver (Sept. 2015). "Deep Reinforcement Learning with Double Q-learning". In: URL: http://arxiv.org/abs/1509.06461.

Helbing, Helge (Sept. 1997). "A Cognitive Model of En-Route Air Traffic Control". In: *IFAC Proceedings Volumes* 30.24, pp. 57–60. ISSN: 14746670. DOI: 10.1016/S1474-6670(17)42222-1. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474667017422221.

Henderson, Peter, Joshua Romoff, and Joelle Pineau (2018). "Where Did My Optimum Go?: An Empirical Analysis of Gradient Descent Optimization in Policy Gradient Methods". In: *European Workshop on Reinforcement Learning*. Vol. 14.

Hermes, P., M. Mulder, M. M. van Paassen, J. H. L. Boering, and H. Huisman (Nov. 2009). "Solution-Space-Based Complexity Analysis of the Difficulty of Aircraft Merging Tasks". In: *Journal of Aircraft* 46.6, pp. 1995–2015. ISSN: 0021-8669. DOI: 10.2514/1.42886. URL: https://arc.aiaa.org/doi/10.2514/1.42886.

Hernandez-Leal, Pablo, Bilal Kartal, and Matthew E. Taylor (Nov. 2019). "A survey and critique of multiagent deep reinforcement learning". In: *Autonomous Agents and Multi-Agent Systems* 33.6, pp. 750–797. ISSN: 15737454. DOI: 10.1007/s10458-019-09421-1.

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver (Oct. 2017). "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1710.02298.

Hester, Todd, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys (Apr. 2017). "Deep Q-learning from Demonstrations". In: URL: http://arxiv.org/abs/1704.03732.

Hilburn, Brian (Jan. 2004). "Cognitive complexity in air traffic control: a literature review". In:

Hilburn, Brian, Carl Westin, and Clark Borst (Apr. 2014). "Will Controllers Accept a Machine That Thinks like They Think? The Role of Strategic Conformance in Decision Aiding Automation". In: *Air Traffic Control Quarterly* 22.2, pp. 115–136. ISSN: 1064-3818. DOI: 10.2514/atcq.22.2.115.

Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov (July 2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: URL: http://arxiv.org/abs/1207.0580.

Hoekstra, J and J Ellerbroek (2016). "BlueSky ATC Simulator Project: An Open Data and Open Source Approach". In: *7th International Conference for Research in Air Transportation (ICRAT)*. ICRAT.

Hoekstra, J.M, R.N.H.W van Gent, and R.C.J Ruigrok (2002). "Designing for safety: the 'free flight' air traffic management concept". In: *Reliability Engineering System Safety* 75.2, pp. 215–232. ISSN: 0951-8320. DOI: https://doi.org/10.1016/S0951-8320(01)00096-5. URL: https://www.sciencedirect.com/science/article/pii/S0951832001000965.

Hoff, Dennis van der (2020). "A Multi-Agent Reinforcement Learning Approach to Air Traffic Control". MA thesis. Technical University of Delft. URL: http://repository.tudelft.nl/.

Horgan, Dan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver (Mar. 2018). "Distributed Prioritized Experience Replay". In: URL: http://arxiv.org/abs/1803.00933.

Huo, Ying, Daniel Delahaye, and Yanjun Wang (June 2018). "Sensitivity Analysis of Closest Point of Approach". In: *ICRAT 2018, 8th International Conference for Research in Air Transportation*. Barcelone, Spain. URL: https://hal-enac.archives-ouvertes.fr/hal-01823194.

Jaakkola, Tommi, Satinder Singh, and Michael Jordan (Nov. 1999). "Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems". In: *Advances in Neural Information Processing Systems* 7.

Juozapaitis, Zoe, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez (2019). "Explainable Reinforcement Learning via Reward Decomposition". In: International Joint Conference on Artificial Intelligence.

Kakade, S and J Langford (2002). *Approximately Optimal Approximate Reinforcement Learning*. Vol. 2. ICML, pp. 267–274.

Kallus, K, D Van Damme, and A Dittmann (1999). *Integrated Task and Job Analysis of Air Traffic Controllers - Phase 2: Task Analysis of En-route Controllers*. Eurocontrol, pp. 1–98.

Kingma, Diederik P. and Jimmy Ba (Dec. 2014). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference for Learning Representations*. URL: http://arxiv.org/abs/1412.6980.

Kirwan, B and M Flynn (2002). *Investigating Air Traffic Controller Conflict Resolution Strategies*. Tech. rep. EUROCONTROL.

Kulkarni, Tejas D, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum (2016). "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2016/file/f442d33fa06832%20082290ad8544a8da27-Paper.pdf.

Law, David J., James W. Pellegrino, Steve R. Mitchell, Susan C. Fischer, Thomas P. McDonald, and Earl B. Hunt (1993). "Perceptual and cognitive factors governing performance in comparative arrival-time judgments." In: *Journal of Experimental Psychology: Human Perception and Performance* 19.6, pp. 1183–1199. ISSN: 1939-1277. DOI: 10.1037/0096-1523.19.6.1183. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/0096-1523.19.6.1183.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. Vol. 86. 11, pp. 2278–2324. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665.

Madumal, Prashan, Tim Miller, Liz Sonenberg, and Frank Vetere (Apr. 2020). "Explainable Reinforcement Learning through a Causal Lens". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03,

pp. 2493–2500. DOI: 10.1609/aaai.v34i03.5631. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5631.

Matignon, Laetitia, Guillaume J. Laurent, and Nadíne Le Fort-Piat (2006). "Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 4131 LNCS - I. Springer Verlag, pp. 840–849. ISBN: 3540386254. DOI: 10.1007/11840817{_}87.

Mercado Velasco, GA, M Mulder, and MM van Paassen (2010). "Air Traffic Controller Decision-Making Support using the Solution Space Diagram". English. In: *Proceedings of the 11th IFAC/IFIP/IFORS/IEA Symposium on Design, Analysis and Evaluation of Human-Machine Systems (IFAC-HMS 2010)*. Ed. by W Chul Yoon. the 11th IFAC/IFIP/IFORS/IEA Symposium on Design, Analysis and Evaluation of Human-Machine Systems (IFAC-HMS 2010) ; Conference date: 31-08-2010 Through 03-09-2010. IFAC, pp. 1–6.

Mercado Velasco, Gustavo Adrian, Clark Borst, Joost Ellerbroek, M. M. Van Paassen, and Max Mulder (Aug. 2015). "The Use of Intent Information in Conflict Detection and Resolution Models Based on Dynamic Velocity Obstacles". In: *IEEE Transactions on Intelligent Transportation Systems* 16.4, pp. 2297–2302. ISSN: 15249050. DOI: 10.1109/TITS.2014.2376031.

Metzger, Ulla and Raja Parasuraman (2006). "Effects of automated conflict cuing and traffic density on air traffic controller performance and visual attention in a datalink environment". In: *International Journal of Aviation Psychology* 16.4, pp. 343–362. ISSN: 10508414. DOI: 10.1207/s15327108ijap1604{_}1.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (Feb. 2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533. ISSN: 14764687. DOI: 10.1038/nature14236.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P Lillicrap, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*.

Navarro, Cristóbal A., Nancy Hitschfeld-Kahler, and Luis Mateu (2014). "A Survey on Parallel Computing and Its Applications in Data-Parallel Problems Using GPU Architectures". In: *Communications in Computational Physics* 15.2, pp. 285–329. ISSN: 1991-7120. DOI: https://doi.org/10.4208/cicp.110113.010813a. URL: http://global-sci.org/intro/article_detail/cicp/7096.html.

Nguyen, Vu, Sebastian Schulze, and Michael A Osborne (Sept. 2019). "Bayesian Optimization for Iterative Learning". In: URL: http://arxiv.org/abs/1909.09593.

Nolan, M. (2010). *Fundamentals of Air Traffic Control*. Cengage Learning. ISBN: 9781435482722. URL: https://books.google.nl/books?id=6yhTiGC3ulcC.

Omidshafiei, Shayegan, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian (Mar. 2017). "Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability". In: URL: http://arxiv.org/abs/1703.06182.

Pack Kaelbling, Leslie, Michael L Littman, Andrew W Moore, and Smith Hall (1996). "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4, pp. 237–285.

Parasuraman, R., T.B. Sheridan, and C.D. Wickens (2000). "A model for types and levels of human interaction with automation". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 30.3, pp. 286–297. DOI: 10.1109/3468.844354.

Parasuraman, Raja, Robert Molloy, and Indramani L. Singh (Jan. 1993). "Performance Consequences of Automation-Induced "Complacency"". In: *The International Journal of Aviation Psychology* 3.1, pp. 1–23. ISSN: 15327108. DOI: 10.1207/s15327108ijap0301{_}1.

Pardo, Fabio, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev (Dec. 2017). "Time Limits in Reinforcement Learning". In: URL: http://arxiv.org/abs/1712.00378.

Parr, Ronald and Stuart Russell (1998). "Reinforcement Learning with Hierarchies of Machines". In: *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*. NIPS '97. Denver, Colorado, USA: MIT Press, pp. 1043–1049. ISBN: 0262100762.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (Dec. 2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: URL: http://arxiv.org/abs/1912.01703.

Pham, Duc Thinh, Ngoc Phu Tran, Sim Kuan Goh, Sameer Alam, and Vu Duong (May 2019). "Reinforcement learning for two-aircraft conflict resolution in the presence of uncertainty". In: *RIVF 2019 - Proceedings: 2019 IEEE-RIVF International Conference on Computing and Communication Technologies*. Institute of Electrical and Electronics Engineers Inc. ISBN: 9781538693131. DOI: 10.1109/RIVF.2019.8713624.

Piot, Bilal, Matthieu Geist, and Olivier Pietquin (2014). "Boosted Bellman Residual Minimization Handling Expert Demonstrations". In: *Proceedings of the 2014th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*. ECMLPKDD'14. Nancy, France: Springer-Verlag, pp. 549–564. ISBN: 9783662448502. DOI: 10.1007/978-3-662-44851-9_35. URL: https://doi.org/10.1007/978-3-662-44851-9_35.

Pohlen, Tobias, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado van Hasselt, John Quan, Mel Večerík, Matteo Hessel, Rémi Munos, and Olivier Pietquin (May 2018). "Observe and Look Further: Achieving Consistent Performance on Atari". In: URL: http://arxiv.org/abs/1805.11593.

Puca, Riccardo, Erik-Jan Van Kampen, Clark Borst, Maarten Tielrooij, and Qi Ping Chu (2014). "Experience-based AI methods for ATC decision-making support". MA thesis. Delft University of Technology.

Rantanen, Esa M and Ashley Nunes (2005). "Hierarchical Conflict Detection in Air Traffic Control". In: *International Journal of Aviation Psychology* 15.4, pp. 339–362. DOI: 10.1207/s15327108ijap1504{_}3.

Rantanen, Esa M. and Christopher D. Wickens (July 2012). "Conflict Resolution Maneuvers in Air Traffic Control: Investigation of Operational Data". In: *International Journal of Aviation Psychology* 22.3, pp. 266–281. ISSN: 10508414. DOI: 10.1080/10508414.2012.691048.

Regtuit, Robert M., Clark Borst, Erik-Jan van Kampen, and M. M. van Paassen (2018). "Building strategic conformal automation for air traffic control using machine learning". In: *AIAA Information Systems-AIAA Infotech at Aerospace, 2018*. 209989. American Institute of Aeronautics and Astronautics Inc, AIAA. ISBN: 9781624105272. DOI: 10.2514/6.2018-0074.

Ribeiro, Marta, Joost Ellerbroek, and Jacco Hoekstra (June 2020). "Review of conflict resolution methods for manned and unmanned aviation". In: *Aerospace* 7.6. ISSN: 22264310. DOI: 10.3390/AEROSPACE7060079.

Ruder, Sebastian (Sept. 2017). "An overview of gradient descent optimization algorithms". In: URL: http://arxiv.org/abs/1609.04747.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2016). "Prioritized Experience Replay". In: *International Conference on Learning Representations*. Puerto Rico.

Schaul, Tom, John Quan, Ioannis Antonoglou, David Silver, and Google Deepmind (2016). "PRIORITIZED EXPERIENCE REPLAY". In: International Conference on Learning Representations (ICLR). ISBN: 1511.05952v4.

Schulman, John, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel (Feb. 2015). "Trust Region Policy Optimization". In: URL: http://arxiv.org/abs/1502.05477.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (July 2017). "Proximal Policy Optimization Algorithms". In: URL: http://arxiv.org/abs/1707.06347.

Seamster, Thomas L., Richard E. Redding, John R. annon, Joan M. Ryder, and Janine A. Purcell (1993). "Cognitive Task Analysis of Expertise in Air Traffic Control". In: *The International Journal of Aviation Psychology* 3.4, pp. 257–283. ISSN: 15327108. DOI: 10.1207/s15327108ijap0304{_}2.

Shao, Kun, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao (Dec. 2019). "A Survey of Deep Reinforcement Learning in Video Games". In: URL: http://arxiv.org/abs/1912.10944.

Simonyan, Karen and Andrew Zisserman (Sept. 2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: URL: http://arxiv.org/abs/1409.1556.

Sun, Junzi, Joost Ellerbroek, Jacco Hoekstra, and Jacco M Hoekstra (2020). "OpenAP: An open-source aircraft performance model for air transportation studies and simulationstransportation studies and simulations". In: *Aerospace* 7.8. DOI: https://doi.org/10.3390/AEROSPACE7080104.

Sutton, R and A Barto (2018). *Reinforcement learning: an introduction*. Second Edition. Cambridge: The MIT Press, p. 225.

Sutton, R.S. and A.G. Barto (1998). *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press.

Sutton, Richard S. (July 1991). "Dyna, an Integrated Architecture for Learning, Planning, and Reacting". In: *SIGART Bull.* 2.4, pp. 160–163. ISSN: 0163-5719. DOI: 10.1145/122344.122377. URL: https://doi.org/10.1145/122344.122377.

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement learning : an introduction*. Second Edition. Cambridge: The MIT Press, p. 526. ISBN: 9780262039246.

Sutton, Richard S., Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial Intelligence* 112.1, pp. 181–211. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(99)00052-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Tavakoli, Arash, Fabio Pardo, and Petar Kormushev (Nov. 2017). "Action Branching Architectures for Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1711.08946.

Tjoa, Erico and Cuntai Guan (July 2019). "A Survey on Explainable Artificial Intelligence (XAI): Towards Medical XAI". In: URL: http://arxiv.org/abs/1907.07374.

Tran, Ngoc Phu, Duc Thinh Pham, Sim Kuan Goh, Sameer Alam, and Vu Duong (Apr. 2019). "An intelligent interactive conflict solver incorporating air traffic controllers' preferences using reinforcement learning". In: *Integrated Communications, Navigation and Surveillance Conference, ICNS*. Vol. 2019-April. Institute of Electrical and Electronics Engineers Inc. ISBN: 9781728118932. DOI: 10.1109/ICNSURV.2019.8735168.

Van Dam, S.B.J., An.L.M. Abeloos, M. Mulder, and M.M. van Paassen (2004). "Functional presentation of travel opportunities in flexible use airspace: an EID of an airborne conflict support tool". In: *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*. Vol. 1, 802–808 vol.1. DOI: 10.1109/ICSMC.2004.1398401.

Van Meeuwen, Ludo W., Halszka Jarodzka, Saskia Brand-Gruwel, Paul A. Kirschner, Jeano J.P.R. de Bock, and Jeroen J.G. van Merriënboer (Aug. 2014). "Identification of effective visual problem solving strategies in a complex visual domain". In: *Learning and Instruction* 32, pp. 10–21. ISSN: 09594752. DOI: 10.1016/j.learninstruc.2014.01.004.

Van Rooijen, S J (2019). "Personalized Automation for Air Traffc Control using Convolutional Neural Networks". MA thesis. Technical University of Delft. URL: http://repository.tudelft.nl/.

Vezhnevets, Alexander Sasha, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu (Mar. 2017). "FeUdal Networks for Hierarchical Reinforcement Learning". In: URL: http://arxiv.org/abs/1703.01161.

Volz, Katherine, Euijung Yang, Rachel Dudley, Elizabeth Lynch, Maria Dropps, and Michael C. Dorneich (2016). "An evaluation of cognitive skill degradation in information automation". In: *Proceedings of the Human Factors and Ergonomics Society*. Human Factors an Ergonomics Society Inc., pp. 191–195. DOI: 10.1177/1541931213601043.

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas (Nov. 2016). "Dueling Network Architectures for Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1511.06581.

Westin, Carl, Clark Borst, and Brian Hilburn (Feb. 2015). "Strategic Conformance: Overcoming Acceptance Issues of Decision Aiding Automation?" In: *IEEE Transactions on Human-Machine Systems* 46.1, pp. 41–52. ISSN: 21682291. DOI: 10.1109/THMS.2015.2482480.

Wickens, Christopher D., John Hellenberg, and Xidong Xu (June 2002). "Pilot Maneuver Choice and Workload in Free Flight". In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 44.2, pp. 171–188. ISSN: 0018-7208. DOI: 10.1518/0018720024497943. URL: http://journals.sagepub.com/doi/10.1518/0018720024497943.

Wickens, Christopher D., C. D. Mavor, and R. Parasuraman (Oct. 1998). "The Future of Air Traffic Control: Human Operators and Automation". In: *Ergonomics in Design: The Quarterly of Human Factors Applications* 6.4, pp. 18–22. ISSN: 1064-8046. DOI: 10.1177/106480469800600407. URL: http://journals.sagepub.com/doi/10.1177/106480469800600407.

Xu, Min, Jeanne M. David, and Suk Hi Kim (Apr. 2018). "The fourth industrial revolution: Opportunities and challenges". In: *International Journal of Financial Research* 9.2, pp. 90–95. ISSN: 19234031. DOI: 10.5430/ijfr.v9n2p90.

Xu, Xidong and Esa M. Rantanen (June 2007). "Effects of air traffic geometry on pilots' conflict detection with cockpit display of traffic information". In: *Human Factors* 49.3, pp. 358–375. ISSN: 00187208. DOI: 10.1518/001872007X197008.

Zhang, Chiyuan, Oriol Vinyals, Remi Munos, and Samy Bengio (Apr. 2018). "A Study on Overfitting in Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1804.06893.