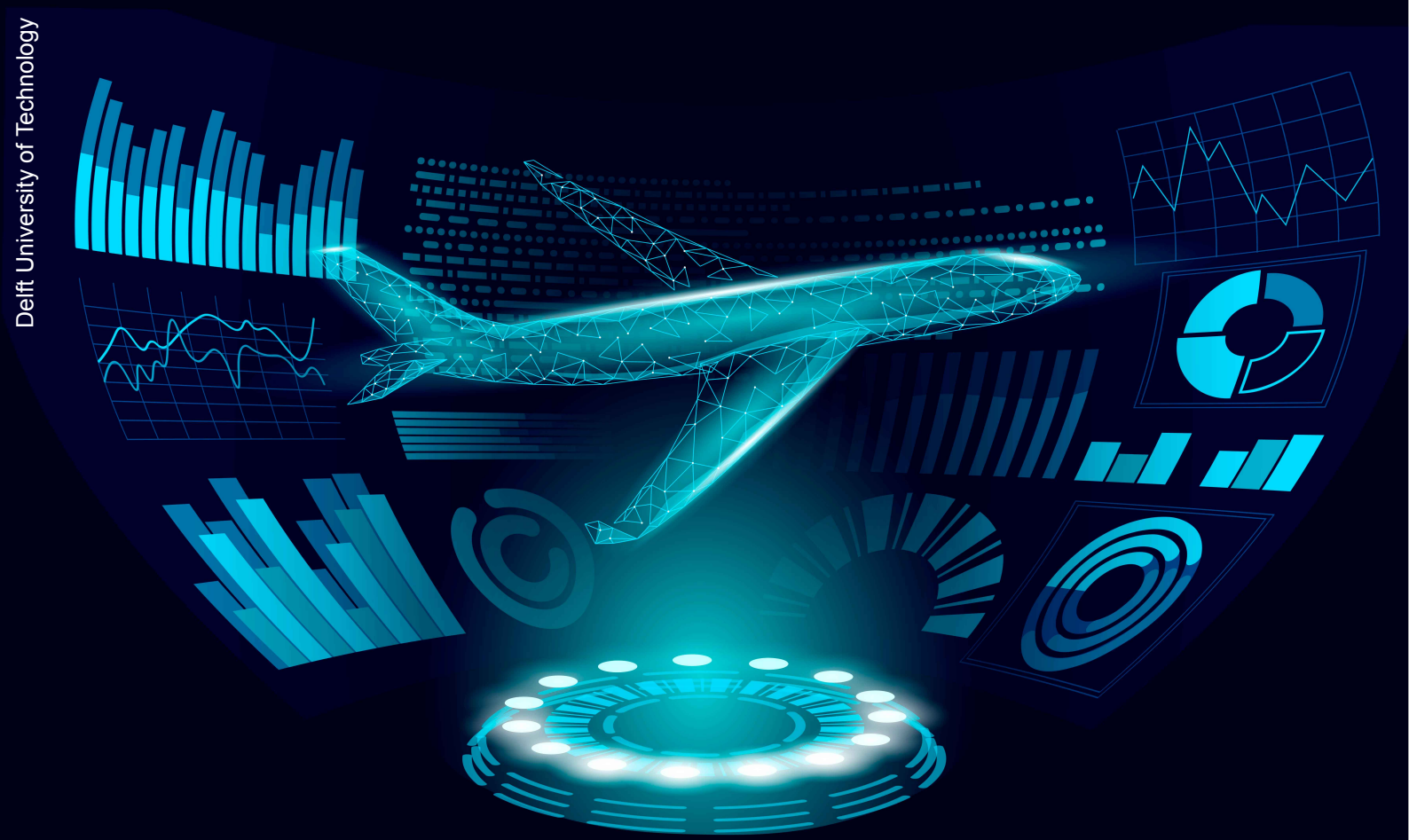
Explainable Reinforcement Learning in Flight Control through Reward Decomposition

MSc Thesis Report

André Lemos



Delft University of Technology

TU Delft

Explainable Reinforcement Learning in Flight Control through Reward Decomposition

by

André Lemos

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on September 23, 2022 at 14:00.

Student number:	5144078	
Project duration:	March 1, 2021 – September 23, 2022	
Thesis committee:	Dr. ir. Coen de Visser,	TU Delft, chair
	Dr. ir. E. van Kampen,	TU Delft, supervisor
	ir. Marc Naeije,	TU Delft, external examiner

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

Cover Image: Adapted from LuckyStep (2020).

TU Delft

Preface

This thesis report concludes my aerospace engineering studies at TU Delft. My journey as a TU Delft student was marked by several people to whom I would like to express my gratitude. First, I would like to thank Dr. ir. Erik-Jan van Kampen for his supervision and for always providing insightful feedback when it was most needed, his support had a great influence on the outcome of this research. I would also like to thank all my friends in Delft for being by my side during the recent global pandemic, the many laughs and calls I had with them were vital for me during that strange time. A special thanks to all my friends back in Portugal which made working from home enjoyable and exciting. To Francisca, I would like to express my gratitude for all the support and encouragement, you were essential during the harsh times I faced during my thesis project and you made the good times even better.

This is the closure of a great chapter in my life, marking the end of my time as a full-time student. I would like to thank my parents for their continuous support, even when my ambitions required them to see their son leave his native country and put a higher financial load on their shoulders. I could have never become the person I am without you, and for that I am eternally grateful! To my sister, thank you for being so understanding and for helping me with uplifting words and gestures.

It is a farewell to TU Delft, but it is also the beginning of a new chapter in my life. I am very grateful to you all and cannot wait to see what the next chapter reserves.

André Lemos
Delft, August 2022

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ABS	Adaptive Backstepping
AFCS	Automatic Flight Control System
AI	Artificial Intelligence
ANN	Artificial Neural Network
ATM	Air Traffic Management
CitAST	Citation Analysis and Simulation Toolkit
CNN	Convolutional Neural Network
CUT	Continuous U-Tree
DASMAT	Delft University Aircraft Simulation Model and Analysis Tool
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DoF	Degrees of Freedom
DP	Dynamic Programming
DP	Decision Point
DPG	Deterministic Policy Gradient
DQfD	Deep Q-Learning from Demonstrations
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DRX	Dominant Reward eXplanation
drDQN	decomposed reward DQN
drDSARSA	decomposed reward Deep SARSA
drQ	decomposed reward Q-Learning
drSARSA	decomposed reward SARSA
HER	Hierarchical Experience Replay
HIGhER	Hindsight Generation for Experience Replay
IATA	International Air Transport Association
IDHP	Incremental Dual Heuristic Programming
INDI	Incremental Nonlinear Dynamic Inversion

KL	Kullback-Leibler
LMUT	Linear Model U-Tree
LOC-I	Loss of Control - In Flight
LTI	Linear Time Invariant
MC	Monte-Carlo
MDP	Markov Decision Process
ML	Machine Learning
MSE	Mean Squared Error
MSX	Minimal Sufficient eXplanation
MSX$^+$	Positive Minimal Sufficient eXplanation
MSX$^-$	Negative Minimal Sufficient eXplanation
NDI	Nonlinear Dynamic Inversion
NN	Neural Network
PPO	Proximal Policy Optimisation
RDX	Reward Difference eXplanation
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RMSProp	Root Mean Squared Propagation
RNN	Recurrent Neural Network
RTS	Real-Time Strategy
RK4	Runge-Kutta Fourth-Order Method
SAC	Soft Actor-Critic
SARSA	State-Action-Reward-State-Action
SGD	Stochastic Gradient Descent
SHAP	Shapley Additive Explanations
TD	Temporal Difference
TD3	Twin-Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimisation
UAV	Unmanned Aerial Vehicle

XAI eXplainable Artificial Intelligence

XRL eXplainable Reinforcement Learning

1 Introduction

1.1. Research Background

Current Automatic Flight Control Systems (AFCSs) typically consist of controllers that rely on look-up tables for gain-scheduling (Stevens et al., 2015). These tables are built by using linearized models of the aircraft dynamics at specific operating points, which presents multiple disadvantages. First, this approach requires many operating points to have proper performance and, if not carefully planned, leaves regions of the flight envelope unchecked. Second, since it relies on knowing the aircraft dynamics, it is unable to deal with sudden changes to the model (such as failures).

Consequently, there is an urge to update these controllers with ones that are robust in all regions of the flight envelope and can tolerate sudden changes to the aircraft model. Reports show that over the 2015-2019 period, Loss of Control - In Flight (LOC-I) accidents alone represented 51% of the total number of fatalities in aviation (IATA, 2019). LOC-I occurs when the aircraft enters a flight regime that is outside its normal flight envelope, usually (but not always) at a high rate, thereby introducing an element of surprise for the flight crew involved. According to Grondman et al. (2018), when such changes occur, ”the control system many times reverts back to reversionary modes or even direct control”, causing the control law to be heavily reduced or even abandoned. In this scenario, the pilot workload substantially increases to deal with the change in dynamics and try to restore aircraft control. In the 2015-2019 period more than 85% of LOC-I accidents resulted in hull losses and fatalities (IATA, 2019).

To address this issue, adaptive flight control has found more and more applications by developing controllers that are able to globally tackle the system dynamics and deal with aircraft model changes (Balas, 2003). The most renown methods in this area are based on either Nonlinear Dynamic Inversion (NDI) or Adaptive Backstepping (ABS), however, these solutions still depend on knowledge about the system dynamics. More recent research retains the NDI advantages while reducing model-dependency (Grondman et al., 2018; Keijzer et al., 2019), but even these techniques need some model information, which is not ideal.

With the advent of Deep Reinforcement Learning (DRL), researchers are now able to design controllers that learn by trial-and-error and do not rely on knowledge about the aircraft dynamics. DRL is a bio-inspired machine learning framework that uses deep neural networks as function approximators and relies on environment interaction to learn. DRL was firstly seen in gaming applications (Mnih et al., 2013), then new algorithms were developed to tackle difficult robotic tasks (Lillicrap et al., 2016) and recently several flight control applications were presented (Cano Lopes et al., 2018; He et al., 2021).

However, despite DRL’s proven ability to handle highly complex control tasks, most of DRL’s aeronautical applications are limited to Unmanned Aerial Vehicles (UAVs). According to Xie et al. (2021), this is mostly due to the opaqueness and inexplicability associated with these solutions, given that they use a large number of parameters which makes it hard for users to interpret the control law applied by the agent. Furthermore, in general, humans are reticent to adopt techniques that are not directly interpretable, tractable and trustworthy (Goodman and Flaxman, 2017). As such, if explanations are not given along with DRL’s predictions, it is unlikely that these solutions will ever be included in a real aircraft’s flight control system.

Moreover, while explainability starts to be well developed for standard ML models and neural networks (Arrieta et al., 2020), there is little research on how to apply explainability in DRL (Heuillet et al., 2021). The eXplainable Reinforcement Learning (XRL) research field addresses this issue by developing methods that increase explainability in DRL, however, the amount of XRL research that targets flight control is scarce.

Therefore, by producing ways to supply interpretable and trustworthy explanations along with the DRL's decisions, one can contribute to the development of new model-free XRL flight control solutions. That is the focus of this research.

1.2. Research Goals

The overarching research objective of this thesis is:

> *Contribute to the development of XRL methods that increase end-user acceptance for Reinforcement Learning (RL) algorithms in flight control, by means of identifying suitable XRL techniques and by successfully implementing one of them in a business jet flight control system.*

End-user acceptance is the primary goal of this research since, in flight control, end-users (e.g. pilots, certification organisations, etc.) are required to always be aware of how the aircraft's automation reaches its decisions (EASA, 2019). If DRL algorithms are not able to supply interpretable and trackable decisions, then end-users will not be able to fulfil this requirement and DRL might never be applied in real flight control systems.

Furthermore, even though building a controller for the full jet aircraft coupled dynamics would present even more relevant results, considering the timeline and resources available for this project, it was decided that the chosen XRL method should only be applied to the longitudinal model of a Cessna Citation 500 aircraft. Using only the longitudinal dynamics still provides good model accuracy while allowing: 1) Reduced model complexity, leading to minimal required changes to the baseline DRL method used in the original XRL algorithm paper, which allows this research to focus more on the explainability part of the solution; and 2) More reliable verification of the explanations generated, given that the model's output is easier to predict. An investigation on the possible applications of the controller developed in this work to the full coupled aircraft dynamics is left as a future research recommendation.

In order to fulfil the research objective, several research questions have been identified:

Q1: What are the biggest factors that contribute to end-user acceptance of XRL methods?

- Q1.1) How to evaluate the increase in explainability provided by an XRL method?
- Q1.2) With the goal of increasing end-user acceptance, in what should the explanation focus on? 1) Focus on explaining the network processing; 2) Focus on explaining the expected future rewards; 3) Focus on where in the input the method is putting the biggest focus; or 4) Focus on aligning the explanation with the user's rationale.
- Q1.3) In order to increase end-user acceptance, how does providing multiple explanation types compare with a single explanation type method?

Q2: What XRL method should be used to automate an attitude control task for a business jet flight control system?

- Q2.1) What are the suitable XRL techniques available?
- Q2.2) Which XRL method has the highest potential to increase end-user acceptance while presenting good performance on the attitude flight control task?
- Q2.3) How does the XRL method chosen perform in a simple but relevant environment, in terms of learning stability and task performance?
- Q2.4) Is the XRL method chosen capable of providing satisfactory explanations in the simple but relevant environment?

Q3: How can a XRL method be used to improve end-user explainability in an attitude control task of a business jet flight control application?

- Q3.1) What are the states and outputs of the XRL agent?
- Q3.2) What manoeuvres will the aircraft perform during testing?
- Q3.3) What is the performance of the XRL method in keeping control of the aircraft during the attitude flight control task?
- Q3.4) Is the XRL method capable of providing satisfactory explanations for the attitude control task?
- Q3.5) How well does the chosen XRL method keep the explanations relevant during the whole task?

Research Questions 1 and 2 are fully answered in the Preliminary Research Part of this report. Research Question 3 is only answered in the conclusions section, however, the components necessary to respond to this question are given in the Scientific Article presented in Part I.

1.3. Research Scope

The scope of this study is limited to: 1) Identifying the biggest factors that contribute to end-user acceptance of XRL methods; 2) Selecting a state-of-the-art XRL method that can solve an attitude flight control task; 3) Testing the selected XRL method on an attitude flight control task, using a simulated environment and considering only longitudinal dynamics.

As such, the increase in explainability supplied by the XRL solution is not evaluated using human-in-the-loop experiments involving actual pilots, neither is the controller going to be tested in a real aircraft during this project. Furthermore, only one explanation method will be used even though a literature search is performed on the effects of presenting several explanation types to end-users.

Given that this work will most likely be the first application of the selected XRL solution to the flight control domain, these research limitations guarantee that the goals defined are in alignment with the research's timeline and available resources.

1.4. Report Outline

The remainder of this report is structured as follows. Part I presents the main results of this research in the form of a scientific article, detailing how reward decomposition explanations can be used to augment end-user explainability in flight control. Part II presents a Preliminary Research containing both a Literature Review that compares the existing XRL state-of-the-art techniques, and a Preliminary Analysis where the selected XRL methods are tested on a simple environment to assess their feasibility and performance. Part III presents additional results to the scientific article including: 1) A detailed description of the state-space model used in this research's main experiment; 2) The hyperparameter tuning phase results; and 3) The state-space model verification and validation results. The thesis is concluded in Part IV by answering the Research Questions previously formulated and by suggesting future research recommendations to expand on this research's findings.

I

Scientific Article

Explainable Reinforcement Learning in Flight Control through Reward Decomposition

André Lemos*
Delft University of Technology, P.O. Box 5058, 2626HS Delft, The Netherlands

Even though Deep Reinforcement Learning (DRL) techniques have proven their ability to solve highly complex control tasks, the opaqueness and inexplicability associated with these solutions many times stops them from being applied to real flight control applications. In this research, reward decomposition explanations are used to tackle this issue and augment DRL end-user explainability. A reward decomposition-based DRL controller is deployed in a longitudinal state-space model of the Cessna Citation 500 aircraft, and it is assessed on two attitude flight control tasks. Furthermore, a new explanation type called Dominant Reward eXplanations (DRX) is presented, which allows users to obtain more global insights than the ones generated by Reward Difference eXplanations (RDX). Results show that the explanations produced lead to straightforward and intuitive insights about the controller's behaviour, capable of improving end-user explainability. Moreover, a small analysis seems to indicate that the decomposed method has similar performance to the one obtained without reward decomposition, however, training time increases considerably. To the author's best knowledge, this is the first application of reward decomposition explanations to the flight control domain.

Nomenclature

$\boldsymbol{s}, \boldsymbol{a}$	=	reinforcement learning environment state and agent action vectors
t	=	discrete time step index
r	=	instantaneous reward
$f(\boldsymbol{s_t}, \boldsymbol{a_t})$	=	state transition function
π, π^*	=	policy and optimal policy
Q^π, Q^π_w	=	action-state value function (Q-function) and parameterised Q-function
$\boldsymbol{w}$	=	network parameter vector
ξ	=	discount factor
ϵ	=	exploration rate
$\boldsymbol{r}$	=	vectorised reward function for reward decomposition
$\boldsymbol{\Delta}(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2)$	=	reward difference explanation of action $\boldsymbol{a}_1$ over $\boldsymbol{a}_2$ in state $\boldsymbol{s}$
$\Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2)$	=	reward difference explanation of action $\boldsymbol{a}_1$ over $\boldsymbol{a}_2$ in state $\boldsymbol{s}$ with respect to reward type c
$\text{MSX}^+(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2)$	=	positive minimal sufficient explanation of action $\boldsymbol{a}_1$ over $\boldsymbol{a}_2$ in state $\boldsymbol{s}$
$\text{MSX}^-(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2)$	=	negative minimal sufficient explanation of action $\boldsymbol{a}_1$ over $\boldsymbol{a}_2$ in state $\boldsymbol{s}$
$\boldsymbol{x}$, $\boldsymbol{u}$	=	aircraft model state and input vectors
q, θ	=	pitch rate and pitch angle
α, γ	=	angle-of-attack and flight-path angle
V, h, n	=	true airspeed, altitude and load factor
δ_e, $\Delta\delta_e$	=	elevator deflection and elevator deflection increment
$\mathcal{N}$	=	standard normal distribution
μ, σ	=	mean and sample standard deviation
$\square_{ref}$	=	reference signal subscript
$\square_{error}$	=	error between reference and output value subscript
$\square_c$	=	reward type c subscript

*Graduate student, Control and Simulation Division, Faculty of Aerospace Engineering, Delft University of Technology.

I. Introduction

Current Automatic Flight Control Systems (AFCS) typically consist of controllers that rely on look-up tables for gain-scheduling [1]. These tables are built by using linearized models of the aircraft dynamics at specific operating points, which presents multiple disadvantages. First, this approach requires many operating points to have proper performance and, if not carefully planned, leaves regions of the flight envelope unchecked. Second, since it relies on knowing the aircraft dynamics, it is unable to deal with sudden changes to the model (e.g. a failure). Consequently, there is an urge to update AFCS controllers with ones that can handle model changes.

With the advent of deep reinforcement learning (DRL), researchers are now able to design controllers that learn by interacting with the environment (trial-and-error approach) and do not rely on knowledge about the aircraft dynamics (model-free solution). DRL is a bio-inspired machine learning framework that uses deep neural networks (DNNs) as function approximators and was firstly seen in gaming applications [2], then new algorithms were developed to tackle difficult robotic tasks [3], and recently several flight control applications were presented [4, 5].

However, despite DRL's proven ability to solve highly complex control tasks, most of its aeronautical applications are limited to small-scale Unmanned Aerial Vehicles (UAVs). According to [6], this is mainly due to the opaqueness and inexplicability associated with these solutions, given that they use a large number of parameters which makes it hard for users to interpret the control law applied by the agent. Moreover, users are generally reticent to adopt techniques that are not interpretable, tractable and trustworthy [7]. This implies the need to supply satisfactory explanations together with DRL's predictions. The eXplainable Reinforcement Learning (XRL) research field addresses this issue by developing methods that increase explainability in DRL.

This project aims to augment end-user explainability in flight control by using existing XRL techniques. When referring to end-users in this work, one refers to domain experts that do not necessarily have DRL expertise (e.g. pilots, certification organisations). End-user explainability is the primary goal since this audience is required to always be aware of how the aircraft's automation reaches its decisions* and, consequently, if DRL algorithms are not able to supply interpretable and trackable decisions, then end-users will not be able to fulfil this requirement and DRL might never be applied to real AFCS.

The quality of an explanation is many times qualitative and subjective, therefore, its content should depend on the audience it addresses [8]. When targeting end-users, [7] states that explanations should have trustworthiness, causality, transferability, confidence and interactivity as main goals. Furthermore, these explanations should not require previous RL knowledge to be understood. However, the number of XRL solutions that comply with these requirements and generate explanations that augment trust and interpretability for end-users is scarce [8]. Furthermore, even though previous research already presented XRL solutions for the aeronautical domain [9–11], most of them are targeted at control experts.

A novel approach based on using Reward Decomposition [12] showed promising results by generating end-user suitable explanations while solving high-dimensional input tasks. In this work, the reward function is decomposed into a sum of different semantically meaningful reward types, which allows for actions to be compared in terms of trade-offs among these different types. Furthermore, [13] performed an user study where this method was tested in a Real-Time Strategy (RTS) game environment, showing that reward decomposition explanations effectively increase trust and interpretability for users when compared to a control group without access to explanations. However, (currently) this approach only works with value-based RL methods, which forces one to discretise the action space when dealing with tasks that have a continuous action space. Nonetheless, this approach is identified as the most promising XRL solution to augment end-user explainability in flight control.

The main contribution of this paper is to augment DRL end-user explainability in flight control by using reward decomposition explanations. In this work, a model-free DRL method that uses reward decomposition is applied to a longitudinal state-space model of the Cessna Citation 500 aircraft to: 1) Solve an attitude flight control task; and 2) Generate end-user suitable explanations for the agent's behaviour. To the author's best knowledge, this is the first application of reward decomposition explanations to flight control. A second contribution is to introduce a new reward decomposition explanation type, which allows users to reach global insights rather than just insights on local predictions. These two contributions are expected to open way for the development of new model-free XRL flight control solutions.

The remainder of this paper is structured as follows. Section II describes the foundations of the methods used in this research. Section III presents the methodology applied by describing the flight control environment, agent-environment interface and DRL agent structure. Additionally this methodology section introduces a novel way to augment explainability by using Dominant Reward eXplanations (DRX). The results are found and discussed in section

*EASA - European Union Aviation Safety Agency. (2015). *Automation and Flight Path Management*. `https://www.easa.europa.eu/automation-and-flight-path-management`. Retrieved December 20, 2021

IV, where the controller and the explanations generated are evaluated on two attitude flight control tasks, and the performance of this method is compared to the one obtained without reward decomposition. Finally, section V presents the conclusions together with future research recommendations that can expand on this project's findings.

II. Foundations

This section describes the foundations of the methods used in this research. It starts by outlining the RL fundamentals, then it presents the Deep Q-Network (DQN) algorithm, and ends by describing the XRL technique used to generate explanations in this work.

A. Reinforcement Learning Fundamentals

There are two main components to a RL problem, an agent and an environment. If the action, state and reward are communicated at discrete time steps in an iterative manner, then at a given time step the agent is in state $\boldsymbol{s}_t \in \mathbb{R}^n$, takes action $\boldsymbol{a}_t \in \mathbb{R}^m$, and receives the reward $r_t \in \mathbb{R}$. After this, the sequence either ends (if $\boldsymbol{s}_t$ was terminal) or the agent moves to $\boldsymbol{s}_{t+1}$, where $\boldsymbol{s}_{t+1}$ is given by a state transition function that is unknown to the agent as described by Eq. (1).

A policy, π, is a mapping that tells the agent which action to take in each state and the goal in RL is to find an optimal policy, i.e. the policy that maximises reward over time. To find this optimal policy the agent uses an action-state value function, $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$, which is an estimate of the discounted return expected from taking action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$ and following policy π thereafter. The action-state value function (or Q-function) is shown in Eq. (2), where $\xi \in [0, 1]$ is the discount factor used to diminish future reward values.

In the learning phase, the agent is initialised with a given Q-function and policy. This policy is used to take actions in the environment and experience rewards. The rewards obtained are then used to update the current Q-function estimates, which leads to a new policy. The agent then acts according to this new policy and the cycle ideally continues until the agent converges to the optimal Q-function and policy, $Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and π^*, respectively.

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{1}$$

$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r_t + \xi \cdot Q^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \tag{2}$$

B. Deep Q-Network

When dealing with tasks that have large state spaces, calculating the Q-function value for each state-action tuple becomes unfeasible and, consequently, DNNs are typically used to estimate the value function. This approach was firstly seen in [2] where, for a given state-action tuple, the DNN (also called Q-network) outputs a Q-function estimate. The authors named this method Deep Q-Network (DQN).

Two features were essential for DQN's success at solving high-dimensional input tasks: experience replay and a target network [2].

1) Experience Replay: while learning DQN temporarily stores transitions, $(\boldsymbol{s_t}, \boldsymbol{a_t}, r_t, \boldsymbol{s_{t+1}})$, in a memory-buffer and, to update the Q-network's parameters, mini-batches of fixed size are randomly extracted from that buffer. This presents multiple advantages: 1) Allows for sample reuse, leading to greater sample efficiency; 2) Correlation between samples is minimised, leading to a reduced variance in the Q-network updates; 3) The algorithm is more stable since the behaviour distribution is averaged over many of its previous states, preventing what is known as the *catastrophic forgetting* problem [14].
2) Target Network: DQN includes a separate network to compute the target in the gradient descent updates. The parameters of this extra network are not trained, instead they are periodically synchronised with the weights from the main Q-network. Without this feature the target would change too rapidly, leading to training instabilities.

As a drawback, DQN-based methods are not suitable for tasks with continuous action spaces. DQN derives the greedy action (the best action for a state) by selecting the one that maximises the value function (i.e. DQN is a value-based method), however, in a task with a continuous action space there is an infinite number of actions, making it impossible for the algorithm to compare every state-action tuple. To deal with this difficulty one can discretise the action space, which allows for value-based methods to be used in these tasks.

It is important to note that there are DRL methods which do not require a discretisation of the action space when dealing with continuous action space tasks [15–17], however, these methods are not value-based solutions. Since this research focuses on augmenting end-user explainability in flight control by using reward decomposition explanations

(an XRL approach that currently only works with value-based solutions), such methods are not described in this work. Furthermore, over time, several improvements were made to the original DQN algorithm [18–20], but considering that this project is the first implementation of reward decomposition explanations to the aeronautical domain, an effort was made to keep the baseline DRL model as close as possible to the one seen in [12], for validation purposes.

C. Explainable Reinforcement Learning via Reward Decomposition

Despite DRL's proven ability to solve highly complex control tasks, the DNNs utilised by these methods usually have thousands of parameters, resulting in an opaque decision-making process [8]. To address this issue, [12] suggests decomposing the reward function into a sum of different semantically meaningful reward types, which allows for actions to be compared in terms of trade-offs among these different types.

1. Multi-agent off-policy RL algorithm implementation

In [12] the authors define a set of semantically meaningful reward types, $c \in C$, and a vector reward function, $\boldsymbol{r} : S \times A \rightarrow \mathbb{R}^{|C|}$. The agent's goal is still to maximise the overall reward, $r(\boldsymbol{s}, \boldsymbol{a})$, that results from summing all the different reward components in $\boldsymbol{r}$, as shown in Eq. (3), where $r_c(\boldsymbol{s}, \boldsymbol{a})$ is the reward value for reward type $c \in C$. Equation (4) shows how these concepts can be extended to the action-value function by using different Q-function components to account for the different reward types, $Q_c^\pi(\boldsymbol{s}, \boldsymbol{a})$. As described by Eq. (5), the greedy action is the one that maximises the overall Q-function that results from summing all the different Q-function components.

$$r(\boldsymbol{s}, \boldsymbol{a}) = \sum_{c \in C} r_c(\boldsymbol{s}, \boldsymbol{a}) \tag{3}$$

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \sum_{c \in C} Q_c^\pi(\boldsymbol{s}, \boldsymbol{a}) \tag{4}$$

$$\boldsymbol{a}' = \underset{a \in \mathbb{R}^m}{\arg\max} \left(\sum_{c \in C} Q_c^\pi(\boldsymbol{s}, \boldsymbol{a}) \right) \tag{5}$$

To solve an environment with this setting the authors introduce an off-policy multi-agent RL algorithm that learns multiple Q-functions, one per reward type. This solution learns the best policy together with the explanation associated, where this explanation consists on telling the user which reward types were most important for the agent's decision. Two versions of this off-policy multi-agent RL solution are presented: 1) Tabular version called decomposed reward Q-learning - drQ; and 2) DQN variant called decomposed reward DQN - drDQN. Given that this research tackles flight control tasks that have large state spaces, only the drDQN version is described in this work. It should be noted that currently this XRL approach only works with value-based DRL methods.

drDQN operates in a similar way to DQN, however, it estimates each Q-function component using a different Q-network, $Q_c^\pi(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w}_c)$, where $\boldsymbol{w}_c$ represents the network's parameters. Furthermore, drDQN uses a replay memory-buffer and several target networks (one target network per reward type). This method's pseudocode is presented in Algorithm 1, where M is the number of training episodes, T is the episode length and k is the mini-batch size. Additionally, epsilon-greedy is mentioned as the exploration method but it can be replaced by other mechanisms. As it can be seen in Algorithm 1, drDQN computes the overall greeedy action, $\boldsymbol{a}_i^+$, and then updates each network's parameters towards the target $y_{c,i} = r_c + \xi \hat{Q}_c(\boldsymbol{s}_{i+1}, \boldsymbol{a}_i^+; \boldsymbol{w}_c^-)$. Intuitively, this leads the different $Q_c(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w}_c)$ to converge towards the value of the overall greedy policy with respect to c [12]. The reward types should be orthogonal and complementary for this solution to work.

2. Extract explanations from the off-policy multi-agent RL solution

Even though comparing the decomposed Q-function components for a given state already provides information as to why an action was taken, the authors in [12] propose two other explanation types: 1) Reward Difference Explanations (RDX), and 2) Minimal Sufficient Explanations (MSX).

RDX provides all the reasons why a certain action is preferred over another. More specifically, if the agent prefers action $\boldsymbol{a}_1$ over $\boldsymbol{a}_2$ then $Q(\boldsymbol{s}, \boldsymbol{a}_1) > Q(\boldsymbol{s}, \boldsymbol{a}_2)$ and, as such, RDX is defined as the difference between the two decomposed Q-vectors, as shown in Eq. (6). If a certain RDX component is positive, $\Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) > 0$, then with respect to reward type c action $\boldsymbol{a}_1$ is preferable over action $\boldsymbol{a}_2$. Contrarily, if that RDX component is negative then, with respect to that

Algorithm 1 drDQN. Adapted from [12].

Initialise replay memory-buffer $\mathcal{D}$ to capacity $N_{\mathcal{D}}$
Initialise the different state-action value functions $Q_c(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w}_c)$ with weights $\boldsymbol{w_c}$
Initialise target state-action value functions $\hat{Q}_c(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{w}_c^-)$ with weights $\boldsymbol{w}_c^- = \boldsymbol{w_c}$
for episode = 1, ..., M **do**
  Initialise episode and get $\boldsymbol{s}_1$
  for $t = 1, \ldots, T$ **do**
    With probability ϵ select a random action $\boldsymbol{a}_t$, otherwise select $\boldsymbol{a}_t = \arg\max_a \sum_c Q_c^{\pi}(\boldsymbol{s_t}, \boldsymbol{a}; \boldsymbol{w}_c)$
    Take action $\boldsymbol{a}_t$, observe $\boldsymbol{r}_t$ and $\boldsymbol{s}_{t+1}$
    Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_t, \boldsymbol{s}_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $\{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{r}_i, \boldsymbol{s}_{i+1}) \,:\, i = 1, ..., k\}$ from $\mathcal{D}$
    for all $c \in C$ **do**
      $\boldsymbol{a}_i^+ = \arg\max_{a'} \left(\sum_{c \in C} \hat{Q}_c(\boldsymbol{s}_{i+1}, \boldsymbol{a}'; \boldsymbol{w}_c^-) \right)$
      $y_{c,i} = \begin{cases} r_c, & \text{for terminal } \boldsymbol{s}_{i+1} \\ r_c + \xi \hat{Q}_c(\boldsymbol{s}_{i+1}, \boldsymbol{a}_i^+; \boldsymbol{w}_c^-), & \text{for non-terminal } \boldsymbol{s}_{i+1} \end{cases}$
      Perform a gradient descent step on $L(\boldsymbol{w}_c) = \sum_{i=1}^{k} \left(y_{c,i} - Q_c(\boldsymbol{s}_i, \boldsymbol{a}_i \,;\, \boldsymbol{w}_c) \right)^2$ with respect to the network parameters $\boldsymbol{w}_c$
      Every N_t steps set $\boldsymbol{w}_c^- = \boldsymbol{w}_c$
    end for
    Set $\boldsymbol{s}_t = \boldsymbol{s}_{t+1}$
  end for
end for

reward type, action $\boldsymbol{a}_2$ is preferable. The magnitude of each RDX component tells the user how desirable one action is over another in relation to a specific reward type.

$$\boldsymbol{\Delta}(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) = \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}_1) - \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}_2) \tag{6}$$

Alternatively, MSX identifies the set of the *most important reasons* why an action is preferable over another. This reduces the amount of information presented in a single explanation, allowing users to not get overwhelmed with the explanations in environments that have a large number of reward types. More specifically, the MSX for actions $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ in state $\boldsymbol{s}_t$ is a tuple (MSX$^+$, MSX$^-$), where "MSX$^+$ and MSX$^-$ are the sets of critical positive and negative reasons for the preference" of action $\boldsymbol{a}_1$ over $\boldsymbol{a}_2$ [12].

To obtain the MSX$^+$ set one needs to first compute the disadvantage of $\boldsymbol{a}_1$ in relation to $\boldsymbol{a}_2$ described in Eq. (7), where I is the identity function. The *disadvantage* is the total magnitude of reasons that prefer $\boldsymbol{a}_2$, and the MSX$^+$ is the smallest set of positive reasons that are required for $\boldsymbol{a}_1$ to outweigh the *disadvantage* calculated, as shown in Eq. (8).

The MSX$^-$ set indicates the critical disadvantages of taking action $\boldsymbol{a}_1$ that makes all the reasons in the MSX$^+$ necessary. To obtain this set, one first computes the *just-insufficient advantage*, which sums all the reasons in MSX$^+$ except the smallest one (refer to Eq. 9). The MSX$^-$ is the smallest set of reasons that, when summed, have a total magnitude greater than $v(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2)$, as described by Eq. (10). When all positive components of RDX are needed for $\boldsymbol{a}_1$ to be preferable over $\boldsymbol{a}_2$, then the MSX will not offer any information compression relative to RDX.

$$d(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) = \sum_{c \in C} I[\Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) < 0] \cdot |\Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2)| \tag{7}$$

$$\text{MSX}^+(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) = \underset{M \in 2^C}{\arg\min} |M|, \quad \text{such that} \quad \sum_{c \in M} \Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) > d(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) \tag{8}$$

$$v(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) = \left(\sum_{c \in \text{MSX}^+} \Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) \right) - \left(\min_{c \in \text{MSX}^+} \Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) \right) \tag{9}$$

$$\text{MSX}^-(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) = \underset{M \in 2^C}{\arg\min} |M|, \quad \text{such that} \quad \sum_{c \in M} -\Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) > v(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) \tag{10}$$

These three explanation types (analysis of the decomposed Q-function components, RDX and MSX) already showed to increase explainability and trust for end-users [13]. Furthermore, [12] states that these explanations might be helpful for developers as well, by helping them to identify the sources for strange agent behaviour.

III. Methodology

This section describes how this work used a drDQN controller in a Cessna Citation 500 model to track a reference pitch angle signal, θ_{ref}, while complying with passenger comfort and altitude safety requirements. It presents the flight control environment, agent-environment interface and drDQN's structure. Furthermore, a new explanation type is proposed, which allows users to obtain more global insights than the ones generated by RDX.

A. Cessna Citation 500 Model

The system to be controlled in this project is a longitudinal linear model of the Cessna Citation 500 aircraft with zero wind and turbulence associated. A linear model is chosen since this research's goal is to be the first application to demonstrate how reward decomposition explanations can augment end-user explainability for DRL in flight control. Using such a model provides good accuracy in regions near the trim condition while allowing: 1) Minimal required changes to the baseline drDQN model described in [12]; 2) More reliable verification of the explanations generated, given that the model's output is easier to predict. After proving that Reward Decomposition explanations augment trust and interpretability for end-users in flight control, more complex baseline DRL methods and aircraft models can be used, however, this exceeds the scope of this project.

The linearized model is based on the high-fidelity nonlinear model presented by the Delft University Aircraft Simulation Model and Analysis Tool (DASMAT) [21]. To trim and linearize the full DASMAT system around a specific operating point the Citation Analysis and Simulation Toolkit (CitAST) was used [22]. First, the full nonlinear model was trimmed around a steady-state flight condition of $h = 2000m$, $\gamma = 0°$ and $V = 90m/s$ (true airspeed). The relevant longitudinal aircraft data obtained from trimming the full nonlinear model is presented in Table 1.

Table 1 Steady-state flight condition obtained from trimming the DASMAT model (only the relevant longitudinal data is presented).

Variable	Trim Value	Units
Aircraft mass	4500	kg
Altitude (h)	2000	m
True airspeed (V)	90	m/s
Thrust engine 1 (T_{N1})	1551.738	N
Thrust engine 2 (T_{N2})	1551.738	N
Load factor (n)	0.995	g

Variable	Trim Value	Units
Pitch rate (q)	0	$°/s$
Angle-of-attack (α)	3.222	°
Pitch angle (θ)	3.222	°
Flight-path angle (γ)	0	°
Sideslip angle (β)	0	°
Elevator deflection (δ_e)	−1.632	°

Second, a longitudinal linear model was obtained from the full trimmed system. This linear model is described by the state-space system shown in Eq. (11). The load factor is included in the state-space output since it is a known indicator for passenger comfort [23], which needs to be taken into account by the drDQN attitude controller. It is important to note that the values of $\boldsymbol{x}$, $\boldsymbol{u}$ and $\boldsymbol{y}$ are always relative to the aircraft's trim state (described in Table 1). This model was verified by giving it both a trim input and a step input. With the trim input the aircraft maintained its steady-state flight condition, and when given a step input the short period and phugoid eigenmodes were checked and behaved as expected.

$$\begin{cases} \dot{\boldsymbol{x}} = A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u} \\ \boldsymbol{y} = C \cdot \boldsymbol{x} + D \cdot \boldsymbol{u} \end{cases}, \quad \text{where} \tag{11}$$

$$\boldsymbol{x} = [q, V, \alpha, \theta, h, x_e]^T, \qquad \boldsymbol{u} = [\delta_e], \qquad \boldsymbol{y} = [q, V, \alpha, \theta, h, n, \gamma]^T$$

The state-space model presented in Eq. (11) was then used to build an RL environment in Python with which the drDQN attitude controller could interact. This environment propagates the internal state $\boldsymbol{x}$ through time in response to the input $\boldsymbol{u}$ by using a fourth order Runge-Kutta (RK4) integration at a sample rate of 100Hz.

B. Agent-Environment Interface

The agent's goal in this project is to track a reference signal for the pitch angle, θ_{ref}, while complying with passenger comfort and altitude safety requirements. To do so, an interface is needed so the agent can receive: 1) Information about the aircraft state; 2) A reference pitch angle to be tracked; and 3) A reward that indicates the effect of its actions towards achieving the goal. A schematic representation of the flight controller interface used in this project is presented in Fig. 1.

To track θ_{ref} the agent commands control input increments, $\Delta\delta_e$, which change the state-space input, δ_e, as described by Eq. (12). Providing the commands as input increments (as opposed to the total deflection) allows for a much higher number of possible elevator configurations without requiring an excessively large action space. This is especially important when using value-based DRL methods since the action space size is one of the main factors that impacts training time [14]. In this experiment the agent commands one of three actions in each time step, $\Delta\delta_e \in \{-0.1^\circ, 0^\circ, 0.1^\circ\}$. However, using this strategy requires the current control deflection, δ_e, to be included in the RL state vector.

The agent outputs actions (and receives rewards) at a frequency of 20Hz, whereas the environment propagates $\boldsymbol{x}$ at 100Hz. Having this lower agent frequency speeds the learning process while allowing the environment to have a high enough frequency to maintain accuracy. Moreover, initial tests on the full nonlinear system found that supplying a $\Delta\delta_e$ of 0.1° (maximum elevator deflection increment allowed) at 20Hz around the trim condition was easily followed by the actuator dynamics.

The environment state supplied to the drDQN attitude controller is shown in Eq. (13). Since the state components can have very different orders of magnitude (e.g. pitch rate values are usually much smaller than altitude ones), all states are normalised in the $[-1, 1]$ range before being fed to the agent.

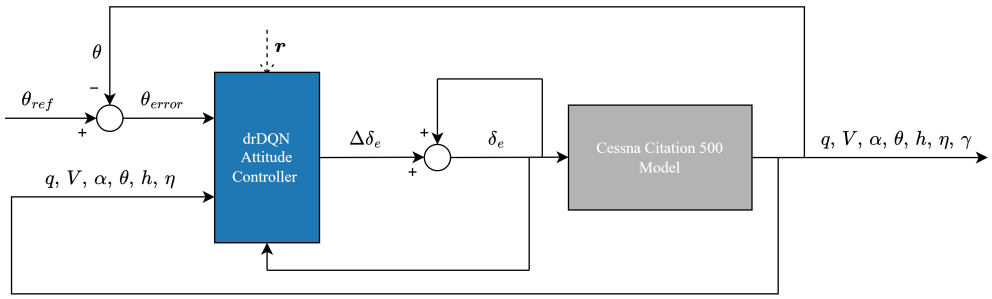


Fig. 1 Schematic representation of the flight controller interface.

$$\delta_{e_t} = \Delta\delta_{e_t} + \delta_{e_{t-1}}, \qquad \text{where} \quad \Delta\delta_{e_t} \in \{-0.1^\circ, 0^\circ, 0.1^\circ\} \tag{12}$$

$$\boldsymbol{s} = [\theta_{error}, q, V, \alpha, \theta, h, n, \delta_e], \qquad \text{where} \quad \theta_{error} = \theta_{ref} - \theta \tag{13}$$

C. Decomposed Reward Function Definition

Three reward types are used: a pitch angle tracking reward, a passenger comfort reward (assessed by the aircraft load factor), and an altitude safety reward. Accordingly, the agent's goal is to track the pitch angle reference while complying with passenger comfort requirements and staying within altitude safety bounds.

Each reward function type was tuned by trial-and-error while giving priority to the controller's compliance with the altitude safety limit of $h_{lim} = 1900$ meters (defined by the author). The pitch angle tracking reward is defined as the (negative) absolute clipped weighted error described in Eq. (14), where the "clip" function bounds the first argument with the second and the third. This reward definition guarantees that r_θ is proportional to the tracking error and belongs to the $[-1, 0]$ range. Alternatively, the passenger comfort reward, r_n, is zero if the load factor is within an acceptable range for passenger comfort (defined by the author), otherwise a negative reward is given as shown in Eq. (15), where $n_{min} = 0.7g$ and $n_{max} = 1.3g$. The altitude reward follows a similar behaviour to the one seen in r_n, and is described in Eq. (16).

$$r_\theta = \text{clip}\left[-\left|\theta_{error} \cdot \frac{1}{w_{clip}}\right|, -1, 0\right], \quad \text{where} \quad w_{clip} = 30^\circ \tag{14}$$

$$r_n = \begin{cases} 0, & \text{if } n_{min} \leq n \leq n_{max} \\ -0.3, & \text{otherwise} \end{cases} \tag{15}$$

$$r_h = \begin{cases} 0, & \text{if } h \geq h_{lim} \\ -0.6, & \text{otherwise} \end{cases} \tag{16}$$

Preliminary tests showed that drDQN presented an unstable learning process while using these three reward functions combined. It was hypothesised that such was caused by the existence of multiple optimal policies, which made it difficult for the algorithm to converge to a specific one. In order to further define what the optimal policy should be, the vector reward function is magnified in regions that are distant from the altitude safety limit, forcing the agent to focus more on the θ_{ref} signal and passenger comfort in those regions (delaying the altitude response). The resulting vector reward function is described in Eq. (17), where $h_{break} = 1915$ meters.

$$\boldsymbol{r} = \begin{cases} 10 \cdot (r_\theta, r_n, r_h), & \text{if } h \geq h_{break} \\ (r_\theta, r_n, r_h), & \text{otherwise} \end{cases} \tag{17}$$

D. drDQN Attitude Controller:

The drDQN attitude controller uses the overall logic presented in Algorithm 1, however, some changes were implemented to further augment performance. Adam is used as the DNN optimiser (instead of stochastic gradient descent) since it has consistently outperformed every other state-of-the-art alternative available [24]. Furthermore, rather than using the Mean Squared Error loss function, this work utilises SmoothL1Loss as seen in [25] which is known to be less sensitive to outliers and, in some cases, prevents exploding gradients.

Since DRL performance highly depends on the choice of suitable hyperparameters [26], an hyperparameter tuning phase was carried out where the number of hidden layers, number of units per layer, learning rate, batch size and target update interval were tuned. Table 2 describes the results obtained from this hyperparameter tuning phase , i.e. the final hyperparameters used. All Q-networks deployed by drDQN in this work use:

1) An input layer with eight units, that is, one unit per state vector component (refer to Eq. 13).
2) Three hidden layers with 128 units each and Rectified Linear Unit (ReLU) activation functions.
3) An output layer with 3 units, that is, one unit per available action (refer to Eq. 12).

The weights and biases of each network are randomly initialised with $\mathcal{N}(\mu = 0, \sigma = \frac{1}{128})$. It is important to note that when the altitude reward is not included in the vector reward function, then optimal performance can be achieved with just 2 hidden layers of 64 units each.

In the learning phase the drDQN controller is trained for 7000 episodes, where each one starts with the aircraft at the trim state (described in Table 1) and lasts 30 seconds. This accounts for approximately $4 \cdot 10^6$ training steps, found to be a good balance between training time and algorithm performance. Epsilon-Greedy is used as the exploration mechanism, with ϵ linearly decaying from a value of 0.75 to 0.1 in 5500 episodes. The controller is trained on two tasks, one where following θ_{ref} does not require the controller to exceed load factor or altitude limits, and another where to obtain optimal performance the agent needs to prioritise staying within load factor and altitude bounds. Learning on these two tasks provides the necessary flexibility for the agent to solve a variety of scenarios in the evaluation phase.

Before applying drDQN to the interface previously described, this implementation was verified in a CliffWorld environment similar to the one seen in [12], where it achieved optimal performance and produced satisfactory explanations.

It should be noted that the controller used in this research only supplies relevant performance in regions near the trim condition since it is deployed on top of a longitudinal linear model, which only presents accurate data for regions near the trim state. A DRL method with higher learning capabilities (e.g. one that includes some of the DQN improvements mentioned in [27]) should be used to obtain robust performance in other flight envelope regions, however, such exceeds the scope of this project.

Table 2 drDQN hyperparameters.

Hyperparameter Name	Value
Number of hidden layers	3
Neurons per hidden layer	128
Memory buffer size	$1 \cdot 10^6$
Batch size	64
Target update interval	8192

Hyperparameter Name	Value
Discount factor	0.99
Learning rate	$5 \cdot 10^{-5}$
ϵ_{start}	0.75
ϵ_{min}	0.1
Episodes until ϵ_{min}	5500

E. Explanation Extraction

This work uses two out of the three explanation types presented by [12]: Q-value bars and RDX. A preliminary analysis showed that MSX presents little information compression when compared to RDX for an environment with just three reward types. Consequently, MSXs are not used in this work.

Even though Q-value bars and RDX increased end-user explainability in [13], they only explain the agent's behaviour locally, that is, for a particular state. As such, it can be hard to understand how each reward type influences the agent's behaviour over an extended period of time. To address this issue a new explanation type called Dominant Reward Explanation (DRX) is introduced in this work where, instead of analysing a single state, DRX considers a task segment (containing several states inside). For each state in the segment considered, DRX computes the MSX^+ (refer to Eq. 8) between the action selected by the agent and every other alternative action, that is, for each state it aggregates the reasons that by themselves would already justify the agent's choice. The collected information is presented in the form of how many times each reward type appeared in the computed MSX^+ sets, which provides insights on how each reward type influences the agent's behaviour during the task segment analysed. The DRX pseudocode is given in Algorithm 2, where $\mathcal{A}$ represents the task action space.

Algorithm 2 Dominant Reward Explanation

input : Task segment limits: t_{start} and t_{end}, number of reward types C, set of reward types $\{r_0, ..., r_C\}$, trained drDQN
output : DRX values for task segment

Initialise empty variable DRX
for $c = 0, 1, ..., C$ **do**:
  $DRX[r_c] = 0$
end for
for $t = t_{start}, \ ..., \ t_{end}$ **do**:
  Identify drDQN's greedy-action, $\boldsymbol{a}'_t$, in state $\boldsymbol{s_t}$ according to Eq. (5)
  for a_k *in* $\mathcal{A} \setminus \{\boldsymbol{a}'_t\}$ **do**:
    Compute $\text{MSX}^+(s = \boldsymbol{s_t}, a_1 = \boldsymbol{a}'_t, a_2 = \boldsymbol{a}_k)$ according to Eq. (8)
    for $c = 0, 1, ..., C$ **do**:
      if $r_c \in \text{MSX}^+(s = \boldsymbol{s_t}, a_1 = \boldsymbol{a}'_t, a_2 = \boldsymbol{a}_k)$ **then**
        $DRX[r_c] = DRX[r_c] + 1$
      end if
    end for
  end for
end for

IV. Results and Discussion

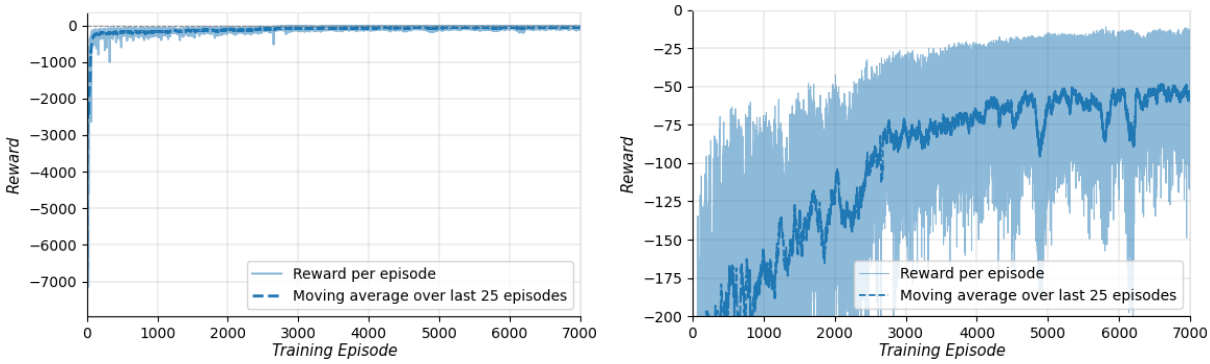
This section presents and analyses the results produced in this research. It starts by describing drDQN's training phase results, and then evaluates the model response on two different attitude flight control tasks:

1) A task where following θ_{ref} does not lead to an altitude safety risk, which allows for a better assessment of the controller's tracking performance and ability to stay within passenger comfort limits.
2) A task where following the reference signal leads to an altitude safety risk and, consequently, a trade-off between tracking θ_{ref} and staying within altitude bounds is needed to obtain optimal performance.

Reward decomposition explanations are presented for both of these tasks which demonstrates drDQN's capacity to augment end-user explainability. This section ends by presenting a small analysis that compares drDQN's performance with the one obtained by DQN.

A. drDQN Training Process

As displayed in Fig. 2, the moving average reward obtained during the learning phase stabilises around episode 6500, indicating that no further training is required. Even though not shown in this report, the three Q-network loss values stabilised around the same episode. It is also noticeable that the reward obtained per episode shows accentuated variations even after the moving average curve convergence, which is caused by: 1) The agent's exploratory actions; and 2) Using two different training tasks in the learning phase. When performed optimally each task leads to different optimal rewards and, as such, it is normal to see these high variations even at later learning stages. Nevertheless, agent convergence is confirmed by the stabilisation of the moving average reward at a value of -50.



(A) Learning curve - Total plot.

(B) Learning curve - Y-axis zoomed-in.

FIG. 2 Learning curve tracking the agent's score over training episodes.

B. Evaluation Scenario 1: No Altitude Safety Risk

This subsection shows how the drDQN attitude controller successfully solves evaluation scenario 1 and augments end-user explainability. The first sub-subsection presents the model response, whereas the second shows how drDQN increases explainability by using RDX and DRX.

1. Model Response

As shown in Fig. 3, for evaluation scenario 1, the attitude controller accurately follows θ_{ref} in regions where the aircraft is not required to exceed passenger comfort limits to have a good tracking performance. However, during the initial four seconds of the task, in order to accurately follow θ_{ref} the controller would have to pitch down at a rate that puts passenger comfort at risk and, consequently, it pitches down at a rate where n is kept at $n_{min} = 0.7g$. Once the aircraft reaches the desired pitch angle (around $t = 4s$), the controller presents good tracking performance until the end of the task given that no other manoeuvres require it to surpass safety limits. Furthermore, tracking θ_{ref} does not require the controller to surpass the altitude limit defined and, therefore, the altitude reward type should have little influence on the controller's actions (something that will be confirmed by the explanations presented in the next sub-subsection).

Minor pitch angle tracking deviations are seen throughout the task, with the θ signal momentarily diverging from θ_{ref} but quickly recovering afterwards. Most likely, these instabilities are due to drDQN failing to completely learn the

optimal policy. It is hypothesised that by using a DRL method with higher learning capabilities (e.g. by including some of the DQN improvements mentioned in [27]) these minor deviations would be reduced.

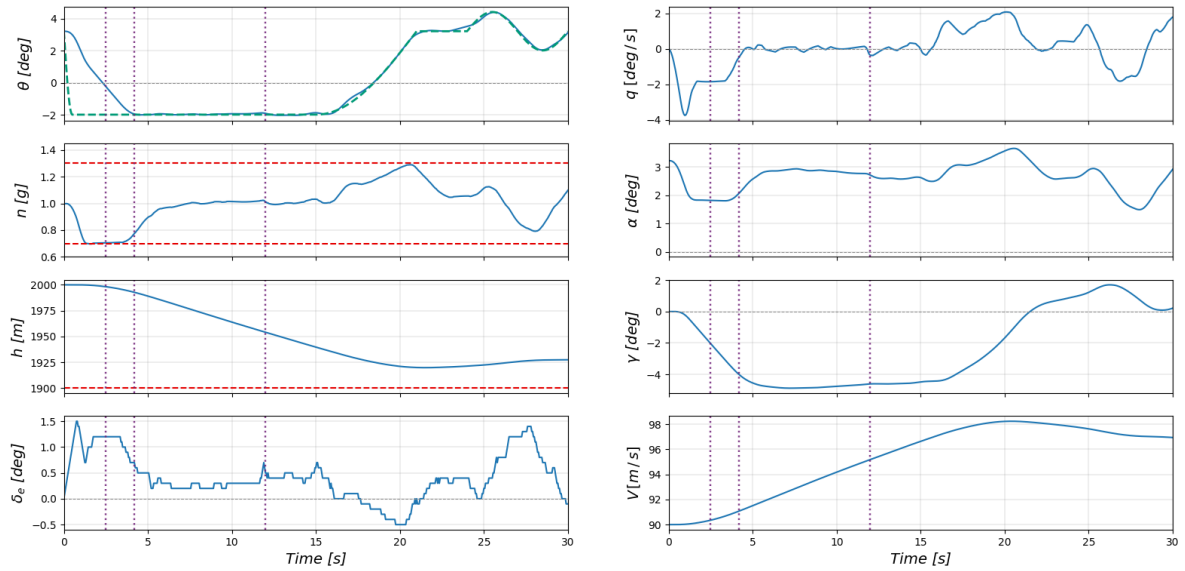


Fig. 3 Aircraft model response with drDQN attitude controller in evaluation scenario 1. Reference signals are shown with green dashed lines, whereas load factor and altitude limits are shown with red dashed lines. Purple dotted lines indicate the time steps for which the RDX is presented in this work.

2. Explanations

By using reward decomposition explanations it is possible to augment explainability in this evaluation scenario. First, this sub-subsection shows how using RDX is better than presenting Q-value bars when targetting explainability, then it analyses several individual states with the help of RDX, and finally it presents the DRX produced for various task segments which allows the reader to gain global insights rather than just local ones.

As described in Fig. 3, at $t = 2.5\,s$ the agent is pitching down to reach θ_{ref}, however, since the load factor is already at its minimum allowed value of $0.7g$ it does this at a slower rate than the one demanded by θ_{ref}. The Q-value bars generated for this time step are presented in Fig. 4, however, from this figure it is not clear why the agent opts for action $\Delta\delta_e = 0°$ given that other actions seem to provide similar returns. In order to determine the best action based on this explanation the user would have to, for each action, sum all the Q-values (one per reward type) and then select the action with the highest Q-value sum. Considering that all returns seem very similar, this is a hard task and Q-value bars provide almost no explainability increase. It should be noted that, in Fig. 4, the "θ track" Q-value bar is more dominant than the ones seen for other reward types since the agent is not expecting any penalty from the altitude or load factor rewards, but it is expecting a penalty from currently having a considerable θ_{error}.

By using RDX (refer to Fig. 5 and Fig. 6) one can directly see the difference between two decomposed Q-vectors, which allows for actions to be compared in terms of trade-offs among different reward types. This provides a much more straightforward and intuitive explanation. As previously stated, if a certain RDX component is positive, $\Delta_c(\boldsymbol{s}, \boldsymbol{a}_1, \boldsymbol{a}_2) > 0$, then with respect to reward type c action $\boldsymbol{a}_1$ is preferable over action $\boldsymbol{a}_2$. Contrarily, if that RDX component is negative then, with respect to that reward type, action $\boldsymbol{a}_2$ is preferable. Fig. 5 shows that, even though a better tracking performance would be obtained by commanding $\Delta\delta_e = +0.1°$ (i.e. by pitching down at a higher rate), the load factor reward has a higher absolute RDX value and, as such, it is preferable to command $\Delta\delta_e = 0°$ so the aircraft stays within passenger comfort limits. Alternatively, Fig. 6 shows that the load factor reward prefers action $\Delta\delta_e = -0.1°$ over $\Delta\delta_e = 0°$ (given that such brings the pitch rate closer to zero, driving n away from $n_{min} = 0.7g$), however, this would lead to a large tracking penalty and, consequently, the agent keeps the same elevator deflection. In Figs. 5 and 6, the altitude reward has little influence on the agent's decision since the aircraft altitude is far from the safety limit. These two figures completely justify the agent's choice for $\Delta\delta_e = 0°$ in $t = 2.5$ s, demonstrating how RDX increases explainability for this state.

As seen for $t = 2.5s$, RDX presents simpler and more intuitive explanations than Q-value bars because it directly describes the decomposed Q-vector difference. This RDX superiority was seen for almost all states analysed, leading the author to recommend the use of RDX over Q-value bars for explainability purposes.

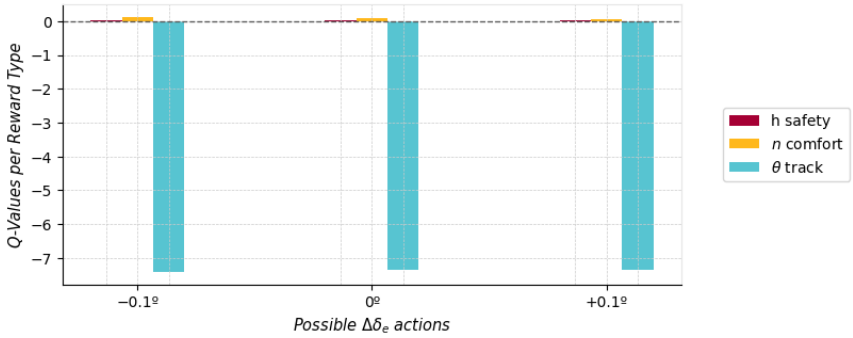


Fig. 4 Q-value bars at $t = 2.5\,s$.

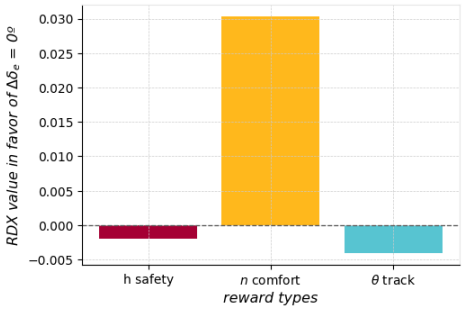


Fig. 5 RDX between action $\Delta\delta_e = 0°$ and $\Delta\delta_e = +0.1°$ at $t = 2.5\,s$.

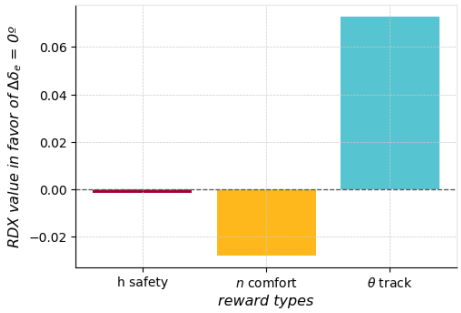


Fig. 6 RDX between action $\Delta\delta_e = 0°$ and $\Delta\delta_e = -0.1°$ at $t = 2.5\,s$.

As shown in Fig.3, around $t = 3.5s$ the agent is closer to the desired θ_{ref} and begins to decrease the elevator deflection, bringing the pitch rate closer to zero and driving n away from $n_{min} = 0.7g$. The RDX plots generated for $t = 4.2s$ can be seen in Figs. 7 and 8, from where it is clear that the load factor reward now has little influence on the agent's decision (passenger comfort is no longer at risk) and the action is selected based on the pitch angle reward. The agent opts for $\Delta\delta_e = -0.1°$ to drive the pitch rate closer to zero given that θ is already close to θ_{ref}.

Even though such is not the deciding factor for the agent's decision, the "h safety" reward at $t = 4.2s$ contradicts intuition since it supports $\Delta\delta_e = +0.1°$, which would drive the altitude closer to the altitude safety limit. It was found that these Q-value estimations that contradict intuition were common in regions where the altitude is far away from h_{lim}. It is hypothesised that such values are presented because the algorithm converged to a policy where the controller only tries to avoid going lower in altitude when it is already close to the altitude safety limit. Consequently, even though the altitude RDX is generally small in regions far away from h_{lim} (the agent does not have a strong preference for one action over another), its value might contradict user intuition. Further research is required to investigate whether this is the right explanation for this behaviour.

It would be valuable to explore if these altitude estimations that contradict intuition can be reduced by either: 1) Changing the reward definition, by providing a more gradual (rather than sparse) altitude reward; or 2) Using a baseline DRL agent with higher learning capabilities (e.g. by including some of the DQN improvements mentioned in [27]). Nevertheless, very rarely does the altitude reward have a relevant influence on the agent's decision in regions far away from h_{lim}. Furthermore, as it will be seen in evaluation scenario 2, once the altitude gets closer to h_{lim} the Q-value estimates become accurate and support intuition.

Once the pitch angle reaches θ_{ref} around $t = 4s$ (refer to Fig. 3), the θ reward dominates most of the agent's decisions until the end of the task given that no other manoeuvre requires the aircraft to exceed safety limits. However, as previously mentioned, minor pitch angle tracking deviations are observed. By analysing the RDX in these deviation

moments it becomes clear that such happens because the agent overestimates the return provided by the load factor reward. To illustrate this, at $t = 12s$ the agent has a load factor of approximately one (far away from passenger comfort limits), however, as shown in Figs. 9 and 10 it selects the greedy action based on the load factor reward and not on tracking θ_{ref}. Nevertheless, this issue is quickly fixed in subsequent states where the agent selects the action based on the pitch angle reward, allowing it to keep a good overall tracking performance. It should be noted that when running the same scenario with DQN rather than drDQN (instead of a vectorized reward, the reward is a single scalar value that results from summing the three reward components used in this experiment), these momentary divergences also happened. However, DQN does not provide explanations for its decisions and, as such, it was hard to identify the causes for this agent behaviour. It is hypothesised that the estimation errors previously mentioned can be reduced by using a more gradual load reward definition combined with an agent that has higher learning capabilities.

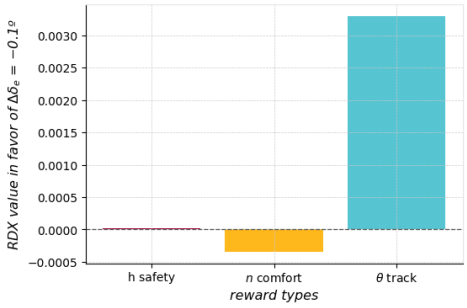


Fig. 7 RDX between action $\Delta\delta_e = -0.1°$ and $\Delta\delta_e = 0°$ at $t = 4.2\,s$.

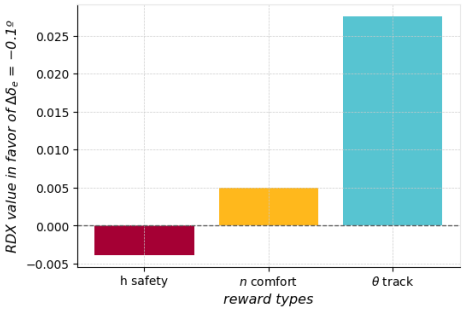


Fig. 8 RDX between action $\Delta\delta_e = -0.1°$ and $\Delta\delta_e = +0.1°$ at $t = 4.2\,s$.

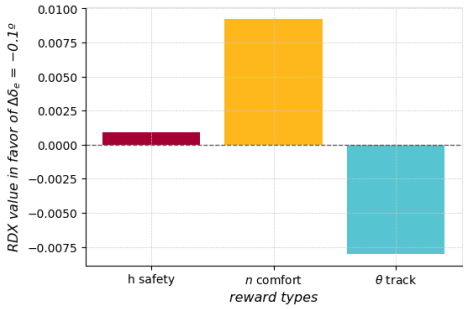


Fig. 9 RDX between action $\Delta\delta_e = -0.1°$ and $\Delta\delta_e = 0°$ at $t = 12\,s$.

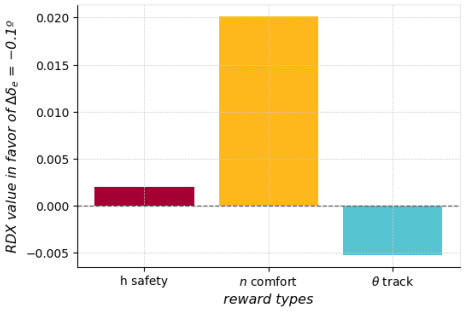


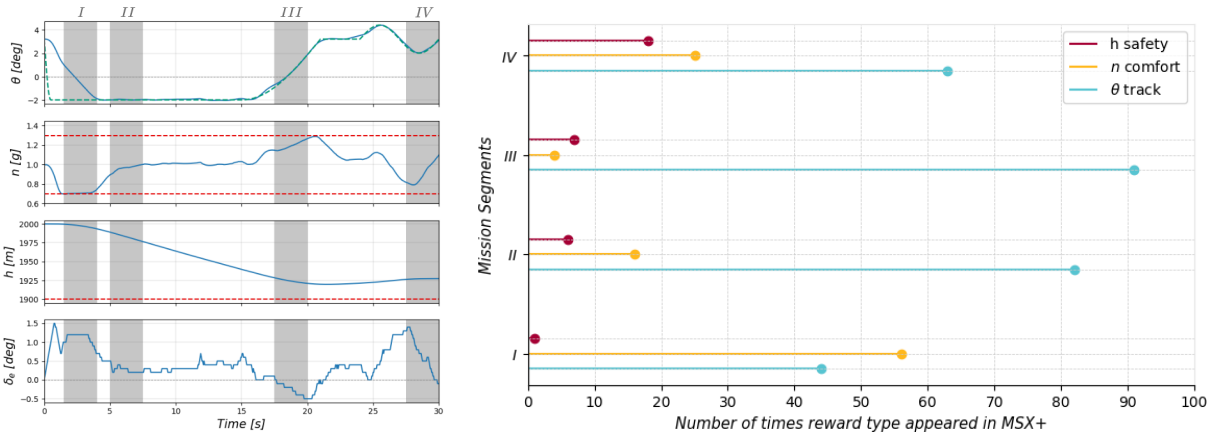
Fig. 10 RDX between action $\Delta\delta_e = -0.1°$ and $\Delta\delta_e = +0.1°$ at $t = 12\,s$.

In order to obtain global insights rather than just findings on individual predictions, this work proposed a new explanation type called DRX. In evaluation scenario 1, four task segments were analysed with DRX and the results are shown in Fig. 11.

- In segment I, the DRX shows that the agent's decisions are based on a trade-off between tracking the reference signal and staying within passenger comfort limits, with the passenger comfort reward having slightly more importance than the tracking one. This was expected given that the load factor is close to its lower limit of $n_{min} = 0.7g$ during the whole segment.
- In segments II and III, the DRX shows that the θ track reward dominates most of the agent's decisions. This was also expected since tracking the reference signal in these segments does not require the controller to exceed safety limits. Furthermore, the load factor reward influences some of the agent's choices, which demonstrates the previously discussed Q-value overestimation problem associated with this reward type. Nevertheless, since the θ track reward dominates most of the decisions, the agent achieves a good overall tracking performance despite minor estimation errors.
- In segment IV, even though the θ track reward still controls most of the agent's choices, the load and altitude rewards appear more often in the calculated MSX^+ than in previous segments. This presence is mostly notorious

in states where the agent has a negative pitch rate since, in those regions, the load factor and altitude rewards combined sometimes present a bigger RDX value than the pitch angle reward. Nevertheless, if taking the action commanded by the altitude or load reward leads to a high θ_{error}, then the pitch angle reward tends to control the decision, leading to an overall good tracking performance.

The information provided by DRX in Fig. 11 allows the user to understand how each reward type influences the agent's behaviour during each task segment. This provides more global insights than the ones presented by RDX, which is only capable of explaining the decisions locally.



(A) Time segments considered by DRX. **(B) DRX plot.**

FIG. 11 DRX produced in evaluation scenario 1.

C. Evaluation Scenario 2: Altitude Safety Risk Present

In the previous scenario analysed, even though the controller obtained a good tracking performance, no altitude safety limit had to be exceeded to track θ_{ref} and, consequently, the altitude reward had little influence over the agent's decisions. This subsection analyses the controller behaviour in a scenario where a trade-off between tracking θ_{ref} and staying within altitude bounds is needed to obtain optimal performance.

The first sub-subsection presents the model response, whereas the second shows how drDQN increases explainability for this scenario by using RDX and DRX.

1. Model Response

Figure 12 shows the model response in evaluation scenario 2, which is exactly the same as scenario 1 until $t = 15s$, however, it delays the θ_{ref} increase until the later stages of the task. This subsection only analyses the regions in which this scenario differs from scenario 1. Given that following the θ_{ref} described would require the aircraft to surpass the altitude safety limit, around $t = 20.5s$ the controller starts to pitch up. This pitch up manoeuvre is performed while complying with passenger comfort requirements (n is always lower than $n_{max} = 1.3g$), and ends when the flight-path angle reaches the zero value, that is, when the aircraft is no longer descending. From that point onward, the agent focuses on keeping γ above 0° so the altitude does not go below h_{lim}, however, it also tries to keep the pitch angle as low as possible (while still maintaining $\gamma > 0°$) to avoid larger θ track reward penalties.

Overall the controller maintains a low tracking error in regions where it is not required to exceed safety limits, and minimises the tracking penalties in regions where it has to prioritise those safety limits.

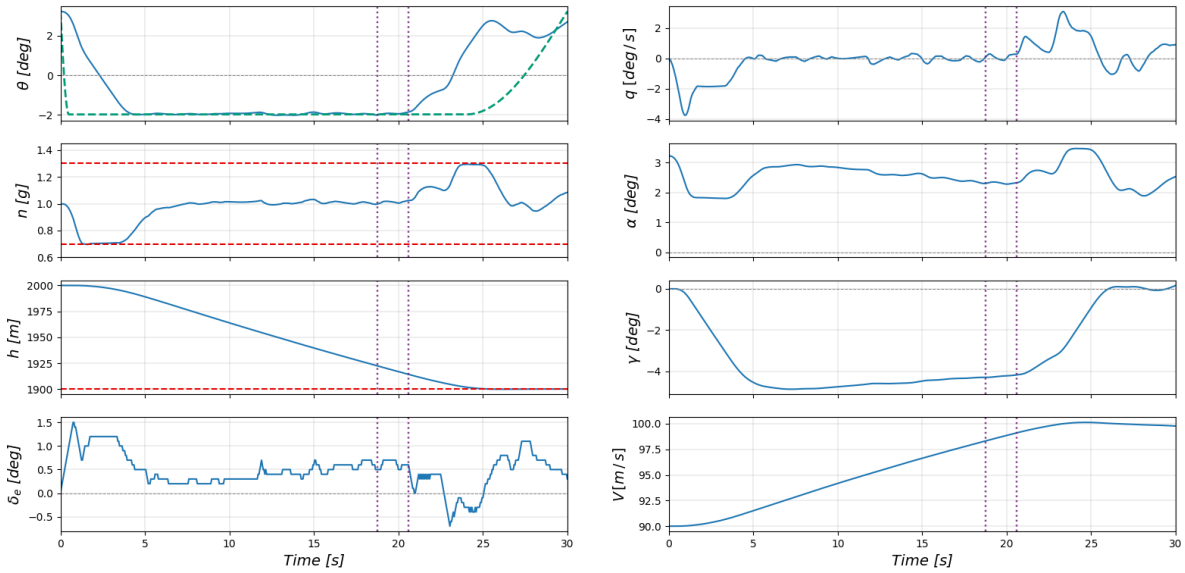


FIG. 12 Aircraft model response with drDQN attitude controller in evaluation scenario 2. Reference signals are shown with green dashed lines, whereas load factor and altitude limits are shown with red dashed lines. Purple dotted lines indicate the time steps for which the RDX is presented in this work.

2. Explanations

Considering only the regions where this scenario differs from evaluation scenario 1, it is especially interesting to investigate why the aircraft starts to pitch up exactly at $t = 20.6s$ and not at another time step. To investigate this, the regions before and after the start of the manoeuvre are analysed with RDX.

At $t = 18.6s$ the controller is two seconds away from starting the pitch up manoeuvre. The RDX generated for this time step is shown in Figs. 13 and 14, from where it is clear that even though the altitude reward prefers action $\Delta\delta_e = -0.1°$ (which would drive the altitude away from $h_{lim} = 1900$ meters), its RDX value is small compared to the pitch angle tracking one. As such, the controller chooses action $\Delta\delta_e = 0°$ based on obtaining the best tracking performance possible.

As time progresses the aircraft keeps descending and the altitude reward gains more influence over the agent's choices. At $t = 20.6s$ the controller starts the pitch up manoeuvre to prevent the aircraft from going below the altitude safety limit. The RDX generated for this time step is shown in Figs. 15 and 16, from where it is clear that the altitude reward now dominates the agent's decision, forcing the controller to command $\Delta\delta_e = -0.1°$ (i.e. driving the altitude away from h_{lim}) even though the pitch angle tracking reward prefers action $\Delta\delta_e = 0°$. Consequently, the manoeuvre starts at $t = 20.6s$ since that is when the altitude reward alone controls the agent's decision.

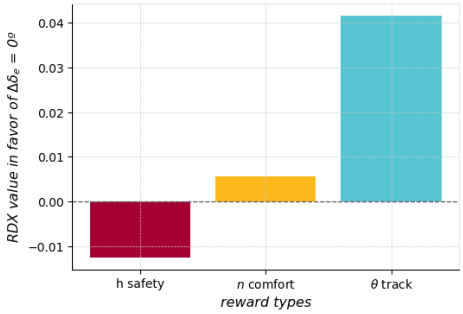


FIG. 13 RDX between action $\Delta\delta_e = 0°$ and $\Delta\delta_e = -0.1°$ at $t = 18.6\,s$.

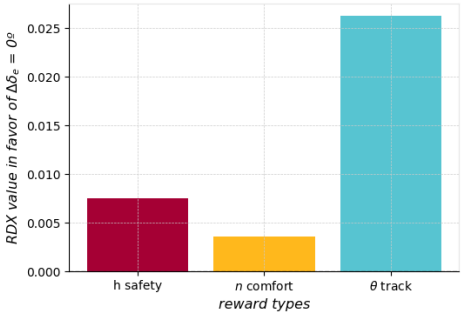


FIG. 14 RDX between action $\Delta\delta_e = 0°$ and $\Delta\delta_e = +0.1°$ at $t = 18.6\,s$.

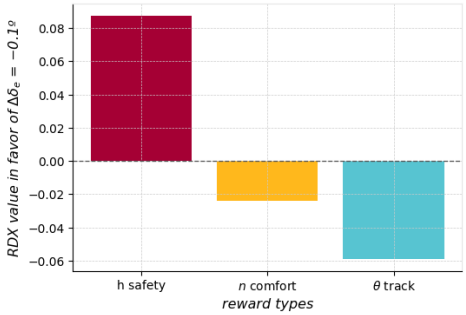


Fig. 15 RDX between action $\Delta\delta_e = -0.1°$ and $\Delta\delta_e = 0°$ at $t = 20.6\,s$.

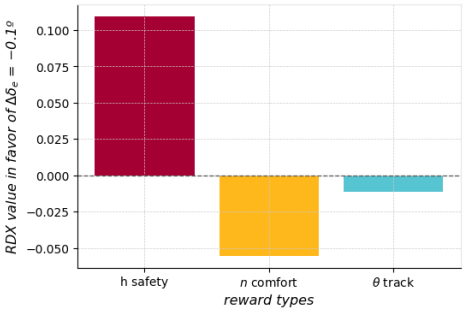


Fig. 16 RDX between action $\Delta\delta_e = -0.1°$ and $\Delta\delta_e = +0.1°$ at $t = 20.6\,s$.

To understand how each reward type influences the agent's behaviour in different task sections, five time segments were analysed with DRX and the results are presented in Fig. 17.

- Segments I and II are the same as the ones seen in scenario 1 and, thus, they were already analysed.
- Segment III refers to the region immediately before the start of the pitch up manoeuvre. The DRX shows that, even though the altitude reward has some influence over the agent's decisions, the actions are mostly selected based on the θ track reward, which keeps the aircraft on a descent and focused on tracking θ_{ref}. Furthermore, this altitude influence in segment III is largely related to stopping the agent from commanding $\Delta\delta_e = +0.1°$ (which would lead to a stepper descent) when such does not lead to a relevant increase in tracking performance.
- In segment IV the DRX shows that most of the agent's decisions are obtained by following a trade-off between staying above h_{lim} (which is the controller's priority) and trying to minimise the θ track penalty. Consequently, even though the controller will mainly focus on not exceeding the 1900 meters altitude limit, it will also keep the altitude close to that limit to avoid even larger θ penalties
- In segment V, the altitude reward influence over the agent's decisions decreases when compared to the previous segment, which is expected considering that the aircraft is no longer descending. However, the large importance given to the passenger comfort reward seems to indicate that the agent did not fully learn the optimal policy for this task segment since the load factor is far away from its limits and, consequently, should not have a large influence over the agent's decisions. It would be interesting to investigate if using a baseline DRL method with higher learning capabilities changes this behaviour.

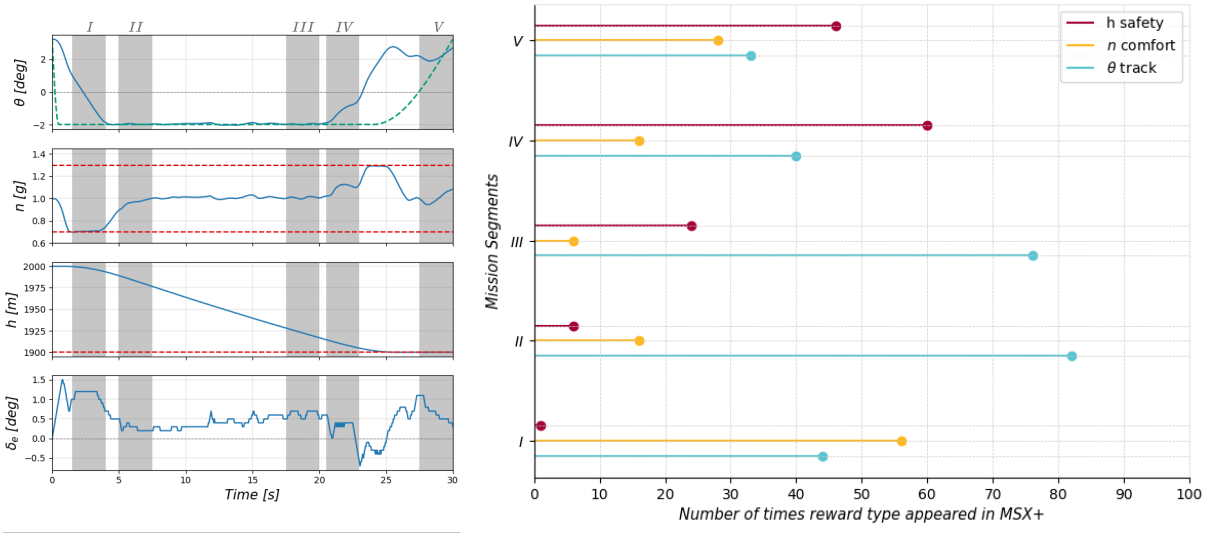
DRX proved to be especially useful for this scenario given that, in segments IV and V, the reward type that controls the agent's action can largely vary between two consecutive states (e.g. depending if the controller needs to decrease the elevator deflection to stay within altitude limits, or if it can wait before doing so to avoid larger θ track penalties). Consequently, in order to arrive at the insights provided by DRX solely based on RDX, the user would have to analyse a large number of explanations. By using the DRX method presented in this work, it is possible to obtain global insights about the agent's behaviour in a more straightforward manner.

D. Additional Tests - DQN vs drDQN

The previous subsections showed how drDQN achieves good performance in the two attitude tasks proposed, and how this method increases end-user explainability. Nevertheless, it is interesting to investigate how drDQN's performance compares with the one obtained by an agent that does not use reward decomposition (e.g. the vanilla DQN presented in [2]) since, as pointed out by [12], there are several reward decomposition strategies available in the literature and each one can lead to different task performances. The goal of this subsection is to compare drDQN's performance with the one obtained by DQN.

In order to use DQN, the vectorised reward function (refer to Eq. 17) was converted into a single scalar value that results from summing the three reward components previously used, and the hyperparameters utilised are the same as the ones described in Table 2. The inputs, outputs and general interface are maintained. Even though more runs would have been desirable, given the available resources and the large training time associated with these algorithms, only 25 runs were collected for each method. The resulting learning curves averaged over 25 runs are presented in Fig. 18, where "σ band" refers to the region between the "$\mu - \sigma$" and "$\mu + \sigma$" curves.

Both learning curves are relatively similar, with drDQN obtaining a smaller reward between episode 2000 and 5000, but ending the learning phase with the same average reward per episode. The loss curves were also checked and both



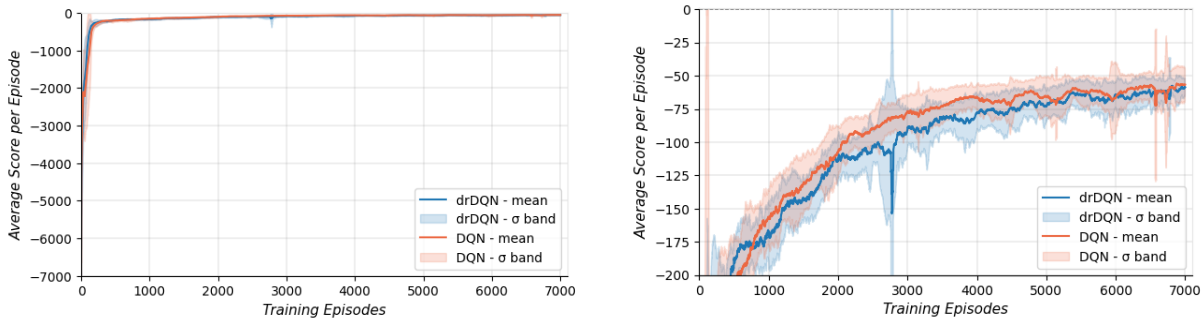
(A) Time segments considered by DRX. **(B) DRX plot.**

FIG. 17 DRX produced in evaluation scenario 2.

methods converged around the same episode. Although performance seems relatively similar, drDQN takes almost three times longer to train since it uses three Q-networks (one per reward type) whereas DQN uses a single network. This training time increases with the number of reward types used by the algorithm, which is something that should be considered before implementing this solution.

Furthermore, DQN presented similar tracking deviations to the ones seen while using drDQN. However, since the former method does not generate explanations, it was unclear why such estimation errors happened and how this behaviour could be improved. Even though such is not this project's main goal, this demonstrates how using reward decomposition can help developers improve their algorithms, given that based on drDQN's explanations several solutions for these tracking deviations were identified.

Overall, this small analysis indicates that, by using the reward decomposition approach outlined in this work, one can increase end-user explainability while maintaining task performance, however, the agent's training time increases considerably. Although useful, this does not lead to a generalisation about this method's performance given that more runs would be necessary to arrive at robust conclusions. It would also be interesting to run these tests using other baseline DRL algorithms and different environments (e.g. test on the full nonlinear aircraft model).



(A) Learning Curves - Total plot. **(B) Learning Curves - Y-axis zoomed-in.**

FIG. 18 DQN and drDQN learning curves averaged over 25 runs.

V. Conclusion

This work demonstrated how end-user explainability can be augmented for DRL methods in flight control by using reward decomposition explanations. It showed that the decomposed DRL algorithm utilised, drDQN, achieved good tracking performance in regions where it was not required to exceed safety limits, and it was able to minimise tracking penalties in regions where it needed to prioritise those limits. Furthermore, drDQN provided end-user suitable explanations that justified the agent's decisions in a straightforward and intuitive way, turning the originally opaque DQN's decision process into a more interpretable one.

Additionally, this work introduced a new explanation type called Dominant Reward eXplanations (DRX), providing insights on how each reward component influences the agent's decisions during a task segment. In the two tasks analysed DRX successfully supplied global insights about the agent's behaviour, which can be very useful given that RDX and Q-value bars only generate local explanations.

Even though the controller achieved a good overall task performance, in specific time steps, it presented tracking deviations from which it quickly recovered afterwards. An RDX analysis showed that these deviations were induced by load factor Q-value estimation errors. It was hypothesised that these errors could be reduced by using a more gradual passenger comfort reward function and a DRL agent with higher learning capabilities. The same tracking deviations were observed when performing these tests with DQN, however, this method does not generate explanations and, as such, it was unclear why such estimation errors occurred and how this behaviour could be improved. This shows how reward decomposition explanations are not only useful to end-users, but can also help developers improve their algorithms by providing insights into strange agent behaviour.

Furthermore, this research conducted a small analysis that compared drDQN's performance with the one obtained by DQN. Results showed that in the 25 runs performed both algorithms displayed similar performance, however, in order to augment end-user explainability drDQN takes considerably longer to train. Moreover, this training time increases with the number of reward types used in the method, something that should be considered before implementing this solution.

To the author's best knowledge, this work is the first application of reward decomposition explanations to the flight control domain, and the results produced are expected to contribute to the development of new model-free XRL flight control solutions. In order to expand on this project's findings, further research should focus on four main aspects. First, it would be interesting to run this experiment using more gradual (rather than sparse) passenger comfort and altitude reward functions to investigate if such leads to an improved agent behaviour for the moments where it previously presented counterintuitive Q-value estimates. Second, further research should evaluate the explanations generated by using human-in-the-loop experiments involving end-users (e.g. pilots). Third, it would be interesting to use this explainability approach in the full nonlinear aircraft model to investigate if this solution also augments explainability while performing more complex manoeuvres in the full nonlinear model. To use this approach in the full model, a DRL agent with higher learning capabilities (e.g. one that includes some of the DQN improvements mentioned in [27]) should be used, since the controller structure presented in this research is not expected to solve the full nonlinear aircraft model. Finally, it would be of great value to develop a version of this XRL solution that can be used with actor-critic methods, given that most flight control DRL solutions available use an actor-critic approach.

References

[1] Stevens, B., Lewis, F., and Johnson, E., *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*, Jonh Wiley & Sons, 2015.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing Atari with Deep Reinforcement Learning," *Neural Information Processing Systems Deep Learning Workshop*, 2013.

[3] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *Proceedings of the 4th International Conference on Learning Representations*, ICLR, 2016, p. 40.

[4] Dally, K., and Kampen, E.-J., "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control," *AIAA SCITECH Forum*, 2022. https://doi.org/10.2514/6.2022-2078.

[5] Azar, A. T., Koubaa, A., Ali Mohamed, N., Ibrahim, H. A., Ibrahim, Z. F., Kazim, M., Ammar, A., Benjdira, B., Khamis, A. M., Hameed, I. A., and Casalino, G., "Drone Deep Reinforcement Learning: A Review," *Electronics*, Vol. 10, No. 9, 2021. URL https://www.mdpi.com/2079-9292/10/9/999.

[6] Xie, Y., Pongsakornsathien, N., Gardi, A., and Sabatini, R., "Explanation of Machine-Learning Solutions in Air-Traffic Management," *Aerospace*, Vol. 8, No. 8, 2021. URL https://www.mdpi.com/2226-4310/8/8/224.

[7] Arrieta, A. B., Díaz-Rodríguez, N., Ser, J. D., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F., "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, Vol. 58, 2020, pp. 82–115. https://doi.org/10.1016/j.inffus.2019.12.012.

[8] Heuillet, A., Couthouis, F., and Díaz-Rodríguez, N., "Explainability in deep reinforcement learning," *Knowledge-Based Systems*, Vol. 214, 2021, p. 106685. https://doi.org/10.1016/j.knosys.2020.106685.

[9] He, L., Aouf, N., and Song, B., "Explainable Deep Reinforcement Learning for UAV autonomous path planning," *Aerospace Science and Technology*, Vol. 118, 2021, p. 107052. https://doi.org/10.1016/j.ast.2021.107052.

[10] van Zijl, J., Nunes, T., and Kampen, E.-J., "Using Explainable Artificial Intelligence to Improve Transparency of Reinforcement Learning for Online Adaptive Flight Control," Master's thesis, Delft University of Technology, 2022. URL http://resolver.tudelft.nl/uuid:66a5fdba-6508-4b6f-b20f-c1766ec4fc7f.

[11] Pizarroso, G., and Kampen, E.-J., "Explainable Artificial Intelligence Techniques for the Analysis of Reinforcement Learning in Non-Linear Flight Regimes," Master's thesis, Delft University of Technology, 2022. URL http://resolver.tudelft.nl/uuid:d278202d-fe0b-4679-8b74-63d3c2f57495.

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

[13] Anderson, A., Dodge, J., Sadarangani, A., Juozapaitis, Z., Newman, E., Irvine, J., Chattopadhyay, S., Olson, M., Fern, A., and Burnett, M., "Mental Models of Mere Mortals with Explanations of Reinforcement Learning," *ACM Transactions on Interactive Intelligent Systems*, Vol. 10, No. 2, 2020. https://doi.org/10.1145/3366485.

[14] Sutton, R., and Barto, A., *Reinforcement Learning: An Introduction*, MIT Press, 2018.

[15] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," *arXiv preprint*, 2017. ArXiv: 1707.06347.

[16] Fujimoto, S., van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, PMLR, 2018, pp. 1587–1596.

[17] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, PMLR, 2018, pp. 1861–1870.

[18] Schaul, T., Quan, J., Antonoglou, I., and Silver, D., "Prioritized Experience Replay," *4th International Conference on Learning Representations*, ICLR, 2016.

[19] van Hasselt, H., Guez, A., and Silver, D., "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30, No. 1, 2016. https://doi.org/10.1609/aaai.v30i1.10295.

[20] Wang, Z., Schaul, T., Hasselt, H., Freitas, N., and Lanctot, M., "Dueling Network Architectures for Deep Reinforcement Learning," *Proceedings of The 33rd International Conference on Machine Learning*, PMLR 48, 2016, pp. 1995–2003.

[21] van der Linden, C., *DASMAT-Delft University Aircraft Simulation Model and Analysis Tool: A Matlab/Simulink Environment for Flight Dynamics and Control Analysis*, Control and Stimulation Series, Delft University Press, 1998. URL http://resolver.tudelft.nl/uuid:25767235-c751-437e-8f57-0433be609cc1.

[22] Borst, C., "CITAST: Citation Analysis and Simulation Toolkit," Tech. rep., Delft University of Technology, Delft, 2004.

[23] Hale, F. J., *Introduction to aircraft performance, selection, and design*, Wiley New York, 1984.

[24] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9*, 2015.

[25] Girshick, R., "Fast R-CNN," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. https://doi.org/10.1109/ICCV.2015.169.

[26] Nguyen, V., Schulze, S., and Osborne, M., "Bayesian Optimization for Iterative Learning," *Advances in Neural Information Processing Systems*, Vol. 33, Curran Associates, Inc., 2020, pp. 9361–9371.

[27] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D., "Rainbow: Combining Improvements in Deep Reinforcement Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32, No. 1, 2018. https://doi.org/10.1609/aaai.v32i1.11796.

II

Preliminary Research[1]

[1]The *Preliminary Research* part was assessed as part of the *AE4020 - Literature Study* course.

2

Literature Review

This chapter presents a literature review where the foundations of the methods used in this research are specified, and the main DRL and XRL state-of-the-art algorithms are outlined. The content presented allows for a comparison between the different state-of-the-art solutions, from which the most promising methods for increasing end-user explainability are chosen.

Section 2.1 presents the foundations of RL whereas section 2.2 presents the theory necessary to understand how RL can be applied to tasks with large (or even continuous) state and action spaces. Only in section 2.3 the state-of-the-art DRL methods are introduced and compared. Section 2.4 addresses the opaqueness problem associated with DRL techniques by introducing the field of XRL and presenting the available XRL state-of-the-art solutions. Lastly, section 2.5 concludes the literature review by selecting the most promising DRL and XRL methods for increasing end-user acceptance in the attitude flight control task this project focuses on.

The content presented in this chapter answers the entirety of Research Question 1 and Sub-Question 2.1.

2.1. Reinforcement Learning Fundamentals

Reinforcement Learning (RL) is a ML framework where the agent learns from interaction with the surrounding environment. It is based on the way that biological agents learn to perform complex tasks (Sutton and Barto, 2018), and its main difference compared to other ML frameworks is that, in RL, the agent learns based on reward and punishment without ever being told how to complete the task (Kaelbling et al., 1996).

This section starts by outlining the basic elements of RL (refer to subsection 2.1.1), which are essential to understand the RL tabular solution methods presented in subsection 2.1.2.

2.1.1. Basic Concepts of Reinforcement Learning

There are two main components to a RL problem, an agent and an environment. Through interaction with the environment the RL agent learns how to perform a task over time, just like humans learn to walk by trial-and-error.

In order to learn how to perform a task optimally the agent takes actions in the environment, which leads the environment to present a new situation and to give the agent a reward (that tells him the effects that its action had towards achieving a certain goal). By trying to maximise the reward it receives from the environment over time, the agent learns to select optimal actions that lead to achieving the goal.

In the literature, the mentioned ”situation” given by the environment is called a state. States are observations that the agent receives from the environment and are considered to be part of the interface between the agent and the environment. It is important to note that the environment might not provide full information to the agent. To illustrate this, in a game of chess, the agent has full information about the environment since the state is represented by the position of the pieces over the chess board,

whereas in a card game the agent generally knows which cards are in its possession but it does not know the other player's cards (thus the agent only accesses partial information).

In order to implement this interface between the agent and the environment, a mathematical formalisation is needed. Such formalisation is achieved through the use of Markov Decision Processes (MDPs).

Finite Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical framework used for modelling decision-making problems (Silver, 2015).

If the action, state and reward are communicated at discrete time steps in an iterative manner, then at a time step k, the agent is in state s_k, takes the action a_k, and receives the reward r_{k+1}. After this, the sequence either ends (if s_k was terminal) or the agent moves to state s_{k+1}. It is said that a state s_k has the Markov Property if it captures all relevant information from the past, that is, all states previous to s_k are no longer needed to predict the future (Sutton and Barto, 2018). Such is the same as stating that:

$$P\{s_{k+1}, r_{k+1} \mid s_k, a_k, s_{k-1}, r_{k-1}, ..., s_0, a_0\} = P\{s_{k+1}, r_{k+1} \mid s_k, a_k\} \tag{2.1}$$

, where P is the probability that the agent moves to state s_{k+1} and gets reward r_{k+1}. In equation 2.1, the reward r_{k+1} is a real number (belonging to a set of rewards $\mathcal{R}$), while action a_k and state s_k are vectors belonging to the action space $\mathcal{A}$ and state space $\mathcal{S}$, respectively.

Now that the Markov Property has been defined, it is possible to define a Markov Decision Process (MDP). A MDP is a decision making process where all states obey the Markov Property. More formally, Silver (2015) defines the MDP as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions .
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}\{s_{k+1}{=}s' \mid s_k{=}s, a_k{=}a\}$.
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[r_{k+1} \mid s_k{=}s, a_k{=}a]$.
- γ is the discount factor, $\gamma \in [0, 1]$.

, where s' is the state at the next time step. The discount factor will be talked about in more detail later. Almost all RL problems can be formalised as MDPs, and partially observable problems can be converted to MDPs. Figure 2.1 outlines the agent-environment interaction present in a MDP.

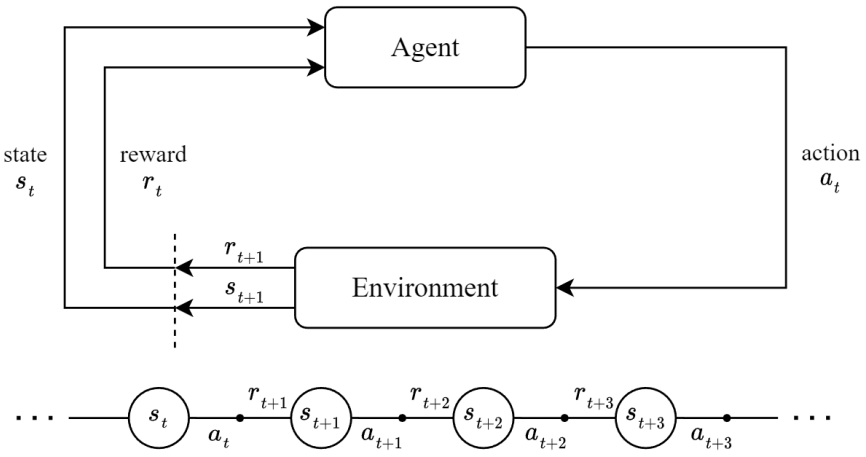


Figure 2.1: The agent-environment interaction in a MDP. Adapted from Sutton and Barto (2018).

Reward Function

The agent's goal is to find optimal actions that maximise the reward over time. However, to do this, the agent does not have access to all future rewards that come from following a particular policy, instead, it receives an immediate reward r_{k+1} given at the end of each time step. This reward can have various designs, it may be sparse (i.e. it is only non-zero when the agent reaches the goal), or it might be more gradual (providing the agent with non-zero rewards throughout the episode). The agent's objective is to maximise the cumulative reward rather than just the immediate one.

Furthermore, not every RL problem has a terminal state. There are several situations where the agent-environment interaction described in figure 2.1 does not break into episodes and goes on without a time limit, leading the cumulative reward to tend to infinity. This type of problems are called *continuous tasks* (Sutton and Barto, 2018).

To deal with these two ideas, designers use a discounted return, G_k, where instead of maximising a cumulative sum of rewards (as previously idealised) the agent strives to maximise a discounted long-term reward. The value of the discounted reward, G_k, at time step k in an episode with a terminal step N is defined in equation 2.2.

$$G_k = r_{k+1} + \gamma \cdot r_{k+2} + \gamma^2 \cdot r_{k+3} + \ldots = \sum_{i=0}^{N} \gamma^i \cdot r_{k+i+1} \tag{2.2}$$

The discount factor $\gamma \in [0, 1]$ can be increased (or decreased) to favour (disregard) long-term planning. For continuous tasks $N = \infty$, but G_k remains bounded as long as $\gamma \in [0, 1]$.

Policy, Value Function and Action-Value Function

The policy π is a function that indicates to the agent what action to take in each state. As expressed in equation 2.3, this policy can be represented as the probability of choosing action a_k in state s_k.

In order to maximise the discounted return, the RL agent uses value functions (Sutton and Barto, 2018). One such value function is the state value function, $V^\pi(s)$, defined in equation 2.4, which is a measure of how beneficial it is to follow policy π from state s_k onward. It is important to mention that $E_\pi[\cdot]$ denotes the expected value of a random variable following a policy π.

Alternatively, one can use the state-action value function, $Q^\pi(s, a)$, which is also known as action-value function. This function represents the discounted return expected from taking action a_k in state s_k, and following policy π thereafter (refer to equation 2.5).

$$\pi(a, s) = \mathcal{P}\{a \mid s_k{=}s\} \tag{2.3}$$

$$V^\pi(s) = \mathbb{E}_\pi[G_k \mid s_k{=}s], \quad \text{for all } s \in \mathcal{S} \tag{2.4}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_k \mid s_k{=}s, a_k{=}a], \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \tag{2.5}$$

A policy π is called optimal if its expected return for all states is higher or equal than any other policy, in which case it is written as π^*. The state value function and state-action value function are maximised when the optimal policy π^* is followed and, on that case, they are called optimal value functions. The optimal state value function and state-action value function are presented in equations 2.6 and 2.7, respectively.

$$V^*(s) = \max_\pi V^\pi(s), \quad \text{for all } s \in \mathcal{S} \tag{2.6}$$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a), \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \tag{2.7}$$

Bellman Equations

In order for RL algorithms to use the previously described value functions one needs to define them in a recursive form, and the Bellman equations allow RL designers to do just that. In equation 2.8 the Bellman equation for the state value function is derived, whereas in equation 2.9 only the final expression for the action-state value function is shown (its derivation can be found at Sutton and Barto, 2018).

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[G_k \mid s_k{=}s] \\ &= \mathbb{E}_{\pi}[r_{k+1} + \gamma \cdot G_{k+1} \mid s_{k+1}{=}s'] \\ &= r_{k+1} + \gamma \cdot \mathbb{E}_{\pi}[G_{k+1} \mid s_{k+1}{=}s'] \\ &= r_{k+1} + \gamma \cdot V^{\pi}(s') \end{aligned} \tag{2.8}$$

$$Q^{\pi}(s,a) = r_{k+1} + \gamma \cdot Q^{\pi}(s',a') \tag{2.9}$$

By using the recursive form of the value functions expressed above, one can rewrite the optimal state value functions in a recursive manner, which yields the Bellman optimality equations:

$$V^{*}(s) = \max_{a \in \mathcal{A}} \mathbb{E}[r_{k+1} + \gamma \cdot V^{*}(s') \mid s_k{=}s, a_k{=}a] = \max_{a \in \mathcal{A}} [r_{k+1} + \gamma \cdot V^{*}(s')] \tag{2.10}$$

$$Q^{*}(s,a) = \mathbb{E}[r_{k+1} + \gamma \cdot \max_{a' \in \mathcal{A}} Q^{*}(s',a') \mid s_k{=}s, a_k{=}a] = r_{k+1} + \gamma \cdot \max_{a' \in \mathcal{A}} Q^{*}(s',a') \tag{2.11}$$

With the Bellman equations defined, it is possible to build methods that allow the agent to find an optimal policy that solves the task. In case the state and action spaces are small, the RL method is able to find an exact solution and it is called a *tabular solution method*. However, in many RL tasks, the state space is enormous and finding an exact solution to the problem becomes unfeasible. In such cases, a parameterised function representation is used to approximate the value functions and the methods are called *approximate solution methods*.

2.1.2. Tabular Solution Methods

When the task state and action spaces are small, tabular solution methods can be used, where the value function is represented in tables or arrays. Even though such is not the case for the business jet application this research focuses on, tabular solution methods are the foundation for advanced approximate solution methods. As such, understanding how they work is essential.

There are mainly two categories of tabular solution methods. Firstly there are Dynamic Programming (DP) methods, which according to Sutton and Barto (2018) refer to a ”collection of methods that can be used to compute optimal policies given a perfect model of the environment as a MDP”. For DP methods, the complete probability distributions of all state transitions is needed and in each iteration the algorithm sweeps through the entire state space $\mathcal{S}$. Since this research focuses on methods that are completely model-free, DP methods will not be explored in this project.

Alternatively, there are model-free tabular solution methods. These solutions are based on either Monte-Carlo (MC) learning or Temporal Difference (TD) learning. Both of these alternatives use experience acquired in sample episodes and do not require any knowledge about the environment. The main difference between these two model-free alternatives is that MC methods have to wait until the end of the episode before the return is known and an update can be performed, whereas TD methods are able to learn after each step within an episode. Given that flight control tasks are generally *continuous tasks*, MC methods will not be considered for this research since they are only suitable for episodic tasks.

Contrarily to Monte-Carlo methods, TD methods can learn at every step of the episode because they bootstrap, meaning, they compute the current value based on an existing estimate from previous iterations. As such, these methods are well suited for *continuous tasks* and are widely used in RL. TD learning solutions compute the error between the target and the current estimate of the value function, and then increment the current estimate by this error multiplied by a learning rate α. Equations 2.12

and 2.13 are examples of how TD learning methods update the value function based on an existing estimate:

$$V(s_k) \leftarrow V(s_k) + \alpha \cdot \left[r_{k+1} + \gamma \cdot V(s_{k+1}) - V(s_k)\right] \tag{2.12}$$

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha \cdot \left[r_{k+1} + \gamma \cdot Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)\right] \tag{2.13}$$

According to Sutton and Barto (2018), a disadvantage linked to this technique is that the target will have a bias associated since the updates are based on an existing estimate of the value function.

Exploration vs Exploitation Trade-Off:

During the learning process, the agent can use two modes to choose its next action. First, it can choose based on exploitation, that is, by selecting the action that maximises the current state-action value function. In this case, it is said that the agent selected the greedy action, which is defined in equation 2.14.

$$a_k = \underset{a \in \mathcal{A}}{\arg\max}\ Q_k(s_k, a) \tag{2.14}$$

The problem with this strategy is that, in the initial stages of the learning phase, the value function estimated by the agent is not accurate. Consequently, if the agent only takes greedy actions then it will never visit regions outside of the current maximum of the estimated value function (Sutton and Barto, 2018). In this scenario, the agent gets stuck on a local maximum without ever discovering the global one.

To address this issue there is a second mode for choosing actions, the exploration mode. In this mode, the agent executes a random action (which might be a non-optimal one) with the objective of reaching unseen state-action combinations and possibly find another maximum. The idea is that, with time, the agent will explore all relevant actions and find the optimal solution. Both exploration and exploitation have to be combined during the learning process in order to reach a stable global solution.

There are several strategies to guarantee that the agent continues to select exploratory actions throughout the learning process. A common strategy is called ϵ-greedy, where the agent selects a random action with a probability of ϵ (the probability of selecting the greedy action is $1-\epsilon+\frac{\epsilon}{\#\,of\ actions}$). Generally, the value of ϵ decreases over time and never reaches zero.

With the introduction of the trade-off between exploration and exploitation another important distinction appears. If there are two modes for choosing actions then two different policies are considered in the algorithm: 1) The target policy, which is the policy that the agent follows when it is acting in a greedy manner; 2) The behavioural policy, which is the policy used to take actions. These two policies only differ when the agent uses the exploration mode to choose its next action.

Depending on what policy the agent uses to update the value function, the RL algorithm can be an on-policy method, or an off-policy one.

On-Policy Methods:

On-Policy methods refer to algorithms that learn based on the behavioural policy, that is, they learn based on the executed action which might not necessarily be the greedy one. A widely used on-policy tabular solution method is SARSA (State-Action-Reward-State-Action). The update rule for SARSA using the state-action value function can be seen in equation 2.15.

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha \cdot \left[r_{k+1} + \gamma \cdot Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k)\right] \tag{2.15}$$

Equation 2.15 shows that the state-action value function is updated with the action taken by the agent at time step k, which might differ from the greedy action. As such, the behavioural policy is the policy that is being optimised in this algorithm.

Off-Policy Methods:
Off-Policy RL refers to algorithms that use the greedy action (which might differ from the executed one) to update the value function. As such, these methods can execute actions from a behavioural policy while using the target policy for the value function update. The most well-known off-policy tabular solution method is the Q-Learning algorithm, and its update rule using the state-action value function can be seen in equation 2.16.

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha \cdot \left[r_{k+1} + \gamma \cdot \max_{a \in \mathcal{A}} Q(s_{k+1}, a) - Q(s_k, a_k)\right] \tag{2.16}$$

In equation 2.16 the learned state-action value function ,$Q(s, a)$, directly approximates the optimal one, $Q^*(s, a)$, independently of the policy being followed (Sutton and Barto, 2018). Off-policy methods present some advantages since they allow for: 1) Continuous exploration; 2) Learning from demonstration; and 3) By not using the behavioural policy in the update, off-policy methods are able to optimise not only on the current sample but also on old data (leading to better sample efficiency).

On the reverse side, these solutions can lead to dangerous behaviour while exploring. If higher rewards are present near a safe-critical boundary, then off-policy methods will learn to stay close to that boundary while neglecting the risk of trespassing it due to exploratory actions. In safe critical situations this behaviour can be dangerous, making off-policy methods better suited for cases where safety during learning is not a concern (offline learning).

Since there are no specifications in the research background that dictate if the model needs to be on-policy or off-policy, both strategies are considered moving forward.

Online and Offline Learning:
At a given point, the learning phase can be done: 1) Online, that is while performing the task; or 2) Offline, where the agent learns to do the task in a separate environment (e.g. a simulated one). Taking these two options into account, in RL the entire learning phase can be done in one of three ways:

- Fully online: The agent learns while performing the task. For this to be the case, one has to make sure that the agent has high sample efficiency and that safety considerations are taken into account (since crashes are not permitted). Given that the parameters of these methods are allowed to change online, they belong to Adaptive Control.
- Initially offline and then online: The initial learning is performed offline where safety is not an issue and the agent has enough time to learn the basics of the task. Then, in a second learning phase, the agent is taken online where it is still able to adapt its parameters to the real system.
- Fully offline: The agent only learns in a simulated environment. The algorithm is transferred to the real system after learning how to perform the task and, once in the real system, the agent cannot change its parameters.

Considering that the research background does not present any specifications that dictate if the model has to learn online or offline, both possibilities are considered moving forward.

Double Learning:
A common drawback pointed to both Q-learning and SARSA is that both of these algorithms involve a maximisation over the estimated value function (Sutton and Barto, 2018), which might introduce a positive bias in the estimation. This positive bias is usually called *maximisation bias*.

To counteract this, *double learning* can be used where two independent estimates called $Q_1(a)$ and $Q_2(a)$ are used. Since both estimates learn from the same environment, each Q-function is an estimate of the true value $q(a)$, for all $a \in \mathcal{A}$. In this strategy, one can use one of the estimates, $Q_1(a)$, to determine the maximising action (refer to equation 2.17), while using the other to provide the estimate for the state-action value function using a^* (refer to equation 2.18). According to Sutton and Barto (2018), this last estimate will be unbiased in the sense that $\mathbb{E}[Q_2(a^*)] = q(a^*)$. This process can be repeated with the two estimates reversed to produce a second unbiased estimate.

$$a* = \arg\max_{a \in \mathcal{A}} Q_1(a) \tag{2.17}$$

$$Q_2(a*) = Q_2\big(\arg\max_{a \in \mathcal{A}} Q_1(a)\big) \tag{2.18}$$

It is important to note that, even though this method learns two estimates, only one of them is updated in each iteration. Consequently, double learning doubles the memory requirements but it does not increase the amount of computations per step.

Now that tabular solution methods have been covered, discrete problems with limited state and action spaces can be solved. However, for the flight control applications this research focuses on more advanced methods are needed to deal with the enormous state spaces these tasks usually have. As such, the next section will expand on the foundations of more advanced RL methods.

2.2. Approximate Solution Methods Foundations

Whenever the task state (or action) space becomes too large, approximate solution methods need to be implemented. To illustrate this, there are more than 10^{150} possible board configurations in a game of Go (Silver et al., 2016), which is more than the number of atoms in the known universe and, consequently, it is not reasonable to expect the exact optimal value function to be found for those problems. In these cases (where flight control tasks are also included) almost every state encountered will never have been visited before, as such, it is necessary to generalise from previous encounters with states that are in some way similar to the current one. This is exactly what approximate solution methods do, where instead of using look-up tables to find the exact solution they use function approximators.

Subsection 2.2.1 presents the main function approximators that RL methods can use when dealing with large state spaces, whereas subsection 2.2.2 expands on one of the most used function approximators in DRL (neural networks). Finally, subsection 2.2.3 introduces the different strategies that approximate solution RL methods can use to solve a given task.

2.2.1. Function Approximation Methods

By using function approximators it is possible to generalise unknown values to neighbouring known ones. In RL agents these function approximators are used to compute an estimate of the value function (or optimal policy). This subsection describes the most well-known function approximation methods.

It is important to mention that non-parametric methods are not described since they are usually memory based and would require unreasonable amounts of memory when dealing with the flight control task this research focuses on. Consequently, only parametric methods are explored and these can be separated into linear and nonlinear methods.

Linear Methods:

When referring to linear methods in this research, one is referring to the function approximator features and not to the function itself. A function approximator might be linear in its features while the resulting function might be nonlinear due to the way those same features are connected.

Generally, with linear methods it is relatively easy to understand why the output assumes a particular value. Furthermore, these methods are already extensively studied and used by other fields of research. Some of the most well-known linear function approximators are coarse coding, tile coding, polynomials and radial-basis functions (Sutton and Barto, 2018). However, these solutions present several disadvantages: 1) They are not as accurate as nonlinear methods when analysing highly complex tasks such as the flight control ones addressed by this project; and 2) Most of them require knowledge about the environment in order to work. To illustrate this second point, in radial-basis functions (which are the natural generalisation of coarse coding to continuous-valued features) the shape of the features will determine the nature of the generalisation and, as such, environment knowledge is required to guarantee good accuracy. Due to these two reasons linear function approximators are discarded from further research.

Nonlinear Methods:
Nonlinear methods include Artificial Neural Networks (ANNs) trained by backpropagation and variations of Stochastic Gradient Descent (SGD). While these methods are not guaranteed to converge, they gained a lot of attention ever since Cireşan et al. (2012) was able to match human performance on an image recognition task by using Neural Networks (NNs). Since then these methods were applied in a multitude of research fields, including flight control research.

Considering that NNs are the basis for the majority of the algorithms discussed in this literature study, the next subsection presents the basic concepts of these function approximators.

2.2.2. Basic Concepts of Neural Networks

In deep learning, there are mainly three general types of neural networks: Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs):

- ANNs: also known as Feed-Forward Neural Networks. They pass information in one direction, through various nodes until it reaches the output node. In an ANN there are no loops, meaning there is no path in which a node's output can influence its input. The ANN's input is usually given in tabular or text form, and these networks are used for a variety of tasks related to classification and regression.
- CNNs: most commonly used for vision-related tasks. Their building blocks are filters (also called kernels) which are used to extract relevant features from the input using a convolutional operation. When combined with pooling layers, these networks become "shift invariant", i.e., they can detect the same object in different parts of the image.
- RNNs: These networks are modified to be able to consider previously entered data and current data at the same time. This is very useful for sequential tasks like Natural Processing Language, which is the main usage for RNNs.

This research focuses on an algorithm that receives inputs from the aircraft sensors (e.g.: altitude, velocity, etc.) and outputs a control command to the aircraft. To do this, the agent uses a neural network that takes the aircraft states and maps them to an optimal policy or value function. Since this mentioned input is not an image nor the task is a sequence prediction task, CNNs and RNNs are discarded from further research. As such, only ANNs are considered for this project.

ANNs' Architecture and Basic Functioning:
An ANN is composed by an input layer, an output layer and a set of hidden layers. The input layer is an one-to-one mapping of the input vector and, therefore, has as many units (from now on called neurons) as the input size. The hidden layers have a variable size, which represents a hyperparameter of the network. Lastly, the number of output neurons is determined by the output vector size.

Figure 2.2 presents an example of the feed-forward neural network structure described, where the network has m inputs, n outputs and 2 hidden layers. Each link in the network has a real-valued weight associated, and each neuron computes a weighted sum of the input signals it receives. Generally to that weighted sum the neuron applies a nonlinear function called the *activation function*, and the result is called a neuron's output.

The weights for each link are usually initialised with random values in the $[-1, 1]$ range whereas the biases for the weighted sum usually start with a zero value. During the learning phase, the network weights (and biases) are updated so that the network is able to, given a certain input, estimate the correct output.

Activation Functions:
For the network to learn complex data patterns some type of non-linearity is needed in the neurons, and this is accomplished through the use of activation functions. If no activation function was present, then a neuron's output would simply be a linear function of the input. The most common activation functions seen are hyperbolic tangent and Rectified Linear Unit (ReLU) for hidden layers, whereas for the output layer it is common to use a linear or sigmoid activation function.

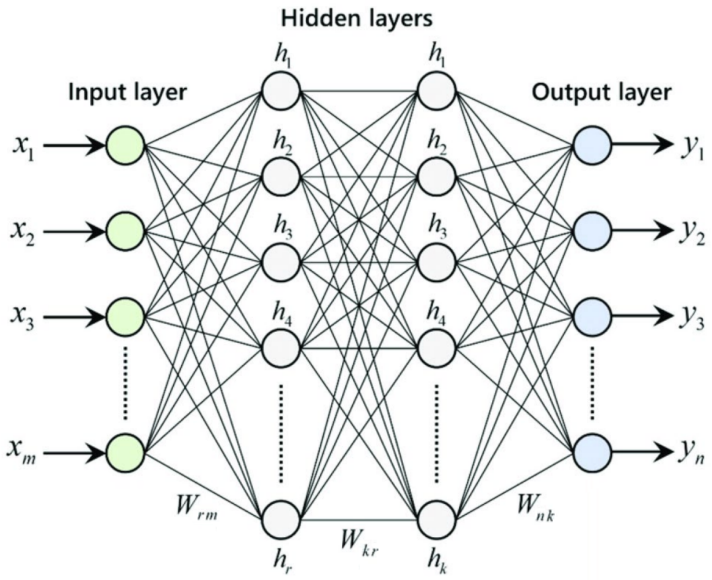


Figure 2.2: ANN's architecture example. Adapted from Esfe et al. (2021).

When considering the possible hidden layer activation functions, Goodfellow et al. (2016) mentions that, even though the hyperbolic tangent function is bounded, it might suffer from exploding and vanishing gradients. To address this, it is common to use the ReLU activation function instead, however, this generally requires more neurons in the network to reach the same accuracy and (due to the fact that it is not bounded) may cause some stability issues.

Loss Function:

For the network weights to converge to an optimal solution during the training phase, they need to be updated in relation to some objective. Generally, this objective is the minimisation of a loss function.

A commonly used loss function is the Mean Squared Error (MSE) between the classification given by the network and the value mentioned in training set. The computation of a single classification loss using the MSE is presented in equation 2.19, where x_i is the input vector, w represents the weights of the network and y_i is the target output vector. The total loss is the average between all classification losses, as seen in equation 2.20.

$$L_i\Big(f(x_i, w), y_i\Big) = \Big(f(x_i, w) - y_i\Big)^2 \tag{2.19}$$

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i\Big(f(x_i, w), y_i\Big) \tag{2.20}$$

The goal is to adjust the weights of the network in a way that minimises the loss function. Given that ANNs are differentiable, to minimise the weights one calculates the gradient of the loss function and then optimises the weights with respect to that gradient.

Backpropagation and Weight Optimisation:

As previously mentioned, the weights of the network are optimised with respect to the gradient of the loss function. Backpropagation is a technique commonly used to calculate this gradient, where the partial derivatives of the loss function are calculated, starting from the output layer to the input layer, using a backward pass. After having this gradient, different optimisation techniques can be used to optimise the NN weights, with SGD being one of the most renown optimisation techniques (Goodfellow et al., 2016).

In SGD, a step is taken in the opposite direction of the loss function gradient and the magnitude of this step is determined by a hyperparameter called learning rate (which is set by the designer). This hyperparameter must be tuned with care since a high learning rate might lead the NN to overshoot the target (and thus never converge), but setting a small learning rate leads the algorithm to take longer to find the optimal solution.

One main drawback that SGD presents is that all weights are updated with the same step size, which is sub-optimal since not all weights need to be changed with the same magnitude. In recent years, new algorithms appeared to fix this SGD disadvantage. The AdaGrad optimisation algorithm uses various learning rates and scales them inversely to the square root of the sum of all their historical squared values. This way, each NN parameter is adjusted according to the effect it has in the loss function (Goodfellow et al., 2016). However, many times, this strategy results in a premature and excessive decrease of the learning rate effectiveness due to the accumulation of squared gradients from the beginning of training.

Root Mean Squared Propagation (RMSProp) modifies the AdaGrad algorithm by changing the gradient accumulation into an exponentially weighted moving average (Goodfellow et al., 2016). RMSProp has shown to be effective while practical, being one of the most renown optimisation algorithms used with deep neural networks. Finally, Adam (which stands for "Adaptive moments") is the best AdaGrad variant seen up to date, and results from the combination of RMSProp with momentum-type optimisers (Kingma and Ba, 2015). According to Goodfellow et al. (2016), in Adam "momentum is incorporated directly as an estimate of the first order moment of the gradient (with exponential weighting)" and this optimiser includes bias correlations to the estimates of the first and second order moments. Its pseudocode is presented in algorithm 2.1. Adam has proved to consistently outperform RMSProp, AdaGrad, and SGD. Consequently, it is the chosen optimiser for this project.

Algorithm 2.1 The Adam algorithm. Adapted from Goodfellow et al. (2016).

Require: Step size ϵ. (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1[$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilisation. (Suggested default: 10^{-8})
Require: Initial Parameters $\boldsymbol{\theta}$
  Initialise 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialise time step t = 0
  while stopping criterion not met **do**
    Sample minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$
    Compute gradient $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \cdot \sum_i L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right)$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Regularisation:

When training the ANN on a set of data the function can: 1) underfit; 2) fit the data correctly; or 3) overfit the data. Overfitting happens when a model learns not only the general patterns of the data, but also the noise in the training set. This happens to a point where, even though the NN perfectly fits the training data, it is not able to make accurate predictions on unseen data which defeats the prediction's purpose.

To avoid overfitting, regularisation techniques are commonly used which make sure that the model does not get overly complex. This way the model remains general and accurate to new data.

Several regularisation techniques exist. In L2 regularisation, all weights are driven closer to the origin so there are fewer outliers and more features are taken into account (instead of overvaluing a single feature). In L1 regularisation some weights are reduced to zero, which removes some features altogether. This is a good strategy for feature selection in problems with a huge number of features. Another option is to use the dropout technique where one randomly removes some NN nodes during training, forcing the system to be simpler. Another option is to use noise robustness, where noise is added to the input so the network becomes more stable to noise variations. Finally, one can use early stopping where instead of optimising towards an optimum weight value w^*, the updates try to reach a regularised optimum $\widetilde{w}$ (Goodfellow et al., 2016). Early stopping limits the overfitting associated with training since it restricts the optimisation procedure to a relatively small volume of parameter space in the neighbourhood of the initial parameter values.

Even though there are more strategies used to prevent overfitting, the methods mentioned above are the ones that deep learning practitioners use the most.

The progressive improvement of computer hardware allowed the development of ANNs to a point where they became useful to various research domains. Reinforcement Learning was no exception, with researchers using ANNs to estimate the value function (or even the agent's policy directly). To do this, networks with more than one hidden layer are generally used, resulting in what is known as a Deep Neural Network (DNN). By combining RL with DNNs, the field of Deep Reinforcement Learning (DRL) appeared.

Before diving into the different DRL methods available, it is useful to first outline the different general approaches that RL approximate solution methods can use to solve a given problem.

2.2.3. Approximate Solution Method Types

As previously stated, the goal of the RL agent is to learn an optimal policy, π^*, and whenever the state (or action) space becomes too large approximate solution methods are needed. A bigger focus was given to ANNs as function approximators since: 1) If given an adequate activation function and network structure, they are able to approximate any real-valued function; and 2) Most of the state-of-the-art DRL algorithms use ANNs.

In this subsection, the different strategies that approximate solution methods can use to solve a task are described. These solutions can either: 1) Learn a value function without learning an explicit policy (i.e. using a critic-only approach); 2) Learn a parameterised policy without learning a value function (i.e. using an actor-only approach); or 3) Learn both (i.e. using an actor-critic approach).

Value-Based Methods (Critic-Only):

Rather than representing the value function as a table (like tabular solution methods), approximate solution value-based methods represent the value function as a parameterised function. To do this, the algorithm uses one of the function approximators previously described. If an ANN is used to approximate the value function then $\hat{v}(s,w) \approx v_w(s)$, with w being the vector that represents the network weights, that is, the value function is estimated by a parameterised function of the network weights.

Value-based methods do not learn any policy explicitly. For the tabular case, one can think of methods like Q-learning and SARSA as algorithms that also did not learn any policy explicitly. In a value-based DRL method, during the learning phase, the weights of the network are optimised in relation to a loss function (e.g. the MSE between the Q-value estimated by the network and the target Q-value). As learning progresses, the estimated Q-values will approximate the target ones and the optimal action for a given state is the one that maximises the estimated value function. As it will be seen in the following sections, the DQN is an example of an approximate solution value-based method.

The main drawback associated with these solutions comes when dealing with continuous action spaces. Since value-based methods do not calculate an explicit policy, they derive the greedy action by selecting the one that maximises the value function. However, in continuous action spaces, these algorithms would have to search through an infinite number of actions to find the greedy one and, consequently, value-based solutions are unsuitable for tasks with continuous action spaces. One way of dealing with this difficulty is by discretising the action space, however, the algorithm continues to be computationally inefficient when compared to methods that include an actor (actor-only or actor-critic solutions).

Policy Gradient Methods (Actor-Only):
Rather than learning a value function and then selecting an action based on the argument that maximises that function, policy gradient methods focus on learning a parameterised policy instead. This parameterised policy can be expressed as $\pi(a_k \mid s_k, \theta) = \pi_\theta(a_k \mid s_k)$, with θ being the parameters of the function approximator. To learn this policy, actor-only methods optimise an objective function $J(\theta)$ with respect to the policy using some sort of optimisation algorithm (e.g. Stochastic Gradient based optimisation). To illustrate this, if the objective function $J(\theta)$ is the value function, then the actor-only method can update its parameters using gradient ascent, as expressed in equation 2.21:

$$\theta_{k+1} = \theta_k + \alpha \cdot \widehat{\nabla J(\theta_k)} \tag{2.21}$$

, where $\widehat{\nabla J(\theta_k)}$ is a "stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument θ_k"(Sutton and Barto, 2018).

Rather than choosing actions based on a value function, these methods choose the action according to the parameterised policy. It is important to note that actor-only methods perform updates at the end of the episode since that is when they have the return on the policy they just acted on.

- Advantages relative to value-based methods:
  - Policy parameterisation allows for tasks with continuous action spaces to be solved.
  - Actor-only methods model the action probabilities, and thus, they are capable of learning stochastic policies. By contrast, valued-based methods do not have a natural way of finding stochastic policies.
  - For a specific problem the policy might be simpler to approximate than the value function, which makes it better to use policy gradient methods for those tasks.

- Disadvantages relative to value-based methods:
  - Since actor-only methods need to wait until the end of the episode to update the parameters of the function approximator, they are not suited for continuous tasks (only episodic ones).
  - Actor-only methods present high variance when estimating the gradient of the discounted return. Essentially, each time the method performs an update, it is using an estimation of the gradient generated by a series of data points collected along the episode (e.g.: $[(s_0, a_0), \ldots, (s_k, a_k)]$). Hence, this estimation can be very noisy which impacts the stability of the algorithm. Nevertheless, some research has been developed to decrease this high variance problem, by for example, introducing a baseline in the computation.
  - Actor-only methods may easily become sample inefficient. These methods base their update in running the policy once, then updating it and throwing away the trajectory. This means that they do not have the ability to reuse old data to improve sample efficiency. Moreover, even if some actions taken were relatively bad but the overall episode score was high, the algorithm will average all actions taken as good. Thus, in order to have an optimal policy, actor-only methods generally require lots of samples.

To address the disadvantages mentioned the actor-critic structure was created, which adds a critic to the actor-only architecture. Adding this critic allows the algorithm to be used in both continuous action spaces and continuous tasks, while also solving the high-variance problem.

Actor-Critic Methods:
Actor-critic methods add a critic to the actor-only structure in order to solve its disadvantages. These solutions use temporal difference learning and bootstrapping (just like value-based methods), which allows them to deal with continuous tasks and to update the model at each time step.

In these methods, the critic receives the state and reward from the environment and computes an estimate of the temporal difference error. Then, the actor chooses an action according to a parameterised policy.

During the training phase, the critic updates the parameterised action-value function, $\hat{Q}_w(s, a)$ $\big($or state-value function $\hat{V}_w(s)$, depending on the algorithm$\big)$ while the actor updates the policy parameters

θ in the direction indicated by the critic. The overall structure of this architecture is presented in figure 2.3. To illustrate this approach, the pseudocode for a Q-function actor-critic method is presented in algorithm 2.2 (Weng, 2018).

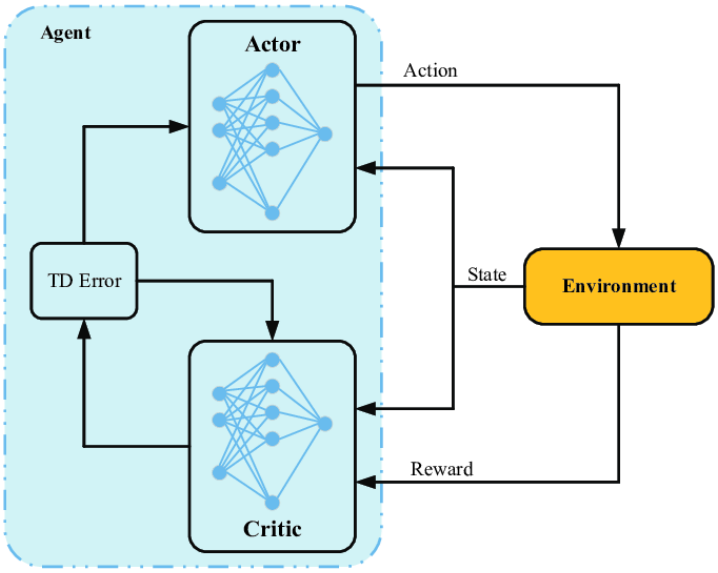


Figure 2.3: Actor-Critic general architecture. Adapted from Giang et al. (2020).

Algorithm 2.2 Simple Q-function Actor-Critic algorithm. Adapted from Weng (2018).

Require: Learning rates α_θ, α_w
  Initialise parameters s, θ, w at random
  Sample initial action $a \sim \pi_\theta(a|s)$
  for $t = 1, \dots, T$ **do**
    Sample next state $s' \sim P(s'|s,a)$ and reward $r_t \sim R(s,a)$
    Sample next action $a' \sim \pi_\theta(a'|s')$
    Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta \ln \pi_\theta(a|s)$
    Compute the correction (TD error) for action-value at time t: $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$
    Use the TD-error to update the Q-function's parameters: $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
    Update $a \leftarrow a'$ and $s \leftarrow s'$
  end for

Algorithm 2.2 shows that both the critic and actor networks are updated in each step. Moreover, there are 2 learning rates, α_θ and α_w, predefined for the policy and value function updates, respectively. This algorithm is on-policy, meaning, the training samples are collected by following the behaviour policy (the same policy that is being optimised), however, there are also off-policy actor-critic methods available. Examples of off-policy actor-critic algorithms are: DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), SAC (Haarnoja et al., 2018), etc. These algorithms are described in the next section of this report.

This section presented the fundamentals of approximate solution methods by not only outlining the available function approximators, but by also exploring the different overall strategies these methods can use. The next section uses these concepts to outline the most relevant DRL solutions for this research.

2.3. Deep Reinforcement Learning

Thanks to the development of ANNs in recent years, DRL (a type of approximate solution method) emerged and gained great popularity throughout the RL research community. These methods use DNNs to estimate a value function and/or policy and proved to be capable of solving a variety of highly complex tasks, ranging from high-dimensional input games (Vinyals et al., 2019) to sophisticated robotic applications (Akkaya et al., 2019).

This section presents the most relevant DRL methods for this research, beginning with value-based DRL methods and then proceeding to policy-based and actor-critic DRL solutions. It finishes with a short note on the different aerospace DRL literature available.

In order to allow for a clear understanding on how the different DRL solutions mentioned in this literature study relate to each other, figure 2.4 presents an overview of all DRL methods discussed in this work.

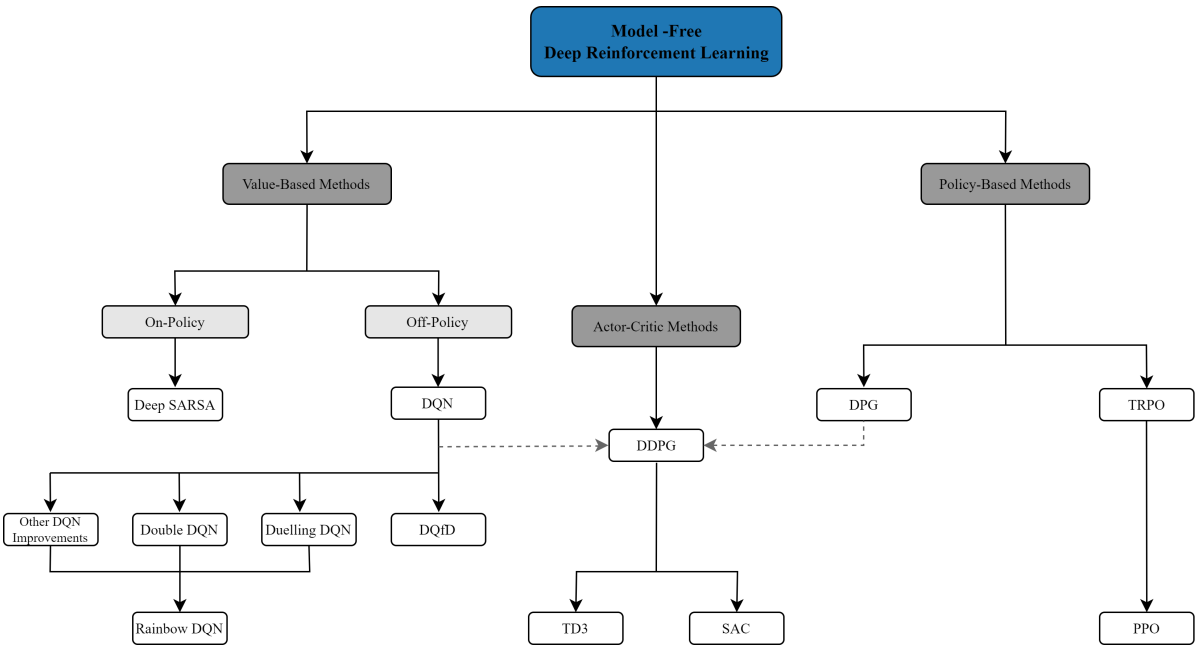


Figure 2.4: Overview of all model-free DRL methods discussed in this section.

2.3.1. DRL Value-Based Methods

DRL value-based methods estimate a value function through the use of DNNs, and then chose the action that maximises the mentioned value function. These method's main drawback relates to their inability to deal with continuous action spaces, as it was already referred in subsection 2.2.3. To deal with this difficulty one can discretise the action space, however, in that case the algorithm is still usually computationally inefficient when compared to methods that include an actor.

Deep Q-Network (DQN):

DQN gained great popularity because it marked the beginning of the DRL field by being the first model-free RL-based algorithm to successfully learn control policies directly from an high-dimensional input (Mnih et al., 2015). Essentially, DQN takes the previously described Q-learning algorithm (off-policy tabular solution) but rather than using a look-up table to approximate the state-action value function, it uses a Deep Neural Network (DNN). This neural network is called Q-network which, for a given state and action, outputs an estimate of the value function.

According to Mnih et al. (2015), two features were essential for DQN's success in solving high-dimensional input tasks effectively, experience replay and a target network (described below). The DQN's pseudocode can be seen in Algorithm 2.3.

Experience Replay:

When using experience replay, transitions $e_t = (s_t, a_t, r_t, s_{t+1})$ are temporarily stored in a memory-buffer and mini-batches of fixed size are extracted randomly from that buffer to perform the Q-network updates. When the memory-buffer reaches its limit, the oldest sample is removed from the buffer and the new one is added.

According to Mnih et al. (2015), this presents many advantages. First, each transition can potentially be used in many of the Q-network updates, allowing for greater sample efficiency. Second, it minimises correlations between the samples which reduces the variance of the updates. Third, it increases the algorithm stability since the behaviour distribution is averaged over many of its previous states. Using experience replay forces this algorithm to learn in an off-policy way, which justifies the choice for Q-learning as its update rule.

Since DQN's first publication, experience replay has been further improved with the creation of prioritised experience replay (Schaul et al., 2016). Prioritised experience replay samples more frequently the transitions with higher TD error, which means that learning is intensified in the regions where the algorithm performs worse. This improved technique outperformed uniform experience replay in most of the test cases presented by Schaul et al. (2016).

Target Network:

DQN includes a separate target network to compute the target y_j in Algorithm 2.3. The weights of this extra network are not trained, instead they are periodically synchronised with the weights from the main Q-network, which allows the later to move in the direction of a steady target between updates. Without this feature, y_j would change too rapidly which would lead to training instabilities.

Algorithm 2.3 The DQN algorithm. Adapted from Mnih et al. (2015).

Initialise replay memory $\mathcal{D}$ to capacity N
Initialise action-value function Q with random weights θ
Initialise target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
for episode $= 1, \ldots, M$ **do**
  Initialise episode sequence with initial state s_1
  for $t = 1, \ldots, T$ **do**
    With probability ϵ select a random action a_t, otherwise select $a_t = \arg\max_a Q_\theta(s, a)$
    Execute action a_t, observe reward signal r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{,if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}_\theta(s_{j+1}, a') & \text{,otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q_\theta(s_j, a_j)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

Double Deep Q-Network (Double DQN):

The concept of double learning described in subsection 2.1.2 can be integrated in value-based DRL methods to create the Double DQN algorithm (van Hasselt et al., 2016). Normal DQN (as it was already seen with Q-learning for the tabular case) presents an overestimation bias problem caused by using the max operator over the estimated value function. Double DQN addresses this issue by decoupling the selection and evaluation of the bootstrap action.

This algorithm introduces a second independent Q-network, which means that it has two separate Q-value estimators, each of which is used to update the other. According to van Hasselt et al. (2016), double DQN reduces the observed overestimation bias and leads to better performance results on most of the tested Atari 2600 games.

Nevertheless, according to Hessel et al. (2018) there is no thorough proof (theoretical or experi-

mental) that double DQN performs better than normal DQN for the majority of tasks. It is proven that on some tasks double DQN performs better, however, without theoretical or experimental proof one cannot generalise such statement for the majority of tasks at this point in time. The same cannot be said about prioritised experience replay which is known to perform better than uniform experience replay for most situations (Schaul et al., 2016).

Dueling Network Architecture:

Wang et al. (2016) introduced the dueling network architecture, which is a structure designed for value-based RL that has two separate estimators: one for the state value function, and another for the state-dependent action advantage function, $A(s,a) = Q(s,a) - V(s)$. This action advantage represents the benefit of choosing "a" over the action suggested by the policy at state "s". Both networks share a convolutional encoder and are then merged with a special aggregator as described by figure 2.5.

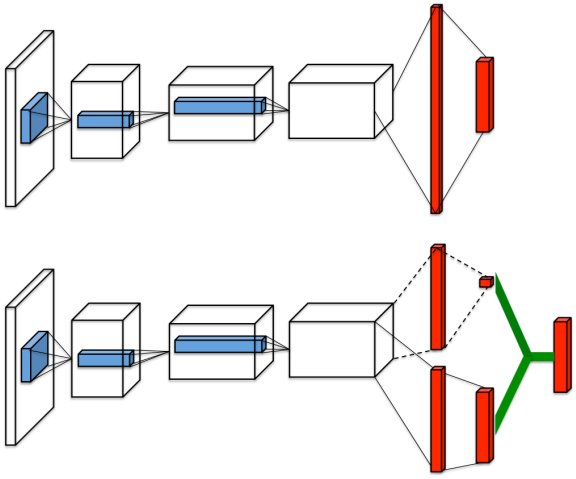


Figure 2.5: The normal Q-network architecture (top) and the dueling Q-Network architecture (bottom). Both networks output Q-values for each action. Adapted from Wang et al. (2016).

The key motivation for the dueling architecture is that, for some tasks, it is unnecessary to know the value of each action at every timestep. By explicitly separating the two estimators, this architecture can learn which states are (or are not) valuable without having to learn the effect of each action in each state. In Wang et al. (2016), Dueling DQN achieves state-of-the-art performance in the Atari 2600 domain by outperforming both the normal DQN and Double DQN with prioritised experience replay. The advantage of this method lies in its ability to generalise learning across actions, which is especially relevant as the number of actions in the task grows (Hessel et al., 2018).

Rainbow:

In Hessel et al. (2018) researchers analysed several improvements made to DQN over time, including the ones previously presented (i.e. prioritised experience replay, double DQN and dueling DQN). The main motivation for this paper was that until its release, researchers made several independent improvements to DQN but it was unclear which extensions were complementary and which could be successfully combined.

In this paper, researchers created an algorithm that combined all previous well-known DQN improvements into a single integrated agent called Rainbow. The specific details of this implementation can be found in Hessel et al. (2018). The results were surprising with Rainbow achieving new state-of-the-art results for 57 Atari 2600 games, both in terms of data efficiency and final performance. With only 44 million frames, the Rainbow agent outperformed any other baseline algorithm, even if they ran for more than 200 million frames (refer to figure 2.6). If a value-based method is chosen to solve the attitude flight control task this research focuses on, then including some of the Rainbow's features might be beneficial to the solution's overall performance.

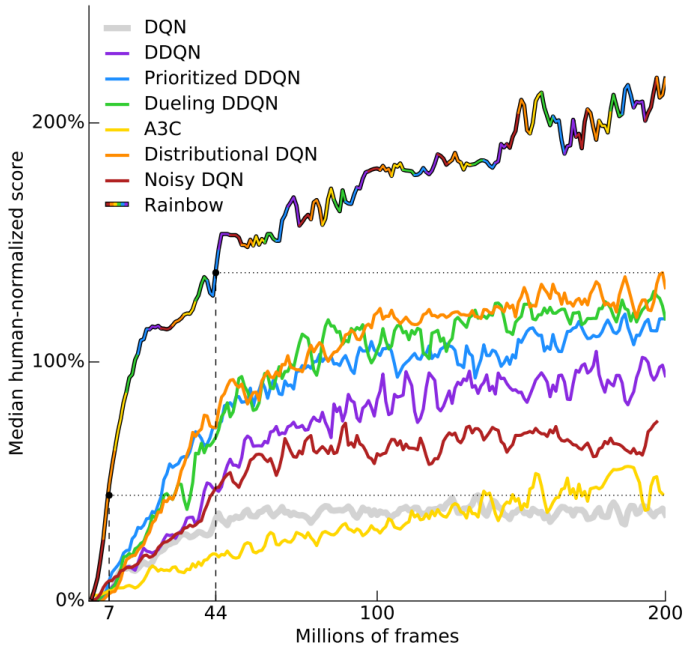


Figure 2.6: Median human-normalised performance across 57 Atari games of various DQN baselines and Rainbow integrated agent. Adapted from Hessel et al. (2018).

Deep Q-Learning from Demonstrations (DQfD):

Even though the previous DRL algorithms achieved major successes in dealing with difficult decision-making problems, they typically require a huge amount of samples before reaching the required results. As an example, in figure 2.6 it is noticeable that the different DQN baselines need millions of frames before reaching human-like performance. In some use-cases this can be a problem.

Hester et al. (2018) proposed an alternative method called Deep Q-Learning from Demonstrations (DQfD) where, through the use of supervised learning and TD, the agent builds an initial policy and value function based on demonstrations. The supervised loss enables the algorithm to learn how to imitate the demonstrator, while TD enables it to learn a value function from which it can continue learning after the demonstration period. When the initial training ends, the agent starts interacting with the environment by using the learned policy and makes improvements on it. DQfD achieves better performance on the first million steps on 41 of the 42 Atari games analysed. Moreover, it takes 83 million steps for Prioritised Dueling Deep Q-Networks to catch up to DQfD's performance, which proves the great sample efficiency achieved by this method.

This might also be an interesting algorithm for explainability purposes since, as it will be shown in the following sections, human's understanding of RL methods depend on the mental model that the user has about the agent (Anderson et al., 2020). Through mimicking the human rationale during the demonstration phase and then improving on the policy produced, this method can generate policies that are inherently easier for humans to understand.

2.3.2. DRL Policy Gradient and Actor-Critic Methods

With the objective of extending DRL to continuous action spaces, various methods that include an actor were created. Specifically, actor-critic methods use two separate learners, one critic and one actor. The critic receives the state and reward from the environment to compute the temporal difference error while the actor chooses an action according to a parameterised policy. The most renown DRL actor-critic methods are presented in this subsection.

Deep Deterministic Policy Gradient (DDPG):

Deep Deterministic Policy Gradient (DDPG) is a model-free off-policy algorithm used for solving tasks with continuous action spaces (Lillicrap et al., 2016). It combines the actor-critic approach seen in the Deep Policy Gradient algorithm (Silver et al., 2014) with DQN's learning method. DDPG's pseudocode is presented in Algorithm 2.4, where it is important to note that both the actor and the critic are DNNs.

Similarly to DQN, DDPG uses experience replay and a target network to minimise correlations between samples and stabilize the DNN's training, however, the target network is adapted to deal with the new actor-critic structure. As such, both the actor and the critic are attributed a target network but instead of directly copying the weights from the main networks (like DQN), DDPG introduces the concept of soft target updates, which greatly improves stability. The weights of each target network are updated in a way that allows them to slowly track the weights from the main network: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, with $\tau \ll 1$.

Finally, DDPG uses batch normalisation to deal with the fact that different components of the state vector might be expressed in different units, leading the components to have different orders of magnitude (e.g.: angles are usually much smaller than positions). With batch normalisation each dimension across the samples in a minibatch has unit mean and variance, and the covariance shift is minimised by applying a running average of the mean and variance. By using batch normalisation, DDPG learned effectively across different tasks without needing to manually ensure that units were within range.

One of the main faults pointed to DDPG is its tendency to overestimate the value function. Several posterior methods successfully attempted to diminish this issue (Fujimoto et al., 2018; Haarnoja et al., 2018). Furthermore, even though DDPG is considered an offline method, there are several applications where researchers successfully implemented this algorithm partially online (Xu et al., 2018).

Algorithm 2.4 The DDPG algorithm. Adapted from Lillicrap et al. (2016).

Randomly initialise critic $Q(s, a \mid \theta^Q)$ and actor $\mu(s|\theta^\mu)$ networks with weights θ^Q and θ^μ.
Initialise target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$.
Initialise replay buffer $\mathcal{R}$
for episode $= 1, \ldots, M$ **do**
  Initialise a random process $\mathcal{N}$ for action exploration.
  Receive initial observation state s_1
  for $t = 1, \ldots, T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise.
    Execute action a_t, observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$.
    Set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}) \mid \theta^{Q'}\right)$
    Update critic by minimising the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$
    Update the actor policy using the sampled policy gradient:
      $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a|\theta^Q\right)\big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)\big|_{s_i}$
    Update the target networks:
      $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
      $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for

Trust Region Policy Optimisation:

The previously mentioned policy gradient methods kept new and old policies close in the parameter space by using a constant step in each policy update (with a length defined by the user). However, even small variations in policy can produce large differences in performance and, thus, a step wrongly taken can have a catastrophic effect in performance. Consequently designers need to be very careful and choose small step sizes in policy gradient methods, which hurts sample efficiency.

Schulman et al. (2015) presented the Trust Region Policy Optimisation (TRPO) algorithm which tries to avoid this issue by, in the updates, taking the largest step possible that improves performance while

satisfying a special constraint (Kullback–Leibler Divergence) on how close the new and old policies are allowed to be. TRPO is an on-policy gradient-based method that achieved robust performance in a variety of tasks, however, it is hard to implement and still suffered from some sample efficiency problems (Schulman et al., 2017).

Proximal Policy Optimisation:

To address the problems associated with TRPO, Schulman et al. (2017) proposed another algorithm named Proximal Policy Optimisation (PPO) that is simpler to implement, faster and more sample efficient, while retaining the benefits of trust region solutions. Just like its predecessor, PPO is an on-policy algorithm that can be used for both discrete and continuous action spaces. This method focuses on the same problem as TRPO, that is, in taking the biggest possible improvement step without causing a policy performance collapse. However, rather than being a complex second-order method like TRPO, PPO uses first-order strategies along with other features to keep the algorithm from taking a too big of a step (OpenAI, 2018a). There are mainly two variants of PPO: PPO-Penalty and PPO-clip.

- PPO-Penalty: uses the same strategy as TRPO and computes an approximate solution to a KL-constrained update. However, instead of making it a hard constraint, PPO-Penalty penalises the KL-divergence in the objective function and "automatically adjusts the penalty coefficient over the course of training so it's scaled appropriately"(OpenAI, 2018a).

- PPO-Clip: does not use KL-divergence at all. Instead, it relies on clipping the objective function which removes incentives for the new policy to get too far from the old one.

Both variants achieve better performance in data efficiency, computational speed and overall performance than its predecessor TRPO. Since trust region methods are different than any other strategy previously presented in this research, the pseudocode for PPO-clip is presented in Algorithm 2.5.

Algorithm 2.5 The PPO-Clip algorithm. Adapted from OpenAI (2018a).

Require: Initial policy parameters θ_0, and initial value function parameters ϕ_0

for k $= 0, 1, 2, \ldots,$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute the advantage estimates $\hat{A}_t$ (using any advantage estimation strategy) based on the current value function V_{ϕ_k}.

Update the policy by maximising the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t) \ , \ g\left(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)\right)\right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Soft Actor-Critic (SAC) and Twin-Delayed Deep Deterministic Policy Gradient (TD3):

To address the DDPG's disadvantages previously described several methods were developed. Two main algorithms emerged from this, Twin-Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) (Fujimoto et al., 2018; Haarnoja et al., 2018). Just like DDPG, these solutions are off-policy and can only be used in environments with continuous action spaces.

In order to address DDPG's overestimation problem, TD3 introduces three critical features. Firstly, it learns from two Q-functions instead of one and uses the smaller estimated value to produce the target in the updates (i.e. using a double learning strategy). Secondly, it updates the policy less frequently than the Q-function which makes sure that the Q-function error is first reduced before using it to update

the policy. Finally, TD3 adds noise to the target action, reducing the chance of choosing overestimated Q-values. According to Fujimoto et al. (2018), this algorithm outperforms DDPG in several control tasks, in both overall return and learning speed.

Alternatively, around the same time that TD3 was published, Haarnoja et al. (2018) proposed another solution named Soft Actor-Critic (SAC). This method optimises a stochastic policy in an off-policy way, which is a mid-way between stochastic policy optimisation and the DDPG approach (OpenAI, 2018b). In SAC the policy tries to maximise a trade-off between the expected return and an entropy parameter, where this entropy is a measure of the randomness in the policy. Similarly to the ϵ-greedy approach, increasing the entropy parameter leads to more exploration. Furthermore, SAC includes clipped double Q-learning and uses an inherently stochastic policy (caused by using the entropy parameter in the policy update) leading to an effect that mimics target policy smoothing. When compared to DDPG, Haarnoja et al. (2018) show that SAC presents a more stable learning process with increased average return and sample efficiency on several control tasks.

In this subsection several algorithms were presented. However, to the author's best knowledge, it is still unclear which of these state-of-the-art policy-gradient methods performs best. Research suggests that comparison between these methods may be biased by: 1) A lack of hyper-parameter tuning, 2) Stochasticity properties of the environment, or 3) Auxiliary details in the algorithm implementation. All of these factors can have a major impact on the method's performance (Engstrom et al., 2020 ; Henderson et al., 2017). As such, the best policy gradient method to use (i.e.: PPO , TD3 or SAC) may depend on the specific application. Since this research focuses on explainability, if a policy gradient method is chosen (instead of a valued-based one), the decision as to which method to use will be based on how easily the XRL solution can adapt itself to the method.

2.3.3. DRL in the Flight Control Domain

In the previous subsections, the main DRL methods available were presented and their major strengths discussed. However, it is important to mention some DRL applications inside the aerospace field since the main practical case-study of this research is an aerospace application. Even though DRL methods have proven their ability to solve highly complex control tasks, current DRL aeronautical applications are mainly focused at small-scale UAVs, which is largely related to the opaqueness and inexplicability associated with most of these solutions (Xie et al., 2021).

In Tang and Lai (2020), DDPG was applied to a fixed-wing aircraft to perform automatic landing in a simulated environment. The training phase was performed with normal flight conditions and the results were validated by using various wind disturbances. Overall, the algorithm presented good results while being robust to the wind disturbances. However, this research did not use the full aircraft model, instead it used a simplified one that only takes into account the aircraft's longitudinal states.

To the author's best knowledge, in order to encounter a DDPG application that uses coupled dynamics, one has to look into UAV research. In Tsourdos et al. (2019), all 6 DoF are considered and coupled dynamics are included. In this work, while tracking single angle changes (e.g. pitch angle) the algorithm's performance was relatively good with only small steady-state errors, however, when changes happened in all 3 angles at the same time (i.e. pitch, roll and yaw angles) large steady state errors were registered.

In Bøhn et al. (2019), PPO is used to control an unmanned flying-wing model with coupled dynamics. The authors justify the PPO's choice by stating that it presents superior performance in quadcopter attitude control when compared to DDPG. Like in the previous example, the agent is trained in a disturbance-free environment, but it is tested under severe wind and turbulence conditions. The trained agent accurately follows attitude references while being robust to disturbances, however, when compared to a PID controller this agent shows only similar overall performance.

Finally, in Dally and Kampen (2021) an offline cascaded SAC controller is implemented inside a Cessna Citation 500 flight control system, using a high-fidelity simulator model with coupled dynamics. Results show that the trained agent not only completes highly coupled manoeuvres, but is also robust to various failure cases (e.g. jammed rudder, aileron effectiveness reduced, atmospheric disturbances, etc.). During training the agent was not exposed to these disturbances, which demonstrates the robustness of the controller. This is an improvement compared to model-based solutions that usually have a difficulty in keeping control of the aircraft during failures, and shows the plausible power of model-free DRL in flight control applications.

2.3.4. Concluding Remarks

In this section, different state-of-the-art DRL methods were presented, however, when attempting to choose a suitable algorithm for this project one has to take into account that the main research goal is to increase end-user acceptance of RL algorithms in flight control. This means that a suitable algorithm does not only need to be capable of solving an attitude flight control task, but also needs to be combined with a XRL method in a way that increases explainability.

As such, before choosing a DRL method based on this section, one has to analyse the different XRL strategies available. Only after this analysis will a solution be picked based on the trade-off between solving the attitude task efficiently and increasing explainability for end-users. The next section will present the main XRL methods available.

2.4. Explainable Reinforcement Learning

In the previous sections, different DRL methods were analysed and the impressive results that these algorithms are capable to achieve were discussed. However, as pointed out by Puiutta and Veith (2020), the more powerful and robust these methods become, the more opaque they usually are. This is generally referred to as the trade-off between model interpretability and model accuracy/performance (Rudin, 2019 ; Arrieta et al., 2020). According to Xie et al. (2021) this is one of the reasons why DRL methods are not yet present in practical aeronautical applications like ATM services and flight control. To address this trade-off and augment RL explainability is the main goal of the Explainable Reinforcement Learning (XRL) field.

First, this section outlines the importance of explainability. After that, the main XRL concepts are defined in order to provide the reader with the necessary tools to understand XRL algorithms (refer to subsection 2.4.2). In subsection 2.4.3, different XRL methods are discussed and their advantages (and disadvantages) are outlined. Finally, subsection 2.4.4 discusses what XRL applications exist inside the aeronautical domain.

2.4.1. The Importance of Explainability

Before outlining the different taxonomies and methods available inside XRL, it is important to understand why explainability is important. As previously mentioned, even though DRL has shown the ability to solve highly complex control tasks, due to the fact that these methods suffer from a lack of explainability, they end up not being used in several industrial areas of interest. Furthermore, increasing explainability can also aid in detecting bugs and improving the overall method, since it helps developers understand aspects of the algorithm that were previously hidden behind the opaqueness of the solution.

Finally, while explainability starts to be well developed for standard supervised learning methods (Arrieta et al., 2020), XRL research is scarce (Heuillet et al., 2021). Consequently, developing the field of XRL is of primary importance for users to be able to take advantage of the many impressive results achieved by DRL solutions.

2.4.2. Useful XRL Concepts

Currently there is no uniform XRL terminology used in the literature, which according to Heuillet et al. (2021) is caused by the newness of this research field. Consequently, authors refer to different terms (e.g. explainability, understandability and interpretability) to describe the same concept. Moreover, some authors refer to transparency as a synonym to comprehensibility (Lipton, 2016), while others draw a distinction between these two terms (Doran et al., 2017). In order to clearly describe how various XRL solutions differ from one another and what are their advantages/disadvantages, it is important to define some of the concepts that are going to be used in this research.

In this work, the terms interpretability and explainability are used to refer to the ”ability to not only extract or generate explanations for the decisions of the model, but also to present this information in a way that is understandable by human users” (Puiutta and Veith, 2020).

Different XRL Audiences:

As outlined in Heuillet et al. (2021), increasing a method's explainability can involve different goals depending on the target audience (i.e. the public to whom the explanation is addressed at). Some of these goals are causality, trust, interactivity, etc. (refer to Arrieta et al., 2020 for the formal definition of these terms). These goals have to be taken into account when generating the explanation since the explanation type may vary depending on the objective, and not all goals are relevant to all audiences.

In table 2.1 the different possible XRL target audiences are presented, and the explainability objectives that the explanation needs to comply to depending on the audience are described (based on Arrieta et al., 2020). By looking at the mentioned table (and by considering this research's objective), one can conclude that the target audience in this project is made of experts in the aeronautical domain, that is, the pilots and every other group that might interact with the flight control system.

Generally, expertise in Artificial Intelligence (AI) is not a requisite for experts in the aeronautical domain. As such, for simplicity, the experts target audience will be called "end-users" from this point onward. Based on table 2.1, it can be said that the XRL solution chosen for this project should focus on trustworthiness, causality, transferability, confidence and interactivity.

Table 2.1: Target audiences in XRL and the different explainability objectives depending on the audience profile. Adapted from Arrieta et al. (2020).

Target Audience	**Description**	**Explainability Purposes**	**Pursued Goals**
Experts (In this research, this audience type is called *end-users*)	Domain experts, model users (e.g.: medical doctors, insurance agents)	Trust in the model, gain scientific knowledge	Trustworthiness, causality, transferability, confidence, interactivity
Users	Users affected by the model decisions	Understand their situation, verify fair decisions	Trustworthiness, informativeness, fairness, accessibility, interactivity, privacy awareness
Developers	Developers, researchers, data scientists, product owners, etc.	Ensure and improve product efficiency, research new functionalities, etc.	Transferability, informativeness, confidence
Executives	Managers, executive board members, etc.	Assess regulatory compliance, understand corporate applications, etc.	Causality, informativeness, confidence
Regulation	Regulatory entities/agencies	Certify model compliance with legislation, audits, etc.	Causality, informativeness, confidence, fairness, privacy awareness

How to evaluate explanations:
In a broad sense, metrics are usually needed to evaluate a method since they allow one to compare the method's performance against a standard. As an example, the total reward obtained and the robustness to disturbances were metrics that this research used to compare different RL algorithms. However, according to Heuillet et al. (2021), comparing XRL solutions is a much more difficult task due to several reasons:

1. As previously stated, there is no official XRL nomenclature that researchers adhere to.
2. The explanation is always relative to a certain audience. Consequently, one has to make sure that all subjects in the evaluation phase belong to the same target audience, which might be hard.
3. The quality of an explanation is not quantitative, but qualitative and subjective. Moreover, there is an individual variability present since two humans can reach different levels of understanding with the same explanation.

To address these problems, user studies are usually conducted in order to evaluate explanations since they allow the conversion of qualitative results into quantitative ones. When applied to end-users, Doshi-Velez and Kim (2017) suggest that the evaluation should focus on assessing the quality of the mental model built by the user after seeing the explanation. Mental models are "internal representations, built upon experiences, and which allow to mentally simulate how something works in the real world" (Heuillet et al., 2021).

To assess the quality of a user's mental model, Hoffman et al. (2018) suggest to focus on two aspects:

- Ask post-task questions about the agent's behaviour (e.g.: How does it work? What does it achieve?)
- Ask participants to make predictions on the agent's next action.

These evaluations are often done using the Likert scales (Heuillet et al., 2021). It is important to note that performing user studies to evaluate a method's interpretability exceeds the scope of this research and is left as a future research recommendation.

2.4.3. State-of-the-art XRL Methods

As previously stated, increasing DRL's interpretability is an essential condition for these solutions to be accepted in several industrial areas (e.g.: medicine, aerospace, etc.). The most relevant XRL methods for this project are discussed below, where only methods that are suitable for end-users are described. The XRL method for the attitude flight control task will be selected based on the trade-off between solving the attitude task efficiently and increasing interpretability for end-users.

To allow for a clearer understanding on how the different XRL solutions discussed in this section relate to each other, figure 2.7 presents an overview of all XRL methods covered in this research.

XRL through a Causal Lens:
In Madumal et al. (2020) causal relationships are used to explain model-free RL agents. The authors justify their choice for a causal model by stating that humans develop and deploy causal models themselves to explain the world around them. In this work, the structural causal model presented in Halpern and Pearl (2005) is extended to include the agent's actions, which creates the *action influence model*. In essence, the action influence model represents the world by using random variables where some of these variables might exert a causal influence over others. This causal influence is described by a set of structural equations called causal relationships (Puiutta and Veith, 2020).

The total *action influence model* is a tuple containing all state-actions ensemble and the set of causal relationships. More precisely, this model offers 'actuals' (providing explanations to why the agent took a certain action) and 'counterfactuals' (which provides explanations for why the agent did not take a certain action). Consequently, this solution not only considers the actions selected by the agent, but also hypothetical actions (and justifies why the agent did not select those actions).

In Madumal et al. (2020), three steps are needed to generate explanations:

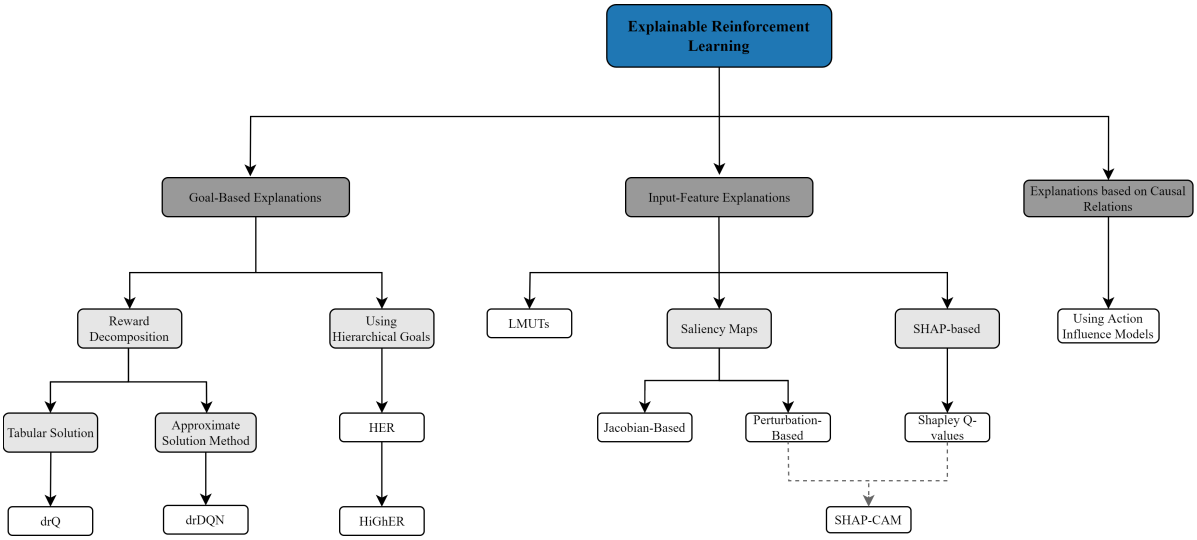


Figure 2.7: Overview of all XRL methods discussed in this section.

1. Define the qualitative causal relationships in the form of a directed acyclic graph (refer to figure 2.8 for an acyclic graph example).

2. Use a multivariate regression model to discover the approximate structural equations.

   · Since uncovering the true structural equations that describe the relationships between variables is hard, this method approximates the structural equations in a way that they are just exact enough to produce the counterfactuals.

3. Generate explanations by traversing the action influence graph from the root to the leaf reward node. Explain a given actual (e.g. Why action A?) by including the counterfactual (e.g. "Why not action B") and by using the structural equations to find the differences between the two.

To not overwhelm the user with information, *minimally complete contrastive explanations* are used, where instead of including all nodes in an explanation the XRL agent only presents the essential ones for user interpretability.

Figure 2.8 describes an explanation generated by this method for a Starcraft II RL agent. Starcraft II is a real-time strategy game with a large discrete state and action space. To present the explanation, the *action influence model* was reduced to four actions and nine state variables. The causal chain for action A_s is depicted in bold arrows whereas the extracted causes for that action are depicted in darkened nodes. The counterfactual (Why not A_b?) is shown in node B. In this scenario, the user wants an explanation for why the agent selected action A_s, and the system answers by stating (refer to figure 2.8) "Because it is more desirable to do action $build\_supply\_depot\,(A_s)$ in order to have more $Supply\_depot\,(S)$ as the goal is to have more $Destroyed\_units\,(D_u)$ and $Destroyed\_buildings\,(D_b)$".

The results presented in this work are a practical and impressive application of causal models to RL. One of the advantages presented by this method is that it is able to adapt itself to several DRL solutions (e.g. DQN, DDPG, SAC). Nevertheless, after being evaluated in a user study, it was found that this algorithm does not increase trust in the model when compared to an agent that did not present explanations. The authors suggest that in order to increase trust, more interaction between the user and the agent is needed.

Furthermore, when applied to the BipedalWalker task (which has a continuous action space) this XRL method was not able to reach a relevant performance. Meaningful performance was only achieved in pure discrete environments. Moreover, this solution needs a clear causal structure present in order to work, something that is not always available beforehand.

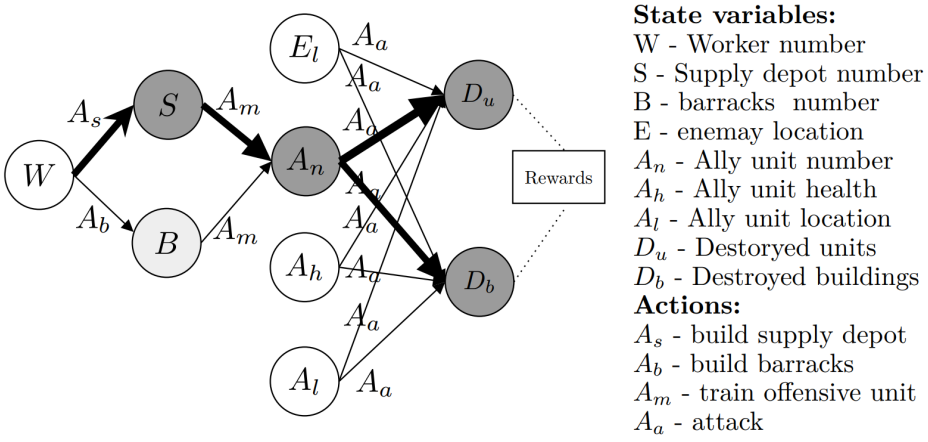


Figure 2.8: Action Influence Graph for a Starcraft II agent. Adapted from Madumal et al. (2020).

XRL based on Hierarchical Goals:

There are several XRL solutions that use Hierarchical Reinforcement Learning and sub-task decomposition to improve explainability (Heuillet et al., 2021). Essentially, Hierarchical RL consists of a high-level agent that divides the task's main goal into sub-goals, and then these sub-goals are pursued by lower-level agents. By solving all the lower-level tasks, the algorithm also solves the high-level one. The idea behind using Hierarchical RL for explainability is based on the premise that by learning what these sub-tasks are, the high-level agent usually forms a representation of the environment that can be made interpretable to humans.

In Andrychowicz et al. (2017), the authors used this idea together with Hierarchical Experience Replay (HER) to solve robotic manipulation tasks that involved grasping and moving an item. In this work, the high-level agent learns a representation of both the environment and the action-value function. The action-value function is then represented as an heat map that illustrates the different Q-values associated with each region of the environment space. By using this heat map, the user understands which areas lead to the biggest reward and is able to predict the sub-goal associated. Even though this method presented a good performance, giving the explanation as an heat-map decreases the interpretability potential of this solution. Furthermore, HER assumes that a mapping between states and goals is available, which might not always be the case since the goal space might differ from the state space and the mapping function might be unknown (Cideron et al., 2019). Finally, HER requires an external human expert to relabel episodes during the training phase, which is undesirable.

To improve on the drawbacks of HER, Cideron et al. (2019) propose the "Hindsight Generation for Experience Replay (HIGhER)" method. This solution replaces the heat maps by a natural language setting that allows the agent to learn from past experiences and to map goals to trajectories, without needing an external human expert.

During training, HIGhER simultaneously learns the language-goal mapping and the navigation policy by interacting with the environment. Furthermore, this solution uses both positive and negative trajectories to learn the mapping function by simply relabelling the language goals when dealing with a negative trajectory. Figure 2.9 describes a section of the HIGhER training process, where it is possible to see that if a positive trajectory is detected then it is added to the replay buffer and to the goal mapper dataset, however, when dealing with a negative trajectory the goal mapper automatically relabels the episode and both trajectories are added to the replay buffer. Consequently, there is no need for an external human expert to relabel trajectories. Finally, the explanation is supplied in a natural language form and, as such, can easily be interpreted by a human user.

This solution successfully solves different BabyAI environment tasks, presenting a clear improvement over other RL baselines while being robust to noise. However, it generally performs worse than the original HER algorithm.

The authors recommend that HIGhER should be used to complement other explanation methods and not as the main explainability solution. When applied by itself, HIGhER does not improve causality and neither does it show why the agent is opting for a certain action instead of another. Furthermore,

there is no explanation regarding the environment features in which the agent focuses on to reach a decision. Consequently, this algorithm will not be considered for the final research.

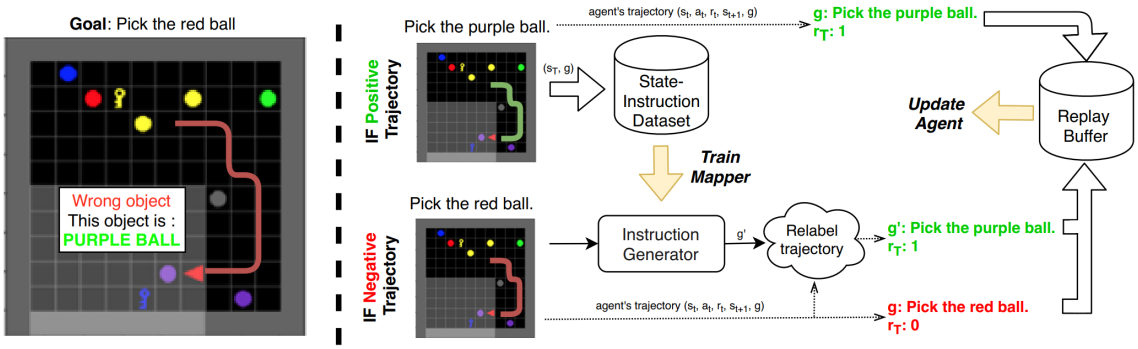


Figure 2.9: HIGhER section of the training procedure scheme. Adapted from Cideron et al. (2019).

XRL through using Linear Model U-Trees (LMUTs):

As previously mentioned, many DRL methods use a DNN to estimate an action-value function, which is a function that contains a lot of implicit knowledge about the environment but is often not analysed in detail. In Liu et al. (2018) a mimic learning framework based on SGD is used to approximate the predictions of the Q-Network by using Linear Model U-Trees (LMUTs). LMUTs are an extension of a previous mimicking framework proposed by Uther and Veloso (1998) named Continuous U-Trees (CUTs) which were also capable of approximating continuous functions. However, while CUTs used constants at the leaf nodes, LMUTs use a linear model which greatly improves their generalisation capability while reducing the number of leaves required in the model. Consequently, LMUTs are simpler and more interpretable to humans.

In Che et al. (2016) tree models were already used to increase interpretability in supervised learning, however, LMUTs are the first solution to adapt this mimicking framework to DRL. Each LMUT leaf node defines a partition cell of the environment input space, which can be interpreted as a discrete state for the decision process. Moreover, each partition cell stores the reward and transition probabilities of performing a certain action in that state, and thus, (in essence) LMUT builds a MDP in each cell.

At the end of the training phase the method will have an ensemble of linear Q-functions, where each leaf node has a single linear model that describes the value function in that section of the input space (Liu et al., 2018). With this ensemble, the LMUT can predict the NN's output for each state. As an example, if it wants to predict the value function for state s_k, it just determines in which leaf node is that state represented and then uses the leaf node's linear model to predict the Q-value. Since this prediction is generated with a linear model, it can be easily interpreted by analysing the feature weights of that model.

In order for LMUT to accurately predict the NN's outputs, the training is split into two phases:

1. Data Gathering Phase: episode transitions are collected and then distributed along the leaf nodes according to the partitions that were made up to that point. This prepares the algorithm for fitting linear models and splitting nodes.

2. Node Splitting Phase: LMUT scans the leaf nodes and updates the different linear models by using SGD. In case SGD does not achieve sufficient improvement on a certain node, then the LMUT determines a new split (dividing the input space into further partitions) and adds the resulting leaves to the current partition cell. As a splitting criterion (i.e. how to find the split in the input space that leads to most improvement) the authors suggest splitting the nodes in a way that generates child nodes whose Q-values contain the least variance.

This solution can be used both offline or online. If used online, the mimic learner gains insight on how to mimic the agent while this agent is ongoing interaction with the environment.

LMUT is considered a transparent method, that is, it is explainable by itself without needing any external processing. Nevertheless, the authors suggest three techniques to extract more information from the model and further increase explainability: 1) Analyse feature importance; 2) Rule extraction; and 3) Highlight super-pixels in the image input that show which segments of the input space are the most important for the agent's decision.

To understand feature importance it is important to know that the LMUT estimated Q-value is calculated by multiplying the linear model weights with the input feature values and, consequently, these weights provide knowledge about what features were most important for the agent's decision. Furthermore, by summing the influence of different features over all nodes it is possible to obtain the overall influence that a feature has when compared to another.

Rule extraction is also used as a way to improve explainability. By combining the input feature range of a cell with the linear model associated with it, a rule for the agent behaviour in that feature range can be extracted. To illustrate this, figure 2.10 presents two rules for the Mountain Car environment, where each rule contains a range for velocity and position (the input features) together with the Q-values for each action ($\boldsymbol{Q} =< Q_{move\_left}, Q_{no\_move}, Q_{move\_right} >$). In the left image the cart is moving towards the left side of the hill with a large velocity and the rule extracted indicates that the action with the biggest Q-value is "move_left" ($Q = -25.2$), since the goal would be to store more gravitational potential energy so the car gains enough velocity to arrive at the top of the hill when coming back. The right image can be interpreted by applying the same logic.

Finally, Liu et al. (2018) suggest using super-pixels in tasks that include an image input. This can be done by highlighting pixels that have a higher feature influence value than the average feature influence value. Since the attitude flight control task explored in this project does not have an image input, this third way will not be further discussed.

To assess the performance of this algorithm, the authors made an empirical evaluation on three different RL environments: Mountain-Car, Cart-Pole and Flappy Bird. Two metrics were used for the evaluation: data coverage (i.e. covering the highest number of environment states possible) and data optimality (i.e. mimicking the NN's predictions accurately). It was found that LMUT outperforms the other tree-model baselines analysed.

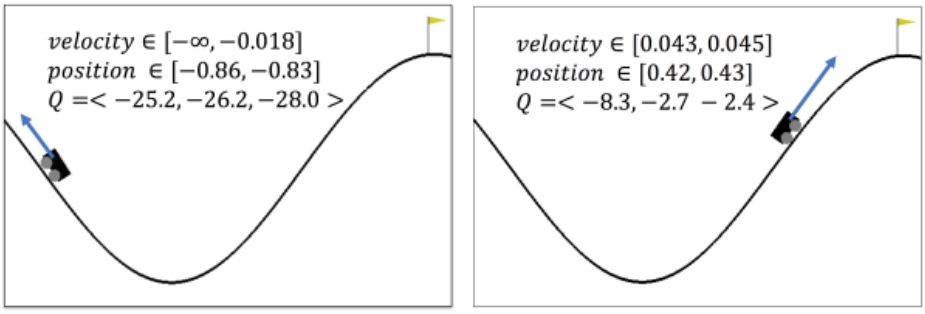


Figure 2.10: Rule extraction example for the Mountain-Car scenario. Adapted from Liu et al. (2018).

XRL through Saliency Maps:

Another interesting explainability approach increasing in popularity consists on using saliency methods to explain the agent's behaviour. These solutions provide explanations in the form of saliency maps, which are filters applied to images that highlight the input areas that most contributed to the agent's decision (Selvaraju et al., 2017 ; Greydanus et al., 2018), making it easy for non-AI experts to interpret the explanation.

Figure 2.11 presents a saliency map generated by a perturbation-based saliency method. To generate this, the algorithm considers a perturbation on the input image that removes information from specific pixels, which helps to understand how removing information from a certain input area affects the agent's policy. This perturbation influence can be quantified by a saliency metric and then presented in a heat-map. As an alternative to using perturbation-based methods, one can use Jacobian-based saliency methods.

However, as pointed out in Heuillet et al. (2021), despite representing a big portion of the available XRL solutions, saliency methods have several disadvantages. First, they need the task to meet a certain set of conditions in order to be reliable (e.g. be implementation invariant or input invariant). Second, even though the image shows the most important input features, it does not show ”why” those are the most important features, which according to Greydanus et al. (2018) is essential to produce satisfactory explanations. For these reasons, saliency methods will not be considered moving forward.

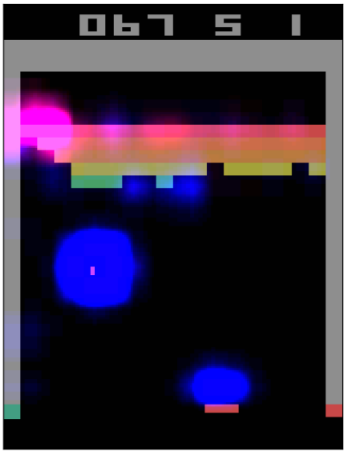


Figure 2.11: Perturbation-Based Saliency Map for an actor-critic model playing an Atari 2600 game. Red indicates the saliency for the critic, while blue indicates the saliency for the actor. Adapted from Greydanus et al. (2018).

XRL through Reward Decomposition:

In RL environments where it is possible to distinguish between different reward types available to the agent, Juozapaitis et al. (2019) suggest using reward decomposition to increase an algorithm's explainability. To do this, the reward function is decomposed into a sum of different semantically meaningful reward types, allowing for actions to be compared in terms of trade-offs among the different types of reward.

In this setting, the reward can be seen as a vector that has several components (one per reward type) and, consequently, the state-action value function can also be seen as a vector where each component accounts for the effect that an action has over a specific reward type. The sum of these various components gives the overall value of the state-action value function.

To solve a task with this decomposed reward vector, the authors introduce an off-policy multi-agent RL algorithm that is capable of learning multiple Q-functions, one for each reward type. The agent learns the best policy together with the explanation associated, where this explanation consists on telling the user which reward types were most important for the agent's decision (Heuillet et al., 2021). Two versions of this off-policy multi-agent RL solution are presented: 1) Tabular version called decomposed reward Q-learning - drQ; and 2) DQN variant called decomposed reward DQN - drDQN. To gain further insight into why an agent prefers an action over another, Juozapaitis et al. (2019) introduce two explanation types: Reward Difference eXplanation (RDX) and Minimal Sufficient eXplanation (MSX). RDX essentially enables the user to understand *all* the reasons why an action has an advantage over another, while MSX identifies a small set of the *most important* reasons why an action is preferable over another. These concepts will be explored in more detail below.

Off-policy multi-agent RL implementation:

Before applying the mentioned method, a small reformulation to the previously presented MDP is needed to account for different reward components. The authors define a set of reward components/types, C, and a vector-valued reward function, $\vec{r} : S \times A \rightarrow \mathbb{R}^{|C|}$, where $r_c(s,a)$ is the reward for type $c \in C$. In this setting, the goal is to maximise the overall reward that results from summing all the different reward components (refer to equation 2.22).

Naturally, these concepts can be extended to the state-action value function by using different Q-function components, $Q_c^{\pi}(s,a)$, to account for different reward types. The best action to take in each

step is the one that maximises the overall Q-function that results from summing all the different Q-function components (refer to equation 2.23). The optimal overall state-action value function and optimal policy are presented in equations 2.24 and 2.25, respectively. For this solution to work the different reward components need to be orthogonal and complementary.

$$r(s,a) = \sum_{c \in C} r_c(s,a) \tag{2.22}$$

$$Q^{\pi}(s,a) = \sum_{c \in C} Q_c^{\pi}(s,a) \tag{2.23}$$

$$Q^{*}(s,a) = \sum_{c \in C} Q_c^{\pi^*}(s,a) \tag{2.24}$$

$$\pi^{*}(s) = \arg\max_{a} Q^{*}(s,a) \tag{2.25}$$

The first off-policy multi-agent RL solution proposed by Juozapaitis et al. (2019) is a tabular method based on the Q-learning algorithm named "decomposed reward Q-learning" (drQ). This algorithm sucessfully solves the extended MDP and its pseudocode can be seen in Algorithm 2.6, where epsilon-greedy is mentioned but this exploration method can be replaced by other mechanisms. As it can be seen, drQ computes the overall greedy action ($a' = \arg\max_a \sum_c Q_c^{\pi}(s,a)$), and then updates each $Q_c(s,a)$ towards $r_{t,c} + \gamma \cdot Q_c^{\pi}(s_{t+1}, a')$. Instinctively, this leads the different Q-components to converge toward the value of the overall greedy policy with respect to c. The authors prove that this solution converges to both the overall optimal policy and to the correct Q-function components as long as the reward components are orthogonal and complementary.

An on-policy alternative named drSARSA is also presented, however, even though the results in the paper show convergence, the authors state that explanations generated by this on-policy method can violate intuition. In drSARSA the Q-components (Q_c) do not display the value of the greedy policy but rather the value of the exploration policy that was followed during learning, which can lead to counter-intuitive explanations. To prevent this on-policy behaviour researchers usually decrease the ϵ value, however, reward decomposition needs to compare the best action against others to generate explanations, and thus, it needs correct Q-value predictions for the non-greedy actions as well. Therefore, even though drSARSA shows convergence this method should not be used for explainability purposes.

Algorithm 2.6 The drQ and drSARSA algorithm. Adapted from Juozapaitis et al. (2019).

Initialise state-action value function Q^0
Observe initial state s_0
Determine initial action $a_0 \leftarrow \epsilon(Q^0, s_0)$
Set $t = 0$
repeat
  Take action a_t, observe $\vec{r_t}$ and s_{t+1}
  Determine next action to take according to ϵ-greedy exploration: $a_{t+1} \leftarrow \epsilon_t(Q^t, s_{t+1})$
  for all $c \in C$ **do**
    $a' \leftarrow \begin{cases} \arg\max_a \sum_c Q_c^t(s,a), & \text{if drQ} \\ a_{t+1}, & \text{if drSARSA} \end{cases}$
    $Q_c^{t+1}(s_t, a_t) \leftarrow Q_c^t(s_t, a_t) + \alpha \cdot \left[r_{t,c} + \gamma \cdot Q_c^t(s_{t+1}, a') - Q_c^t(s_t, a_t) \right]$
    $t \leftarrow t + 1$
  end for
until convergence

When dealing with tasks that have a large state space an approximate solution method is needed. As such, the authors introduce a DQN-variant called drDQN (decomposed reward DQN) that operates

in a similar way to DQN, however, it estimates each Q-function component by using a different function approximator, $Q_c(s, a; \theta_c)$, where θ_c represents the NN's parameters. Like DQN, drDQN uses a replay memory buffer and several target networks (one per reward type). More specifically, drDQN samples minibatches of experience tuples, $(s_i, a_i, \vec{r}_i, s'_i) \ : \ i = 1, ..., k\}$, from the replay buffer and uses them to update the parameters, θ_c, by applying gradient descent on the loss function described in equation 2.26. Periodically, the target parameters are updated with the learning parameters.

The authors also define a deep version of drSARSA called drDSARSA. Again, even though experiments prove that drDSARSA finds the optimal policy, due to the fact that it learns from an exploratory policy, it many times generates counter-intuitive explanations.

In order to test the performance of the methods described, Juozapaitis et al. (2019) consider two environments: a CliffWorld environment and the Lunar Lander environment from OpenAI Gym (Brockman et al., 2016). Both drQ and drSARSA quickly learn how to solve the cliffworld environment and provide good explanations as to why the agent took a certain action. For the lunar lander case, the reward function was decomposed into 8 different reward types, and both drDQN and drDSARSA learned a policy that safely lands the spacecraft. However, only drDQN was able to provide good explanations for the agent's decisions. This proves drDQN's ability to deal with large state spaces.

$$L(\theta_c) = \sum_{i=1}^{k} \left(y_{c,i} - Q_c(s_i, a_i \, ; \, \theta_c) \right)^2 \tag{2.26}$$

$$y_{c,i} = \begin{cases} r_c, & \text{for terminal } s'_i \\ r_c + \gamma Q_c(s'_i, a_i^+; \theta'_c), & \text{for non-terminal } s'_i \end{cases} \tag{2.27}$$

$$a_i^+ = \arg\max_{a'} \sum_{c \in C} Q_c(s'_i, a', \theta'_c) \tag{2.28}$$

Extracting explanations from the off-policy multi-agent RL algorithm:

Even though observing the decomposed Q-function components, Q_c, already provides information as to why a certain action was taken, the authors suggest two other explanation types to provide further insight: Reward Difference Explanations (RDX) and Minimal Sufficient Explanations (MSX).

Essentially, RDX provides all the reasons why the agent prefers a certain action over another in a given state. Stating that the agent prefers a_1 over a_2 is the same as saying that $Q(s, a_1) > Q(s, a_2)$, as such, RDX is defined as the difference between these two decomposed Q-vectors (refer to equation 2.29). If a certain component of $\Delta(s, a_1, a_2)$ is positive then, with respect to that reward type, action a_1 is preferable to action a_2. Contrarily, if it is negative then with respect to that reward type action a_2 is preferable. The magnitude of each RDX component tells the user how desirable one action is over another in relation to a specific reward type.

$$\Delta(s, a_1, a_2) = \vec{Q}(s, a_1) - \vec{Q}(s, a_2) \tag{2.29}$$

Alternatively, MSX identifies a small set with the *most important* reasons why an action is preferable over another. This is useful when there is a large number of reward types in the environment, since the user can rapidly become overwhelmed with the amount of information displayed. In this scenario, identifying a smaller set of reasons that still serve to explain the agent's decision is critical for a satisfactory explanation.

The MSX for actions a_1 and a_2 in state s is a tuple $\left(\text{MSX}^+, \text{MSX}^-\right)$, "where MSX^+ and MSX^- are the sets of critical positive and negative reasons for the preference" of action a_1 over a_2 (Juozapaitis et al., 2019). In order to obtain the first component of this tuple (the positive MSX), one has to first compute the *disadvantage* of a_1 in relation to a_2 (expressed in equation 2.30), which represents the total magnitude of reasons that prefer a_2 over a_1. It is important to note that "I" in equation 2.30 is the identity function. As expressed in equation 2.31, the MSX^+ is the smallest set of positive reasons that are required for a_1 to outweigh the *disadvantage* calculated.

The second component of the MSX tuple is the MSX^- that has the role of answering the question "What are the critical disadvantages of a_1 over a_2, that makes all the reasons in MSX^+ necessary?" (Juozapaitis et al., 2019). To compute this, one first needs to define the *just-insufficient advantage* of a_1 in relation to a_2, $v(s, a_1, a_2)$, expressed in equation 2.32 which sums all the reasons present in MSX^+ except the smallest one. Then, the MSX^- is the smallest set of reasons that when summed together have a total magnitude greater than the *just-insufficient advantage* (refer to equation 2.33).

It is important to note that when all positive components of RDX are needed for a_1 to be preferable over a_2, then the MSX will not offer any information compression relative to RDX.

$$d(s, a_1, a_2) = \sum_{c \in C} I[\Delta_c(s, a_1, a_2) < 0] \cdot |\Delta_c(s, a_1, a_2)| \tag{2.30}$$

$$\text{MSX}^+ = \underset{M \in 2^C}{\arg\min} |M|, \quad such\ that \quad \sum_{c \in M} \Delta_c(s, a_1, a_2) > d(s, a_1, a_2) \tag{2.31}$$

$$v(s, a_1, a_2) = \Big(\sum_{c \in \text{MSX}^+} \Delta_c(s, a_1, a_2) \Big) - \Big(\min_{c \in \text{MSX}^+} \Delta_c(s, a_1, a_2) \Big) \tag{2.32}$$

$$\text{MSX}^- = \underset{M \in 2^C}{\arg\min} |M|, \quad such\ that \quad \sum_{c \in M} -\Delta_c(s, a_1, a_2) > v(s, a_1, a_2) \tag{2.33}$$

The insights provided by these explanations serve not only to augment end-user interpretability about the agent's decision, but also to detect issues in the implementation. The authors demonstrate this by implementing an ill-suited method (HRA agent presented in van Seijen et al., 2017) for the lunar lander environment. After training, this solution is not able to safely land the spacecraft and instead prefers to crash itself into the ground without ever firing the engine. Figure 2.12 shows the RDX for actions "no_operation" and "fire-main-engine" before the lander crash. By analysing this figure, it is possible to see that the HRA agent prefers "no-operation" (which leads to a crash) to avoid any penalties related to fuel-cost, high-velocity or de-stabilization of the lander. Without using decomposed Q-values and RDX it might be difficult to arrive at such insights.

Contrarily to HRA's poor performance, drDQN successfully solves the lunar lander environment. In figure 2.13, the MSX for actions "fire-main-engine" and "no_operation" is presented for a state near the landing site. Intuitively, in this state the agent should focus on avoiding a crash, however, by analysing the MSX it is clear that the shaping rewards dominate the agent's decision (the crash reward has a negligible influence). When reasoning about these results, one has to recognise that optimising the shaping rewards leads to a good environment reward and, consequently, the agent mainly focuses on optimising the shaping rewards. It would be difficult to arrive at such conclusions without using this approach, proving yet again the effectiveness of this solution to increase explainability.

Figure 2.12: RDX (a_1 = no_operation ; a_2 = fire-main-engine) for ill-suited method in Lunar Lander environment before a crash. Adapted from Juozapaitis et al. (2019).

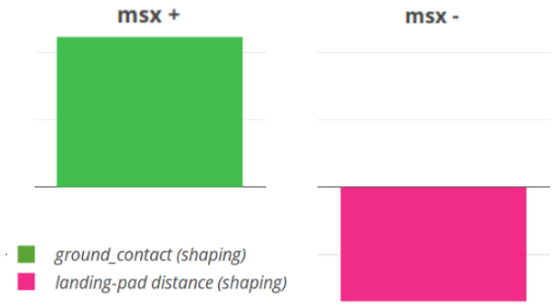


Figure 2.13: MSX (a_1 = fire-main-engine ; a_2 = no_operation) for drDQN in Lunar Lander environment near landing site. Adapted from Juozapaitis et al. (2019).

XRL through using SHAP values:

To conclude this overview on the state-of-the-art XRL methods it is important to discuss the usage of SHAP (SHapley Additive exPlanations) values to increase explainability, given that this strategy has gained great popularity over recent years. Lundberg and Lee (2017) suggest using a SHAP value to measure the marginal contribution of every input feature to the model's output, allowing users to know which input features contributed most to the prediction.

In this method an explanation model, $g(z')$, is obtained through a linear combination of Shapley values ϕ_i with a vector z_i' of maximum size M, as described by equation 2.34.

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i \cdot z_i' \tag{2.34}$$

Lundberg and Lee (2017) present 6 different methods based on SHAP values and define a class for this type of solution, calling methods like this "additive feature attribution methods". These algorithms prove to be more consistent with human intuition when compared to other renown solutions like LIME (Ribeiro et al., 2016) and DeepLift (Shrikumar et al., 2017). An example of how SHAP values can increase explainability in DRL will be presented in subsection 2.4.4.

Even though SHAP methods present reliable results, they can lack causality when presented to end-users and be computational expensive if some assumptions are not met (e.g. feature independence). Due to these drawbacks, this method is not considered to be the best solution to increase interpretability for end-users and will not be considered moving forward.

2.4.4. XRL in the Flight Control Domain

In the last subsection, an overview on the state-of-the-art XRL methods was presented. However, it is important to mention XRL applications inside the flight control domain since this research's main task is to augment explainability in an attitude flight control task.

Due to how recent this research field is, there is little XRL research available in the literature and there is even less research that directly targets XRL inside the flight control domain. One of the few aeronautical XRL applications can be found in He et al. (2021), where authors apply a TD3 controller to a quadrotor in order to have it fly autonomously in an unknown outdoor environment. Two input types were given to the controller: raw depth images recorded with a camera and UAV state information collected with sensors (relative position to the goal and current velocity). The controller was able to output actions in 3 DoF: forward linear velocity (v_{xy}), climbing linear velocity (v_z) and steering angular velocity (ϕ). It was trained for 200k steps in a high-fidelity simulation environment and then validated with real world tests. Results indicated that it was able to navigate to the goal position successfully, while being computationally faster than other conventional approaches.

Furthermore, the authors used feature importance to improve the algorithm's interpretability. Considering that the controller received two different input types, then two different feature importance methods were implemented: one for the camera images and another for the sensor data. To retrieve feature importance values from the camera images an adapted saliency map solution called SHAP-CAM was

used, whereas for the sensor information the authors used SHAP values. SHAP-CAM combined the GRAD-CAM approach (Selvaraju et al., 2017) with SHAP values.

Figure 2.14 shows the network focus depending on the input received and the action type considered, from where it is possible to see that depending on the action type (e.g. forward linear velocity vs steering angular velocity) the network focuses on different image regions. Moreover, figure 2.15 shows the SHAP summary plots, where the influence that different input features have in generating the action outputs is described. From these plots it is possible to see that CNN features (image input) dominate linear velocity decisions, while the angle error ξ (sensor input) is the most important feature to determine the command steering angular velocity.

Even though such explanations might help the user to get a better grasp about the agent's focus, the authors mention that these explanations can be considered shallow since no reference is given on why the network combines the features in the way described by SHAP.

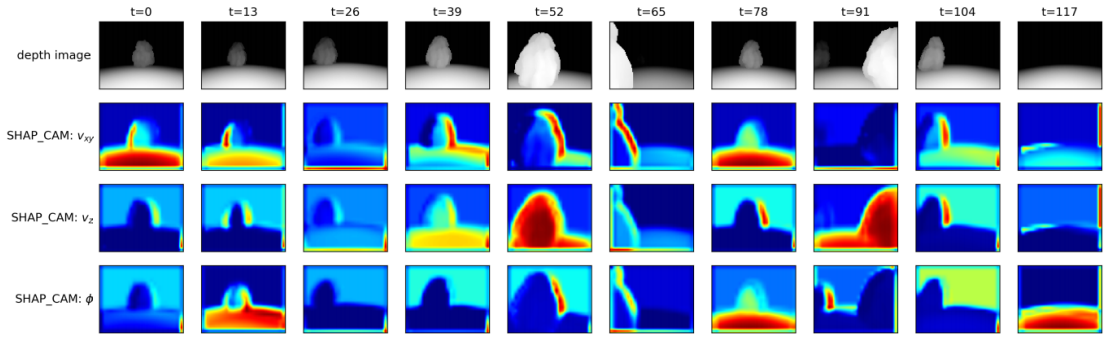


Figure 2.14: Depth image and SHAP-CAM at 10 different timesteps during the evaluation episode. Adapted from He et al. (2021).

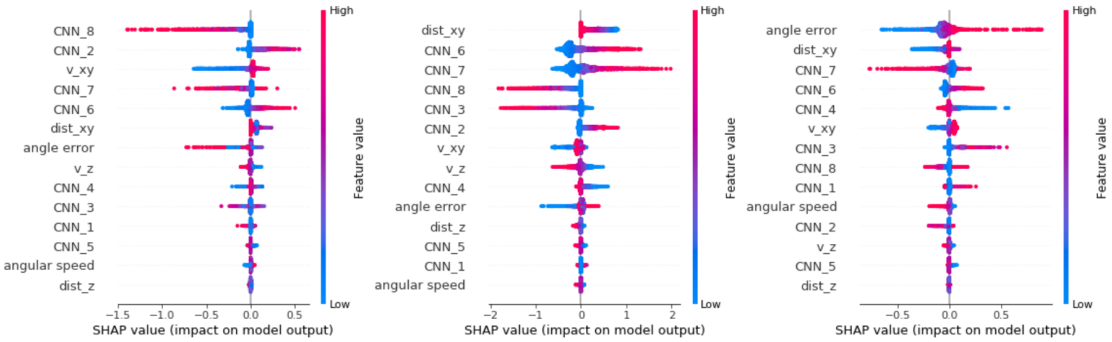


Figure 2.15: Feature importance analysis over 20 trajectories for the 3 DoFs available. Adapted from He et al. (2021).

More recently, van Zijl et al. (2022) used SHAP explanations to obtain insights on the behaviour of a DRL controller implemented in a high-fidelity model of the Cessna Citation 500 aircraft. In this work, two controllers based on Incremental Dual Heuristic Programming (i.e. a lateral and a longitudinal one) were trained online to track altitude and roll angle reference signals. Results showed that the controllers accurately followed the references signals supplied. The learned strategy was then explained with SHAP, where the authors present an innovative way to obtain linear expressions that describe the agent's policy in regions where the DNN's weights are constant. By having access to these linear expressions, control experts can obtain detailed insights about the input-output mapping used by the controller and, therefore, increase explainability.

Pizarroso and Kampen (2022) expanded on the efforts of the previous paper by applying the same controller to nonlinear flight regimes that included high angles-of-attack and large sideslip angles. This work showed that the XRL solution presented in van Zijl et al. (2022) works even when the IDHP agent presents nonlinear behaviour, making SHAP explanations a highly attractive XRL solution for control experts.

Nevertheless, control expert knowledge is required to interpret why the agent implements the input-output mapping described in the SHAP explanation. As such, this solution might lead to explanations that lack causality when presented to end-users, which as mentioned by Arrieta et al. (2020), is an essential condition for end-user explainability.

2.4.5. One explanation method vs Multiple explanation methods

In the previous subsections the main aspects of XRL have been covered. However, to answer Q1.3 one still needs to investigate how providing multiple explanation types compares with a single explanation type method (when the goal is to increase end-user explainability).

In Anderson et al. (2020), an empirical study was carried out to investigate the impact of explanations on non-expert's understanding about RL agents. Two explanation methods were used: reward decomposition and perturbation-based saliency maps. In this study 124 participants saw a DRL agent play a Real-Time Strategy (RTS) game, and the participants were divided into 4 groups: 1) control group with no access to explanations; 2) Group with access to reward decomposition explanations; 3) Group with access to saliency maps; 4) Group with access to both reward decomposition explanations and saliency maps - called everything group. At the end of watching a snippet of the agent playing the RTS game, all 4 groups were evaluated on both their understanding about why the agent took certain actions and what would be its next action, which are both questions that help evaluate the participant's mental-model about the RL agent.

The implemented RL agent consisted on an approximate solution method that used a neural network to estimate the Q-function. To generate reward decomposition explanations, the main reward was divided into six different components (e.g. Enemy fort damage , town/city damage, etc.). Results indicated that the combination of both saliency maps and reward decomposition was needed to achieve the best mental model score over the control group. Figure 2.16 shows that the everything group not only presented significantly better mental models than the control group, but also presented better models than any of the other groups. Furthermore, the fact that both the reward decomposition and saliency map groups achieved a better mental model than the control group but worse than the everything group suggests that each explanation type brings its own strengths to increase explainability.

Even though results show that the overall participant performance was better when having access to both explanations, in some Decision Points (DPs), participants with access to only one explanation method actually outperformed the everything group. After inquiry, it was found that (only in those DPs) the everything group felt overwhelmed by the amount of information and the limited time they had to make a decision. Alternatively, single explanation groups did not have to deal with excessive information and were able to make a better judgement. This reinforces the importance of not overwhelming the users while trying to present complete explanations.

Figure 2.17 presents the time that each group takes to respond for each DP. The ”X” in the figure displays how much time a user would spend solving the DP if they had spent as long as the control, plus the average time that the saliency group takes over the control, plus the average time that the reward group takes over the control. The results show that as a drawback for dealing with more information, the everything group takes more time per decision. Furthermore, since the everything group is placed above the ”X” at all DPs, this means that the time that one takes to consider both explanations together surpasses the sum of considering both explanations individually (i.e. summing the reward decomposition extra time over the control with the saliency map group extra time over the control). This is important to consider when providing explanations in a time constricted environment, since presenting additional information leads to users taking longer to reach a decision.

Anderson et al. (2020) conclude that: 1) One must be careful when designing explanations since there is no method that fits into all situations; and 2) The amount of information presented must be taken into account, not only to avoid overwhelming the user but also to make sure that the user does not take too long to reach a decision. However, if the solution makes sure that users are not overwhelmed and that time addition is not a problem, then having more than a single explanation type usually improves the user's mental model.

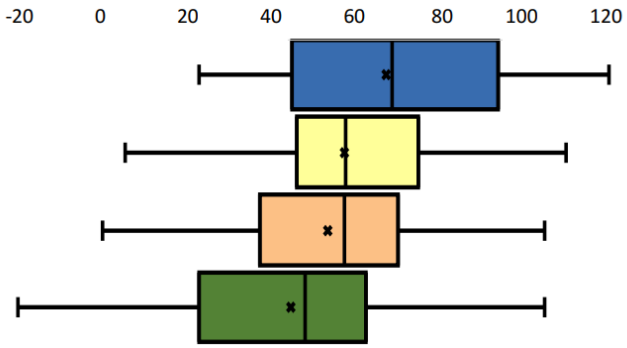


Figure 2.16: Participant's final mental model scores. Blue - Everything group ; Yellow - Reward Decomposition ; Orange - Saliency maps ; Green - Control Group. Adapted from Anderson et al. (2020).

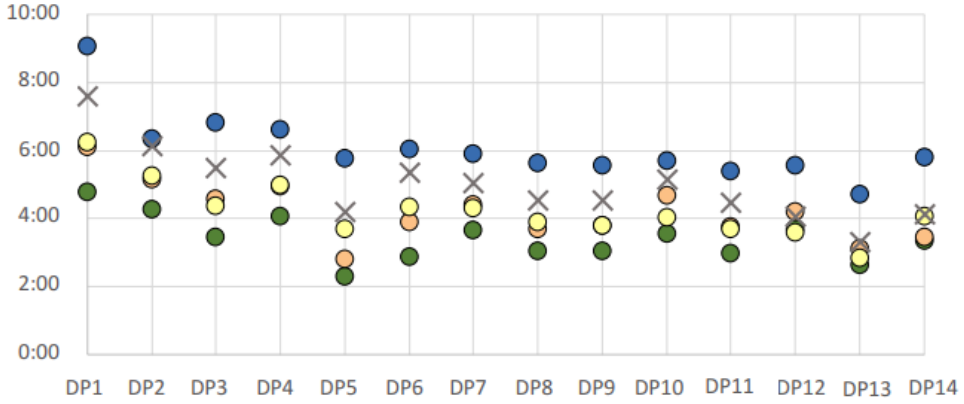


Figure 2.17: Average task time in each user decision point (DP). Blue - Everything group ; Yellow - Reward Decomposition ; Orange - Saliency maps ; Green - Control Group; X - explained in the text. Adapted from Anderson et al. (2020).

2.5. Chapter Conclusion

This chapter's purpose was to answer Research Question **Q1** and Sub-Question 2.1. To do this, this literature review covered various RL topics, from its fundamental concepts to the different state-of-the-art DRL algorithms. Furthermore, the field of XRL was introduced and several state-of-the-art end-user suitable XRL solutions were presented.

Sub-Question 1.1 was answered in subsection 2.4.2, where it was discussed how to properly evaluate an XRL method. It was stated that (currently) there are no specific metrics to evaluate the interpretability increase supplied by a XRL solution due to several factors: 1) There is no official XRL taxonomy; 2) The explanation is relative to a specific audience; 3) The quality of an explanation is many times qualitative and subjective. To deal with this, researchers typically perform user studies which allows them to convert qualitative results into quantitative ones. In these user studies, the mental model created by the user after seeing the explanation is assessed by asking two types of questions: 1) Post-task questions about the agent's behaviour; 2) Questions about predicting the agent's future actions. This responds to Q1.1, however, it is important to note that performing an user study exceeds the scope of this work and, therefore, it is left as a future research recommendation.

To the author's best knowledge, Anderson et al. (2020) is the only study that compares two different explanation types targeted at the same audience class. In this work (refer to subsection 2.4.5), reward decomposition explanations (focus on explaining expected future rewards) were compared against saliency map explanations (focus on explaining where in the input the agent is focusing on). The authors conclude that there is no perfect solution that achieves the best results on all DPs and that

different explanation types provide different strengths. This starts to address Q1.2, since it is possible to say that there is no official recommendation on what the focus of the explanation should be. However, even though there is no official recommendation, by analysing the literature presented in this chapter it is possible to hypothesise about the best explanation type for end-users.

Anderson et al. (2020) provide evidence that explanations that focus on 1) Comparing future expected returns; or 2) The most important input regions; lead to better mental models for non-AI experts when compared to a control group. Contrarily, methods that focus on aligning themselves with the user's rationale (e.g. Madumal et al., 2020) did not have much success in improving user's trust. This might change in the future when the latter methods become more interactive, however, for the purpose of this research such methods are discarded. The last explanation type to be addressed in Q1.2 focuses on explaining the network processing. These explanations are highly related to the informativeness goal, which based on table 2.1 is only relevant to Developers and Executives (not to end-users). Therefore, methods that focus on explaining the network processing are also discarded from this research.

The previous arguments fully answer Q1.2, where the two explanation types that show the most promising results for increasing end-user acceptance are: 1) Methods that focus on comparing expected future returns; and 2) Methods that focus on where in the input is the agent focusing on to reach a decision. Consequently, the final algorithm chosen for this research will use one of these explanation types.

As it was seen in subsection 2.4.5, using more than one explanation type at once leads to users taking more time to reach a decision and, in some cases, it might overwhelm them with too much information. In case users are overwhelmed, the mental model they have about the agent is worse than the one generated by users with access to a single explanation type. Nevertheless, if the additional decision time is not an obstacle to the task and one makes sure that users do not get overwhelmed with information, then having multiple explanation types generates better mental models when compared to a single explanation type method. This answers Q1.3, however, it is important to mention that using more than one explanation solution and making sure that users do not get overwhelmed exceeds the scope of this work.

The goals that an explanation should focus on when targeting end-users were presented in table 2.1. When these goals are fulfilled, a satisfactory explanation is achieved. Consequently, the biggest factors that contribute to end-user acceptance were already identified which, when combined with the previous Sub-question answers, fully responds to Research Question **Q1**.

In section 2.3, various DRL methods were presented and analysed. However, the main focus of this research is to develop a XRL solution for a flight control system of a business jet aircraft and, consequently, one has to take into account not only how effective the solution is at solving the task but also how satisfactory the explanations are to end-users. Since there are various DRL solutions capable of solving complex control tasks, but there are fewer XRL options suitable for end-users, this research will focus first on finding the XRL method with the highest potential to increase explainability for end-users. After having the XRL solution, a DRL method that not only solves the task but is also compatible with the explainability method will be chosen.

The most renown XRL methods that are suitable for end-users were presented in subsection 2.4.3. From this subsection two main options arise as satisfactory solutions for the attitude flight control task: Linear Model U-Trees (LMUT) and Reward Decomposition. This answered Q2.1.

- Linear Model U-Trees (Liu et al., 2018) builds an ensemble of linear Q-functions, where each tree node has a linear model that describes the DRL output in a specific section of the input space. It presents good accuracy, can be used both online and offline, and it has an inherently transparent structure (requiring fewer leaves when compared to other tree-model baselines). Additionally, one can use feature importance and rule extraction with this approach to further enhance explainability for end-users. By using feature importance it is possible to explain where in the input the agent focuses on to reach a decision, which is a proven way to increase a model's interpretability (refer to the Q1.2 answer).

  As a disadvantage, even though LMUTs are capable of dealing with continuous state spaces, this method will most likely have difficulties solving tasks with continuous action spaces. This limits

the amount of DRL methods compatible with the solution, and demands a discretization of the action space when dealing with the attitude flight control task.

- Reward Decomposition (Juozapaitis et al., 2019) decomposes the RL reward into sums of semantically meaningful reward types, allowing for actions to be compared in terms of trade-offs among the different reward components. This approach achieves similar task performance to the one obtained with the algorithm without decomposition, however, it consumes more computational resources given that the number of NNs required increases with the number of reward types. Furthermore, it presents explanations that compare the expected future rewards obtained by selecting different actions and, consequently, it is in accordance with the Q1.2 answer. To further increase explainability one can use RDX and MSX, which as seen in Anderson et al. (2020), is a proven way to increase a model's interpretability. Finally, it can help developers identify "bugs" in the implementation since it allows for insight's into strange agent behaviour. All of this makes reward decomposition a highly attractive solution.
  As disadvantages, this approach only works with value-based DRL solutions, which limits the amount of compatible DRL methods available and demands a discretization of the action space when dealing with the attitude flight control task.

Even though these two solutions demand a discretization of the action space when dealing with the attitude flight control task, to the author's best knowledge, they are the ones that present the highest potential to increase end-user acceptance for DRL methods. Most XRL methods either seem to be only suitable for users with previous AI knowledge, or do not have a proved ability for improving the user's mental model about the RL agent.

In summary, based on this literature review chapter two methods were selected: 1) Linear Model U-Trees (LMUT) and 2) Reward Decomposition. These two solutions generate and present explanations in a very different way and, therefore, a direct comparison based on the literature is not possible. The final XRL solution to implement in the attitude flight control task will only be chosen in the next chapter, where a simple implementation of both methods is carried out to test the feasibility and performance of both XRL solutions.

Regarding which underlying DRL algorithm to use with the XRL methods selected, both XRL approaches only work with discretised action spaces, something that highly limits the amount of relevant DRL solutions. As such, a DQN-based algorithm is selected since it is compatible with the two XRL methods while also being able to solve an attitude flight control task. As a disadvantage, this method has a difficulty in dealing with large action spaces, however, since this research's main goal is to augment explainability and not to obtain a new state-of-the-art task performance, the DQN-based algorithm is still the most attractive solution.

In the next chapter, both the LMUT and Reward Decomposition methods are implemented in a simple but relevant environment to test their feasibility and performance. For this first implementation a DQN algorithm very similar to the one seen in Mnih et al. (2015) will be used, that is, without prioritized experience replay and other RAINBOW improvements. This maximises compatibility with the selected XRL solutions since it mimics the way in which Liu et al. (2018) and Juozapaitis et al. (2019) first implemented their algorithms. However, when moving to the final attitude flight control task, it might be needed to include some of the DQN improvements mentioned in subsection 2.3.1. Nevertheless, this research's primary goal is to improve DRL explainability and not to maximise its efficiency, as such, an effort will be made to keep the model as simple as possible (for explanation verification purposes) while still solving the attitude flight control task.

3

Preliminary Analysis

In the previous chapter, two XRL methods were identified as having high explainability potential for tackling the attitude control task of a business jet flight control system. Furthermore, the literature review chapter fully responded to Research Question 1 and Sub-question 2.1. This chapter answers the remainder of Research Question 2 by implementing these two XRL solutions in a simple but relevant environment to test for feasibility and compare their performance. Through this analysis, one of the methods is selected as the most suitable solution for the attitude flight control task. Moreover, having the main algorithm implemented in a simple environment provides the confidence and knowledge necessary for an implementation in the more complex business jet aircraft flight control system.

Section 3.1 presents the implementation details and results obtained by using the first XRL method chosen during the literature review, LMUTs. In section 3.2 the implementation details and results obtained while using Reward Decomposition (second XRL method selected) are presented. Finally, section 3.3 answers the remainder of Research Question 2 by selecting a single method as the most suitable one for this project's main task.

3.1. Linear Model U-Trees Implementation

Linear Model U-trees (LMUTs) was the first method implemented in this preliminary analysis. In order to improve the author's knowledge about the algorithm and test for feasibility, an initial implementation in the OpenAI Gym MountainCar-v0 environment (Brockman et al., 2016) was performed following the same implementation details as in the original paper. The code was built based on the source code provided by the authors (available at Liu and Schulte, 2018) and the hyperparameters used were the same as the ones seen in the original publication.

When testing on a computer with 16gb of RAM and an Intel Core i7 processor at 2.6GHz, the algorithm ran for more than 12 hours without ever converging to a solution. Given that the attitude flight control task has a much larger input space than the one seen in the MountainCar-v0 environment, it is hypothesised that this training time will make it impossible to use LMUTs in this project's main task given the resources available. This is a relevant drawback that is not evident in the original paper (Liu et al., 2018). Furthermore, the LMUT quickly grows to a size where having a transparent structure no longer improves interpretability since there are too many nodes for the user to track.

When looking at the training process to identify the most computationally expensive tasks it is seen that, while in the learning phase and for each step, the LMUT checks every tree node individually and runs SGD on the linear model inside the node to check: 1) If it is worth to split the node further; or 2) If it already achieves the desired performance for that section of the input space. If the algorithm determines that the node has to be split, then the agent checks and compares every possible way to split the node. This allows LMUT to find the best split for mimicking purposes. At the beginning of the training phase (when the tree is still small) this is not a problem, however, as the training phase progresses the tree length quickly grows to sizes that make learning a computationally expensive task.

It is important to mention that there is a chance that these results are caused by an implementation error. However, this is a small chance given that: 1) This implementation uses Liu and Schulte (2018)

source code; 2) The hyperparameters seen in the original paper are maintained; 3) There is a valid hypothesis as to why this happens. Consequently, LMUTs are discarded as a relevant method for this research and the final solution will be based on Reward Decomposition.

3.2. Reward Decomposition Implementation

Reward Decomposition (Juozapaitis et al., 2019) was the second method implemented in this preliminary analysis. Given the results seen in the previous section, it was already known that Reward Decomposition was to be chosen as the final solution for this project's main task. Consequently, the environment choice was changed to one that allowed easy verification of the explanations produced, as opposed to targeting an environment where one could easily compare Reward Decomposition with LMUTs. This simplified the RDX and MSX explanation validation process.

Subsection 3.2.1 presents the simulation environment and the proposed decomposed reward vector. The algorithm implementation is detailed in subsection 3.2.2, whereas the performance results obtained are outlined and discussed in subsection 3.2.3. Lastly, subsection 3.2.4 presents the explanation results produced while using this solution.

3.2.1. Simulation Environment and Decomposed Reward Vector

For this preliminary analysis the environment was chosen based on two main criteria:

- Be complex enough to allow the testing of drDQN learning abilities.
- Be intuitive enough to enable the developer to easily validate the explanations generated.

Considering these two conditions an environment was developed based on the one presented by Juozapaitis et al. (2019), however, some changes were applied to allow for a more thorough analysis of the algorithm and explanations generated. The OpenAI gym library was used to make the mentioned changes. The new environment is a grid-world (discrete state space), where each cell can contain: cliffs, monsters, coins, gold bars or treasures. It is called *CliffWorld* and its general structure can be seen in figure 3.1, where the agent is represented in cell (0,0), but it can start the episode in any white cell (the start position is chosen randomly). At each step the agent has four possible discrete actions: "Up" , "Down", "Left" and "Right". Furthermore, an episode ends whenever the agent steps into a grey coloured cell or the limit of 50 steps per episode is reached.

In this environment, the decomposed reward vector has the following reward types: *[cliff, coin, gold, monster, time, treasure]*. With the exception of the time reward, these reward components represent the contents of the agent's current cell and the specific reward obtained by stepping into such cells is indicated in table 3.1. The time reward is a penalty given to the agent for not reaching the end of the episode, that is, while the agent is in a white cell it receives a "-2" reward per step (as indicated in table 3.1). Without this reward type, the optimal policy would consist on always going for the treasure, however, with this addition it will depend on the agent's starting position.

At each step the agent receives its current position from the environment. More specifically, 20 Boolean signals are sent (one for every cell in the environment) from which only one signal is true.

3.2.2. drDQN and drDSARSA Implementation

Two different algorithms are implemented to solve this environment, drDQN and drDSARSA (previously presented in subsection 2.4.3). drDSARSA is only implemented to test reward decomposition's ability to use on-policy algorithms, not for explainability purposes. This is the case since Juozapaitis et al. (2019) mention that the explanations generated by this on-policy method might be counter-intuitive, but very few references are given as to how drDSARSA compares to drDQN in terms of task performance. Nevertheless, the topic of how to improve the explanations generated by this on-policy algorithm exceeds the scope of this project.

The PyTorch ML framework is used to implement both methods. Each algorithm is trained for 2000 episodes, and at every 25 episode interval an evaluation phase (of 50 episodes) is performed. These

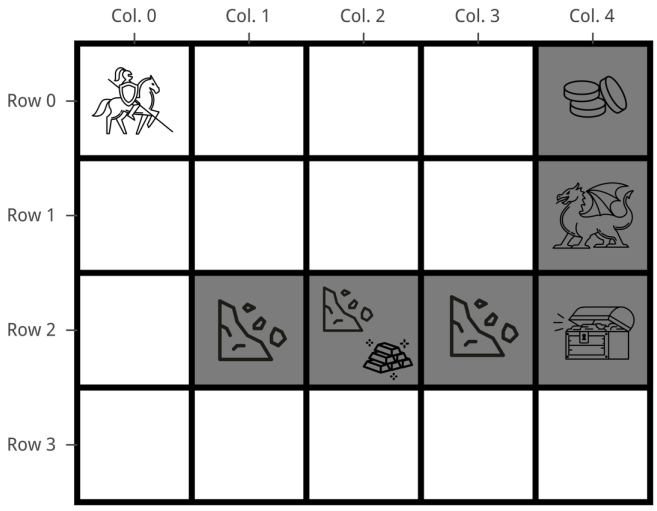


Figure 3.1: CliffWorld environment structure.

Table 3.1: Reward values depending on the reward type.

Reward Name	**Reward Value**
cliff	-10
coin	+4
gold	+10
monster	-20
time	-2
treasure	+15

evaluation phases allow one to see how the performance changes as training progresses and how the explanations vary with time. This procedure is repeated for 5 runs and the final results are averaged across the runs. The environment seed varies from run to run, but it is the same for the two methods so they are both trained under the same conditions. The seed used in the evaluation phase is always the same.

Two metrics are used for convergence and performance comparison: 1) The average score over the last 150 episodes, that is, a moving average; 2) The loss function value. The loss function used by both agents is the SmoothL1Loss, which is less sensitive to outliers when compared to MSE and in some cases can prevent exploding gradients (Girshick, 2015). Furthermore, the Adam optimiser is used instead of the conventional SGD and the justification for this choice was given in the previous chapter. To guarantee that the agent continues to select exploratory actions throughout the learning process, ϵ-greedy is used. ϵ starts with a value of 0.9 and linearly decays to 0.1 until episode 1250, keeping that same value thereafter (refer to figure 3.2). Furthermore, a constant learning rate of 0.01 and a discount factor of 0.99 are used.

Given that this experiment is a small application that uses a similar grid-world structure to the one presented by Juozapaitis et al. (2019), the hyperparameters described are the same as the ones used in the original paper. However, it is important to note that an hyperparameter tuning phase will be carried out when moving to the more complex attitude flight control task.

The Q-network architecture is the same for every agent regardless of the Q-function reward type it estimates. There are 6 different reward types in this modified Cliffworld and, consequently, there are 6 Q-networks used in this method. The structure of each single Q-network is presented in figure 3.3, where the input layer has 20 neurons (one neuron per environment cell) and the output layer uses 4

neurons (one for each possible action). No hidden layers are used given that such keeps the model simpler and the solution was capable of solving the task without them. However, when moving to the main attitude flight control task hidden layers with ReLU activation functions will be used.

For learning, a minibatch size of 64 samples is used and the memory replay buffer has a total size of 100 000 samples. The target network is updated every 100 steps. A summary of all the hyperparameters used in this experiment is presented in table 3.2, where ”Update Frequency” refers to the number of actions selected by the agent between two successive Adam updates. It is important to mention that these hyperparameters are the same for both drDQN and drDSARSA.

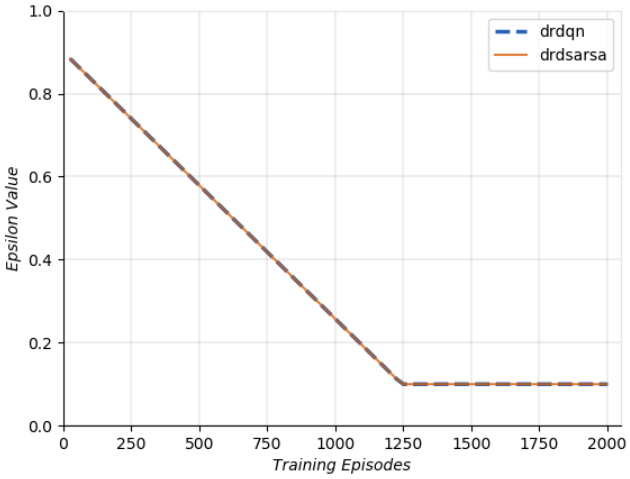


Figure 3.2: Epsilon value change over the training process.

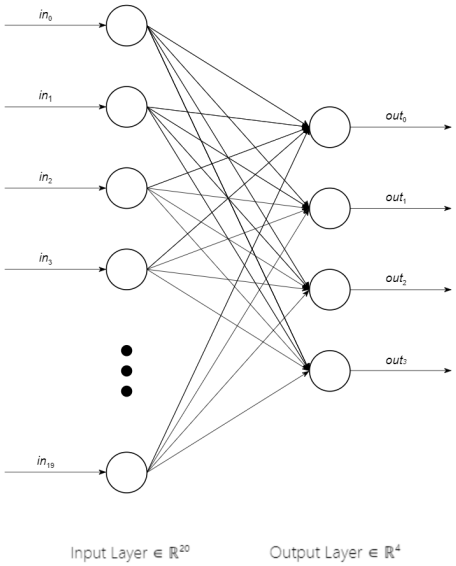


Figure 3.3: Single Reward Type Q-Network Architecture (the overall method uses 6 Q-Networks with this same architecture - one per reward type).

Table 3.2: drDQN and drDSARSA hyperparameter summary.

Hyperparameter Name	Implementation Value
Learning rate (α)	0.01
Discount factor (γ)	0.99
Minibatch size	64
Memory buffer size	100 000
Initial epsilon (ϵ) value	0.9
Final epsilon (ϵ) value	0.1
Update frequency	1

3.2.3. Algorithm Training and Performance Results

The training process for both agents took around 4 minutes when performed on a computer with 16gb of RAM and an Intel Core i7 processor at 2.6GHz. All plots presented in this subsection are the average results across 5 runs (and do not refer to any specific run).

The drDQN and drDSARSA learning curves obtained for this experiment are shown in figures 3.4 and 3.5, respectively. From these figures it is clear that, even though the training score tends to increase as the training phase progresses (i.e. the moving average increases with the episode number), there is still a high variability in the score obtained from episode to episode. This variability is caused by the exploration that is present during the whole training phase, which forces the agent (with a small probability) to take non-optimal actions. These results also suggest that drDQN converges faster to higher rewards when compared to drDSARSA, which will be clear in the evaluation results.

The loss function value is used to check for algorithm convergence and the results are shown in figures 3.6 and 3.7 for drDQN and drDSARSA, respectively. Even though drDQN presents a high loss value variation in the initial training phase, these variations diminish as training progresses. Furthermore, not only does drDQN converge faster to lower loss values, it also achieves lower overall loss values than the ones presented by drDSARSA, which tends to stabilise around a loss of 0.1 (10x higher than the ones presented by drDQN). Finally, it is important to address the small peak observed in the drDQN loss plot around episode 1550. This small variation happened only in run 1 (out of the 5 runs), but its effect is still noticeable in the averaged plot presented in figure 3.6. This change is also seen in the drDQN's Q-network weights and is probably caused by an exploration induced policy variation that happened only in run 1, from which the network quickly recovers.

During the different evaluation phases (that occur every 25 episodes) the ϵ value is temporarily set to zero, which forces the algorithm to follow the policy it believes to be optimal. This allows for a direct performance comparison between different learning stages and algorithms. The evaluation scores obtained by the two XRL methods can be seen in figure 3.8. drDQN clearly converges faster to the optimal policy, but both algorithms end up with the same final policy (which can be seen in figure 3.9). More precisely, figure 3.8b shows that drDQN converges around episode 500 while drDSARSA takes 1400 episodes to find the same optimal policy.

The analysis performed confirms drDQN's superiority when compared to drDSARSA. Juozapaitis et al. (2019) state that drDSARSA should not be used for explainability purposes since it can generate counter-intuitive explanations, however, this work suggests that drDSARSA also takes more time converge and ends-up with higher loss values when compared to drDQN. It might be the case that such results vary with the environment used, however, these results should still be taken into account when considering a drDSARSA implementation.

Before analysing the explanations generated, the weights and biases of the 6 Q-networks were checked to further confirm the algorithm's convergence. Only the drDQN weights and biases are presented in this report given that: 1) Each method uses 6 different networks (and each network has many weights and bias); 2) drDSARSA is not considered for the main task of this research.

Apart from run number 1 where there is a small variation in the drDQN's biases around episode 1550, the results across runs are very similar with the biases converging to a constant value somewhere around episode 300 to 500, depending on the run. The output layer bias values for the treasure reward Q-network are presented in figures 3.10a and 3.10b for run 1 and 3, respectively. The peak seen around episode 1550 for run 1 relates to the same policy deviation mentioned during the loss plot analysis, from which the agent quickly recovers.

Figure 3.11 shows the weights for the treasure reward Q-network output neuron 0 during run 3, from which it is clear that the weights converge to a constant value around episode 400. To prove that other Q-networks also converge, figure 3.12 shows the weights for the coin reward Q-network output neuron 3 in different runs. In that figure two subfigures are presented: 1) A plot for run 0, which also represents what happens for run 2, 3 and 4; 2) A plot for run 1. The weights converge before episode 500 for all runs, with exception of run 1 where there is a small peak around episode 1550, just like in the loss and bias plots.

It should be mentioned that these results were common across all 6 Q-networks, which are not all presented to avoid repeated and unnecessary information.

This analysis shows that drDQN successfully solves the modified CliffWorld environment, however, it should be noted that a larger network (with an hyperparameter tuning phase and nonlinear activation functions) would probably solve this environment in considerably less episodes. Nevertheless, for the purpose of this preliminary analysis this simple implementation is sufficient.

In the next subsection the drDQN's explanations are shown to demonstrate how this approach increases end-user explainability.

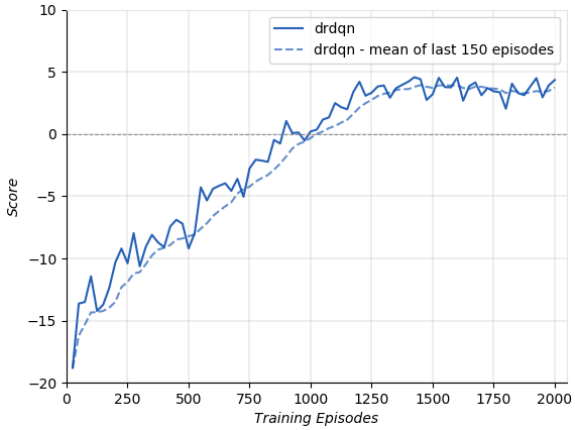


Figure 3.4: drDQN training score over training episodes.

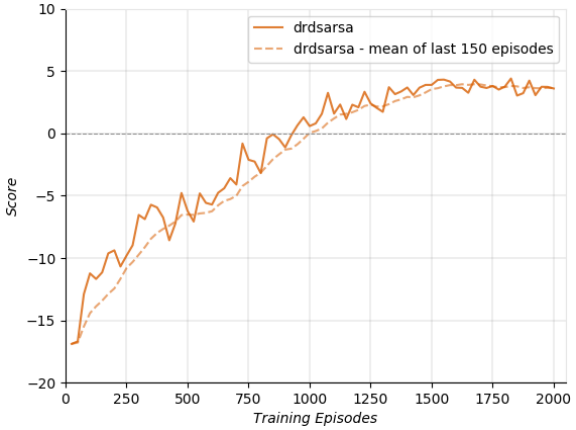


Figure 3.5: drDSARSA training score over training episodes.

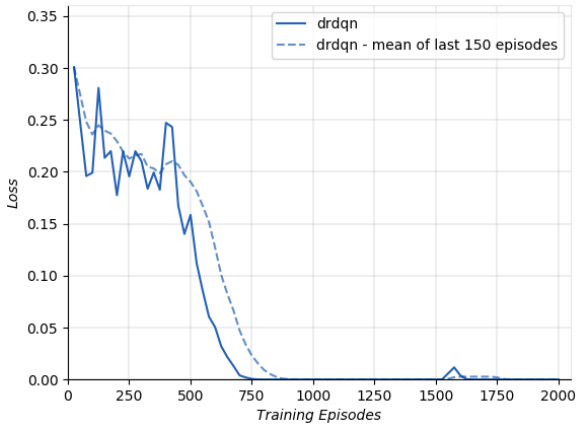


Figure 3.6: drDQN loss over training episodes.

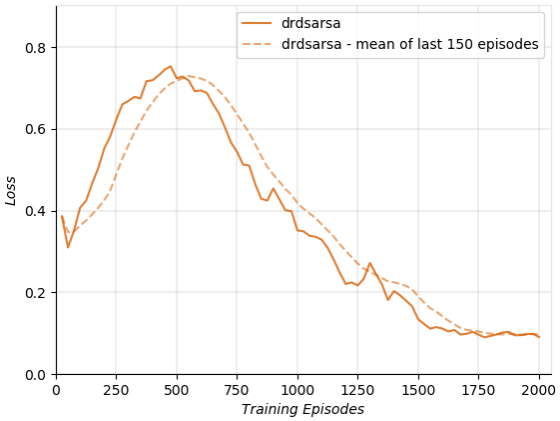
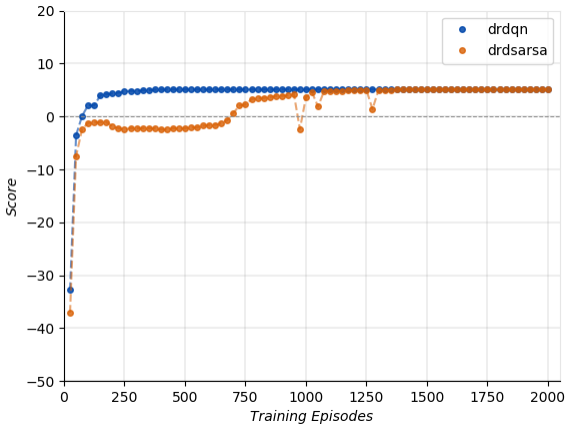
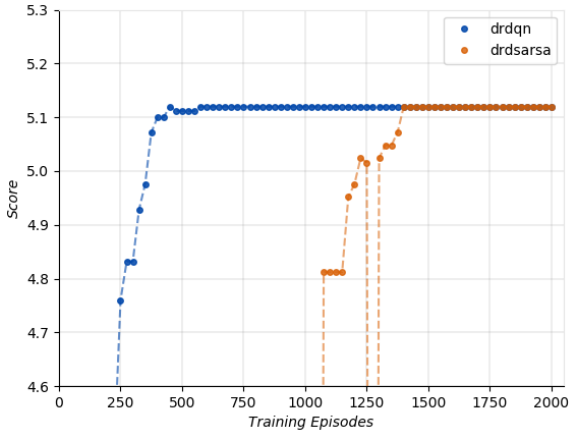


Figure 3.7: drDSARSA loss over training episodes.



(a) Testing Score - Complete Plot.



(b) Testing Score - Y-axis zoomed plot.

Figure 3.8: Testing score (evaluation phase) obtained in different points of the training phase.

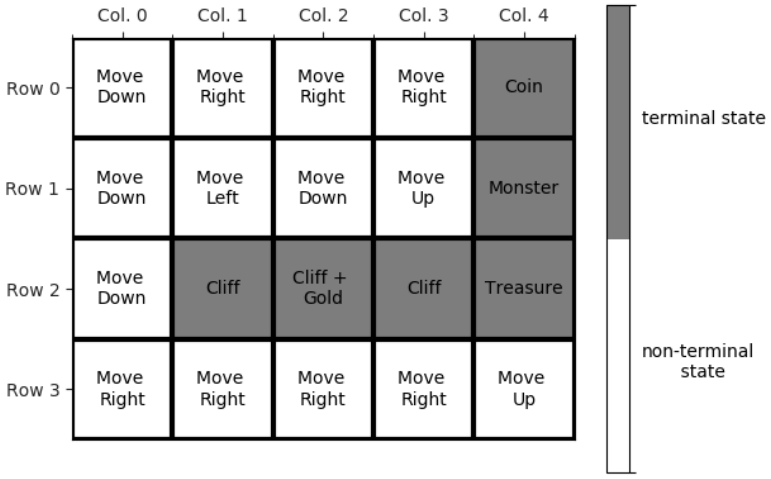
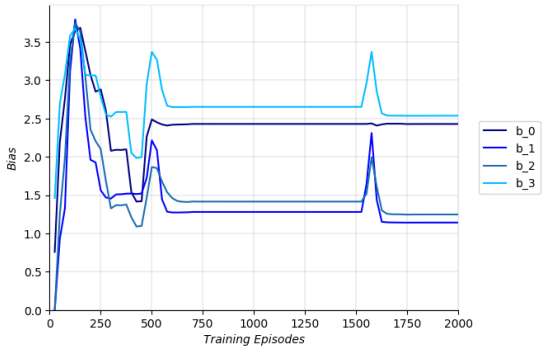
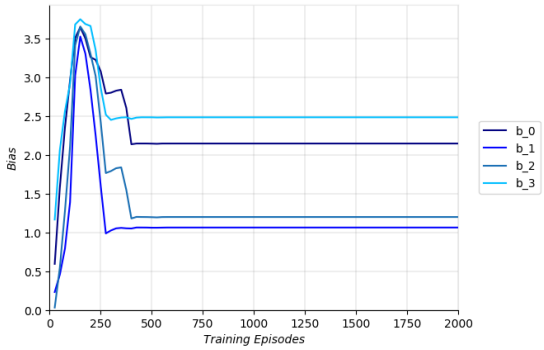


Figure 3.9: drDQN and drDSARSA policy at the end of the training phase (same policy for both algorithms).



(a) Run 1.



(b) Run 3.

Figure 3.10: drDQN - Treasure Q-network output layer biases during run 1 and 3.

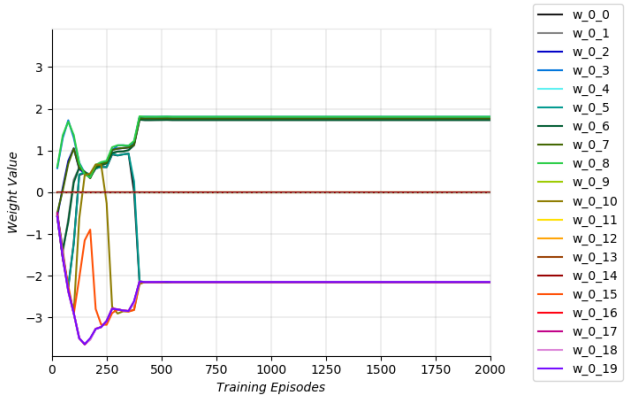
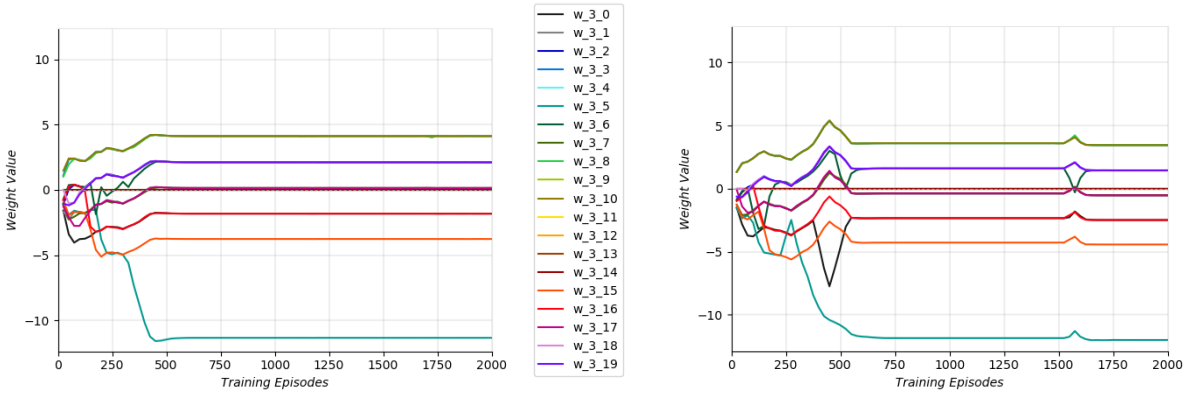


Figure 3.11: drDQN - Treasure Q-network output neuron 0 weights during run 3.



(a) Run 0.

(b) Run 1.

Figure 3.12: drDQN - Coin Q-network output neuron 3 weights during run 0 and 1.

3.2.4. Explainability Results

The previous subsection showed that drDQN successfully solves the modified CliffWorld environment. This subsection explains the reasons behind drDQN's behaviour by extracting explanations from the decomposed agent, which are presented in 3 forms: 1) Q-value bars; 2) RDX bars; and 3) MSX bars.

First, this subsection describes the way in which the explanations generated in this work differ from the ones presented by Juozapaitis et al. (2019). Then the explanations generated by the agent are analysed, demonstrating how this solution increases interpretability. Finally, the explanations generated at different stages of the learning process are compared to show how this approach can provide insights into the agent's training process.

Changes in the Explanation Generation Process:

When compared to Juozapaitis et al. (2019), this research presents two modifications in the way that explanations are generated and presented to the user. Such modifications were implemented with the objective of increasing drDQN's interpretability even further.

As mentioned in the previous chapter, Juozapaitis et al. (2019) successfully compare actions by analysing the trade-offs among different reward types using RDX and MSX explanations. For most cases this approach works flawlessly, however, when both actions lead to similar future rewards (indistinguishable to the user's eye), it becomes hard for the user to know that the agent is choosing one action over another based on very small estimated differences. In this case, the explanation might easily become confusing.

One such example of this dynamic happens in the CliffWorld state (1,2). In this state, the action ”Move Down” leads to a future reward of zero [$Cliff(-10) + Gold(+10) = 0$] while heading to the coin reward also provides a future reward of zero [$Coin(+4) + 2 \times Time\,(2 \times -2) = 0$]. The RDX plot generated by following the original paper's method is presented in figure 3.13, from where it is difficult to understand why the agent is choosing ”Mode Down” instead of ”Move Up”.

To increase user's understanding in these situations an alert signal is added, which informs the user whenever the two estimated future rewards compared differ less than 1% from one another. The updated RDX plot for state (1,2) is presented in figure 3.14, where it is clear that both actions lead to similar rewards and that the agent is choosing one of them based on very small estimated differences. It is hypothesised that: 1) This helps the user to quickly understand that both actions lead to similar rewards; and 2) It might increase trust in the model, since the algorithm is now capable of validating other strategies as optimal solutions (other than the one executed by the agent).

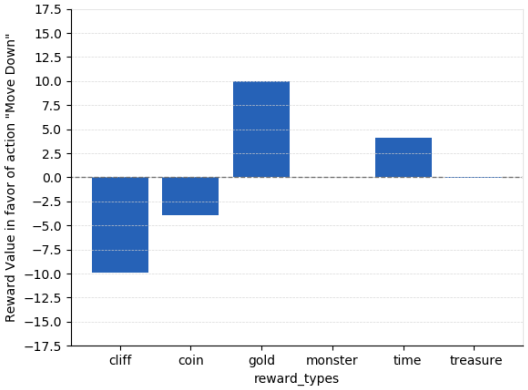


Figure 3.13: Old RDX plot between action ”Move Down” and ”Move Up” for state (1,2) using drDQN.

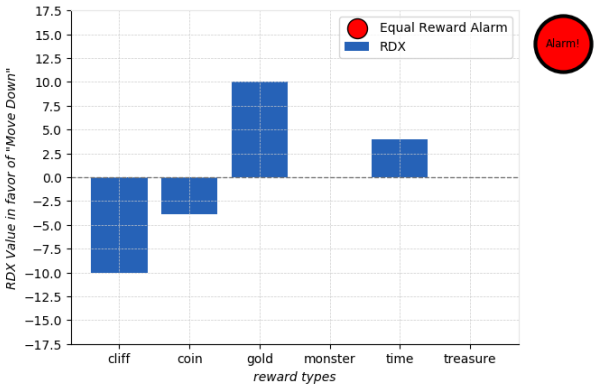


Figure 3.14: Updated RDX plot between action ”Move Down” and ”Move Up” for state (1,2) using drDQN.

The second (and final) modification to the way in which explanations are produced tackles boundary cases where the MSX^+ is only slightly bigger than the MSX^-. In these situations, generating the explanations as described by Juozapaitis et al. (2019) might lead the MSX plot to present incomplete information. State (0,2) is a good example of this dynamic. Intuitively, in this state the action ”Move Down” leads to a negative reward [$Gold(+10) + Cliff(-10) + Time(-2) = -2$] and the action ”Move Right” (towards the coin) leads to a positive reward [$Coin(+4) + Time(-2) = +2$]. Therefore, the action ”Move Right” is preferable. This logic is perfectly described by the RDX plot in figure 3.15, where the RDX time value is zero since the number of steps required to reach the gold bar is the same as the number of steps to reach the coin.

However, if one follows the original MSX definition (Juozapaitis et al., 2019), then the resulting MSX plot for state (0,2) is the one presented in figure 3.16. In this figure the coin reward is not present, leading to an incomplete explanation. The root of this problem can be traced back to the fact that the RL agent does not have access to the real environment reward values and has to work with estimates learned during training. In this specific case, while learning the agent estimates that the absolute value of the (negative) cliff reward is slightly bigger than the positive reward obtained from the gold bars, even if this difference is practically insignificant to the user. With this estimate error, the cliff reward alone has an absolute value that is bigger than the sum of all rewards in favour of ”Move Down” and, therefore, the MSX^+ tuple by definition only contains the cliff reward. This leads to an incomplete explanation since the coin reward is essential for the user to understand the preference for the action ”Move Right”.

To address this, a modification is made to the MSX generation process, where if the equal reward alarm is not activated and the difference between MSX^+ and MSX^- is less than 3%, then the next highest reward type in favour of the action chosen by the agent is added to the MSX^+ set. This small change effectively solves the boundary cases previously described, and the updated MSX plot for state (0,2) is presented in figure 3.17.

The two modifications made to the explanation generation process guarantee that the model produces complete and satisfactory explanations, even in the boundary cases described. Now that the required modifications have been performed, it is possible to demonstrate how using drDQN leads to better end-user interpretability.

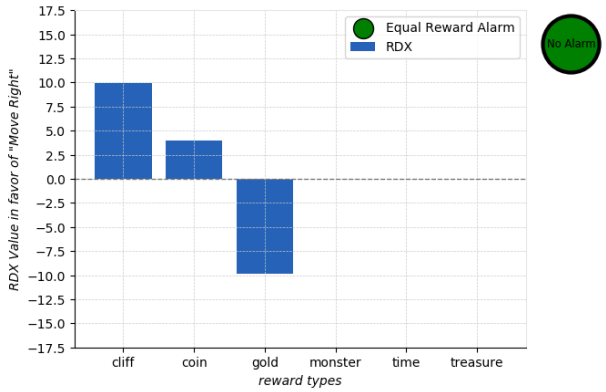


Figure 3.15: Updated RDX plot between action ”Move Right” and ”Move Down” for state (0,2) using drDQN.

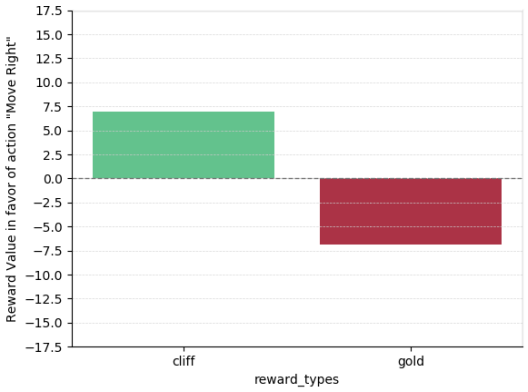


Figure 3.16: Old MSX plot between action ”Move Right” and ”Move Down” for state (0,2) using drDQN.

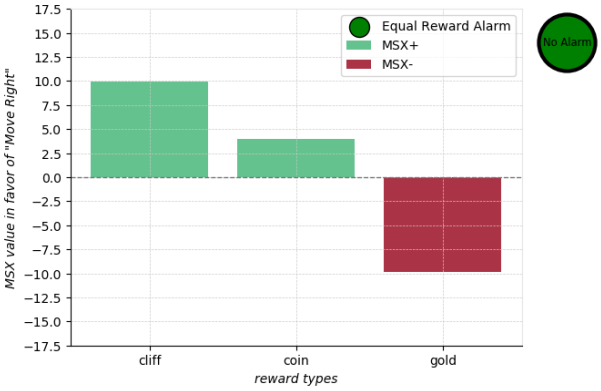


Figure 3.17: Updated MSX plot between action ”Move Right” and ”Move Down” for state (0,2) using drDQN.

Demonstrating the increase in explainability provided by drDQN:

Below, several relevant states of the modified CliffWorld environment are analysed to show how reward decomposition increases explainability for end-users. The explanations are presented in 3 forms: 1) Q-value bars; 2) RDX bars; and 3) MSX bars.

State (0,0):

In state (0,0) the optimal action is ”Move Down” (towards the treasure reward) since it provides an overall reward of +1 [$Treasure(+15) + 7 \times Time\,(7 \times -2) = +1$]. The actions ”Move Up” and ”Move Left” provide an immediate negative reward of -2 (due to the time reward component) and do not change the environment state. The action ”Move Right” leads to an overall reward of -2 [$Coin(+4) + 3 \times Time\,(3 \times -2) = -2$]. This same rationale can be easily extracted from the Q-value bars presented in figure 3.18, which shows the power of reward decomposition explanations.

To provide further insight into why action ”Move Down” is better than action ”Move Right”, the RDX plot is presented in figure 3.19. From that figure, it is clear that the treasure reward alone (which favours action ”Move Down”) completely overrides all reasons in favour of the action ”Move Right”. This provides a complete explanation as to *”why”* the agent prefers ”Move Down” over ”Move Right”.

The MSX plot for state (0,0) between actions ”Move Down” and ”Move Right” is presented in figure 3.20. It is evident that this plot is in accordance with the MSX definition since the treasure reward alone

is enough for the agent to decide to ”Move Down” and the time reward is the critical disadvantage that makes all the reward components present in the MSX$^+$ necessary. However, the user might need access to the coin reward value for the explanation to be considered complete and satisfactory. Hence, the use of the MSX explanation is not advisable for this case.

The limited usefulness of MSX explanations was observed not only for this state but also for many other states. Generally, MSX either: 1) Offered very little information compression when compared to RDX; or 2) Presented incomplete information, leading to an unsatisfactory explanation. This limited usefulness is further discussed in this chapter’s conclusion (refer to section 3.3).

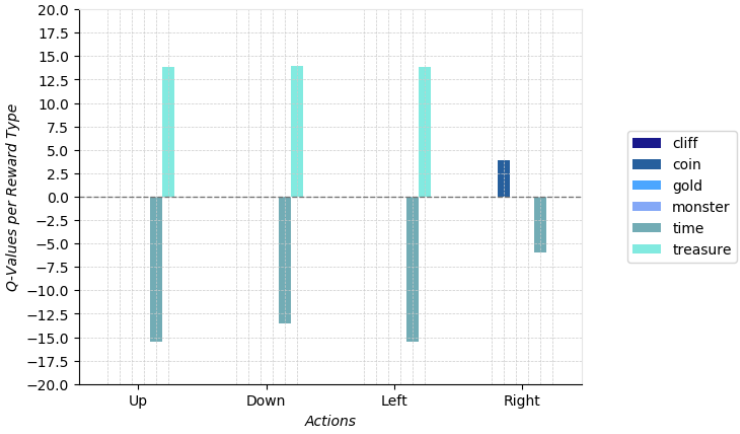


Figure 3.18: Q-Value bars for state (0,0) using drDQN.

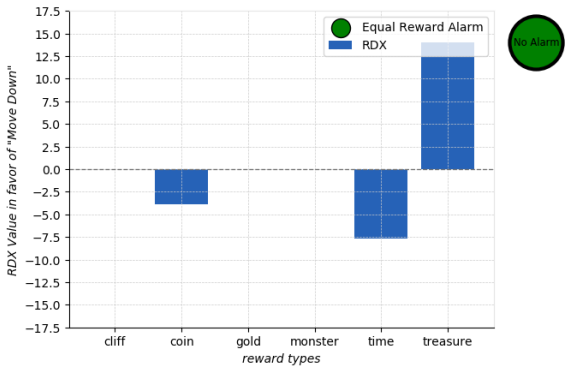


Figure 3.19: RDX plot between action ”Move Down” and ”Move Right” for state (0,0) using drDQN.

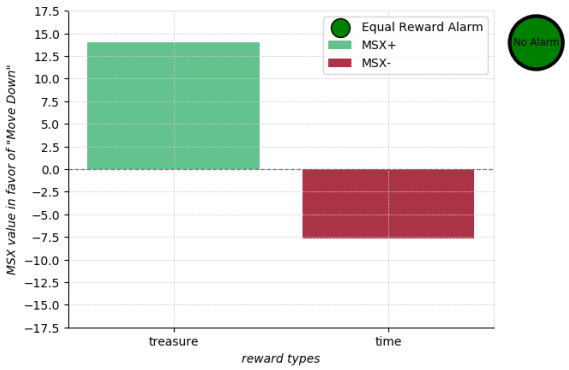


Figure 3.20: MSX plot between action ”Move Down” and ”Move Right” for state (0,0) using drDQN.

State (1,1):

State (1,1) might lead the user to be confused about why the agent moved in the treasure direction (”Move Left”) instead of moving to the gold bar (”Move Right”). By using such an intuitive environment it is easy to understand what the best action is and to verify the explanations produced.

Moving in the treasure direction provides an overall reward of +1 [$Treasure(+15) + 7 \times Time\,(7 \times -2) = +1$] whereas going for the gold bar provides a future reward of -2 [$Gold(+10) + Cliff(-10) + Time(-2) = -2$]. Hence, ”Move Left” is preferable. This rationale can be extracted from the Q-value bars presented in figure 3.21. However, with Q-value bars the user needs to perform calculations to arrive at that conclusion, that is, calculate the overall reward obtained for each action and then compare those values to see which option provides a bigger reward. By using RDX it is possible to obtain a much more straightforward explanation on the agent’s behaviour.

RDX reduces the amount of information presented by directly showing the reward difference between two actions. The RDX plot presented in figure 3.22 shows that the gold reward ”cancels” with

the cliff reward and that the treasure reward (which favours ”Move Left”) is bigger than the time reward. Consequently, the action ”Move Left” is preferable. This example shows how using RDX explanations leads to more straightforward conclusions and possibly lower decision times when compared to Q-value bars.

For this state the MSX plot shown in figure 3.23 contains a complete explanation, however, it offers very little information compression when compared to the RDX explanation. As such, the MSX presents (again) very little usefulness.

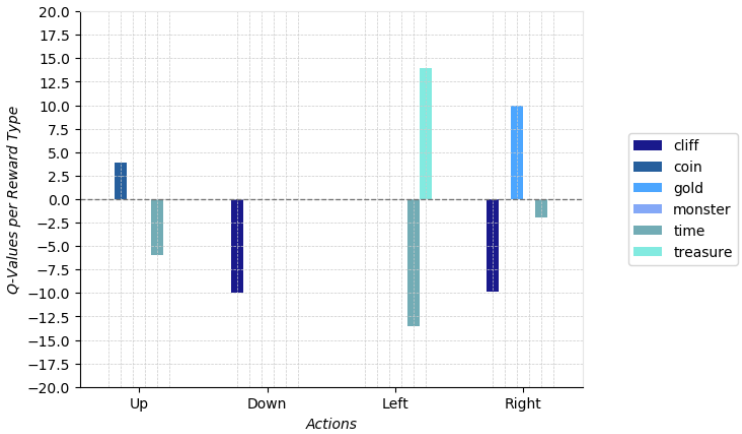


Figure 3.21: Q-Value bars for state (1,1) using drDQN.

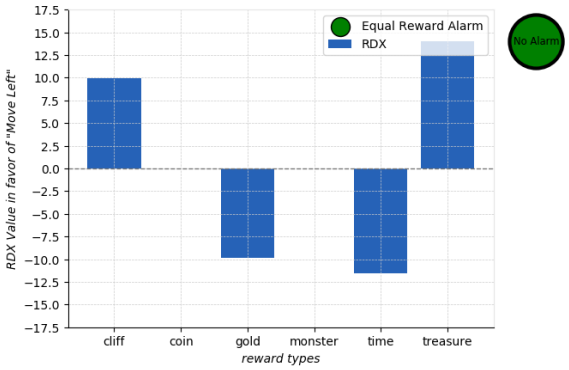


Figure 3.22: RDX plot between action ”Move Left” and ”Move Right” for state (1,1) using drDQN.

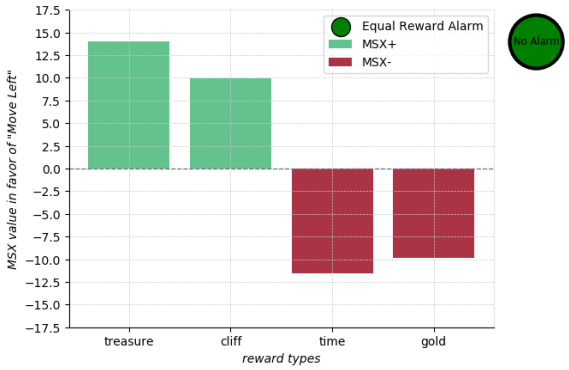


Figure 3.23: MSX plot between action ”Move Left” and ”Move Right” for state (1,1) using drDQN.

State (1,2):

The final state analysed in this work demonstrates how reward decomposition explanations can validate strategies that are different from the one followed by the agent. As described in table 3.3, in state (1,2) three actions lead to the same future reward: ”Move Up”, ”Move Down” and ”Move Right”. By referring to the final policy (shown in figure 3.9) it is clear that the agent chooses the action ”Move Down”, however, it is hard to understand why the agent opts for such an action when other options provide similar future rewards. In this case the Q-value bars (presented in figure 3.24) do not help to understand this either, since all 4 actions seem to provide similar overall returns.

In this case, the RDX explanations presented in figures 3.25 and 3.26 are essential to understand the agent’s decision:

1. Figure 3.25 shows that ”Move Down” is preferable over ”Move Left” since the cliff reward ”cancels” with the gold reward but the time reward (that favours ”Move Down”) outweighs the treasure reward (that favours ”Move Left”).

2. Figure 3.26 shows that the equal reward alarm is active while comparing actions ”Move Down” and ”Move Right”. As such, both actions lead to similar future rewards and the agent is simply choosing one of them based on small estimated differences learned during the training phase.

This demonstrates how RDX plots increase a method’s interpretability and how including an alarm signal can be essential for situations where two actions lead to similar future rewards.

Just like in the previous state analysed, the MSX explanations presented in figures 3.27 and 3.28 offer very little information compression when compared to RDX.

Table 3.3: Future Expected Reward for each action in state (1,2).

Action	Total reward	Explanation based on the environment
Move Up	0	$Coin(+4) + 2 \times Time\,(2 \times -2) = 0$
Move Down	0	$Cliff(-10) + GoldBar\,(-10) = 0$
Move Left	-1	$Treasure(+15) + 8 \times Time\,(8 \times -2) = -1$
Move Right	0	$Coin(+4) + 2 \times Time\,(2 \times -2) = 0$

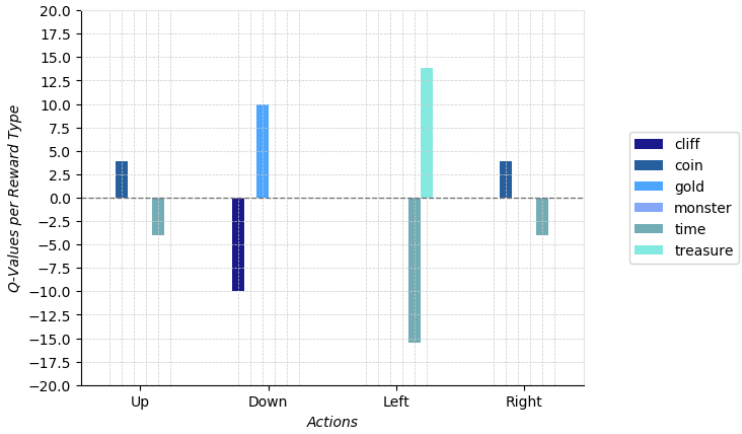


Figure 3.24: Q-Value bars for state (1,2) using drDQN.

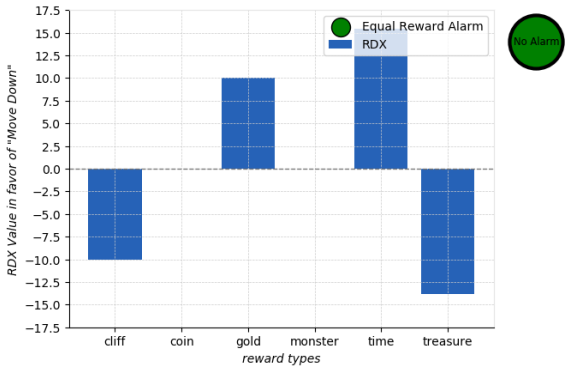


Figure 3.25: RDX plot between action ”Move Down” and ”Move Left” for state (1,2) using drDQN.

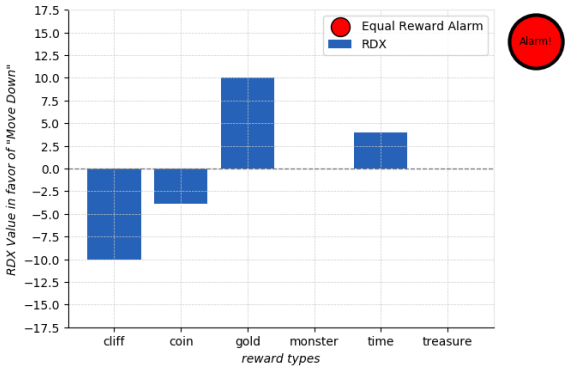


Figure 3.26: RDX plot between action ”Move Down” and ”Move Right” for state (1,2) using drDQN.

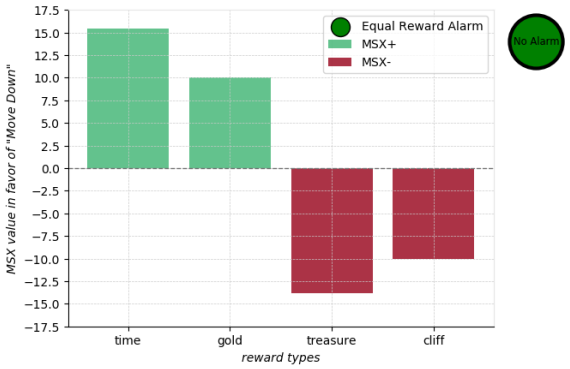


Figure 3.27: MSX plot between action ”Move Down” and ”Move Left” for state (1,2) using drDQN.

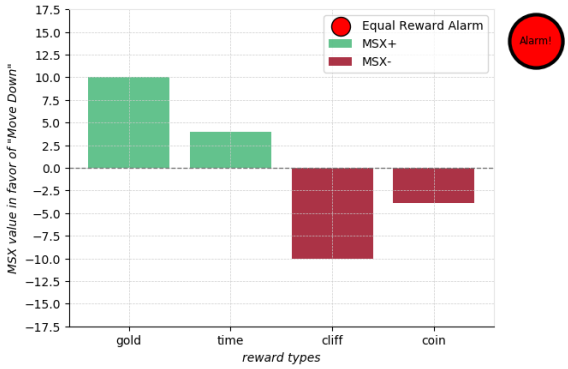


Figure 3.28: MSX plot between action ”Move Down” and ”Move Right” for state (1,2) using drDQN.

With these three relevant states analysed, it is clear how Reward Decomposition can be used to increase a method’s interpretability. Moreover, it is unclear how some of the insights mentioned in this subsection could be noticed without using a Reward Decomposition solution. For example, it would be hard to know that in state (0,2) the agent estimates that the absolute value of the (negative) cliff reward is slightly bigger than the positive value from the gold reward, even though the environment provides the same absolute reward value for both.

Explanations generated for an under-trained agent:

Unlike other XRL methods, Reward Decomposition does not have to wait for the network weights to converge in order to generate accurate explanations. Consequently, this solution can produce explanations during the whole training process, allowing for insights on why the agent’s decisions vary as learning progresses. Even though such insights might not be relevant to end-users since they usually require previous RL knowledge, they can improve a developer’s knowledge about the algorithm.

To illustrate this, figure 3.29 presents the Q-value bars for an under-trained drDQN in state (0,0). When comparing these results with the ones retrieved at the end of the training phase (refer to figure 3.18), it is clear that the under-trained agent underestimates the treasure reward obtained from moving down, which indicates that the agent needs more exploration to achieve a better estimate for that reward type. With time and exploration, the agent eventually arrives at a correct estimate for the treasure reward (as it can be seen from figure 3.18) leading to a change in the optimal action in state (0,0) from ”Move Right” to ”Move Down”.

Contrarily to what is seen with the treasure reward, the under-trained drDQN already provides an almost exact estimate for the reward obtained by moving in the coin direction [$Coin(+4) + 3 \times Time\,(3 \times -2) = -2$], which suggests that the agent already explored that alternative path.

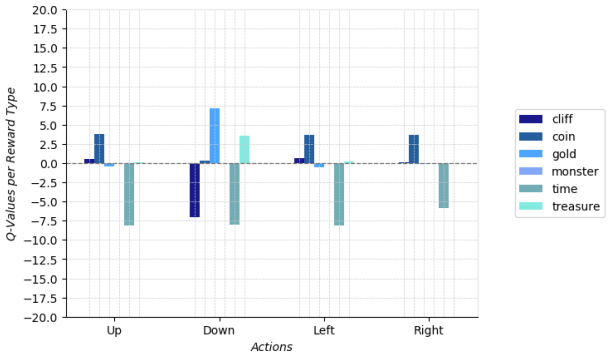


Figure 3.29: Q-Value bars for state (0,0) using under--trained drDQN.

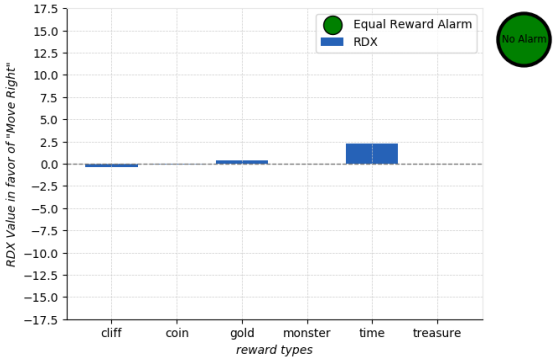


Figure 3.30: RDX plot between action ”Move Right” and ”Move Up” for state (0,0) using under-trained drDQN.

This demonstrates how using Reward Decomposition can provide insights about the agent's learning process and how it might help developers *"debug"* their implementations.

3.3. Chapter Conclusion

This chapter's purpose was to perform a preliminary analysis that compared the two XRL methods selected during the literature review, LMUTs and Reward Decomposition. By using the insights obtained a final XRL solution is chosen for this research's main task.

In section 3.1, it was clear that LMUTs are unsuitable for this research since, in the simple environment tested, they presented a training time that would make it impossible to use this solution in this project's main task given the available resources. After analysis, two causes were identified for LMUTs difficulty in dealing with large state spaces: 1) For every training step, LMUT needs to run SGD in every tree node; and 2) When performing a node split, this algorithm checks all possible ways to split the input space in order to find the split that maximises performance. Moreover, the tree quickly grows to sizes where its transparent structure no longer benefits interpretability. Given all these factors, LMUTs are no longer considered a relevant solution for this project.

In section 3.2, Reward Decomposition was explored by implementing both drDQN and drDSARSA in the Cliffworld environment. drDSARSA was only implemented to compare its task performance relative to the off-policy drDQN, and not to produce satisfactory explanations. Such is the case since, as noted by Juozapaitis et al. (2019), drDSARSA can provide counter-intuitive explanations due to learning from an exploratory policy. The topic of how to extract satisfactory explanations from on-policy methods using Reward Decomposition exceeds the scope of this research.

Both algorithms achieved optimum task performance in a modified CliffWorld environment. Nevertheless, drDQN presented superior learning performance since it obtained considerably lower loss values and was faster to converge to the optimal solution. This performance difference might be related to the environment in which both algorithms were implemented, however, these results are still worth considering when using a Reward Decomposition approach.

The results obtained while using LMUTs combined with the optimum performance achieved by drDQN answers the remaining of Q2.2. This Research Question was partially answered in the last chapter where it was stated that LMUTs and Reward Decomposition were the solutions with the highest potential to increase end-user acceptance in an attitude flight control task. In this chapter it was shown that Reward Decomposition presents a clear superiority when compared to LMUTs. Therefore, Reward Decomposition is the method with the highest potential to increase end-user acceptance while presenting good performance in this project's main task, which fully answers Q2.2.

During subsection 3.2.3 it was clear that not only does drDQN achieve optimal performance in the CliffWorld environment, it also presents stable performance across all 5 different test runs (which fully answers Q2.3).

With drDQN's task performance verified, subsection 3.2.4 demonstrated how this approach increases end-user explainability. It was seen that by analysing the trade-offs among different reward types (mainly using Q-value bars and RDX plots) the agent's decisions were explained in an intuitive and satisfactory way. This answers Q2.4 given that drDQN effectively increased explainability for situations where the reasons for the agent's decisions were not obvious.

Furthermore, it was stated that Reward Decomposition does not have to wait for the network weights to converge in order to extract explanations from the agent. Consequently, this solution allows developers to see how decisions change as learning progresses, uncovering training insights that would probably be hidden otherwise (e.g. where in the environment the agent still lacks exploration).

During the literature review chapter it seemed that presenting MSX explanations would be an effective and simple way to augment a method's interpretability, however, this preliminary analysis showed otherwise since such explanations either: 1) Offered very little information compression when compared to RDX; or 2) The information presented in the MSX figure was not enough to provide a satisfactory explanation. Furthermore, all the agent's decisions were already explained in a complete and not overwhelming manner by RDX and, consequently, there was no need for MSX explanations.

Nevertheless, it is important to note that this limited usefulness associated with MSX explanations

might be related to the specific environment used. If the number of reward types drastically increases, then MSX plots can be an effective way to present explanations without overwhelming the user. However, since the number of reward types is not expected to drastically increase for the attitude flight control task, the explanations generated for the final task will focus only on Q-value bars and RDX plots.

In this preliminary analysis Research Questions Q2.2, Q2.3 and Q2.4 were answered. It was stated that drDQN should be the XRL method used to automate the attitude control task of a business jet flight control application, which fully answers Research Question **Q2**. Given that Research Question **Q1** was already answered in the previous chapter, the remainder of this work will aim to answer Research Question **Q3**, that is, it will focus on implementing drDQN in an attitude flight control task and in extracting satisfactory explanations for the agent's behaviour.

III

Additional Results

4

Cessna Citation 500 Model

In the Scientific Article provided in Part I the longitudinal state-space model of the Cessna Citation 500 aircraft was presented, however, the state-space matrices coefficients were not shown. This chapter presents the complete model description for anyone interested in replicating the results from this research.

In order to get the aircraft model, the high-fidelity nonlinear DASMAT system was first trimmed around a steady-state flight condition of $h = 2000m$, $\gamma = 0°$, and $V = 90m/s$ (true airspeed). The relevant longitudinal aircraft data obtained from trimming the full nonlinear DASMAT model is presented in Table 4.1.

Table 4.1: Steady-state flight condition obtained from trimming the DASMAT model (only the relevant longitudinal data is presented).

Variable	Trim Value	Units
Aircraft mass	4500	kg
Altitude (h)	2000	m
True airspeed (V)	90	m/s
Thrust engine 1 (T_{N1})	1551.738	N
Thrust engine 2 (T_{N2})	1551.738	N
Load factor (n)	0.995	g
Pitch rate (q)	0	$°/s$
Angle-of-attack (α)	3.222	$°$
Pitch angle (θ)	3.222	$°$
Flight-path angle (γ)	0	$°$
Sideslip angle (β)	0	$°$
Elevator deflection (δ_e)	-1.632	$°$

Then, the trimmed model was linearized with the CitAST toolkit, resulting in the longitudinal state-space system described in Eq. (4.1). It should be noted that the values of $\boldsymbol{x}$, $\boldsymbol{u}$ and $\boldsymbol{y}$ are always relative to the aircraft's trim state described in Table 4.1. The procedures carried out to verify and validate this model are described in chapter 6.

$$\begin{cases} \dot{\boldsymbol{x}} = A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u} \\ \boldsymbol{y} = C \cdot \boldsymbol{x} + D \cdot \boldsymbol{u} \end{cases}, \quad \text{where} \tag{4.1}$$

$$\boldsymbol{x} = [q, V, \alpha, \theta, h, x_e]^T, \qquad \boldsymbol{u} = [\delta_e], \qquad \boldsymbol{y} = [q, V, \alpha, \theta, h, \eta, \gamma]^T,$$

$$A = \begin{bmatrix} -2.029822 & 0.001763 & -8.118122 & 0 & -5.630001 \cdot 10^{-6} & 0 \\ -0.250405 & -0.018827 & 4.171416 & -9.800333 & 6.543958 \cdot 10^{-5} & 0 \\ 0.969240 & -0.002482 & -1.417502 & 8.982614 \cdot 10^{-16} & 1.078785 \cdot 10^{-5} & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8.881784 \cdot 10^{-16} & -89.998500 & 89.998500 & 0 & 0 \\ 0 & 1 & 0 & -7.105427 \cdot 10^{-13} & 0 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} -12.662334 \\ -0.452766 \\ -0.145891 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0.282295 & 0.022776 & 12.938837 & 0 & -9.931858 \cdot 10^{-5} & 0 \\ 0 & 9.869867 \cdot 10^{-18} & -1 & 1 & 0 & 0 \end{bmatrix},$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1.338912 \\ 0 \end{bmatrix}$$

5

Hyperparameter Selection

Since DRL performance highly depends on the choice of suitable hyperparameters (Nguyen et al., 2020), an hyperparameter tuning phase was carried out in this work, where the number of hidden layers, number of units per layer, learning rate, batch size and target update interval were tuned. The results from the hyperparameter tuning phase are described in this chapter.

It should be noted that due to this research's timeline and the fact that drDQN takes considerably longer to train than DQN, this hyperparameter tuning phase was carried out using DQN and not drDQN. Although such is not ideal, the author still recognises the added value of incorporating this tuning phase given that, in the preliminary analysis, drDQN's performance was very similar to the DQN one.

Furthermore, each hyperparameter was tuned sequentially and, for each value considered, five runs were executed before choosing the best hyperparameter value. Even though more runs would have been desirable, performing only five runs guaranteed that the tuning phase was aligned with the available project resources and timeline. The only hyperparameter for which more runs were performed was the number of hidden layers, since with five runs it was not clear what the best hyperparameter value would be.

First, the number of hidden layers was tuned by testing DQN's performance with two, three and four hidden layers of 128 units each. The other hyperparameters used while tuning the number of hidden layers are described in Table 5.1. The resulting learning and loss curves averaged over 25 runs are presented in Fig. 5.1 and 5.2, respectively, where "σ band" refers to the region between the "$\mu - \sigma$" and "$\mu + \sigma$" curves. . From these results it becomes clear that using two hidden layers leads to a lower task score and higher loss values. Additionally, it leads to a larger σ band, indicating that there is a higher variability between runs. Consequently, the use of two hidden layers is discarded from this research.

When deciding between using three or four hidden layers, Figs. 5.1 and 5.2 show that both options result in similar task scores and loss values, however, using three hidden layers leads to a higher variability between runs as pointed out by the σ band. Nevertheless, using four hidden layers considerably increases training time when compared to three hidden layers. Given that the overall results are similar and that there is a considerable training time difference, the author opted to use three hidden layers with 128 units each.

Table 5.1: DQN hyperparameters used while tuning the number of hidden layers hyperparameter.

Hyperparameter Name	Value
Number of hidden layers	2, 3 or 4
Neurons per hidden layer	128
Memory buffer size	$1 \cdot 10^6$
Batch size	128
Target update interval	8192

Hyperparameter Name	Value
Discount factor	0.99
Learning rate	$1 \cdot 10^{-4}$
ϵ_{start}	0.75
ϵ_{min}	0.1
Episodes until ϵ_{min}	5500

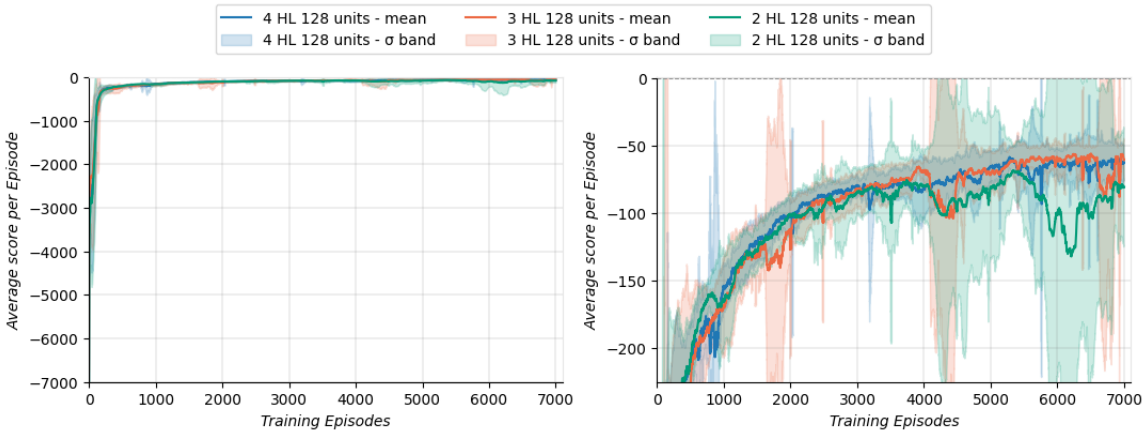


Figure 5.1: Learning curves averaged over 25 runs obtained for DQN while testing with different numbers of hidden layers. Left: Total plot. Right: y-axis zoomed-in plot.

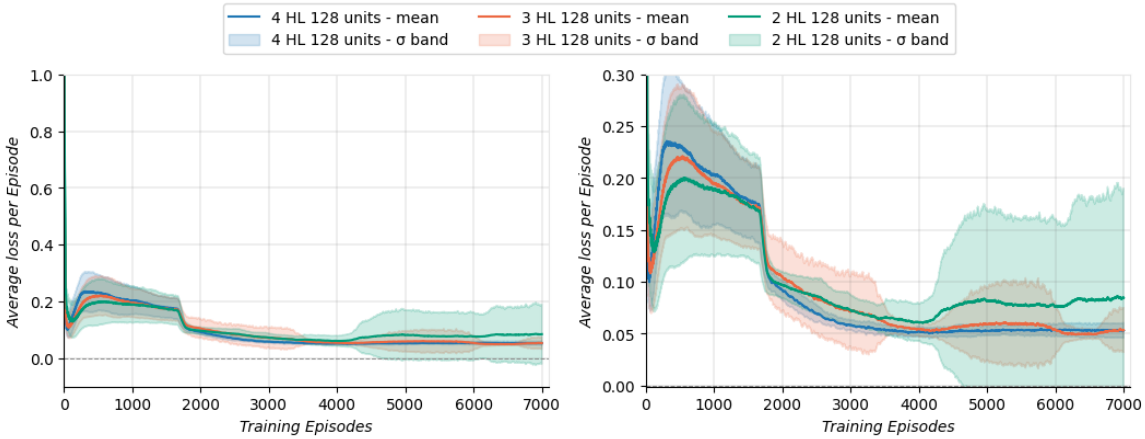


Figure 5.2: Loss curves averaged over 25 runs obtained for DQN while testing with different numbers of hidden layers. Left: Total plot. Right: y-axis zoomed-in plot.

After this initial tuning, the preliminary choice for three hidden layers with 128 units each was compared against similar settings that used fewer units per layer. More specifically, it was compared against using three hidden layers of 64 units, and with using four hidden layers of 64 units. The resulting learning and loss curves averaged over five runs are presented in Figs. 5.3 and 5.4, respectively. From these results it is clear that using three hidden layers of 128 units achieves a better task score and lower loss values than the other two alternatives. Consequently, the drDQN used in this work will use three hidden layers with 128 units each.

The next hyperparameter tuned was the learning rate, where four choices were considered: 0.0005, 0.0001, 0.00005 and 0.00001. In this tuning step, a DQN with three hidden layers of 128 units was used, and the other hyperparameters were the same as in the previous tuning step (refer to Table 5.1). Figures 5.5 and 5.6 show that learning rates of 0.0005 and 0.00001 lead to lower task scores and higher loss values than the other two alternatives. Consequently, they are not considered to be a suitable choice for this research.

When deciding between using a learning rate of 0.0001 or 0.00005, Fig. 5.5 shows that even though the final task score is similar for both options, a learning rate of 0.00005 presents a more stable learning curve. As such a learning rate of 0.00005 was selected for this research.

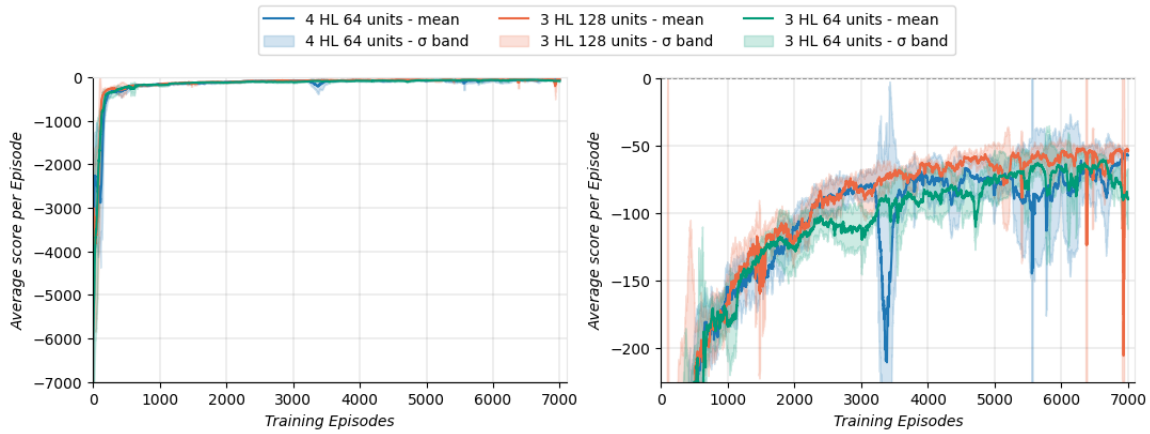


Figure 5.3: Learning curves averaged over 5 runs obtained for DQN while testing with different hidden layer settings. Left: Total plot. Right: y-axis zoomed-in plot.

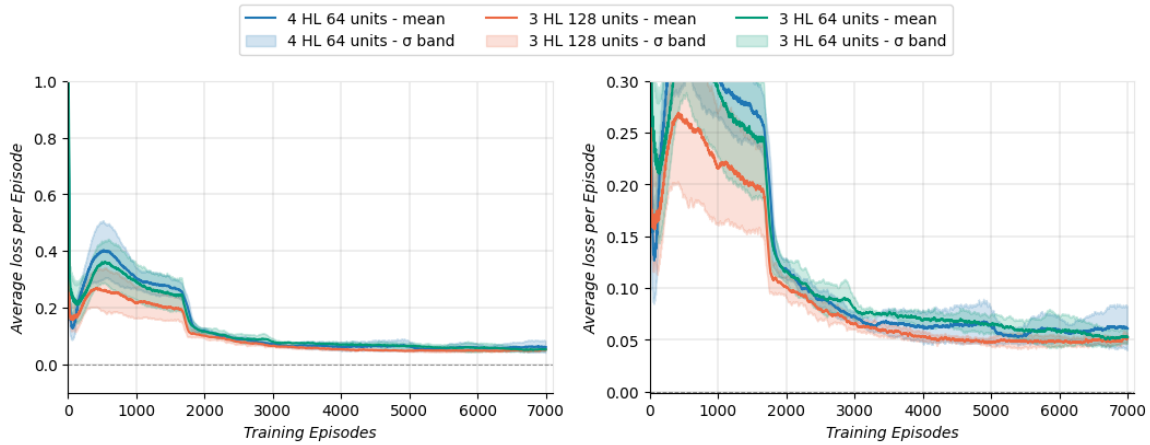


Figure 5.4: Loss curves averaged over 5 runs obtained for DQN while testing with different hidden layer settings. Left: Total plot. Right: y-axis zoomed-in plot.

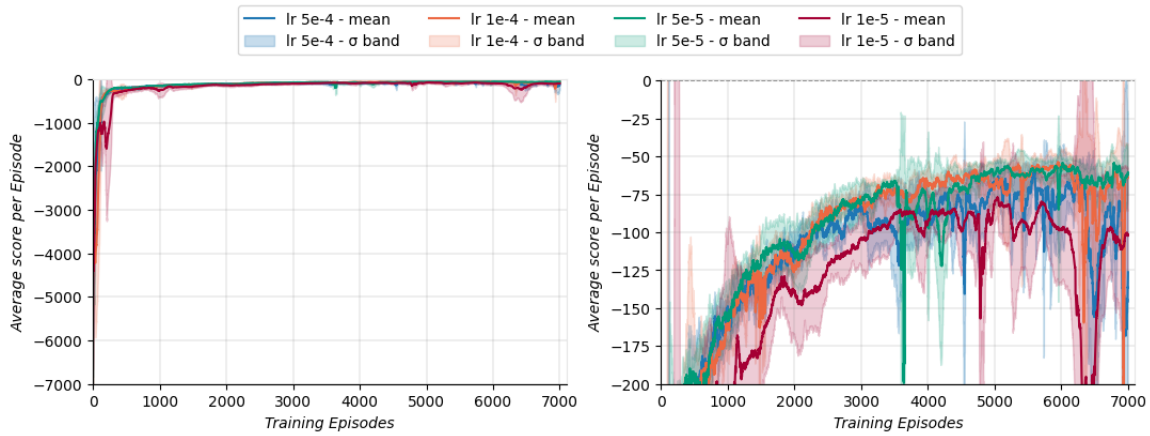


Figure 5.5: Learning curves averaged over 5 runs obtained for DQN while testing with different learning rates. Left: Total plot. Right: y-axis zoomed-in plot.

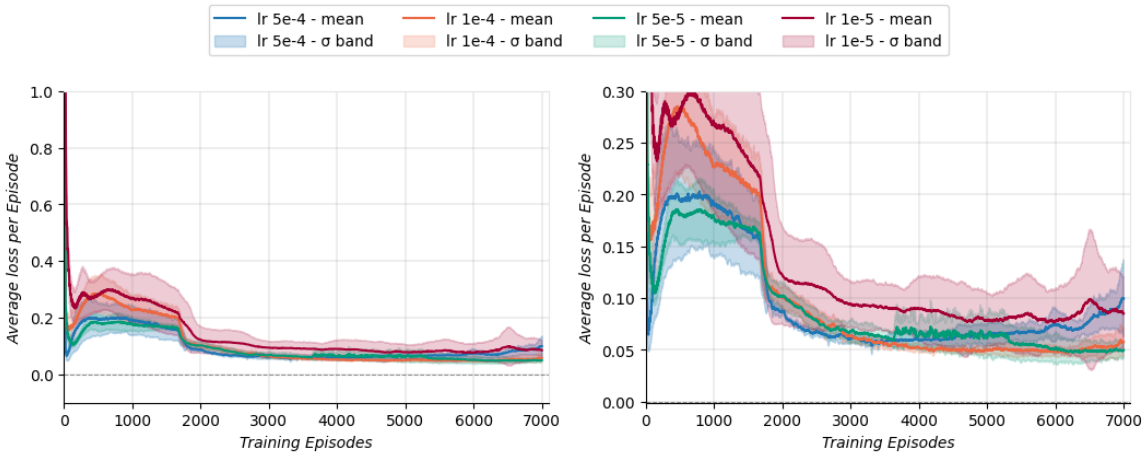


Figure 5.6: Loss curves averaged over 5 runs obtained for DQN while testing with different learning rates. Left: Total plot. Right: y-axis zoomed-in plot.

The next hyperparameter tuned was the batch size, where four batch sizes were considered: 32, 64, 128, 256. The other hyperparameters used while tuning the batch size are described in Table 5.2, which result from merging the data obtained in previous tuning steps with the initial hyperparameters used. The learning and loss curves averaged over five runs obtained while tuning the batch size are presented in Figs. 5.7 and 5.8, respectively. By analysing these figures one can see that:

- A batch size of 256 leads to the lowest task score and results in an unstable performance.
- A batch size of 128 leads to a slightly lower task score than the one obtained by batch sizes of 32 and 64. Additionally, it presents a higher σ band in the loss plot, which indicates that there is more variability between runs.
- In Fig. 5.7 a batch size of 64 slightly outperforms a batch size of 32. However, given that this is a very small difference and only five runs were performed, it is hard to make a robust statement about which alternative is best for this experiment.

Based on this analysis, the best batch sizes for this experiment are 32 and 64, with a batch size of 64 slightly outperforming the 32 one. However, more runs are necessary to arrive at a more robust conclusion. To make a final decision, 25 extra runs were performed for these two batch size alternatives, and the results can be seen in Figs. 5.9 and 5.10. From these two figures it is clear that, even though the overall performance for both choices is similar, a batch size of 64 presents a more stable learning curve and a smaller σ band in the loss plot. Consequently, a batch size of 64 was selected for this research.

Table 5.2: DQN hyperparameters used while tuning the batch size hyperparameter.

Hyperparameter Name	Value
Number of hidden layers	3
Neurons per hidden layer	128
Memory buffer size	$1 \cdot 10^6$
Batch size	32, 64, 128 or 256

Hyperparameter Name	Value
Target update interval	8192
Discount factor	0.99
Learning rate	$5 \cdot 10^{-5}$
ϵ_{start}	0.75
ϵ_{min}	0.1
Episodes until ϵ_{min}	5500

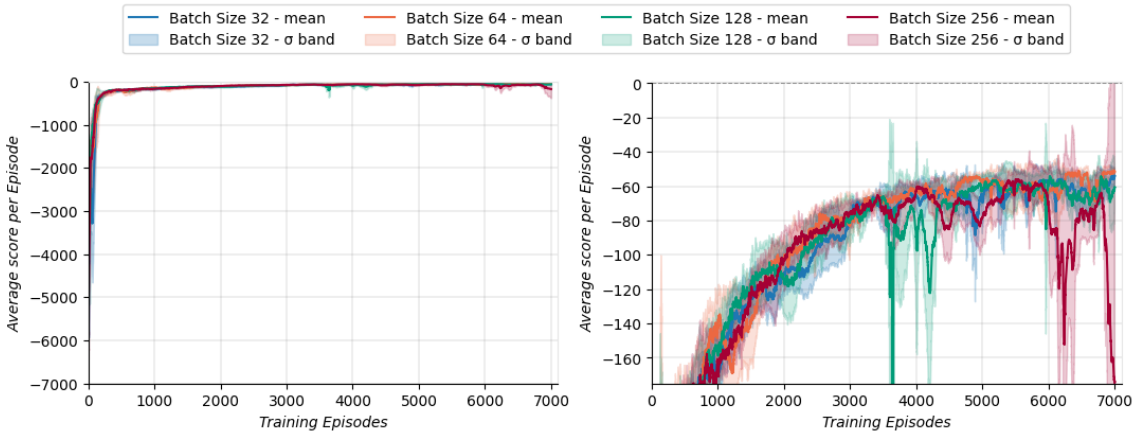


Figure 5.7: Learning curves averaged over 5 runs obtained for DQN while testing with different batch sizes. Left: Total plot. Right: y-axis zoomed-in plot.

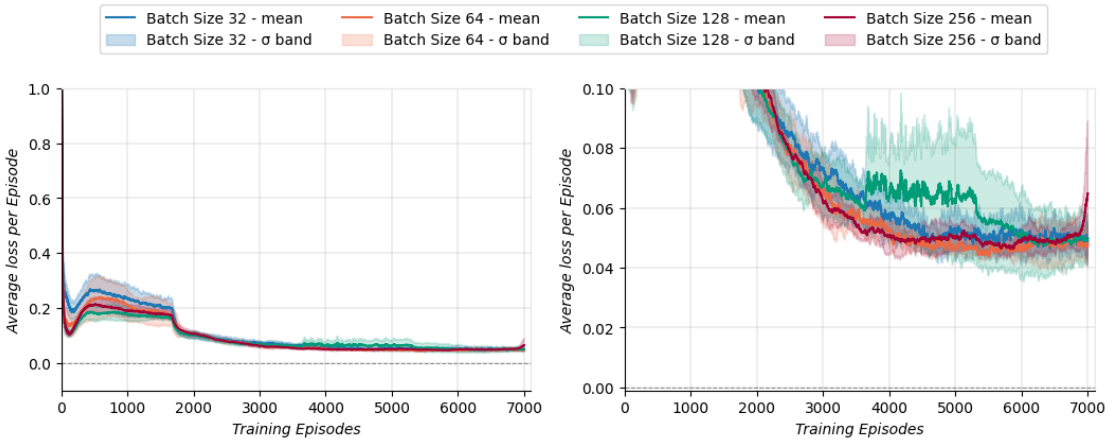


Figure 5.8: Loss curves averaged over 5 runs obtained for DQN while testing with different batch sizes. Left: Total plot. Right: y-axis zoomed-in plot.

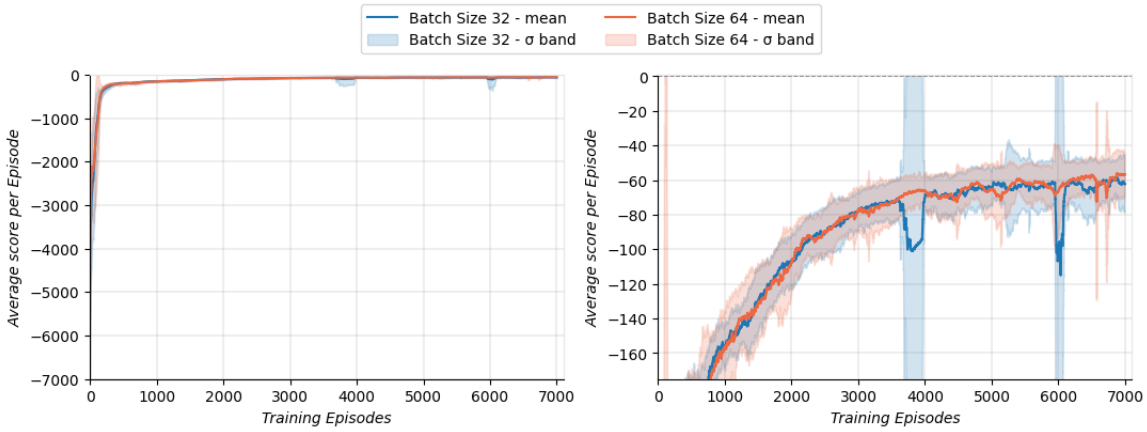


Figure 5.9: Learning curves averaged over 25 runs obtained for DQN while testing with batch sizes of 32 and 64. Left: Total plot. Right: y-axis zoomed-in plot.

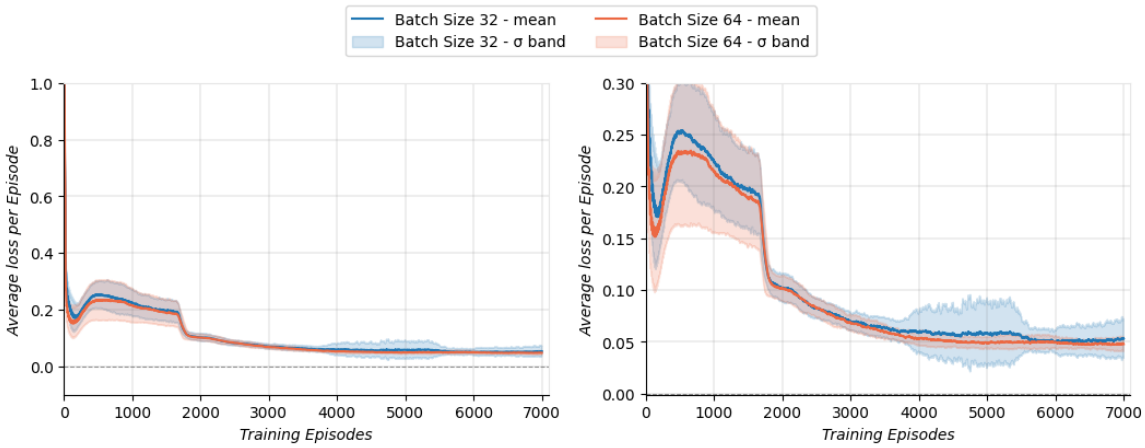


Figure 5.10: Loss curves averaged over 25 runs obtained for DQN while testing with batch sizes of 32 and 64. Left: Total plot. Right: y-axis zoomed-in plot.

The final hyperparameter tuned was the target update interval, where four alternatives were compared: 4096, 8192, 16384 and 32768. The other hyperparameters used while tuning the target update interval are the same as the ones described in Table 5.2, however, a batch size of 64 was used taking into account the last tuning step performed. The learning and loss curves obtained while tuning the target update interval are shown in Figs. 5.11 and 5.12, respectively. From these results one can see that:

- A target update interval of 32768 obtains the lowest task score and, consequently, is disregarded as a suitable choice for this work.
- The other three alternatives lead to very similar task score and loss curves.

Considering that three alternatives obtained similar learning performance (i.e. 4096, 8192 and 16384), the target update interval of 8192 used in previous tuning steps was maintained. This concludes the hyperparameter tuning phase.

To summarise this chapter's information, the final hyperparameters selected for this research's main experiment are outlined in Table 5.3.

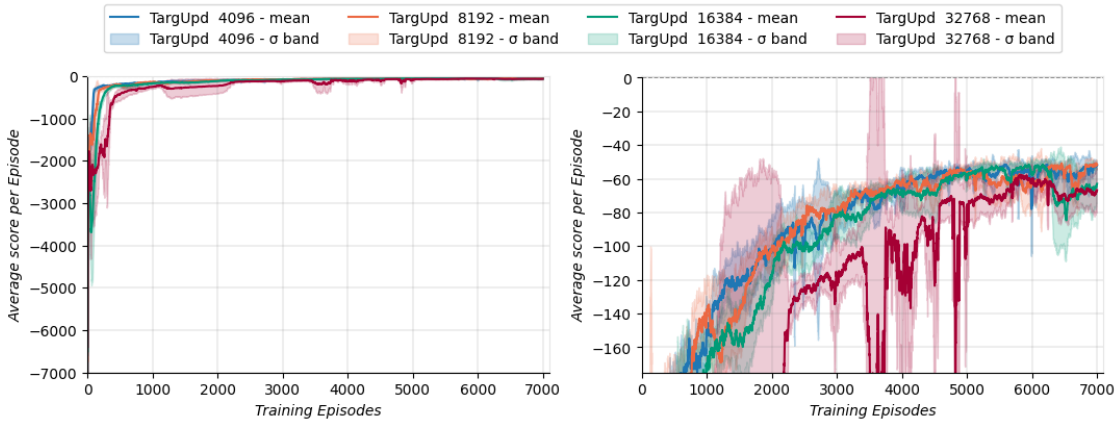


Figure 5.11: Learning curves averaged over 5 runs obtained for DQN while testing with different target update intervals. Left: Total plot. Right: y-axis zoomed-in plot.

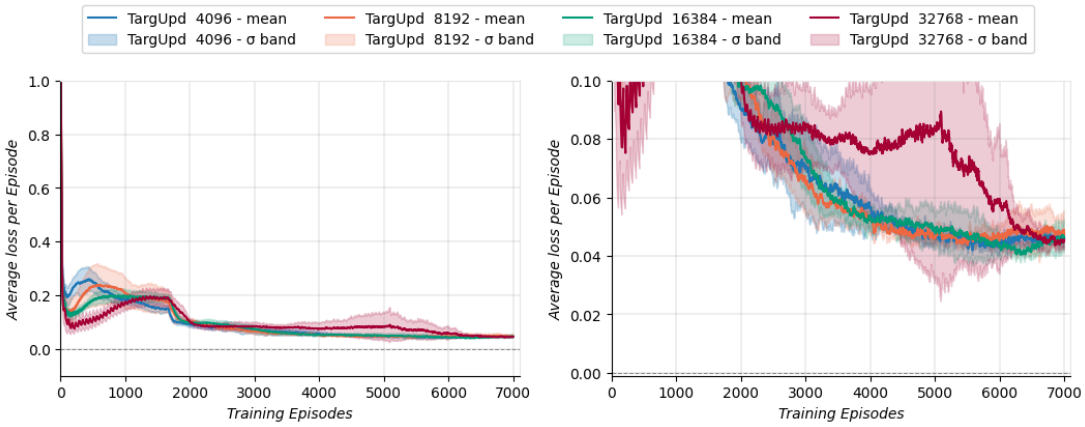


Figure 5.12: Loss curves averaged over 5 runs obtained for DQN while testing with different target update intervals. Left: Total plot. Right: y-axis zoomed-in plot.

Table 5.3: Final drDQN hyperparameters.

Hyperparameter Name	Value
Number of hidden layers	3
Neurons per hidden layer	128
Memory buffer size	$1 \cdot 10^6$
Batch size	64
Target update interval	8192

Hyperparameter Name	Value
Discount factor	0.99
Learning rate	$5 \cdot 10^{-5}$
ϵ_{start}	0.75
ϵ_{min}	0.1
Episodes until ϵ_{min}	5500

6

Verification and Validation

Before using the aircraft model and controller presented in this research, several verification and validation steps were carried out to ensure the validity of the results. This section presents the verification and validation phase of both the Cessna Citation 500 model and the DRL controller used.

6.1. Cessna Citation 500 Model

As described in the Scientific Article (refer to Part I), this project used a longitudinal state-space system that is based on the DASMAT high-fidelity nonlinear model of the Cessna Citation 500 aircraft (van der Linden, 1998). First, the full DASMAT model was trimmed and linearised around a specific operating point using the CitAST Toolkit (Borst, 2004). Second, the resulting state-space model was used to build an RL environment in Python with which the drDQN controller could interact. This environment propagates the internal state $\boldsymbol{x}$ through time in response to an input $\boldsymbol{u}$ by using a fourth-order Runge-Kutta (RK4) integration at a sample rate of 100Hz.

This aircraft model was verified by analysing both its trim and step responses. As described by Fig. 6.1, the model simply maintained its steady-state flight condition when given the trim elevator deflection input, which was expected. Alternatively, Fig. 6.2 shows that when faced with a positive elevator step input at $t = 10s$, the aircraft model responds with a negative pitch rate which, by using aircraft dynamics knowledge, is the expected response. Moreover, in this step response the two longitudinal eigenmodes are noticeable, with the short-period mode affecting mainly the first two seconds after the step input, and the phugoid mode dominating the rest of the response. It is apparent that the short-period mode mainly affects the angle-of-attack, whereas the phugoid oscillations lead to a slow interchange between kinetic energy (velocity) and potential energy (altitude) around a new equilibrium state, which is also the expected response. Since the model presents the expected trim and step responses, the state-space system used is verified.

As for the validation phase, both the DASMAT and CitAST tools have already been used extensively in previous works. Examples of previous research that validated these tools can be found in:

- van der Linden (1998)
- van Hoek et al. (2017)
- van Ingen et al. (2021)

Since the aircraft model used in this project was already validated by previous research, this work did not perform any further validation steps.

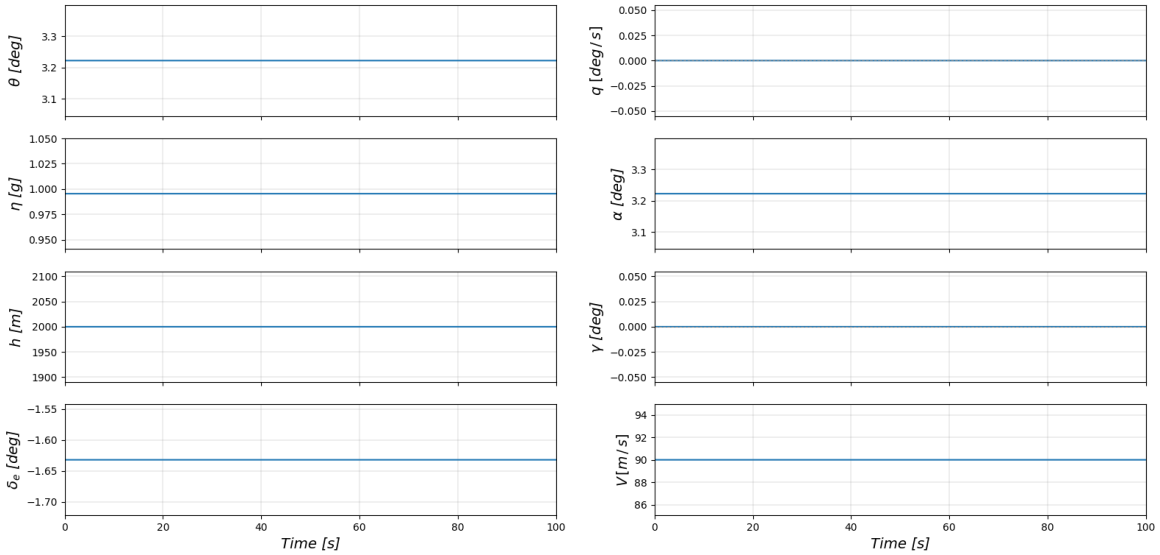


Figure 6.1: Aircraft model trim response.

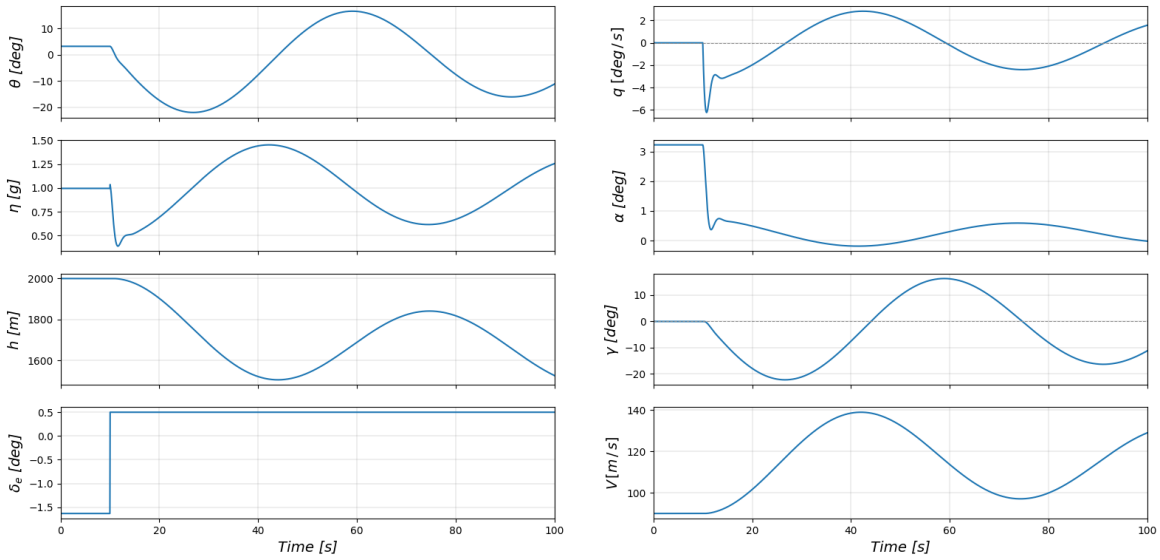


Figure 6.2: Aircraft model step response.

6.2. Reward Decomposition Controller

The reward decomposition controller was verified during the preliminary analysis chapter of this report, where it was applied in a CliffWorld environment similar to the one seen in Juozapaitis et al. (2019). In this environment, the controller achieved optimal performance and generated end-user suitable explanations.

As for the controller's validation, since this research's objective is to generate explanations that increase end-user DRL explainability, a proper validation phase is hard to perform unless human-in-the-loop experiments involving end-users are carried out. Nevertheless, taking into account the answer to research question **Q1**, the explanations generated by this solution are in alignment with end-user explainability requirements. Further research is needed to approve this validity.

IV

Closure

7

Conclusion

This chapter starts by recalling and answering the research questions previously identified. Then it proceeds to summarise the main contributions of this work, and ends by assessing to what degree the overarching research objective of this thesis was accomplished.

The first research question, presented again below, is answered by using the concepts outlined in Part II.

Q1: What are the biggest factors that contribute to end-user acceptance of XRL methods?

Q1.1) How to evaluate the increase in explainability provided by an XRL solution?

Q1.2) With the goal of increasing end-user acceptance, in what should the explanation focus on? 1) Focus on explaining the network processing; 2) Focus on explaining the expected future rewards; 3) Focus on where in the input the method is putting the biggest focus; or 4) Focus on aligning the explanation with the user's rationale.

Q1.3) In order to increase end-user acceptance, how does providing multiple explanation types compare with a single explanation type method?

In chapter 2, it was stated that (currently) there are no established metrics to evaluate a method's explainability and, as such, researchers usually focus on assessing the mental model produced by users after seeing the explanation. Typically, user studies are carried out to assess these mental models, where two question types are focused: 1) Post-task questions about the agent's behaviour; 2) Questions on predicting the agent's future actions. This allows researchers to convert qualitative and subjective results into quantitative ones. Hence, Q1.1 is answered since, to evaluate the increase in explainability provided by an XRL solution, one should resort to user studies where the user's mental model is assessed.

Furthermore, it was stated that when targeting end-user explainability, the explanations should focus on either: 1) Explaining the expected future rewards; or 2) Explaining where in the input the method is putting the biggest focus to reach a decision. Research indicates that these two explanation types increase end-user explainability when compared to a control group with no access to explanations (Anderson et al., 2020). The other two explanation types mentioned in Q1.2 are not considered ideal for end-users since:

- To the author's best knowledge, there are no methods that focus on aligning the explanation with the user's rationale that effectively increase trust in the model. One of the most renown methods in this category uses causal models, however, this approach did not increase trust for end-users (Madumal et al., 2020). Such might change in the future when these solutions become more interactive, but currently the use of these methods is not recommended when targeting end-user acceptance.

- Methods that focus on explaining the network processing are also not recommenced since they put a big focus on the informativeness goal, which is not essential for end-user acceptance. Moreover, to interpret the explanations generated by these solutions one usually needs AI knowledge, which makes them unsuitable for end-users.

This fully answers Q1.2.

During the Literature Review it was also shown that presenting more than one explanation type leads to users taking longer to reach a decision and, in some cases, might even overwhelm them with too much information. In case users are overwhelmed, the mental model developed after seeing the explanation is worse than the one obtained by users with access to a single explanation type. However, if the additional time is not an obstacle to the task and the amount of information displayed does not overwhelm users, then presenting multiple explanation types seems to increase explainability when compared to a single explanation type method. This provides an answer to Q1.3.

Additionally, Arrieta et al. (2020) claim that to augment explainability for end-users the explanations should aim to enhance trustworthiness, causality, transferability and confidence. Combining these goals with the explanation types previously selected as ideal for end-users provides a guideline for generating explanations that increase end-user explainability. This answers research question **Q1**.

The next research question relates to what specific method should be used in this research's main task. This question is answered by using the concepts outlined in Part II.

Q2: What XRL method should be used to automate an attitude control task for a business jet flight control system?

- Q2.1) What are the suitable XRL techniques available?
- Q2.2) Which XRL method has the highest potential to increase end-user acceptance while presenting good performance on the attitude flight control task?
- Q2.3) How does the XRL method chosen perform in a simple but relevant environment, in terms of learning stability and task performance?
- Q2.4) Is the XRL method chosen capable of providing satisfactory explanations in the simple but relevant environment?

The Literature Review chapter described the most relevant XRL solutions for this research, which answered Q2.1. From this chapter, LMUTs and reward decomposition were selected as the XRL solutions that presented the highest potential to increase end-user explainability in the attitude flight control task targeted by this research. This choice was mainly justified by the fact that most XRL solutions available either provide explanations that require previous AI knowledge or do not have a proven ability to increase user's trust in the model. However, a direct comparison between these two methods based solely in the literature was difficult since, in their original publications, they are implemented in very different environments. To choose the best solution for this project, chapter 3 presented a preliminary analysis that compared both methods in a simple but relevant environment.

From the Preliminary Analysis chapter, it was clear that LMUTs are unsuitable for this project since they present a training time that would make it impossible to implement this method in a flight control task, given the available resources. Additionally, during the learning phase, the LMUT tested quickly grew to sizes where its inherently transparent structure no longer benefited explainability.

Alternatively, the reward decomposition solution implemented, drDQN, presented good performance both in terms of learning stability and task score. As such, reward decomposition is selected as the XRL solution with the highest potential to increase end-user explainability in the attitude flight control task targeted by this research, which answers Q2.2. Moreover, since drDQN presented good learning stability and task performance across all runs, Q2.3 is also answered. It should be noted that the preliminary analysis also showed that reward decomposition on-policy methods should be avoided since they can generate counter-intuitive explanations due to learning from an exploratory policy.

To respond to Q2.4 the explanations generated by drDQN during the preliminary analysis were analysed. It was seen that, by using Q-value bars and RDX plots, this solution effectively increased explainability in situations where the reasons for the agent behaviour were unclear. Additionally, drDQN provided insights into how the agent's decisions varied as learning progressed.

Furthermore, MSX plots did not improve explainability during the preliminary analysis since they either: 1) Offered very little information compression when compared to RDX; or 2) The reward types presented in the MSX plot were not enough to provide a satisfactory explanation. It was concluded that MSX plots might only be advantageous in environments with a very large number of reward types. However, for all other cases, RDX already provides explanations that are complete, sound and not overwhelming. This answers Q2.4 given that, by using RDX, the reward decomposition solution was able to provide satisfactory explanations.

Research question **Q2** is fully answered as well considering that Q2.1, Q2.2, Q2.3 and Q2.4 were already responded, and that drDQN was selected as the XRL method to be used in the attitude flight control task targeted by this project.

The last research question is centred around integrating a reward decomposition agent on a Cessna Citation 500 flight control system (under a simulated environment), and on testing its performance. The answer to this question can be found in the Scientific Article provided in Part I.

Q3: How can a XRL method be used to improve end-user explainability in an attitude control task of a business jet flight control application?

- Q3.1) What are the states and outputs of the XRL agent?
- Q3.2) What manoeuvres will the aircraft perform during testing?
- Q3.3) What is the performance of the XRL method in keeping control of the aircraft during the attitude flight control task?
- Q3.4) Is the XRL method capable of providing satisfactory explanations for the attitude control task?
- Q3.5) How well does the chosen XRL method keep the explanations relevant during the whole task?

This research used a longitudinal linear model of the Cessna Citation 500 aircraft, and the controller's goal was to track a reference signal for the pitch angle, θ_{ref}, while complying with passenger comfort and altitude safety requirements. To achieve this goal, the agent commanded control input increments to the elevator, $\Delta\delta_e$, which (when compared to providing the total deflection) allowed for a much higher number of possible elevator configurations without demanding an excessively large action space. However, this strategy required δ_e to be included in the RL state vector. The load factor was also included as a state to ease the agent's task of complying with passenger comfort requirements. As such, the state vector selected for the agent to achieve optimal performance in the attitude flight control task was $\boldsymbol{s} = [\theta_{error}, q, V, \alpha, \theta, h, n, \delta_e]$.

Given that the mentioned state components can have very different orders of magnitude (e.g. pitch rate values are usually much smaller than altitude ones), all states were normalised in the $[-1, 1]$ range before being fed to the agent. This choice for the controller design answers Q3.1.

The XRL controller was evaluated on two tasks : 1) A task where following θ_{ref} did not lead to an altitude safety risk, which allowed for a better assessment of the controller's tracking performance and ability to stay within passenger comfort limits; and 2) A task where following the reference signal would lead to an altitude safety risk and, consequently, a trade-off between tracking θ_{ref} and staying within altitude bounds was needed to obtain optimal performance. Using these two tasks allow one to see the different controller capabilities, thereby answering Q3.2.

With regards to performance, results showed that drDQN achieved good tracking performance in regions where it was not required to exceed safety limits, and it was able to minimise tracking penalties in regions where it needed to prioritise those limits. Minor pitch angle tracking deviations were seen throughout the two evaluation tasks, with the aircraft's pitch angle momentarily diverging from the reference signal but quickly recovering afterwards. This answers Q3.3. It was hypothesised that these deviations could be reduced by: 1) Using a more gradual (rather than sparse) reward function for the passenger comfort and altitude reward types; and 2) Using a DRL method with higher learning capabilities (e.g. by including some of the DQN improvements mentioned in Hessel et al., 2018).

Furthermore, a small analysis of 25 runs showed that drDQN's performance is very similar to the one obtained with DQN, however, it takes considerably longer to train. Moreover this additional training time increases with the number of reward types used. Although useful, this analysis does not lead to robust insights about drDQN's performance given that more runs would be necessary to arrive at stronger conclusions.

Questions Q3.4 and Q3.5 are answered below where the explainability increase supplied by drDQN is analysed. Results from the Scientific Article showed that, while targeting local explainability, RDX plots lead to simpler and more intuitive insights than Q-value bars because they directly describe the decomposed Q-vector difference. By using RDX it was possible to obtain satisfactory explanations for the agent's actions, including for the moments where the agent presented counterintuitive behaviour (e.g. the pitch angle tracking deviations previously mentioned). Still, in regions far away from the altitude safety limit the agent many times presented counterintuitive Q-value estimates for the altitude reward. Even though these estimates did not have a relevant influence on the agent's decision (small Q-value relative to the pitch angle tracking one), they can be confusing for the user. It is hypothesised that such estimates can be fixed by using a more gradual reward function for the altitude reward.

Furthermore, RDX can only explain the agent's decisions locally, that is, for a particular state. To address this, a new explanation type called DRX was proposed, which provides insights on how each reward type influences the agent's behaviour over a task segment. In the two tasks analysed, DRX successfully provided explanations that led to global insights in a straightforward and intuitive way.

It should be noted that the tracking deviations previously mentioned for drDQN were also observed while using DQN. However, given that DQN does not generate explanations, it was unclear why such estimation errors occurred and how this behaviour could be improved.

By using both RDX and DRX the controller was able to supply end-user suitable explanations that justified the agent's decisions and turned the originally opaque DQN's decision process into a more interpretable one. This answers Q3.4. Moreover, Q3.5 is also answered since the explanations maintained their relevance throughout both tasks.

Considering that Q3.1, Q3.2, Q3.3, Q3.4 and Q3.5 were answered, and that the Scientific article showed how to use reward decomposition explanations in an attitude flight control task to improve end-user explainability, research question **Q3** is fully responded.

With the three research questions answered, the research objective (recalled below) can be assessed.

> *Contribute to the development of XRL methods that increase end-user acceptance for RL algorithms in flight control, by means of identifying suitable XRL techniques and by successfully implementing one of them in a business jet flight control system.*

From the Literature Review that compared the current state-of-the-art XRL techniques, this research identified reward decomposition explanations as the XRL solution with the highest potential to increase end-user explainability. Then, a reward decomposition based DRL controller was implemented in a business jet flight control system, where it presented a good tracking performance and produced satisfactory explanations. As such, the research objective was accomplished.

Nevertheless, the system controlled in this project is a longitudinal linear model of the Cessna Citation 500 aircraft which can only provide accurate data in regions near the trim condition and, consequently, the controller only presents relevant performance near the trim state. As such, in its current condition, this controller has limited applicability. Still, it is an initial effort to increase end-user explainability in DRL and it is expected to contribute to the development of new model-free XRL flight control systems.

The next section includes recommendations that one can use to expand on this project's findings.

8

Future Research Recommendations

As previously mentioned, to the author's best knowledge, this work is the first application of reward decomposition explanations to the flight control domain. However, in its current state, the proposed controller has limited applicability since it only supplies relevant performance around the trim condition. In order to expand on this project's findings, several future research recommendations are given below.

- It would be interesting to investigate if using a more gradual (rather than sparse) reward function for the passenger comfort and altitude reward types leads to more intuitive explanations in the regions where the algorithm presented counterintuitive Q-value estimates.
- Even though this project performed a small analysis that compared the decomposed algorithm performance with the one obtained by the algorithm that does not use reward decomposition, a more thorough investigation is needed to arrive at stronger conclusions. Furthermore, this analysis should use different environments and baseline DRL methods to clarify if there is a change in performance when moving from the vanilla algorithm to the decomposed one.
- Since this research's goal is to increase end-user explainability, the explanations produced should be assessed by using human-in-the-loop experiments involving end-users (e.g. pilots). Although such was out of scope for this research, performing these experiments not only validate the explainability increase supplied by this technique, but can also lead to insights on how to improve the quality of the explanations. As stated during the Literature Review, these experiments should focus on assessing the mental model produced by end-users after seeing the explanations by using: 1) Post-task questions about the agent's behaviour; and 2) Questions on predicting the agent's future actions.
- It would be interesting to apply this controller in the full nonlinear aircraft model to investigate if reward decomposition explanations maintain their relevance while analysing more complex manoeuvres in the full nonlinear system. However, to do so, a baseline DRL agent with higher learning capabilities should be used given that the controller built in this research is not expected to be able to solve the full nonlinear aircraft model.
- The main limitation associated with the reward decomposition approach used in this research is that (currently) it only works with value-based methods. Therefore, it would be of great value to develop a version of this solution that can be used with actor-critic methods, given that most flight control DRL techniques available use an actor-critic approach.

Bibliography

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., & Zhang, L. (2019). Solving Rubik's Cube with a Robot Hand. *arXiv preprint*. arXiv: 1910.07113.

Anderson, A., Dodge, J., Sadarangani, A., Juozapaitis, Z., Newman, E., Irvine, J., Chattopadhyay, S., Olson, M. L., Fern, A., & Burnett, M. M. (2020). Mental Models of Mere Mortals with Explanations of Reinforcement Learning. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *10*, 1–37.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., & Zaremba, W. (2017). Hindsight Experience Replay. *Advances in Neural Information Processing Systems*, *30*, 5048–5058.

Arrieta, A. B., Díaz-Rodríguez, N., Ser, J. D., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115. https://doi.org/10.1016/j.inffus.2019.12.012

Balas, G. J. (2003). Flight Control Law Design: An Industry Perspective. *European Journal of Control*, *9*(2), 207–226. https://doi.org/10.3166/ejc.9.207-226

Bøhn, E., Coates, E. M. L., Moe, S., & Johansen, T. (2019). Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy optimization. *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, *7*, 523–533. https://doi.org/10.1109/ICUAS.2019.8798254

Borst, C. (2004). *CITAST: Citation Analysis and Simulation Toolkit* (tech. rep.). Delft University of Technology. Delft.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint*. arXiv: 1606.01540.

Cano Lopes, G., Ferreira, M., da Silva Simões, A., & Luna Colombini, E. (2018). Intelligent Control of a Quadrotor with Proximal Policy Optimization Reinforcement Learning. *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, 503–508. https://doi.org/10.1109/LARS/SBR/WRE.2018.00094

Che, Z., Purushotham, S., Khemani, R. G., & Liu, Y. (2016). Interpretable Deep Models for ICU Outcome Prediction. *Annual Symposium proceedings. AMIA Symposium*, *2016*, 371–380.

Cideron, G., Seurin, M., Strub, F., & Pietquin, O. (2019). Self-Educated Language Agent With Hindsight Experience Replay For Instruction Following. *arXiv preprint*. arXiv: 1910.09451.

Cireşan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column Deep Neural Networks for Image Classification. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.1109/CVPR.2012.6248110

Dally, K., & Kampen, E.-J. V. (2021). Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control. *AIAA SCITECH 2022 Forum*. https://doi.org/10.2514/6.2022-2078

Doran, D., Schulz, S., & Besold, T. (2017). What Does Explainable AI Really Mean? A New Conceptualization of Perspectives. *arXiv preprint*. arXiv: 1710.00794.

Doshi-Velez, F., & Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint*. arXiv: 1702.08608.

EASA. (2019). *Automation and Flight Path Management* [European Union Aviation Safety Agency]. https://www.easa.europa.eu/automation-and-flight-path-management (accessed: 21.12.2021)

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., & Madry, A. (2020). Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. *arXiv preprint*. arXiv: 2005.12729.

Esfe, M. H., Eftekhari, S. A., Hekmatifar, M., & Toghraie, D. (2021). A well-trained artificial neural network for predicting the rheological behavior of MWCNT–Al2O3 (30–70%) oil SAE40 hybrid nanofluid. *Nature*. https://doi.org/10.1038/s41598-021-96808-4

Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. *Proceedings of the 35th International Conference on Machine Learning*, *80*, 1587–1596. http://proceedings.mlr.press/v80/fujimoto18a/fujimoto18a.pdf

Giang, H., Hoan, T., Thanh, P., & Koo, I. (2020). Hybrid NOMA/OMA-Based Dynamic Power Allocation Scheme Using Deep Reinforcement Learning in 5G Networks. *Applied Sciences*, *10*, 4236. https://doi.org/10.3390/app10124236

Girshick, R. (2015). Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1440–1448. https://doi.org/10.1109/ICCV.2015.169

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Goodman, B., & Flaxman, S. (2017). European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Magazine*, *38*, 50–57.

Greydanus, S., Koul, A., Dodge, J., & Fern, A. (2018). Visualizing and Understanding Atari Agents. *Proceedings of the 35th International Conference on Machine Learning*, *80*, 1792–1801. https://proceedings.mlr.press/v80/greydanus18a.html

Grondman, F., Looye, G., Kuchar, R. O., Chu, Q. P., & Kampen, E.-J. V. (2018). Design and Flight Testing of Incremental Nonlinear Dynamic Inversion-based Control Laws for a Passenger Aircraft. *2018 AIAA Guidance, Navigation, and Control Conference*. https://doi.org/10.2514/6.2018-0385

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *Proceedings of the 35th International Conference on Machine Learning*, *80*, 1861–1870.

Halpern, J. Y., & Pearl, J. (2005). Causes and Explanations: A Structural-Model Approach. Part I: Causes. *The British Journal for the Philosophy of Science*, *56*(4), 843–887. https://doi.org/10.1093/bjps/axi147

He, L., Aouf, N., & Song, B. (2021). Explainable Deep Reinforcement Learning for UAV autonomous path planning. *Aerospace Science and Technology*, *118*, 107052. https://doi.org/10.1016/j.ast.2021.107052

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2017). Deep Reinforcement Learning that Matters. *arXiv preprint*. arXiv: 1709.06560.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. *32*(1). https://doi.org/10.1609/aaai.v32i1.11796

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., Dulac-Arnold, G., Agapiou, J., Leibo, J., & Gruslys, A. (2018). Deep Q-learning From Demonstrations. *Proceedings of 32nd AAAI Conference on Artificial Intelligence*, *32*(1). https://doi.org/10.1609/aaai.v32i1.11757

Heuillet, A., Couthouis, F., & Díaz-Rodríguez, N. (2021). Explainability in deep reinforcement learning. *Knowledge-Based Systems*, *214*, 106685. https://doi.org/10.1016/j.knosys.2020.106685

Hoffman, R. R., Mueller, S. T., Klein, G., & Litman, J. (2018). Metrics for Explainable AI: Challenges and Prospects. *arXiv preprint*. arXiv: 1812.04608.

IATA. (2019). *Safety Report 2019*. IATA (International Air Transportation Association).

Juozapaitis, Z., Koul, A., Fern, A., Erwig, M., & Doshi-Velez, F. (2019). Explainable Reinforcement Learning via Reward Decomposition. *Proceedings at the International Joint Conference on Artificial Intelligence*. A Workshop on Explainable Artificial Inteligence.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237–285. https://doi.org/10.1613/jair.301

Keijzer, T., Looye, G., Chu, Q. P., & Kampen, E.-J. V. (2019). Design and Flight Testing of Incremental Backstepping based Control Laws with Angular Accelerometer Feedback. *AIAA Scitech 2019 Forum*. https://doi.org/10.2514/6.2019-0129

Kingma, D., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations, , ICLR 2015*.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *Proceedings of the 4th International Conference on Learning Representations, ICLR 2016*.

Lipton, Z. (2016). The Mythos of Model Interpretability. *arXiv preprint*. arXiv: 1606.03490.

Liu, G., & Schulte, O. (2018). *Github repository: uTree_mimic_mountain_car*. https://github.com/Guiliang/uTree_mimic_mountain_car (accessed: 24.11.2021)

Liu, G., Schulte, O., Zhu, W., & Li, Q. (2018). Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees. *2018 European Conference on Machine Learning and Practice of Knowledge Discovery in Databases*.

LuckyStep. (2020). *Airplane flying HUD screen analysis. Low poly 3D vector illustration*. Digital Image. Shutterstock. https://www.shutterstock.com/pt/image-vector/airplane-flying-hud-screen-analysis-flight-1545254336 (accessed: 21.12.2021)

Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 4768–4777.

Madumal, P., Miller, T., Sonenberg, L., & Vetere, F. (2020). Explainable Reinforcement Learning through a Causal Lens. *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, *34*(03), 2493–2500. https://doi.org/10.1609/aaai.v34i03.5631

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv preprint*. arXiv: 1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533. https://doi.org/10.1038/nature14236

Nguyen, V., Schulze, S., & Osborne, M. (2020). Bayesian Optimization for Iterative Learning. *Advances in Neural Information Processing Systems*, *33*, 9361–9371.

OpenAI. (2018a). *Proximal Policy Optimisation*. https://spinningup.openai.com/en/latest/algorithms/ppo.html (accessed: 06.01.2022)

OpenAI. (2018b). *Soft Actor-Critic*. https://spinningup.openai.com/en/latest/algorithms/sac.html (accessed: 08.01.2022)

Pizarroso, G., & Kampen, E.-J. (2022). *Explainable Artificial Intelligence Techniques for the Analysis of Reinforcement Learning in Non-Linear Flight Regimes* (Master's thesis). Delft University of Technology. http://resolver.tudelft.nl/uuid:d278202d-fe0b-4679-8b74-63d3c2f57495

Puiutta, E., & Veith, E. (2020). Explainable Reinforcement Learning: A Survey. *Machine Learning and Knowledge Extraction*, *12279*, 77–95. https://doi.org/10.1007/978-3-030-57321-8_5

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*, 206–215. https://doi.org/10.1038/s42256-019-0048-x

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized Experience Replay. *4th International Conference on Learning Representations, ICLR 2016*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust Region Policy Optimization. *Proceedings of the 32nd International Conference on Machine Learning*, *37*, 1889–1897. https://proceedings.mlr.press/v37/schulman15.html

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint*. arXiv: 1707.06347.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *2017 IEEE International Conference on Computer Vision (ICCV)*, 618–626. https://doi.org/10.1109/ICCV.2017.74

Shrikumar, A., Greenside, P., & Kundaje, A. (2017). Learning Important Features Through Propagating Activation Differences. *Proceedings of the 34th International Conference on Machine Learning*, *70*, 3145–3153. https://proceedings.mlr.press/v70/shrikumar17a.html

Silver, D. (2015). Lecture 2: Markov Decision Processes [PowerPoint Presentation]. url: https://www.davidsilver.uk/teaching/.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–489. https://doi.org/10.1038/nature16961

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. *Proceedings of the 31st International Conference on Machine Learning*, *32*(1), 387–395. https://proceedings.mlr.press/v32/silver14.html

Stevens, B., Lewis, F., & Johnson, E. (2015). *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. Wiley.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tang, C., & Lai, Y.-C. (2020). Deep Reinforcement Learning Automatic Landing Control of Fixed-Wing Aircraft Using Deep Deterministic Policy Gradient. *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 1–9. https://doi.org/10.1109/ICUAS48674.2020.9213987

Tsourdos, A., Dharma Permana, I. A., Budiarti, D. H., Shin, H.-S., & Lee, C.-H. (2019). Developing Flight Control Policy Using Deep Deterministic Policy Gradient. *2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, 1–7. https://doi.org/10.1109/ICARES.2019.8914343

Uther, W. T. B., & Veloso, M. M. (1998). Tree Based Discretization for Continuous State Space Reinforcement Learning. *Proceedings of the 15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, 769–774. https://www.aaai.org/Papers/AAAI/1998/AAAI98-109.pdf

van der Linden, C. (1998). *DASMAT-Delft University Aircraft Simulation Model and Analysis Tool: A Matlab/Simulink Environment for Flight Dynamics and Control Analysis*. Delft University Press. http://resolver.tudelft.nl/uuid:25767235-c751-437e-8f57-0433be609cc1

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *30*(1), 2094–2100. https://doi.org/10.1609/aaai.v30i1.10295

van Hoek, M., de Visser, C. C., & Pool, D. M. (2017). Identification of a Cessna Citation II Model Based on Flight Test Data. *4th CEAS Specialist Conference on Guidance, Navigation and Control*. http://resolver.tudelft.nl/uuid:c0a57513-38b7-4d3a-898c-fa57c3e7ac2e

van Ingen, J., de Visser, C. C., & Pool, D. M. (2021). Stall Model Identification of a Cessna Citation II from Flight Test Data Using Orthogonal Model Structure Selection. *AIAA Scitech 2021 Forum*, 1725. https://doi.org/10.2514/6.2021-1725

van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., & Tsang, J. (2017). Hybrid Reward Architecture for Reinforcement Learning. *Advances in Neural Information Processing Systems*, *30*, 5392–5402. https://proceedings.neurips.cc/paper/2017/file/1264a061d82a2edae1574b07249800d6-Paper.pdf

van Zijl, J., Nunes, T., & Kampen, E.-J. (2022). *Using Explainable Artificial Intelligence to Improve Transparency of Reinforcement Learning for Online Adaptive Flight Control* (Master's thesis). Delft University of Technology. http://resolver.tudelft.nl/uuid:66a5fdba-6508-4b6f-b20f-c1766ec4fc7f

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., … Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*, 1–5. https://doi.org/10.1038/s41586-019-1724-z

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *Proceedings of The 33rd International Conference on Machine Learning*, *48*, 1995–2003. https://proceedings.mlr.press/v48/wangf16.html

Weng, L. (2018). *Policy Gradient Algorithms*. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html (accessed: 05.01.2022)

Xie, Y., Pongsakornsathien, N., Gardi, A., & Sabatini, R. (2021). Explanation of Machine-Learning Solutions in Air-Traffic Management. *Aerospace*, *8*(8), 224. https://www.mdpi.com/2226-4310/8/8/224

Xu, Z., Tang, J., Meng, J., Zhang, W., Wang, Y., Liu, C. H., & Yang, D. (2018). Experience-Driven Networking: A Deep Reinforcement Learning Based Approach. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 1871–1879. https://doi.org/10.1109/INFOCOM.2018.8485853