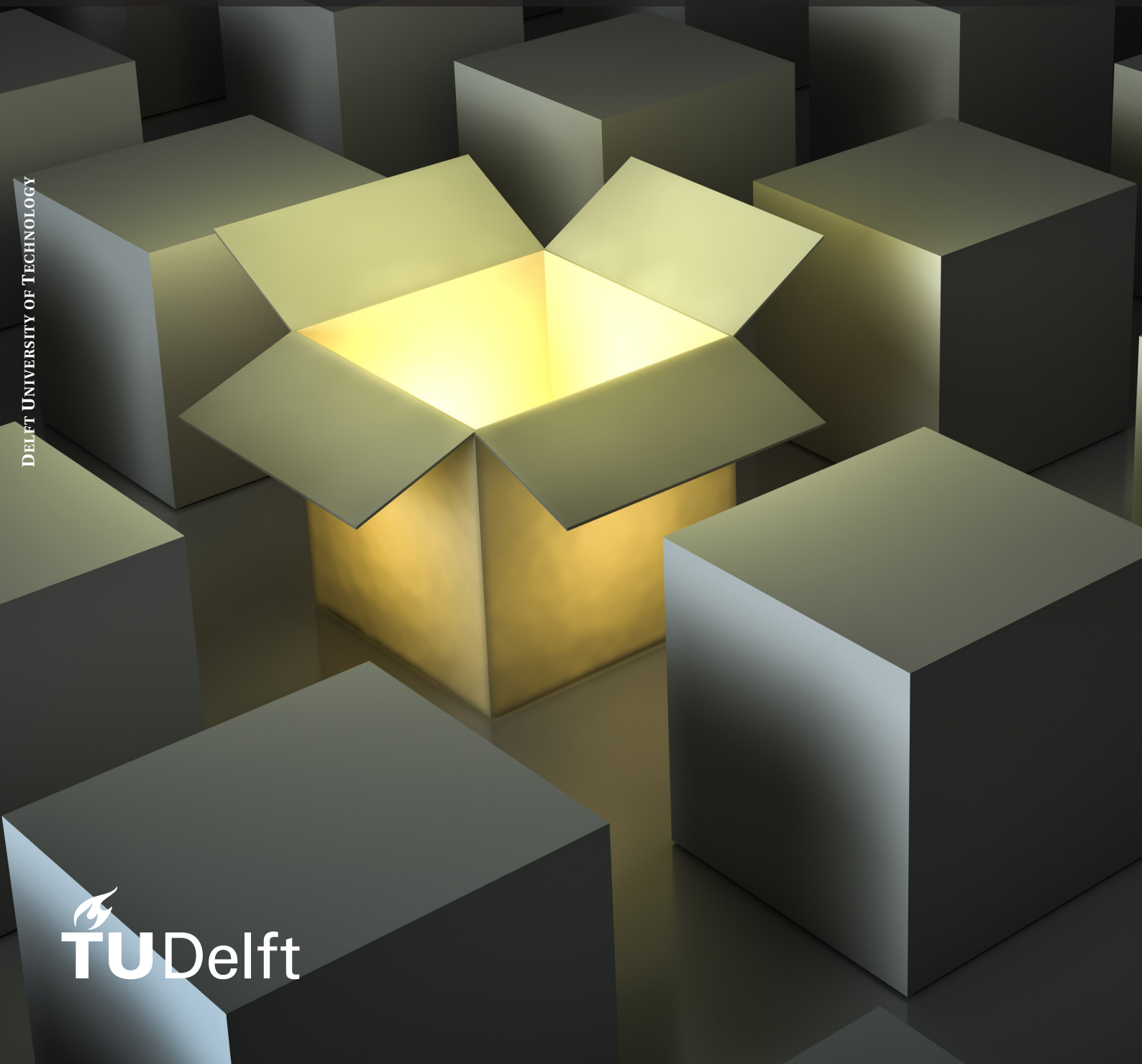
Using Explainable Artificial Intelligence to Increase Transparency of Reinforcement Learning for Flight Control

Breaking Open the Black Box

MSc Thesis Report

J.A.J. van Zijl



Delft University of Technology

TUDelft

Using Explainable Artificial Intelligence to Increase Transparency of Reinforcement Learning for Flight Control

Breaking Open the Black Box

MSc Thesis Report

by

J.A.J. van Zijl

to obtain the degree of
Master of Science in Aerospace Engineering
at Delft University of Technology

Student number: 4380185
Date: January 28, 2022

Thesis committee:

Dr. ir. M. M. van Paassen	Delft University of Technology	Committee chair
Dr. ir. E. van Kampen	Delft University of Technology	Supervisor
T. Nunes, MSc	Delft University of Technology	Supervisor
Dr. ir. E. Mooij	Delft University of Technology	External examiner

Preface

This master thesis report concludes my amazing time as an aerospace engineering student in Delft, at the Control & Simulation department. The subject of explainable artificial intelligence presented in this report sparked my interest the first time Erik-Jan mentioned it as a potential graduation subject, and this interest has only grown during the last year. I find it fascinating that we as humans cannot only use AI for automation, but can now also learn from it. My hope is that you will also learn about, and from AI while reading this thesis report.

There are multiple people I would like to thank for their support and their direct and indirect contribution to this thesis report. First of all, I would like to express my deepest gratitude towards Erik-Jan and Tiago for the great support and guidance during the last year. Every week I was looking forward to our meeting, including interesting and sometimes philosophical discussions about (explainable) artificial intelligence. The positive energy and attention you were able to give on a weekly basis enabled me to keep learning and improving, almost like a reinforcement learning agent. Your detailed feedback helped me tremendously. Furthermore I would like to express my appreciation towards my friends, for studying together and therefore making the graduation process under the sometimes tight pandemic restrictions more bearable. Our discussions sparked many ideas, and kept me sharp during the often long days of studying. Last, but definitely not least, I would like to thank my parents and sisters for always supporting me, not only during the past year, but also during the rest of my time in Delft.

Job van Zijl
Den Haag, January 2022

Table of Contents

List of Figures

List of Tables

List of Acronyms

A2C Advantage Actor-Critic.

ACD Adaptive Critic Designs.

ADDHP Action-Dependent Dual Heuristic Programming.

ADGDHP Action-Dependent Globalized Dual Heuristic Programming.

ADHDP Action-Dependent Heuristic Dynamic Programming.

ADP Approximate Dynamic Programming.

AFCS Automatic Flight Control System.

AI Artificial Intelligence.

ARL Approximate Reinforcement Learning.

CWSD Constant Weight Segment Detection.

DDPG Deep Deterministic Policy Gradient.

DHP Dual Heuristic Programming.

DL Deep Learning.

DP Dynamic Programming.

DQN Deep Q Network.

DRL Deep Reinforcement Learning.

GDHP Globalized Dual Heuristic Programming.

GPI Generalized Policy Iteration.

HDP Heuristic Dynamic Programming.

IDHP Incremental Dual Heuristic Programming.

LIME Local Interpretable Model-agnostic Explanations.

MC Monte Carlo.

MDP Markov Decision Process.

ML Machine Learning.

MSE Mean Squared Error.

MSX Minimal Sufficient eXplanations.

NN Artificial Neural Network.

PID Proportional-Integral-Derivative.

PPO Proximal Policy Optimization.

RDX Reward Difference eXplanations.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

SHAP Shapley Additive Explanations.

SL Supervised Learning.

TD Temporal Difference.

UAV Unmanned Aerial Vehicle.

USL Unsupervised Learning.

XAI Explainable Artificial Intelligence.

XRL Explainable Reinforcement Learning.

1 | Introduction

1.1. Motivation

Artificial Intelligence (AI), defined as the ability of an automation agent to independently interpret and learn from external data to achieve specific outcomes by flexible adaptation [1], is currently widespread in society, as it is employed by numerous companies, governmental organizations, and other institutions [2]. Applications of AI include image recognition [3], control of autonomous vehicles [4] and autonomous planning [5]. The variety of applications and their complexity will increase extensively in the coming years [2]. This acceleration of AI applications and its widespread adoption can be attributed to the tremendous increase in capabilities of AI during the last 20 years, due to advancements in computer hardware, improved optimization algorithms, the emergence of open-source AI libraries and the rise of *Big Data* [6] [7]. *Machine Learning* (ML) is the subset of AI, where machines learn from external data or experience, without being explicitly programmed [8]. ML can be divided into the following three forms: *supervised learning* (SL), *unsupervised learning* (USL), and *reinforcement learning* (RL). A subset of ML, *Deep Learning* (DL), is recently gaining traction within the academic world [2]. DL encapsulates all ML techniques utilizing one or more artificial Neural Networks (NNs). These function approximators are currently used extensively in the academic and professional world due to their high approximation power and flexibility [2]. The terminologies of AI, ML, and DL are summarized in Figure 1.1.

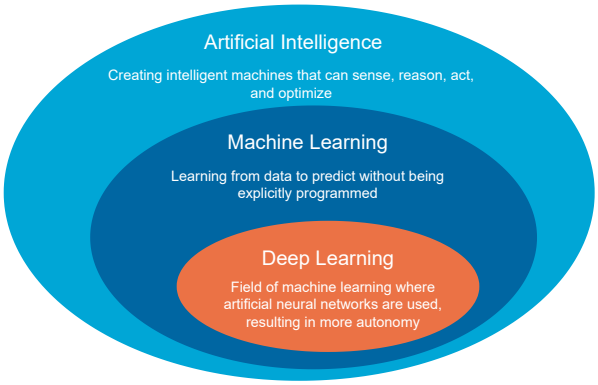


Figure 1.1: Terminology of artificial intelligence, adapted from [8].

In SL, the ML algorithm learns from a training set with labeled data, aiming to extend this learned functionality to data sets the algorithm has not seen before. Common applications in SL include classification, such as image classification, and regression analysis like market forecasting. USL on the other hand learns from unlabeled data, reducing the need for the time-consuming labeling process. USL algorithms analyze and cluster these unlabeled data sets, aiming to discover patterns in large amounts of data. Common applications of USL include clustering tasks, such as finding similarities between customers, and dimensionality reduction as pre-processing of large data sets. Finally, RL differentiates itself from SL and USL as the ML algorithm makes a series of decisions, learned from experience through trial and error [9]. In RL, an agent picks an action at every time step, based on the current state of the agent's environment, to maximize its reward over time. This interaction process is illustrated in Figure 1.2. By interacting with its environment and observing the reward signal, the agent learns which actions are encouraged or penalized. As RL addresses problems where a series of decisions should be made, RL applications include real-time decision-making processes, robotic navigation and control of autonomous vehicles. *Deep Reinforcement Learning* (DRL), which combines RL with NNs, has recently achieved super-human performance in video games [10] and outperformed the world's best *Go* player [11]. Where initial RL algorithms were only capable of simple navigation tasks due to their limited complexity, the field of DRL has resulted in successful completion of complex tasks and DRL gaining more traction in the academic world [12].

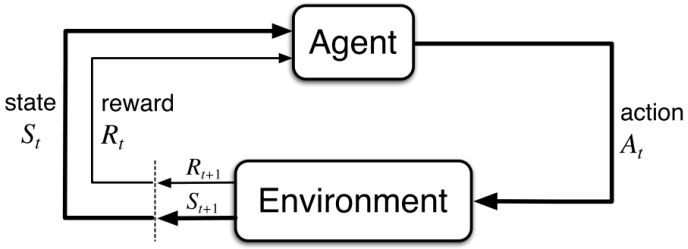


Figure 1.2: Schematic representation of the reinforcement learning process [9].

Due to the increasing performance of DRL, flight control using DRL is an increasingly active research field [13]. Current automatic flight control systems often use classic linear control techniques such as PID control. As the encountered dynamics change during flight, for example due to changes in altitude, gain scheduling methods are used to change between predetermined operating conditions. Not only is engineering this gain scheduling a time-consuming process, it also results in sub optimal control as the nonlinear flight dynamics are approximated using linear control. Furthermore, this traditional control is only suitable around the predetermined operating points, allowing little robustness to unknown flight dynamics and fault-tolerance. The complexity of deep NNs, consisting of multiple *hidden layers* of *neurons*, and their nonlinear *activation functions*, allows interaction with the complex and nonlinear dynamics encountered by aerospace vehicles. Examples include the inverted flight of a helicopter [14] and attitude control of an *Unmanned Aerial Vehicle* (UAV) [15]. Not only do DRL techniques show superior tracking performance with respect to traditional flight control methods like *PID* control [16], the flexibility of NNs allows adaptability to different flight circumstances and fault-tolerance [17].

The increasing complexity of ML techniques comes with a major drawback. Where the initial ML models were easy to *interpret* due to their limited complexity, state-of-the-art ML models are considerably more *opaque* due to their many parameters [18]. Current ML models are often referred to as *black box* models, due to their lack of *transparency* [19]. As current models are already considered opaque, and ML models are becoming more and more complex, future ML models are likely to become more opaque [18]. *Interpretability* is defined in this thesis as the ability to explain something in terms understandable to a human, and *transparent* is defined as being inherently understandable. *Explainability* on the other hand is defined as the ability to make a decision-making process understandable to humans, while still providing an accurate notion of this decision-making process.

This thesis aims to increase the transparency of RL algorithms used for flight control. For aerospace applications, the lack of transparency is a major drawback when comparing DRL with classical techniques like PID control. Before DRL algorithms are allowed for use in aerospace vehicle control, it is essential for control engineers, other experts and regulatory agencies that the working principles can be explained. Therefore, the need arises for *eXplainable Reinforcement Learning* (XRL) for flight control. Flight control engineers or other experts working with RL for flight control will be able to enhance the performance of the flight control since XRL provides more insight into the decision-making process of the RL agent. Finally, another promising advantage of explainable RL for flight control is the reduced development time. DRL models are often hard to debug due to the complex and nonlinear models, dependence on the agent's environment and the design of the reward function. Having an explainable model could help debugging whenever the algorithm does not work as intended, potentially reducing the development time by a significant amount.

1.2. CHALLENGES

Generally, the model interpretability of a machine learning model reduces as the model complexity is increased [18], as displayed in Figure 1.3. Note that in this figure, complex models are regarded as more accurate. However, simple models can be perfectly accurate for simple tasks. Complex models like deep learning on the other hand, are not guaranteed to be always accurate. The complex and nonlinear nature of aerospace vehicle dynamics requires the model to be, in itself, complex which in RL implies the use of DRL models. Explaining these extensive and nonlinear models, often with many continuous inputs, in a clear way is a big challenge. Therefore, *breaking open the black box* of DRL is a complex task in itself. Secondly, what is defined as a clear explanation is open to interpretation and depends on the addressed audience. Currently, there is

no defined metric how the explainability for ML models can be assessed. Thirdly, another big challenge in XRL for flight control is that every flight control application requires its own explanation. As an example, the control strategy employed by a small UAV using monocular vision for obstacle avoidance should be explained differently than the control strategy learned for fault-tolerant control of an airliner. No standard method exists, and while the working principles can be described in general to explain RL for flight control, this will reduce the accuracy of the explanation and cause fine details specific to a certain application to be missed. Finally, DRL for flight control, and especially explainability, is a very young research field, with limited previous research. At time of writing, research into *eXplainable Artificial Intelligence* (XAI) is accelerating [18], however only one publication could be found focusing on explaining RL for flight control [20].

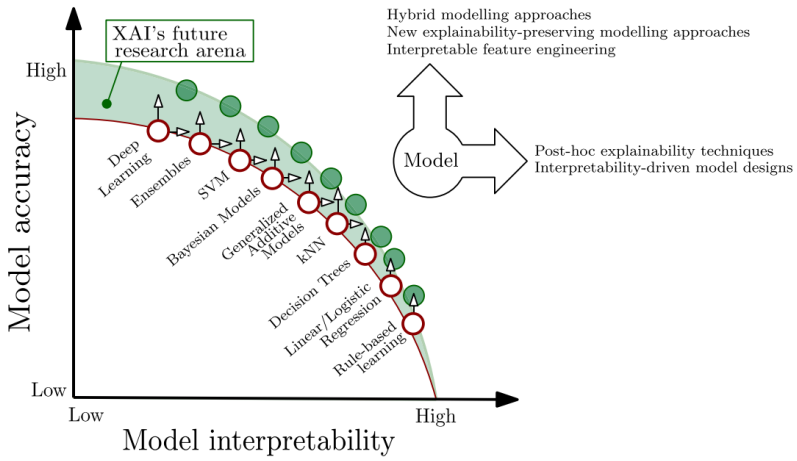


Figure 1.3: Trade-off between model interpretability and performance, and a representation of the area of improvement where the potential of XAI (eXplainable Artificial Intelligence) techniques and tools resides [18].

1.3. RESEARCH GOAL AND RESEARCH QUESTIONS

The research goal of this thesis is defined as follows:

Research goal

Improve the transparency of reinforcement learning algorithms for flight control, by investigating existing explainable artificial intelligence techniques, and applying the most promising techniques to explain flight control using a state-of-the-art reinforcement learning method.

This thesis aims to contribute to this research goal by answering the following research questions:

Research questions

1. **How can reinforcement learning (RL) methods be used for autonomous flight control?**
   1.1. What is the state-of-the art in RL for flight control?
   1.2. Which aspects of RL for flight control are currently the least transparent?
   1.3. Which working principles of RL for flight control are most helpful to explain for flight control engineers or other experts working with the algorithms?
2. **How can eXplainable Artificial Intelligence (XAI) techniques be used to gain more insight into the working principles of reinforcement learning?**
   2.1. What is the state-of-the-art of transparent RL algorithm design?
   2.2. What is the state-of-the-art of post-hoc explainability techniques suitable for RL?
3. **How can explainable reinforcement learning techniques be used to increase the transparency of RL for adaptive flight control?**
   3.1. Which XAI techniques can be integrated with RL for adaptive flight control?
   3.2. How can the most important flight control inputs be determined using XAI and how do these influence the actuator output?
   3.3. What is the influence of changing model weights to the accuracy of explanations produced by the XAI technique?

1.4. REPORT OUTLINE

The remainder of this thesis report is structured as follows. Part I contains the main findings of the conducted research in the form of a scientific paper, presenting how SHAP can be used to explain the learned strategy of an online adaptive RL framework controlling six-dimensional nonlinear flight of a Cessna Citation I aircraft. The utilized RL framework is *Incremental Dual Heuristic Programming* (IDHP), a state-of-the-art adaptive critic design for adaptive and fault-tolerant control. First, the paper presents how SHAP can be used to explain the input-output mapping of the actor module of the RL framework, responsible for applying the learned strategy. Secondly, the paper presents how segments with near-constant weights can be distinguished, to facilitate SHAP computations and explain the control strategy over time. Finally, the paper presents how the learned control strategy of IDHP can be accurately explained linear models. Part II contains the literature review used to select SHAP out of existing XAI techniques, and the preliminary analysis used to assess the strengths and weaknesses of SHAP for explaining RL for low-level flight control [1]. In Part III, additional results are presented. First the linear explanation models obtained in Part I are validated by using these as control laws for the Cessna Citation simulation. Secondly, it is shown how SHAP can be used to illustrate adaptive properties of RL for flight control. Finally, the thesis is concluded in Part IV, presenting the answers to the formulated research questions and including recommendations for future research.

[1] Part II has been graded as part of the AE4020 Literature Study course

I

SCIENTIFIC PAPER

Using Explainable Artificial Intelligence to Improve Transparency of Reinforcement Learning for Online Adaptive Flight Control

J.A.J. van Zijl* T. M. M. Nunes†, *supervisor* E. van Kampen‡, *supervisor*
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Deep Reinforcement Learning (DRL) shows great potential for flight control, due to its adaptability, fault-tolerance, and as it does not require an accurate system model. However, these techniques, like many machine learning applications, are considered black-box as their inner workings are hidden. This paper aims to break open the black box of RL for adaptive flight control by applying Shapley Additive Explanations (SHAP). The generated explanations are aimed at control experts, but can be useful for anyone interested in RL for adaptive flight control. This research proposes a novel Constant Weight Segment Detection (CWSD) algorithm, facilitating the use of eXplainable Artificial Intelligence techniques to adaptive RL. The algorithm and its usefulness are tested on an Adaptive Critic Design controlling a high-fidelity model of a Cessna Citation aircraft. It is demonstrated that SHAP in combination with CWSD provides detailed and useful insights into the relation between input and output of the RL algorithm. Using SHAP, linear relations between input and output are discovered, simplifying the understanding of the learned strategy.

I. Introduction

Automation in vehicle control is currently gaining traction due to the increasing adoption and projected growth of Unmanned Aerial Vehicles (UAVs) [1] and the development of self-driving cars [2]. These automatic control systems require adaptability in order to operate in complex environments. Where one of the dominant challenges for self-driving cars is the highly dynamic urban environment [3], the main challenges for Automatic Flight Control Systems (AFCS) are the nonlinear and uncertain flight dynamics [4]. Typically, the AFCS of modern passenger aircraft is designed using classical control theory switching between multiple linear controllers tuned around predetermined operating conditions through gain-scheduling [5]. As the performance and stability of the AFCS designed through gain-scheduling is heavily dependent on the accuracy of the flight model, extensive verification and validation of this model is required. Not only is engineering this control technique a complex and time-consuming process, gain-scheduling also does not guarantee stability for coupled dynamics [6] and allows only limited autonomous control under adverse, hazardous flight conditions [7] and allows little *fault-tolerance*.

Reinforcement Learning (RL), one of the three domains of Machine Learning (ML), allows for adaptive control without an accurate system model [8]. This ML framework bridges the gap between traditional optimal control, adaptive control, and bio-inspired learning techniques [9]. The field of Deep Reinforcement Learning (DRL), which combines RL with artificial Neural Networks (NNs), is currently gaining traction in the academic world [10], has achieved super-human performance in video games [11] and outperformed the world's best *Go* player [12]. While RL shows potential for online adaptive flight control and numerous other applications, the increasing complexity of ML techniques comes with a major drawback. Where initial ML models were easy to interpret due to their limited complexity, state-of-the-art ML models are considerably more *opaque* due to their many parameters [13]. The attitude policy NN in [14] for example consists of 9 input neurons, 2 layers with 64 hidden neurons, and 6 output neurons, resulting in 5056 weights and 134 biases. From this type of NN it is not easy to interpret the control law that is learned by the agent. The current trend of growing complexity in AI will further limit its interpretability [13]. Many modern ML models are often referred to as *black box* models, as one can observe the input and output of the model, but the inner workings are hidden [15]. This lack of transparency limits the *trustworthiness* of the chosen actions, which can be a serious obstacle for real-life applications involving RL [16]. The research field of eXplainable Artificial Intelligence

*Graduate Student, Faculty of Aerospace Engineering, Control and Simulation Department, Delft University of Technology
†PhD Student, Faculty of Aerospace Engineering, Control and Simulation Department, Delft University of Technology
‡Assistant Professor, Faculty of Aerospace Engineering, Control and Simulation Department, Delft University of Technology

(XAI) aims to improve the interpretability of AI. This paper aims to break open the black box of RL algorithms for flight control, by increasing its transparency using existing XAI techniques.

The target audience is a key aspect regarding explainability of ML models, as it determines the specific level of detail and content of the explanation [13]. The content and form of the explanation are therefore dependent on the audience, and the quality of the explanation is assessed by this audience. This research targets aerospace control experts as well as RL experts and developers, to facilitate development of RL for flight control. However, other audiences such as control engineers unfamiliar with RL can also learn from the insights into the inner workings of RL for flight control. Aerospace control and RL experts desire insights into the working principles of their algorithms for multiple reasons. First of all, state-of-the-art RL algorithms are often hard to debug due to their complexity. Improved transparency for RL can ease detecting and solving possible errors of the algorithms. Furthermore, better understanding of the algorithms allows assessment of the robustness to unknown circumstances. Therefore, the RL expert may gain confidence in the algorithm if it shows logical reasoning. Finally, RL experts may be able to learn from new insights provided by the algorithm, made possible using explainable RL. These goals require technical and in-depth explanations, which will be provided in this paper.

RL is used for flight control in different hierarchical levels of autonomy. High-level control includes autonomous decision making, path planning, trajectory generation, and obstacle avoidance, as these tasks are ultimately responsible for creating the trajectory. Low-level control comprises trajectory following using sensors and actuators. Within these levels of automation, RL is used for at least 4 distinct applications. High-level RL control applications include goal-based navigation [17–20] and autonomous obstacle avoidance [21, 22] , while low-level RL control applications include adaptive flight control [23–26], and flight controller tuning [27, 28]. [29] shows how XAI can be used for UAV goal-based navigation. From the four categories, adaptive control is the most active research field, due to the large potential of NNs for adaptive control [30]. However, research into explainability of RL for adaptive flight control is currently scarce.

The main contribution of this paper is to improve the transparency of RL techniques for adaptive flight control using Shapley Additive Explanations (SHAP), a feature relevance XAI technique. A second contribution of this paper is to facilitate the use of XAI techniques for adaptive control with continually changing NN parameters. In this work, a novel algorithm, Constant Weight Segment Detection (CWSD), is proposed to enable SHAP computations in the continually learning environment of online adaptive flight control. A state-of-the-art RL framework for adaptive control is applied to simulate flight of a Cessna Citation I aircraft, after which the learned control strategy is explained using SHAP. It is shown that SHAP in combination with CWSD can provide useful explanations, giving meaningful insights into the inner workings of RL for adaptive flight control. One of the insights that will be shown is the discovery that the strategy learned by the adaptive critic design for a simulated mission profile can be accurately represented using linear expressions.

This paper is structured as follows. Section II presents how RL is used for adaptive flight control, including the foundations of the Incremental Dual Heuristic Programming framework used as the example algorithm for this research. Furthermore XAI techniques are introduced, including the method chosen for this research. Section III presents the explanation methodology and facilitation of SHAP for explaining adaptive RL. Subsequently, Section IV presents the flight simulation, used as example for the explanations. In Section V the controller is tested and its control strategy is explained for an online training phase and a representative mission profile including online learning. Furthermore, the effect of segment length on the explanation accuracy is investigated. Finally, the conclusions are presented in Section VI.

II. Foundations

This section introduces how RL is used for adaptive flight control, after which the selected framework is presented. Subsequently, XAI techniques suitable for RL are presented. Finally, the selected XAI method is introduced.

A. Reinforcement Learning for Adaptive Flight Control

State-of-the-art RL frameworks such as Proximal Policy Optimization (PPO) outperform PID control for UAV attitude control on multiple metrics including rise time, and average tracking error [24]. Adaptive Critic Designs (ACDs) use NNs for approximation of the policy function (the *actor* module) and the value function (the *critic* module), and in some variants for approximation of the global system dynamics [31]. By applying temporal-difference (TD) learning, ACDs have shown accurate tracking and stable adaptive control for continuous and nonlinear problems, such as the automatic landing of an aircraft [32], for six degree-of-freedom flight of a business jet [33], and for flight control of a fighter jet [34]. While the ACD frameworks used in these researches and other RL frameworks such as PPO do not

require prior information about the controlled system, an offline learning phase is required. Recently developed ACD frameworks such as IDHP [26], remove the need for this prior learning phase due to the introduction of incremental control techniques. This framework allows online adaptive control of systems with unknown nonlinear dynamics [26], shows superior tracking when compared to PID control and other ACDs, and shows robustness to unknown system parameters and fault-tolerance during six-degrees of freedom nonlinear simulation of a Cessna Citation business jet [35]. IDHP is used in this research as the example framework to show how XAI can be applied to explain RL for flight control. While the chosen application is an adaptive RL control technique, the proposed explanation methodology is relevant for any RL application using actor-critic structures.

ACDs, PPO and many other state-of-the-art RL algorithms such as Deep Deterministic Policy Gradient all share the use of the-actor-critic structure, in which the actor represents the control policy π used to select an action $\mathbf{u}_t \in \mathcal{R}^{n\times 1}$, where n is the number of actions, based on the current state $\mathbf{x}_t \in \mathbb{R}^{m\times 1}$, where m is the number of states, and reference state $\mathbf{x}_t^R \in \mathbb{R}^{p\times 1}$, where p is the number of reference states, as illustrated by Equation (1). Following this chosen action and the state of the environment, the controlled system changes according to the potentially nonlinear discrete-time state transition, represented by the function f in Equation (2). Following this change in state, the agent receives a reward r_t. The sum of discounted future rewards from a certain state is defined as the value or cost-to-go (V), which is estimated by the critic. The definition for the cost-to-go for this paper is shown in Equation (3), where the discount factor $\gamma \in [0, 1]$ is used to diminish future rewards. The policy π is optimized during training to maximize the cost-to-go. Therefore, the reward signal should reinforce desired control behavior

$$\mathbf{u}_t = \pi(\mathbf{x}_t, \mathbf{x}_t^R) \tag{1}$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{2}$$

$$V(\mathbf{x}_t, \mathbf{x}_t^R) = \sum_{l=t}^{\infty} \gamma^{l-t} r_t \tag{3}$$

B. Incremental Dual Heuristic Programming

The IDHP framework described in the following subsection and used throughout the paper is adapted from [35]. The reward function, given by Equation (4), is defined as the negative weighted difference between $\mathbf{x}_t^R$ and $\mathbf{x}_{t+1}$, to penalize discrepancies between the reference and actual state. The boolean selection matrix $\mathbf{P} \in \mathbb{R}^{p\times m}$ selects the controlled states from $\mathbf{x}$, while the symmetric weighing matrix $\mathbf{Q} \in \mathbb{R}^{p\times p}$ controls the relative cost between the p controlled states. IDHP does not use the scalar reward itself for optimizing the approximator weights, but utilizes $\frac{\delta r_{t+1}}{\delta \mathbf{x}_{t+1}} \in \mathbb{R}^{1\times m}$: the derivative of the reward function with respect to the state vector $\mathbf{x}_{t+1}$ as given by Equation (5).

$$r_{t+1} = r(\mathbf{x}_t^R, \mathbf{x}_{t+1}) = -\left[\mathbf{P}\cdot\mathbf{x}_{t+1} - \mathbf{x}_t^R\right]^T \mathbf{Q}\left[\mathbf{P}\cdot\mathbf{x}_{t+1} - \mathbf{x}_t^R\right] \tag{4}$$

$$\frac{\delta r_{t+1}}{\delta \mathbf{x}_{t+1}} = \frac{\delta r(\mathbf{x}_t^R, \mathbf{x}_{t+1})}{\delta \mathbf{x}_{t+1}} = -2\left[\mathbf{x}_{t+1} - \mathbf{x}_t^R\right]^T \mathbf{QP} \tag{5}$$

The schematic representation of the feedforward signal flow of the used IDHP framework for a single time step is presented in Figure 1, including the three trainable parametric modules: the critic approximating the derivative of the value function with respect to the state vector $\hat{\lambda}_t(\mathbf{x}_t, \mathbf{x}_t^R, \mathbf{w}_c) = \frac{\delta \hat{V}(\mathbf{x}_t)}{\delta \mathbf{x}_t}$, the actor approximating the optimal control strategy $\hat{\pi}(\mathbf{x}_t, \mathbf{x}_t^R, \mathbf{w}_a)$ and the incremental system model. The reference signal $\mathbf{x}^R$ originates from the outer loop reference, while the state vector $\mathbf{x}$ is the system output from the previous time step. The z^{-1} blocks represent one step time delays. Not only are the NNs themselves black-box elements, the complex structure of IDHP includes an incremental model and intensive routing which further limits the transparency of the framework.

1. Critic and Actor Neural Networks

The critic and actor use single hidden layer fully connected multilayer perceptron NNs, schematically illustrated in Figure 2. The critic estimates the partial derivatives of the value function with respect to the state vector, while the actor NN approximates the control policy by mapping $(\mathbf{x}, \mathbf{x}^R)$ to the actuator deflections $\mathbf{u}$. Both NNs use the same input structure and each has one hidden layer consisting of 10 neurons utilizing the hyperbolic tangent activation function

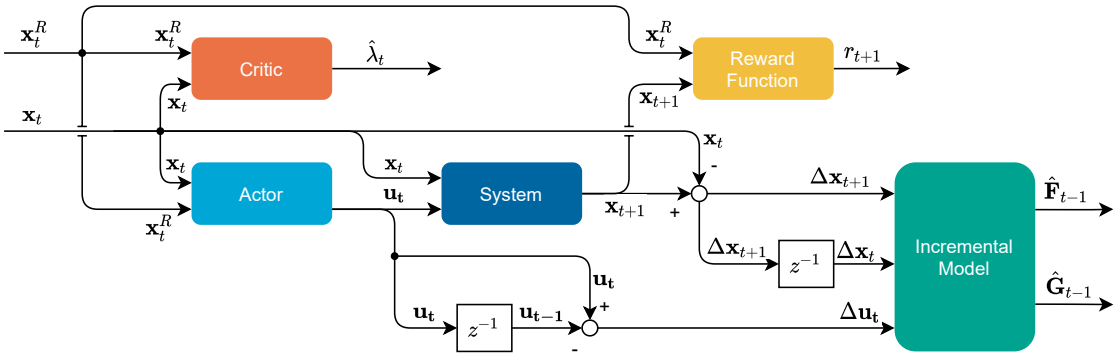
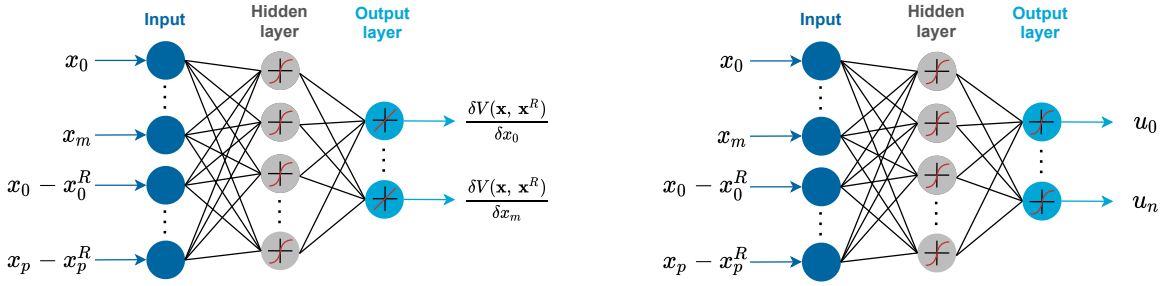


Fig. 1 Feedforward flow of the IDHP framework during a single time step, adapted from [35].

with limits $[-1, 1]$. The output layer neurons of the critic use linear activation functions such that the output of this NN is unbounded. The output layer neurons of the actor use hyperbolic tangent functions bounded on the domain complying with the physical limits of the control actuators introduced in Section IV. The NNs do not use neuron bias because the trim values of the aircraft are known to the controller, which will be further elaborated on in Section III.



(a) Critic, with linear activation for the output

(b) Actor, with hyperbolic tangent activation for the output

Fig. 2 Schematic representation of the neural network approximators. Neuron bias is not applied and both networks use hyperbolic tangent activation functions for the hidden layer.

The update rule of the critic is defined in Equation (6), where $\mathbf{e}_c(t)$ is defined as in Equation (7) [26]. The update rule of the actor is presented in Equation (8) [26].

$$\mathbf{w}_c(t+1) = \mathbf{w}_c(t) - \eta_c \cdot \mathbf{e}_c(t)^T \cdot \frac{\partial \hat{\lambda}(\mathbf{x}_t)}{\partial \mathbf{w}_c(t)} \tag{6}$$

$$\mathbf{e}_c(t) = \hat{\lambda}(\mathbf{x}_t) - \gamma \hat{\lambda}(\mathbf{x}_{t+1}) \frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} - \frac{\partial r_t}{\partial \mathbf{x}_t} \tag{7}$$

$$\mathbf{w}_a(t+1) = \mathbf{w}_a(t) - \eta_a \cdot \left[\frac{\partial r_t}{\partial \mathbf{u}_t} + \gamma \lambda(\mathbf{x}_{t+1}) \mathbf{G}_{t-1} \right] \cdot \frac{\partial \pi(\mathbf{x}_t, \mathbf{x}_t^R, \mathbf{w}_a(t))}{\partial \mathbf{w}_a(t)} \tag{8}$$

2. Incremental Model

IDHP differentiates itself from other ACD frameworks by using an incremental model for approximation of the system dynamics, exploiting the high-frequency measurements of $\mathbf{x}$ and $\mathbf{u}$. This technique can successfully manage nonlinear systems if the sample rate of the measurements is sufficiently high [36]. The incremental model is presented in Equation (9), where the state increment $\Delta\mathbf{x}_{t+1} = \mathbf{x}_{t+1} - \mathbf{x}_t$, the system matrix $\mathbf{F}_{t-1} = \mathbf{F}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, the control effectiveness matrix $\mathbf{G}_{t-1} = \mathbf{G}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, and the control increment $\Delta\mathbf{u} = \mathbf{u}_t - \mathbf{u}_{t-1}$. Assuming that the sampling

frequency is sufficiently high and the system is relatively slow-varying, Equation (9) describes a linear time-varying approximation of the nonlinear aircraft dynamics [26]. The update rules used for this model are derived in [37].

$$\Delta \mathbf{x}_{t+1} \approx \mathbf{F}_{t-1}\Delta \mathbf{x}_t + \mathbf{G}_{t-1}\Delta \mathbf{u}_t \tag{9}$$

C. Explainable Artificial Intelligence

The research field of XAI aims to solve the black box problem of state-of-the-art AI techniques, where the many model parameters and complex structures limit model interpretability. Generally speaking, there is a trade-off for AI models between interpretability and model approximation power as illustrated in Figure 3. The inner workings of models such as decision trees for example is easy to understand, but their modelling power is limited. NNs on the other hand are among the most powerful tools in terms of modelling power, but are difficult to understand due to their many parameters and nonlinearities. Interpretability in this paper is defined as the ability to explain something in terms understandable to a human [13]. Models are considered opaque or black box if their inner workings are hidden [15], while the antonym of this property is transparent. Most AI models cannot be described using these two extremes, but are best described on the spectrum between transparent and black box.

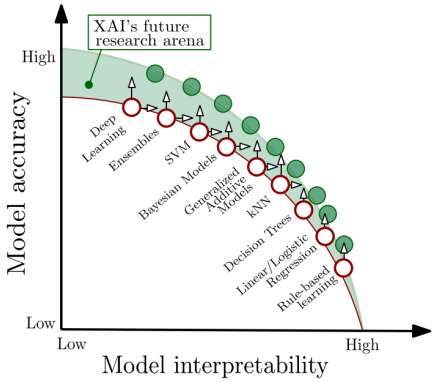


Fig. 3 Trade-off between model interpretability and performance, and a representation of the potential of explainable artificial intelligence techniques, adapted from [13].

While XAI is becoming a more intensive research field [13], research into *eXplainable Reinforcement Learning* (XRL) is relatively sparse [38]. Currently, most XRL solutions apply transparent design where interpretability is one of the design drivers. Examples include reward decomposition to generate explanations using trade-offs in the reward signal [39], and explainable navigation using fuzzy RL with linguistic terms and logical IF-THEN statements [40]. The other suite of XAI techniques, *post-hoc explainability*, show more potential for explaining RL for flight control as they do not require compromises of accuracy for interpretability and because these techniques are often *model-agnostic*, meaning that they are not designed with a specific framework in mind and are therefore suitable for any ML model. *Grad-CAM* is an example of a model-agnostic post-hoc XAI technique [41], generating visual explanations of ML models utilizing convolutional NNs. *Local Interpretable Model-agnostic Explanations* (LIME) [42] and *SHapley Additive eXplanations* (SHAP) [43] are *feature relevance* XAI techniques, aiming at ranking and/or measuring every feature's influence on the prediction output of the ML model [13]. While being developed for supervised and unsupervised learning, SHAP has been proven to provide useful explanations for RL applications such as the longitudinal acceleration control of a car [44] and for goal-based UAV navigation [29].

SHAP is used in this research to explain the input-output mapping of the actor NN for multiple reasons. First of all, SHAP is a post-hoc explainability tool, therefore no compromises are required in terms of reduced complexity of the implemented ML model for increased transparency. Secondly, SHAP is model-agnostic, indicating that it can be used for any ML model, allowing the explanation methodology to be used with any RL framework. Finally, SHAP is more accurate than LIME, as the output of SHAP's explanation model is proven to always match the original model [43]. A disadvantage of SHAP is its computational expense, which is considerably higher than LIME. However, as this research aims at providing meaningful and trustworthy insights of the RL algorithms for control experts, accuracy is deemed more important than computational expense.

D. Shapley Additive Explanations

Inspired by Shapley values from cooperative game theory, SHAP [43] provides explanations for any ML model using the additive feature attribution shown in Equation (10), where $f(x)$ is the original model (not to be confused with the state transition model), $g(x')$ a simplified explanation model and x'_i, the simplified input using the mapping function $x = h_x(x')$. The base value, $\phi_0 = f(h_x(0))$, represents the model output without knowledge of any of the features. Features are defined as the observation space, the states returned by the environment used as input by the NN. By calculating the marginal contributions of all features in all possible permutations, SHAP determines the SHAP values ϕ, representing the contributions of every feature to the model output.

$$f(x) \approx g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{10}$$

To apply SHAP for NNs with nonlinearities, this paper uses *DeepSHAP* [45], a model-specific algorithm of SHAP developed using the DeepLIFT algorithm [46]. This algorithm locally linearizes nonlinearities in the NN to compute the gradients between input and output, and uses a set of background samples to calculate the marginal contributions under all permutations, resulting in a guarantee of local accuracy [29]. This background set is often a random subset of all provided samples. In DeepSHAP, the mapping function $x = h_x(x')$ represents whether a feature is missing from the data set. As this paper assumes that all states are constantly measurable, all features are always assumed to be known and hence $x'_i = 1 \; \forall \; i$. Furthermore, as the local accuracy holds under DeepSHAP, Equation (10) can be rewritten as follows:

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i \tag{11}$$

III. Explanation Methodology for Adaptive Reinforcement Learning

This section presents how SHAP can be used to explain the inner workings of adaptive RL. First, the general explanation strategy for actor-critic structures with SHAP will be introduced. Subsequently, the technique for facilitating SHAP for adaptive control using the constant weight segment detection algorithm will be presented. Finally, the characteristics and possibilities of two specific SHAP plots will be explained.

A. Explaining Actor Input-Ouput Mapping using SHAP

Many state-of-the-art RL frameworks developed for low-level autonomous control, including IDHP, use the actor-critic structure as foundation of the framework. The actor and critic modules are often supplemented with additional framework-specific elements like the incremental model for IDHP [26], a target critic network [35], or experience replay for Deep Deterministic Policy Gradient [24]. This results in opaque decision-making, as not only the framework structure is complex, the inner workings of the individual elements including the NN of the actor and critic are also often hidden. However, only the actor module holds the policy function $\pi(\mathbf{x}, \mathbf{x}^R)$, ultimately responsible for choosing the actions based on the measured state and desired state. Therefore, the actor contains the learned strategy represented by $\mathbf{w}_a$, while the other elements are responsible for the learning process. The proposed methodology to better understand the inner workings of actor-critic RL is to extract this learned strategy by explaining the input-output mapping of the actor module using SHAP, as illustrated in Figure 4.

By recording the input of the actor NN and its weight vector $\mathbf{w}_a$, the SHAP values ϕ can be computed. Using SHAP's *DeepExplainer*, a copy of the actor NN is created with the same model parameters. Then, using the recorded inputs and the explanation model, the SHAP values are calculated. For the Shapley permutation calculations, a set of background samples is required. Generally speaking, the accuracy of the explanation increases when the number of background samples increases, but the computation time increases exponentially. Currently a rule of thumb for the number of background samples does not yet exist. In this research, 200 background samples are used for the SHAP calculations, unless differently specified.

B. Constant Weight Segment Detection

The SHAP explanation model requires constant model parameters. The actor weight vector $\mathbf{w}_a$ in adaptive flight control, however, is often changing due to the online learning. As non-constant model parameters have not yet been studied for SHAP models, a novel Constant Weight Segment Detection (CWSD) technique is proposed. The goal of

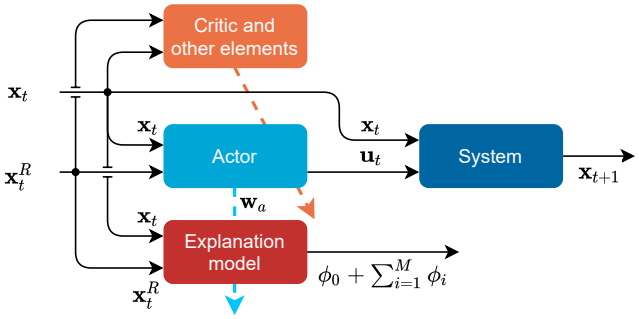


Fig. 4 General explanation strategy for actor-critic RL frameworks, illustrated for the IDHP example. The orange arrow indicates backpropagation for optimizing the actor parameters, and the blue arrow represents copying of the actor network parameters for the SHAP analysis.

this algorithm is to generalize the behavior of the adaptive agent, by selecting segments with near-constant weights. However, different weights do not necessarily imply that the input-output mapping has changed. Therefore it is possible that CWSD selects multiple segments where the input-output mapping is actually similar. Whether these segments show different learned behavior can be analyzed using SHAP or potentially other XAI techniques. This method, summarized in Algorithm 1, selects time segments in which $\mathbf{w}_a$ is kept within a certain variation, based on the tunable maximum allowable weight deviation Δw_{max}, minimum segment length S_{min}, and maximum segment length S_{max}. Starting from $t = 0$, the actor weight vector $\mathbf{w}_a(t)$ is marked as a border b when any of its elements deviates more than Δw_{max} with respect to the previous border. The definition of this maximum deviation is presented in Equation (12), where ϵ_{CW} is a tunable parameter determining Δw_{max} as a percentage of the maximum deviation in $\mathbf{w}_a$. Another reason for defining $\mathbf{w}_a(t)$ as a border is when the number of time steps since the previous border surpasses the maximum segment length parameter S_{max}. After the borders have been determined, the segments between borders are selected if their length is more than the minimum segment length S_{min}. The resulting output is $\mathbb{S}$, the set of constant weight sections with potentially unequal lengths. The SHAP analysis, following the CWSD, uses $\mathbf{w}_a$ at the centre point of the segment. The sample coverage metric, defined in Equation (13) where $L_s \in \mathbb{N}$ represents segment length and N is the total number of samples, assesses the ratio between the samples covered by the constant weight segments and the total number of samples. An example outcome of the CWSD algorithm can be seen in Figure 12, where the borders are represented using vertical blue lines, and the selected regions by the blue shaded area between the borders.

$$\Delta w_{max} = \epsilon_{CW} \cdot (\max \mathbf{w}_a - \min \mathbf{w}_a) \tag{12}$$

$$\text{sample coverage} = \frac{\sum_{s=0}^{S} L_s}{N} \tag{13}$$

With CWSD, a trade-off can be made between accuracy of the constant weights section approximations, and generalization of the dynamics, using ϵ_{CW}. A low ϵ_{CW} setting allows little weight variance within a segment, often resulting in many shorter segments. The function approximation of these shorter segments is generally better compared to segments detected with higher ϵ_{CW} settings, but the use of many segments prevents generalization of the dynamics. Furthermore, it will be shown that segments that are too short can result in accurate global function approximation, but poor local approximation. With a higher ϵ_{CW} setting, CWSD is more lenient regarding weight variance, often resulting in fewer and longer segments, with a reduced function approximation accuracy. However, this limited number of segments is beneficial for generalization of the dynamics.

The S_{min} parameter allows a trade-off between sample coverage and SHAP accuracy. Generally speaking, the accuracy of the SHAP values increases when the number of background samples increases. Therefore, SHAP values within longer constant weight segments will likely be more accurate. Currently, a rule of thumb regarding the recommended number of SHAP background samples for RL does not yet exist. Another difficulty is that there is yet no method of assessing the accuracy or certainty of SHAP values. Higher S_{min} settings will often result in reduced sample coverage, as potential segments shorter than S_{min} will be excluded. Furthermore, S_{min} can be used to prevent very short segments which, as mentioned for ϵ_{CW}, can exhibit poor local accuracy. Finally, the S_{max} parameter does not control a

Algorithm 1: Constant Weight Segment Detection

input : Actor parameter vector $\mathbf{w}_a \in \mathcal{R}^{MxN}$, maximum allowable deviation ϵ, minimum segment length S_{min}, maximum segment length S_{max}
output : Constant weight segments $\mathcal{S}$
determine maximum allowable weight change $\Delta w_{max} \leftarrow \epsilon \cdot (\max(\mathbf{w}_a) - \min(\mathbf{w}_a))$;
initialize empty border list $\mathcal{B}$;
initialize last border vector $\mathbf{w}_{lim} \leftarrow \mathbf{w}_a(t = 0)$;
$\mathcal{B}$.insert(0);
for *t=0, 1, 2, ..., N* **do**
  for *m=0, 1, 2, ..., M* **do**
    if $\mathbf{w}_a(m) - \mathbf{w}_{lim}(m) > \Delta w_{max}$ ***or*** $t - b_{last} > S_{max}$ **then**
      $\mathcal{B}$.insert(t) ;
      break
    end
  end
end
set number of borders $B \leftarrow \mathcal{B}.length$;
for *b=0, 1, 2, ..., B-1* **do**
  if $\mathcal{B}(b + 1) - \mathcal{B}(b) > S_{min}$ **then**
    $\mathcal{S}$.insert(b) ;
  end
end

trade-off, but can be used to force long segments into multiple shorter segments, if segments with similar length are desired for example. The relation between number of background samples and accuracy of the SHAP values will be investigated by comparing the SHAP results of the same time series data, with altering S_{min} and S_{max} values.

C. Combined Dependence Plot

The SHAP dependence plot shows the relation between feature value x_i and SHAP value ϕ_i for one of the NN's inputs. This allows detailed investigation of the relation between feature value and its effect on the model output. To compare this relation between the different segments, the novel *combined dependence plot* is proposed. This plot shows the relation between x_i and ϕ_i for every segment on the same plot. Furthermore, the dependence plots for all features can be combined in the same figure. By including the dependence plots of the other features and using the same scale for ϕ, *feature importance* can be estimated by comparing the range of ϕ. A feature with higher importance will have a larger range and vice versa.

IV. Flight Simulation

This section presents the flight simulation of a Cessna Citation II controlled with IDHP, serving as an example to investigate how SHAP can be used to explain RL for adaptive flight control. The flight control simulation is adapted from [35]. First, the high-fidelity simulation model will be introduced. Secondly, the flight controller including IDHP will be presented together with the used hyperparameters.

A. Cessna Citation Simulation Model

The data used is generated using a high-fidelity, six degrees-of-freedom, non-linear model of the Cessna Citation 500, developed by Delft University of Technology. The model includes engine and actuator dynamics as well as sensor models. The state space of the simulation is presented in Equation (14), and the action space in Equation (15). The roll angle is described using φ, to prevent confusion with the SHAP values. p, q and r are the roll rate, pitch rate and yaw rate, respectively. V_{tas} is the aircraft's true airspeed. α, β and θ are the angle of attack, sideslip angle and pitch angle, respectively. Finally, H is the aircraft's altitude.

For this simulation, a sampling frequency of 50 Hz is used. The thrust of the aircraft is controlled using autothrottle,

and the built-in yaw damper is disabled to allow full control authority of the rudder. Furthermore perfect measurements are assumed, therefore the sensor models are not used. The asymmetric actuator deflection saturation limit for δ_e is [-20.05, 14.90] deg. The symmetrical saturation limits for δ_a and δ_r are respectively ± 37.24 deg and ± 21.77 deg.

$$\mathbf{x}^T = [p \quad q \quad r \quad V_{tas} \quad \alpha \quad \beta \quad \varphi \quad \theta \quad H] \tag{14}$$

$$\mathbf{u}^T = [\delta_e \quad \delta_a \quad \delta_r] \tag{15}$$

B. Flight Controller

The flight controller used to control the simulated Cessna Citation is schematically illustrated in Figure 5. Assuming no significant coupling between longitudinal and lateral controller, the IDHP framework is used for angular rate control, for both longitudinal and lateral control. Rate control is less complex than angle control due to the direct relation between actuator output and angular rate. PID controllers are used for the outer loop control, resulting in p_{ref} for the lateral IDHP controller controlling δ_e, and q_{ref} for the longitudinal controller controlling δ_a and δ_r. β_{ref} is always set to 0, to prevent undesired sideslip. The boolean selection matrices $\mathbf{P}$ as defined in Equation (5) are set up to select the relevant controlled states. $\mathbf{Q}_{lon}$ is 1, as the longitudinal controls one state. The lateral controller uses $\mathbf{Q}_{lat} = \text{diag}(1, 100)$ as the $\beta - \beta_{ref}$ error is often significantly smaller than the $p - p_{ref}$ error.

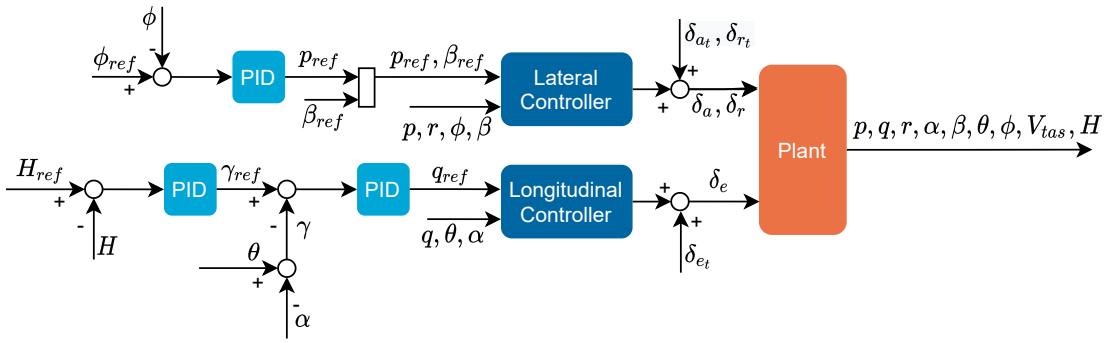


Fig. 5 Schematic representation of the flight controller, adapted from [35].

The hyperparameters of both the lateral and longitudinal IDHP controllers are similar: the learning rate of the actor, η_a, is fixed at 1, and the learning rate of the critic, η_c, is fixed at 2. The discount factor $\gamma = 0.8$. The NN weight vectors $\mathbf{w}_a$ and $\mathbf{w}_c$ are randomly initialised with $N(\mu = 0, \sigma = 0.05)$.

V. Results and Discussion

This section presents the results and discussion of the conducted experiment. First, the longitudinal and lateral controller are trained using pitch and roll rate tracking tasks, after which the learned control behavior is explained using a waterfall plot, to introduce how SHAP can be used to explain RL for flight control, and using SHAP summary plots to show feature importance and relation between input-output mapping of the actor. Secondly, the trained IDHP controller is used to simulate a typical flight profile including climbs, descents, and coordinated turns, after which the learned control strategy is explained. Both the training and mission profile phase use online control, indicating that the framework is learning on-the-go from the data it generates by executing its policy. As the model parameters change during the online adaptive flight, CWSD is used after the simulation to generalize the control strategy. As the CWSD algorithm requires the complete $\mathbf{w}_a$ vector over time, the SHAP analysis is performed after the simulation. Finally, the effect of segment length and of background samples on the explanation model accuracy is investigated.

A. Training Phase

In this section, the training process of both the longitudinal and the lateral controllers is discussed, after which the learned strategy is explained using SHAP. As the model parameters are stable when training converges, the training

process is relatively simple process to explain how SHAP can be used to interpret the learned strategy of RL for flight control, before moving on to the mission profile with continuously changing weights.

1. Simulation

Similar to [35], the longitudinal and lateral controller are trained separately such that the system matrix $\hat{\mathbf{F}}$, control effectiveness matrices $\hat{\mathbf{G}}$ and actor and critic parameters $\mathbf{w}_a$ & $\mathbf{w}_c$ converge to stable values. To minimize coupling effects, both controllers are trained while keeping the actuators associated with the other controller constant at trim value. The longitudinal controller is trained using a sinusoidal pitch rate tracking task with an amplitude of 5° and a frequency of 0.2 Hz, during a period of 60 s as shown in Figure 6.

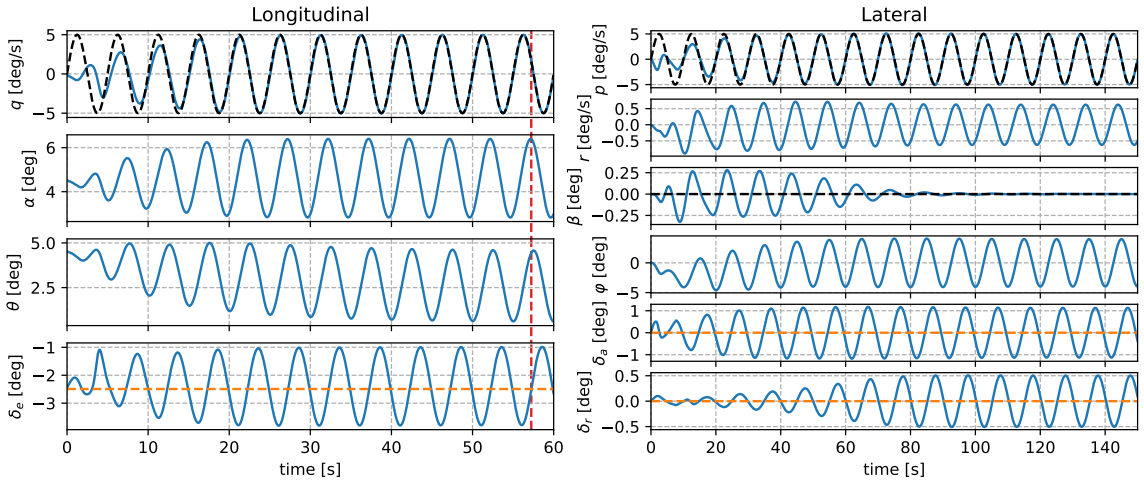


Fig. 6 Input and output of the actor neural network during the individual longitudinal and lateral training phase. Reference values are indicated using the black dashed lines, while the trim conditions are illustrated using the orange dashed lines. The sample indicated with the red dashed line for the longitudinal controller is explained in Figure 8.

The lateral controller is trained using a roll rate tracking task with an amplitude of 5° and frequency of 0.1 Hz and $\beta_{ref} = 0$. The training phase of the lateral controller is 150 s, as the task of minimizing sideslip is more complex than a rate tracking task, due to the rudder deflection being more closely related to yawing rate than to the sideslip angle. The elevator deflection δ_e and the aileron deflection δ_a have a relatively direct relation to q and p, respectively. The relation between the rudder deflection δ_r and the sideslip angle β is more difficult to learn. To force state-space exploration, exponentially decaying sinusoidal excitation is used for δ_e and δ_a. This excitation is a damped sine wave, with an amplitude of 1°, the frequency is 0.2 Hz, and the time constant τ is 5 s. The excitation decays over time such that controllers do not learn to compensate for the decaying signal. The sinusoidal excitation is not used for δ_r as the yawing movement is excited through the aircraft's Dutch roll eigenmode. Figure 6 shows the input and output of the actor NN during the training phase for both the longitudinal and the lateral controller. Stable tracking of p_{ref} and q_{ref} is obtained in 15 seconds, while β is minimized after approximately 90 seconds. The reason for the slower minimization of β will be explained using the system identification visualization.

Figure 7 presents the convergence of the system model estimates and the actor and critic weights. Most parameters of $\hat{\mathbf{F}}$ and $\hat{\mathbf{G}}$ converge in approximately 15 seconds, except for the parameters corresponding to β and δ_r, as the yawing motion is more complex to estimate due to the rudder not being directly excited and the control surfaces being more related to angular rotation than to the sideslip angle [35]. The stable model identification parameters allow $\mathbf{w}_c$ and $\mathbf{w}_a$ to converge after approximately 40 seconds for both the longitudinal and the lateral controller. As these last 20 seconds of the training phase show constant weights, this time segment is used to explain the learned strategy using SHAP.

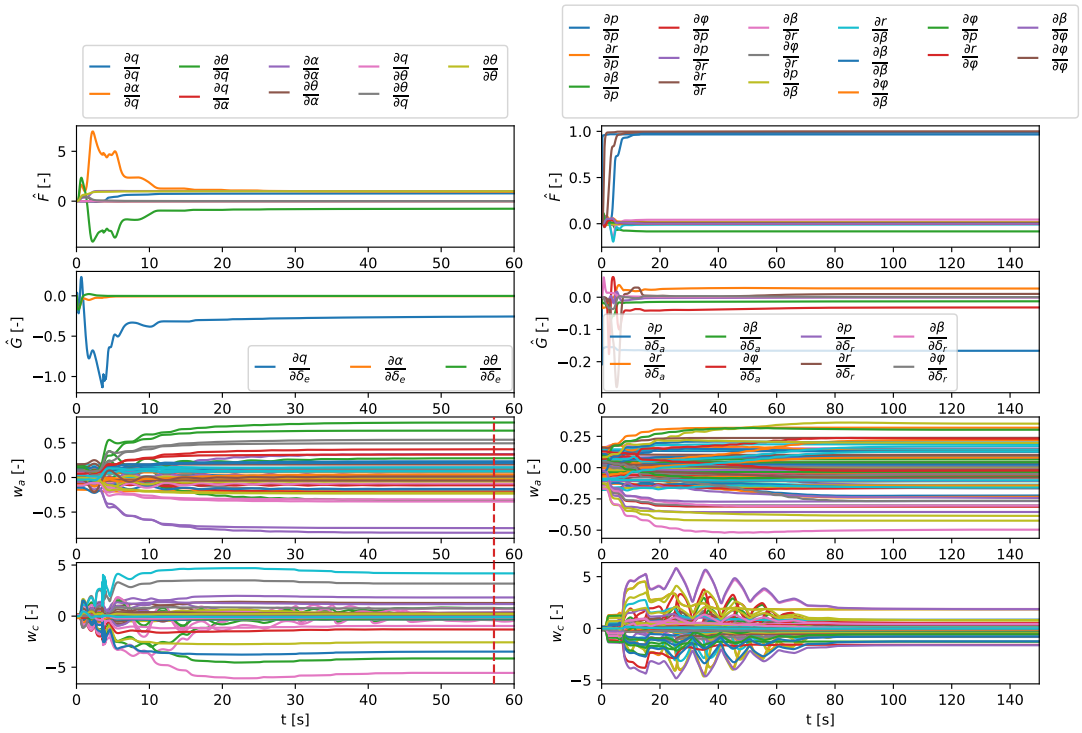


Fig. 7 Convergence of the system and control effectiveness matrices and the actor and critic weights during training of the longitudinal (left) and lateral (right) controllers. The sample at the time step indicated with the red line of the longitudinal controller is explained in Figure 8.

2. Explanation

To introduce how SHAP can be used to explain RL for flight control, the waterfall plot will be presented first. This plot is less complex compared to other SHAP plots as it explains one single prediction as opposed to global insights. The waterfall plot is created for the time instance indicated using the red dashed line in Figure 6 and Figure 7, and the used SHAP values are calculated based on all samples during the last 20 seconds of the longitudinal training phase. It is assumed the weights are near-constant during this period, as the actor weights have converged before it. From these last 20 seconds, 200 randomly selected instances are selected as the SHAP background samples. Figure 8 shows the waterfall plot for the longitudinal controller at $t = 57.24s$, the time instance marked with the red dashed line in Figure 6 and Figure 7. This time step is selected as it displays both positive and negative SHAP values. Starting from the SHAP base value, calculated as the average NN output during the last 20 seconds $\phi_0 = E[\delta_e] = 0.181$ deg, the individual SHAP components ϕ_i are added, resulting in the final actor output $\delta_{e,SHAP} = -0.024$ deg. This model output is the deviation from the elevator trim setting $\delta_{e_t} = -2.49$ deg. Note that the trim value is not the same as the SHAP base value, as ϕ_0 is computed as the average model output during the provided time instances and therefore changes depending on the provided time window. Using the individual SHAP values ϕ_i, the waterfall plot shows how every feature contributes to the final model output. Given the time instance illustrated in Figure 8 for example, both q and θ have an important contribution, pushing or pulling δ_e relative from its base value. The provided value of θ has a positive contribution, while q for this sample has a negative contribution. The contribution from α and err_q is less significant for this sample. As SHAP measures the marginal contribution, which is often dependent on the distribution of the other feature values, the SHAP value for a given feature value can change depending on the other features. For example, $\phi_q = 0.4833$ for $q = 0.16524$ deg/s according to Figure 8. While keeping q constant but changing the value of the other features, it is possible that ϕ_q changes. To investigate this interaction behavior, more samples are required.

While the waterfall plot is useful for local interpretability and to introduce the basic principles of SHAP, it does not offer global insights such as average feature importance, relation between feature value and SHAP value, and insights into interaction behavior. The SHAP summary plot provides global interpretability by showing feature importance and

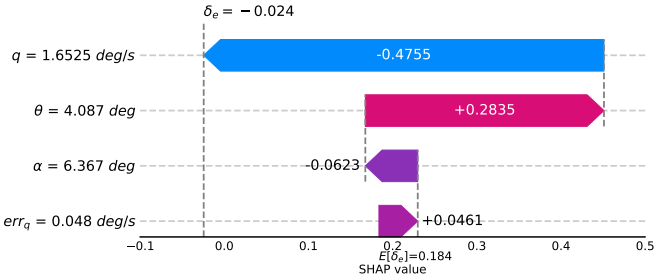


Fig. 8 Waterfall plot for δ_e at $t = 57.24$ s, showing the positive and negative contributions to δ_e, starting from the base value $E[\delta_e] = 0.181$ deg. The model output $\delta_e = -0.024$ deg represents the deviation from the elevator trim value $\delta_{e_t} = -2.49$ deg.

a qualitative description of the relation between feature value and SHAP value. Figure 9 shows the SHAP summary plot of δ_e, based on all $\mathbf{x}$ samples and the near-constant $\mathbf{w}_a$ during the last 20 seconds of the longitudinal controller training. Figure 9 shows that the high feature importance of q and θ in the waterfall plot Figure 8 is not incidental, as these two features are significantly more important than α and err_q. Furthermore, Figure 8 shows the qualitative relation between SHAP value on the x-axis and feature value, colored from low represented with blue to high represented in red. This explains that high pitch rates enforce negative δ_e and vice versa, while high θ enforces positive δ_e and vice versa. The exact relation between feature value and SHAP value can be observed using the dependence plots, which will be introduced to explain the mission profile, together with a more extensive analysis of the learned strategy.

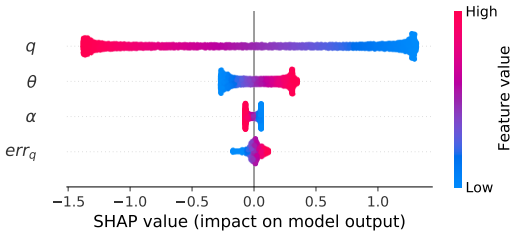
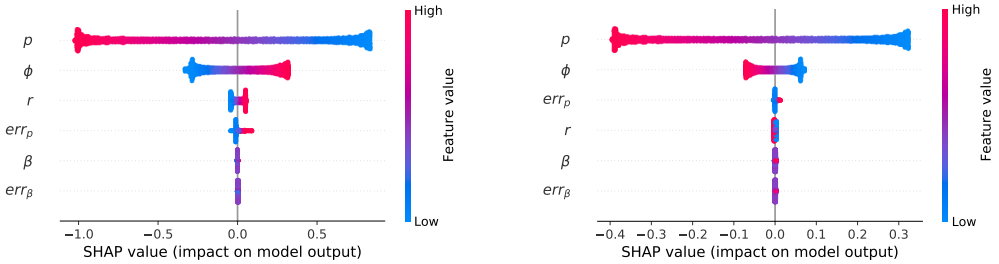


Fig. 9 SHAP summary plot for δ_e, using all samples of the last 20 seconds of longitudinal controller training.

Figure 10 shows the summary plots for δ_a and δ_r, based on the last 20 seconds of the lateral controller tuning. For both these actor outputs, p is the most important feature, followed by φ. The other features are significantly less important. Similar to δ_e, the exact relations between the feature values and their corresponding SHAP values are presented during the analysis of the full mission profile.



(a) δ_a output (b) δ_r output

Fig. 10 SHAP summary plots of the lateral controller based on the last 20 seconds of the training phase.

B. Full Mission Profile

After the online training phase, the converged incremental model parameters and NN parameters of the actor and critic are used to initialize the longitudinal and lateral controller to simulate a typical mission profile where the online-learning longitudinal and lateral controller are used simultaneously. This section first presents the results from the mission profile. Then the learned strategy of the longitudinal controller is explained using the CWSD algorithm and SHAP. Finally, the learned control strategy of the lateral controller is explained.

1. Simulation

The mission profile, similar to [35], includes a climb, descent, and multiple left and right turns executed during 270 s. While the converged incremental model and actor and critic parameters are used to initialize the longitudinal and lateral controllers, online learning continues during flight. Figure 11 shows the executed mission profile, starting at $H = 2000$ m and $V_0 = 90$ m/s. For the longitudinal controller, the altitude mission profile is converted into q_{ref} using the PID controller. For the lateral controller, the coordinated bank profile is converted into p_{ref} using the PID controller, which is forwarded to the IDHP agent together with $\beta_{ref} = 0$.

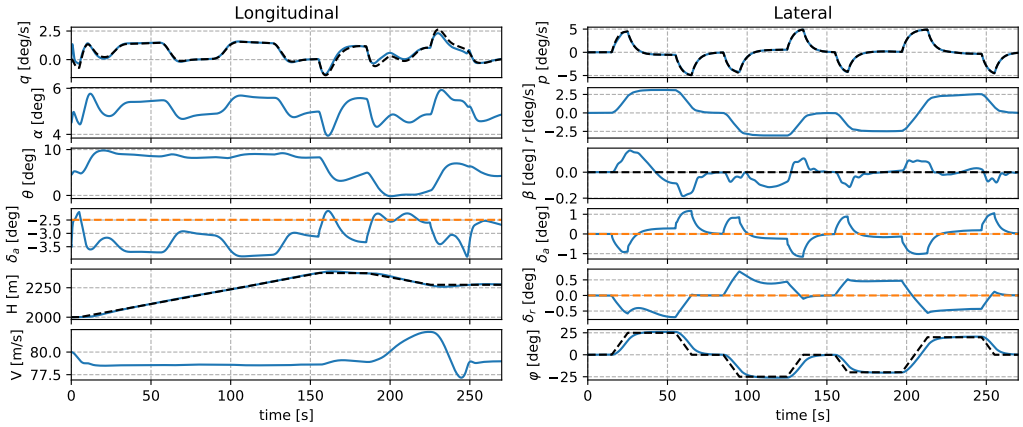


Fig. 11 Input and output of the actor neural network during the combined mission profile, for both the longitudinal and the lateral controller. The reference values for both IDHP and the PID controllers are shown in black dashed lines, while the actuator trim values are shown using orange dashed lines.

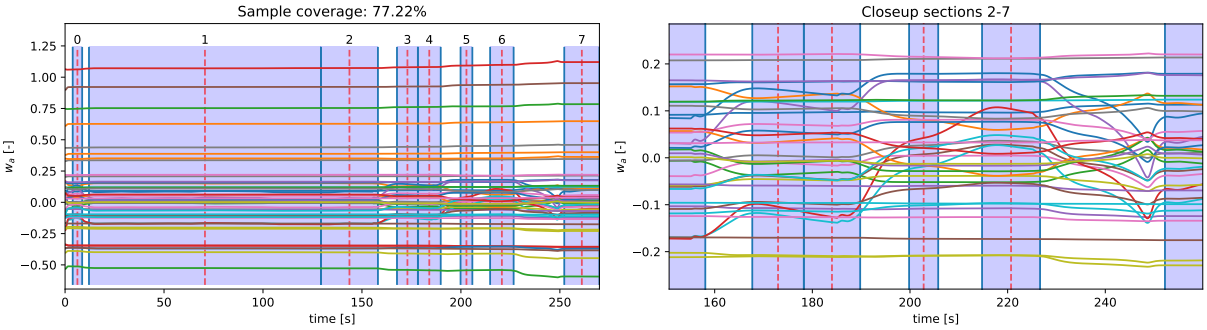
2. Explanation Longitudinal Controller

The changing actor model parameters of the longitudinal agent are presented in Figure 12, together with the selected segments, the result of the CWSD algorithm applied to the longitudinal controller. The CWSD algorithm uses $\epsilon_{CW} = 0.05$, $S_{min} = 200$, and $S_{max} = \infty$. The ϵ_{CW} parameter is found by trial and error, and the S_{min} parameter prevents segments that are too short, which lead to poor local fit as will be described in more detail later. The borders of the selected segments are indicated using the vertical blue lines, while the vertical red dashed lines indicate the centre of the segments where the model weights are selected for the SHAP analysis.

To present and compare the SHAP values from all segments, the combined dependence plots are used. Figure 13 shows these plots for δ_e, where linear relations between feature value and SHAP value for every segment of δ_e can be observed. The summations of neurons and nonlinear activation functions allow nonlinear relations, but the learned strategy between feature value and SHAP value is apparently linear. In the rest of this paper, the relationships between feature value and SHAP value will be described using linear relationship properties like slope and bias. Not only are the relationships between feature value and SHAP value for δ_e linear, the slope of the relations appears to be similar for many segments. Particularly the relationships of q and err_q appear to be parallel, while slight variations can be observed for α and θ.

Using the information provided in the combined dependence plot for δ_e, Figure 13, the following observations can be made about the learned behavior regarding elevator control during the mission profile:

- The pitch rate q is the most dominant feature for δ_e, as it shows the largest range of SHAP values in Figure 13. The



(a) CWSD result

(b) Closeup sections 2-7

Fig. 12 Actor weight matrix w_a of the longitudinal controller during the mission profile, with the segments selected by the constant weight segment detection algorithm. The blue vertical lines represent borders of the selected segments, while the red dashed lines indicate the time instance where the weights are selected. The closeup of sections 2-7 with a reduced vertical range is included to clarify why certain regions are defined as constant-weight sections.

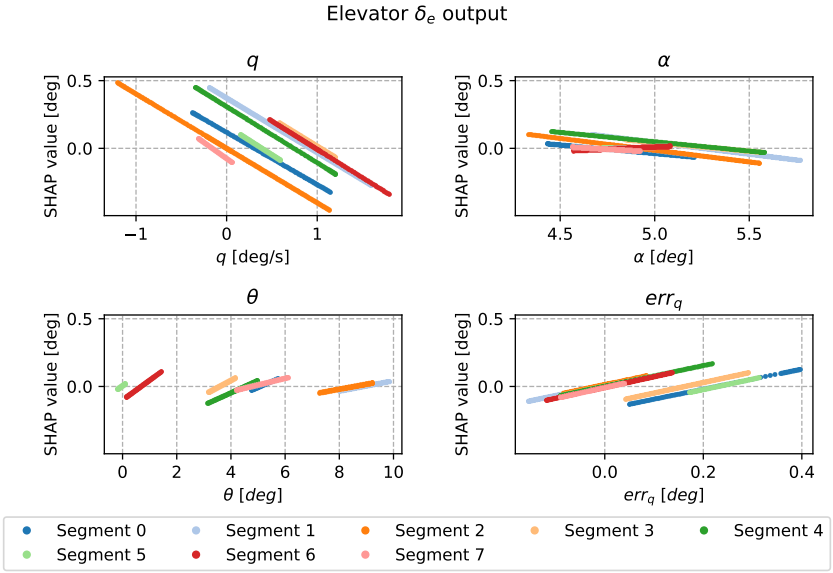


Fig. 13 Combined dependence plot for δ_e during the mission profile.

relation between pitch rate and its SHAP values is negative, implying that negative δ_e is enforced during pitch-up maneuvers. Interestingly, this is in theory destabilizing control behavior, as negative δ_e result in additional pitch-up. However, the slope a_q is significantly less than a_{err_q} hence the overall elevator control behavior is still stabilizing. The reason for the negative slope is likely the large damping factor C_{m_q} mainly caused by the fuselage and the horizontal stabilizers. To counteract this damping factor during pitching maneuvers, the longitudinal IDHP agent has learned to use the elevator.

- The importance of α is lower when compared to q, but is still significant. The explanation for the negative slope is similar to that of a_q: the stabilizing C_{m_α} from the horizontal stabilizers is counteracted with the elevator. The reason for the shifting slopes is currently unknown. Potentially there is connection with the shifting slopes of θ, as these features together form the flight path angle $\gamma = \theta - \alpha$. Further research is required to investigate whether this is the right explanation.
- The importance of the pitch angle θ is minimal compared to the other features. The relation between θ and ϕ_θ appears to be negative for low θ values, and around 0 for higher θ values. As mentioned, there might be feature

interaction with α.
- The pitch rate error $q - q_{ref}$ is the second most important feature and has the steepest slope of all features for the elevator. If the error is positive, indicating too high pitch rate, positive δ_e is enforced, lowering the pitch rate and vice versa.

While the linear relations presented in Figure 13 make the control strategy learned by IDHP during this mission profile easier to understand, more simplifications can be made, starting from the additive feature property as shown in Equation (16). While the slope of the linear relationship is often close to constant for most segments, the linear bias changes significantly. Assuming that the relation between feature value and SHAP value is linear, and using the property of no bias in the NNs, it can be shown that the bias in the combined dependence plots is irrelevant.

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x_i' \tag{16}$$

The combined dependence plots show that all partial dependence relations can be approximated using a linear relationship as represented by Equation (17). Assuming this linear expression is valid for all features, this linear representation can be inserted into the additive feature property (Equation (16)) resulting in the expression shown in Equation (18).

$$\phi_i = a_i \cdot x_i + b_i \tag{17}$$

$$g(x') = \phi_0 + \sum_{i=1}^{M} [a_i \cdot x_i + b_i] \tag{18}$$

As the actor NN does not use bias, $\delta_e = \delta_{e_{SHAP}} = 0$ when $\mathbf{x} = 0$. Inserting this condition in Equation (18) shows that the SHAP base value is equal to the negative sum of bias values as shown in Equation (19).

$$\phi_0 = -\sum_{i=1}^{M} [b_i] \tag{19}$$

Using this expression, Equation (16) can be simplified to the form as shown in Equation (20). The learned strategy for δ_e can therefore be modelled as in Equation (21), showing that only the slopes of the relations between feature value and SHAP value are relevant.

$$g(x') = \sum_{i=1}^{M} [a_i \cdot x_i] \tag{20}$$

$$\delta_{e_{lin}} = a_q \cdot q + a_\alpha \cdot \alpha + a_\theta \cdot \theta + a_{err_q} \cdot err_q \tag{21}$$

The SHAP and linear representation models for δ_e are verified by comparing their outputs with the actual deflection δ_e and the output of the actor NN with the constant weights assumption in Figure 14. The constant weight model $\delta_{e_{CW}}$ represents the elevator deflection as commanded by IDHP, using fixed weights during the segment. The SHAP model and linear representation exactly match the constant weight model, verifying both the accuracy of the SHAP values, and the legitimacy of the linear representation.

3. Explanation Lateral Controller

As with the longitudinal controller, the segments are created based on the changing weight vector $\mathbf{w}_a$, as shown in Figure 15. The same CWSD parameters are used: $\epsilon_{CW} = 0.05$, $S_{min} = 200$ and $S_{max} = \infty$. As the actor NN of the lateral IDHP agent controls both δ_a and δ_r, the presented segments are similar for these two actuators.

The combined dependence plot for δ_a is presented in Figure 16. As with δ_e, all relations between feature value and SHAP value are linear, hence the linear representation can be used to describe the learned strategy of the controller. The linear representation for δ_a is shown in Equation (22).

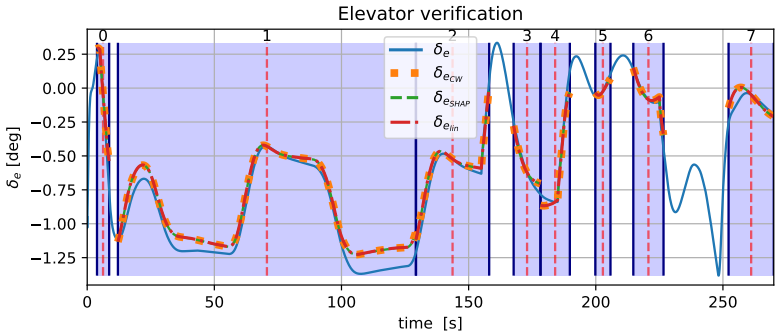


Fig. 14 Verification of the elevator SHAP model and linear representation model, by comparing their output with the actual δ_e and the constant weight model.

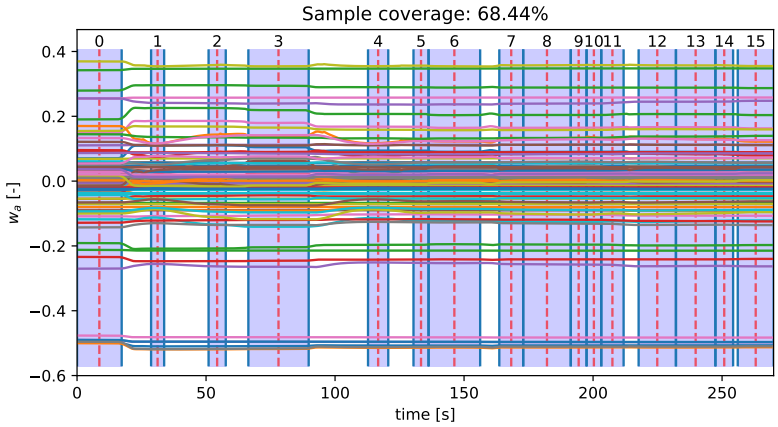


Fig. 15 Actor weight matrix w_a of the lateral controller during the mission profile, with the segments selected by the constant weight segment detection algorithm. The blue vertical lines represent borders of the selected segments, while the red dashed lines indicate the time instance where the weights are selected.

Using the combined dependence plots for δ_a, Figure 16, the following conclusions about the learned aileron strategy can be drawn:

- The roll rate p has strong influence on δ_a, and a negative effect. Hence, during rolling, the agent has learned to maintain the ailerons to sustain the roll. Similar to the pitch rate for δ_e, the negative relation between p and its SHAP values is destabilizing, but is required to sustain the angular motion. During a rolling maneuver, there is a strong damping moment C_{l_p} due to the wings. The agent has learned to counteract this moment using the ailerons.
- The yawing moment r is less significant, and shows a positive relation. This learned behavior can likely be attributed to the Dutch roll eigenmode. During positive yawing movement, e.g. the nose of the aircraft is turning to the right, the left aircraft wing has more velocity with respect to the right wing. This induces a positive rolling moment, which is counteracted with positive δ_a.
- The sideslip angle β is irrelevant for δ_a.
- The importance of the roll angle φ is relatively low, but the relation between roll angle and SHAP value changes between positive and negative during the mission profile. The reason for this is unknown. Further research is required to investigate whether this behavior is useful or not.
- The influence of the error $p - p_{ref}$ is significant, and positive, similar to $q - q_{ref}$ for δ_e. When the error is positive, and hence the roll rate is larger than required, ϕ_{err_p} is positive to encourage counteractive rolling.
- Similar to β the error $\beta - \beta_{ref}$ appears to be irrelevant for δ_a.

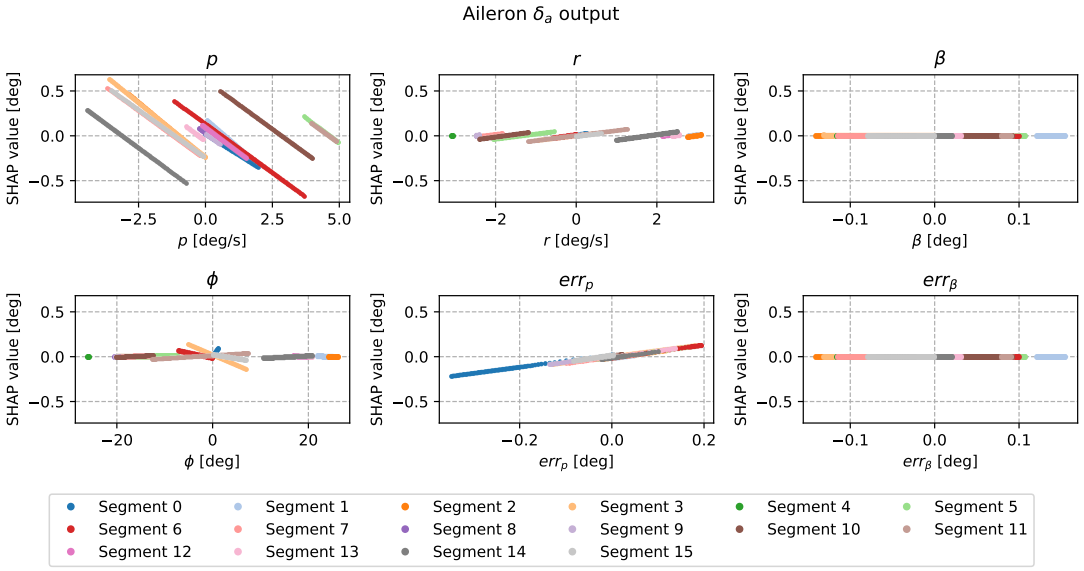


Fig. 16 Combined dependence plots for δ_a during the mission profile.

The combined dependence plots for δ_r are presented in Figure 17.
Similar to δ_e and δ_a, the relations between feature value and SHAP value are linear for δ_r. The following characteristics of the learned behavior for the rudder can be extracted from the combined dependence plots for δ_r:

- During rolling maneuvers, the two wings cause adverse yaw due to different induced drag forces. The corresponding C_{n_p} derivative is negative for the Cessna Citation I aircraft. Therefore, during a roll to the right, the wings induce a negative yawing moment. The negative SHAP values for positive p encourage positive yawing, cancelling the adverse yaw.
- The roll rate r is insignificant for δ_r.
- The β and $\beta - \beta_{ref}$ features are in fact similar, as $\beta_{ref} = 0$. Interestingly, these features are irrelevant for δ_r. Apparently, the other features are more effective at canceling sideslip.
- The roll angle φ is significant for δ_r and has a negative effect. This behavior is likely learned to ensure a coordinated turn.

Due to the linear relations between feature value and SHAP value, the control strategy for δ_a and δ_r can be represented as a linear controller, as shown in Equation (22) and Equation (23), respectively. These models, and the SHAP explanation models for δ_a and δ_r are verified with the actual deflections, and the output of the IDHP controller when assuming constant weights in Figure 19.

$$\delta_{a_{lin}} = a_{p_{\delta_a}} \cdot p + a_{r_{\delta_a}} \cdot r + a_{\beta_{\delta_a}} \cdot \beta + a_{\phi_{\delta_a}} \cdot \phi + a_{err_{p\,\delta_a}} \cdot err_p + a_{err_{\beta\,\delta_a}} \cdot err_\beta \tag{22}$$

$$\delta_{r_{lin}} = a_{p_{\delta_r}} \cdot p + a_{r_{\delta_r}} \cdot r + a_{\beta_{\delta_r}} \cdot \beta + a_{\phi_{\delta_r}} \cdot \phi + a_{err_{p\,\delta_r}} \cdot err_p + a_{err_{\beta\,\delta_r}} \cdot err_\beta \tag{23}$$

C. Minimum Segment Length Analysis

As stated in Section III, there is no rule of thumb yet for the number of background samples to analyse RL using SHAP. Furthermore, to maximize sample coverage of CWSD, it would be beneficial to use the shortest segments possible. The maximum number of background samples within a segment is limited to the number of samples covered by the segment. Therefore, segments that are too short may result in inaccurate SHAP values as there are not enough background samples. To investigate the effect of segment length, and therefore also the effect of background samples, on the SHAP analysis for RL, this section explores what happens when S_{min} is minimized.

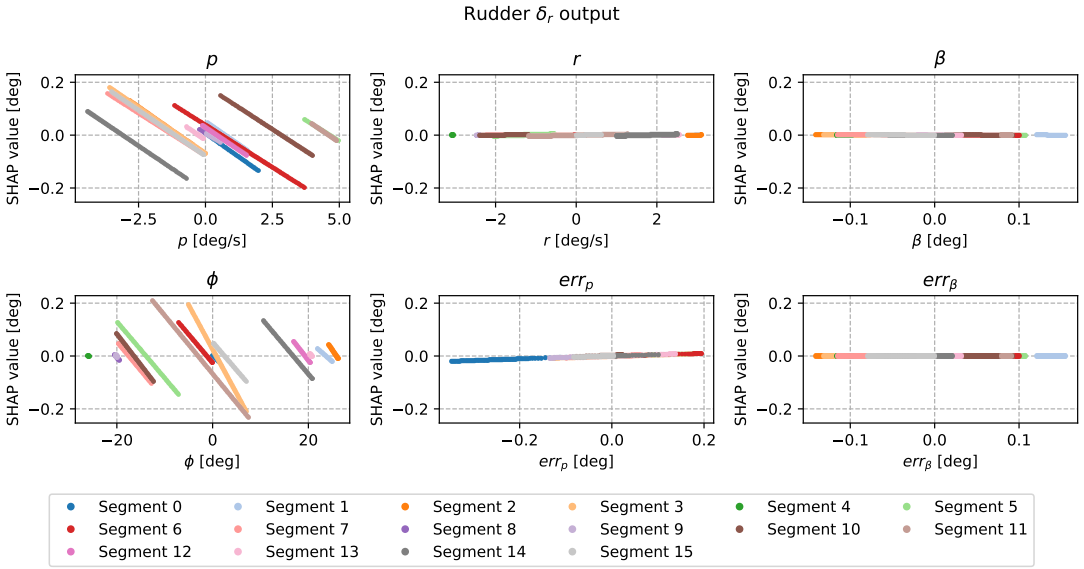
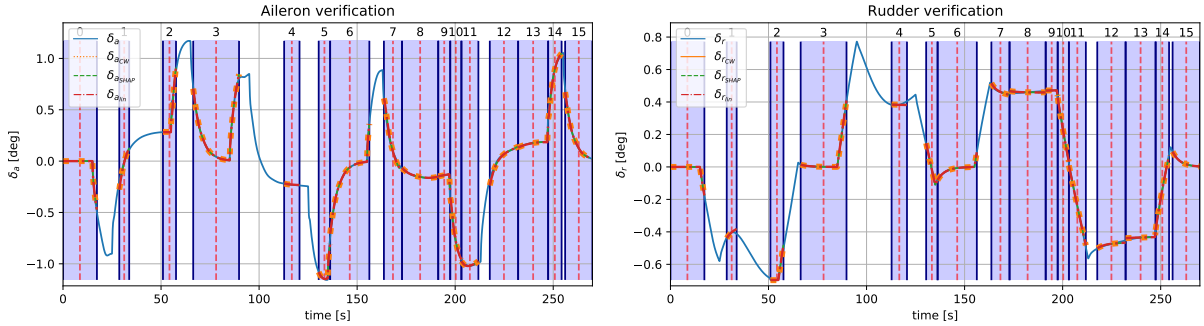


Fig. 17 Combined dependence plots for δ_r during the mission profile.

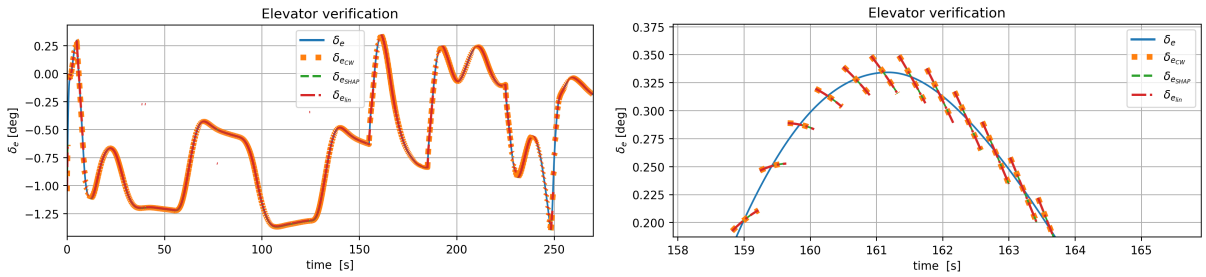


(a) Aileron δ_a (b) Rudder δ_r

Fig. 18 Verification of the SHAP model and the linear representation for the lateral controller.

The same mission profile as shown in Figure 11, and the same actor weights $\mathbf{w}_a$ over time of the longitudinal controller as depicted in Figure 12 are used. The CWSD parameters S_{min} and S_{max} are now set to 20 and 20 respectively, to force all segments to be 20 samples long. The SHAP values are calculated for every segment, using all 20 samples as background samples for the permutations. As there are now 640 segments, the combined dependence plots are too cluttered. However, from the verification plot, Figure 19, several useful conclusions about segment length can be drawn.

As expected, Figure 19 shows significantly more sample coverage than the original SHAP analysis for the mission profile. $S_{min} = 20$ results in a sample coverage of 99.56%. Furthermore, the global accuracy looks better the SHAP analysis using $S_{min} = 200$. However, locally the models show inaccuracies regarding the actual elevator deflection, as shown in the close-up. Upon close inspection it can be observed that $\delta_{e_{CW}} = \delta_{e_{SHAP}} = \delta_{e_{lin}}$, and that these three models do not match the actual δ_e. Hence, these inaccuracies are caused by the constant weight assumption, and not by inaccuracies of SHAP. The ability of SHAP to match $\delta_{e_{CW}}$, the output it is trying to explain, with minimal background samples can likely be attributed to DeepSHAP's effective method of calculating the gradient of the NN's output with respect to its inputs.



(a) Global accuracy

(b) Close-up showing local inaccuracies

Fig. 19 SHAP and linear representation models verification for δ_e, now using $S_{min} = 20$, $S_{max} = 20$ to force shorter segments.

D. Discussion of the Results

The preceding subsections show that SHAP can provide useful insights into the inner workings of RL for adaptive flight control, by identifying the relations between the actor's input and the commanded actuator deflections. Where previous research focused on explaining high-level goal based navigation, this research investigated how XAI can be used to explain RL for low-level adaptive flight control. The identified input-output mappings can help control experts, the main target audience, recognize potential agent misbehavior by comparing the learned strategy with their expert domain knowledge. Furthermore, the assessment of feature importance can help control experts in limiting the input space used by the RL framework.

As constant model parameters are required by SHAP, the CWSD algorithm allows identification of segments with near-constant weights. The generated explanations are proven to match the IDHP output under the constant weights assumption. However, by fixing the weights in the selected segments, the learned control behavior is generalized, resulting in inaccuracies between the actual model output and the model output using fixed weights. Using the CWSD approach, there will always be a trade-off between generalization of the dynamics and accurate representation of the learned behavior. Control experts can use CWSD's ϵ_{CW} and S_{min} parameters to control this trade-off. Furthermore, the relations between feature value and SHAP value identified in this research are all linear, likely due to the relatively simple NN architecture and the relatively simple inner-loop rate tracking task. The functionality of the proposed combined dependence plots may be reduced when the input-output mappings are more nonlinear, potentially cluttering the plots.

While explaining the input-output mapping of the actor allows interpretation of the learned strategy, a limitation of this approach is that only the "what" is explained, and not the "why". Control expert knowledge is required to interpret why the control strategy is employed and to assess if this is desired behavior. Interpreting the interaction between the incremental model, critic and actor could potentially lead to insights into the reason for the values of the actor parameters, and therefore explain the "why". Another limitation of the presented research is the lack of human-in-the-loop experiments with control experts to assess the quality and level of detail of the generated explanations. Future research should therefore include control experts for assessment of the explanation quality and usefulness during RL algorithm design. Finally, another limitation of the presented approach is that the explanation methodology is only applied to one adaptive RL framework and for one flight application. Further research should therefore test the flexibility of SHAP and CWSD by applying these techniques on other adaptive RL frameworks and different control tasks.

VI. Conclusions

This work has explored how the transparency of RL techniques for adaptive flight control can be improved using the existing XAI technique SHAP. To facilitate XAI techniques requiring constant weights, an algorithm was developed to select time segments with near-constant model parameters, under the assumption the the learned control strategy does not change within these segments.

The constant weight segment detection algorithm successfully detects regions where the learned strategy is near-constant, while still significantly generalizing the control behavior. It was shown for the IDHP controller that SHAP can accurately explain the model, even with minimal background samples. Therefore. assuming that minimum segment length is not a limitation for SHAP, the S_{min} parameter can be mainly used for generalization of the agent's dynamics. Namely, short segments will result in high sample coverage, but also many different segments to analyze and

explain. Using the CWSD parameters, RL developers can trade-off accuracy and generalization according to their needs. Furthermore, the usefulness of CWSD is not limited to flight control, as it can be used for any adaptive DRL technique.

Based on the selected segments, SHAP provides useful and detailed insights into the input-output mapping of the actor. Detailed explanations can be created using the local interpretability waterfall plot, to explain specific complex situations. Furthermore, the SHAP summary plot can be used to easily identify feature importance and the global relation between feature value and SHAP value. To quantitatively investigate this relationship, and compare the learned behavior of distinct segments, the proposed combined dependence plot can be used.

Control experts can use SHAP for multiple purposes during control design. First of all, the feature importance provided by SHAP can be used to select the most relevant and critical features, and therefore minimize algorithm complexity. This could prove particularly useful for small UAV design, where the mass and computational budget is limited. Furthermore, SHAP can be used to accurately identify the contribution of algorithm inputs on the final output. The linear relations discovered in this research could for example be used to design simple proportional controllers if the computational budget is limited.

Further research into explainable RL for flight control using SHAP should investigate two main topics. First of all, the quality of the explanations should be assessed using human-in-the-loop experiments, involving RL control experts. Secondly, the applicability to other RL frameworks and more complex control tasks should be investigated. The application chosen in this research resulted in simple linear explanations, likely due to the relatively simple rate control task. It would be interesting to see whether nonlinear relations can also be discovered using the proposed methodology. Another interesting research topic is using SHAP to illustrate the adaptive properties and fault-tolerance of RL techniques for flight control.

References

[1] Gupta, A., Afrin, T., Scully, E., and Yodo, N., "Advances of UAVs toward Future Transportation: The State-of-the-Art, Challenges, and Opportunities," *Future Transportation*, Vol. 1, 2021, pp. 326–350. https://doi.org/10.3390/futuretransp1020019.

[2] Litman, T., "Autonomous vehicle implementation predictions: Implications for transport planning," *Victoria Transport Policy Institute*, 2020.

[3] Rao, Q., and Frtunikj, J., "Deep Learning for Self-Driving Cars: Chances and Challenges," *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, Association for Computing Machinery, New York, NY, USA, 2018, p. 35–38. https://doi.org/10.1145/3194085.3194087.

[4] Santoso, F., Garratt, M., and Anavatti, S., "State-of-the-Art Intelligent Flight Control Systems in Unmanned Aerial Vehicles," *IEEE Transactions on Automation Science and Engineering*, Vol. 15, No. 2, 2018, pp. 613–627. https://doi.org/10.1109/TASE.2017.2651109.

[5] Vesely, V., and Ilka, A., "Gain-scheduled PID controller design," *Journal of Process Control*, Vol. 23, 2013, pp. 1141–1148. https://doi.org/10.1016/j.jprocont.2013.07.002.

[6] Singh, S., and Padhi, R., "Automatic path planning and control design for autonomous landing of UAVs using dynamic inversion," *Proceedings of the American Control Conference*, 2009, pp. 2409–2414. https://doi.org/10.1109/ACC.2009.5160444.

[7] Garcia, G., and Keshmiri, S., "Adaptive and resilient flight control system for a small unmanned aerial system," *International Journal of Aerospace Engineering*, 2013. https://doi.org/10.1155/2013/289357.

[8] Khan, S. G., Herrmann, G., Lewis, F. L., Pipe, T., and Melhuish, C., "Reinforcement learning and optimal adaptive control: An overview and implementation examples," *Annual Reviews in Control*, Vol. 36, No. 1, 2012, pp. 42–59. https://doi.org/https://doi.org/10.1016/j.arcontrol.2012.03.004.

[9] Williams, J., "Reinforcement learning of optimal controls," *Artificial Intelligence Methods in the Environmental Sciences*, 2009, pp. 297–327. https://doi.org/10.1007/978-1-4020-9119-3_15.

[10] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A., "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, Vol. 34, No. 6, 2017, p. 26–38. https://doi.org/10.1109/msp.2017.2743240.

[11] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533. https://doi.org/10.1038/nature14236.

[12] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Van Den 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., and Hassabis, D., "Mastering the game of Go with deep neural networks and tree search," *Nature*, Vol. 529, No. 7587, 2016, pp. 484–489. https://doi.org/10.1038/nature16961.

[13] Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, 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.

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

[15] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D., "A Survey of Methods for Explaining Black Box Models," *ACM Computing Surveys*, Vol. 51, No. 5, 2018. https://doi.org/10.1145/3236009.

[16] Gunning, D., and Aha, D., "DARPA's explainable artificial intelligence program," *AI Magazine*, Vol. 40, No. 2, 2019, pp. 44–58. https://doi.org/10.1609/aimag.v40i2.2850.

[17] Bayerlein, H., De Kerret, P., and Gesbert, D., "Trajectory Optimization for Autonomous Flying Base Station via Reinforcement Learning," *IEEE Workshop on Signal Processing Advances in Wireless Communications, SPAWC*, Vol. 2018-June, 2018. https://doi.org/10.1109/SPAWC.2018.8445768.

[18] Furfaro, R., and Linares, R., "Waypoint-Based generalized ZEM/ZEV feedback guidance for planetary landing via a reinforcement learning approach," 2017, pp. 401–416.

[19] Miller, D., and Linares, R., "Low-thrust optimal control via reinforcement learning," *Advances in the Astronautical Sciences*, Vol. 168, 2019, pp. 1817–1834.

[20] Li, Y., Eslamiat, H., Wang, N., Zhao, Z., Sanyal, A., and Qiu, Q., "Autonomous waypoints planning and trajectory generation for multi-rotor UAVs," *DESTION 2019 - Proceedings of the Workshop on Design Automation for CPS and IoT*, 2019, pp. 31–40. https://doi.org/10.1145/3313151.3313163.

[21] Zhao, Y., Zheng, Z., Zhang, X., and Liu, Y., "Q learning algorithm based UAV path learning and obstacle avoidence approach," *Chinese Control Conference, CCC*, 2017, pp. 3397–3402. https://doi.org/10.23919/ChiCC.2017.8027884.

[22] Ma, Z., Wang, C., Niu, Y., Wang, X., and Shen, L., "A saliency-based reinforcement learning approach for a UAV to avoid flying obstacles," *Robotics and Autonomous Systems*, Vol. 100, 2018, pp. 108–118. https://doi.org/10.1016/j.robot.2017.10.009.

[23] Fei, F., Tu, Z., Zhang, J., and Deng, X., "Learning extreme hummingbird maneuvers on flapping wing robots," *Proceedings - IEEE International Conference on Robotics and Automation*, Vol. 2019-May, 2019, pp. 109–115. https://doi.org/10.1109/ICRA.2019.8794100.

[24] Koch, W., Mancuso, R., West, R., and Bestavros, A., "Reinforcement learning for UAV attitude control," *ACM Transactions on Cyber-Physical Systems*, Vol. 3, No. 2, 2019. https://doi.org/10.1145/3301273.

[25] Scorsoglio, A., Furfaro, R., Linares, R., and Gaudet, B., "Image-based deep reinforcement learning for autonomous lunar landing," *AIAA Scitech 2020 Forum*, Vol. 1 PartF, 2020. https://doi.org/10.2514/6.2020-1910.

[26] Zhou, Y., van Kampen, E.-J., and Chu, Q., "Incremental model based online dual heuristic programming for nonlinear adaptive control," *Control Engineering Practice*, Vol. 73, 2018, pp. 13–25. https://doi.org/10.1016/j.conengprac.2017.12.011.

[27] Lee, S., and Bang, H., "Automatic Gain Tuning Method of a Quad-Rotor Geometric Attitude Controller Using A3C," *International Journal of Aeronautical and Space Sciences*, Vol. 21, No. 2, 2020, pp. 469–478. https://doi.org/10.1007/s42405-019-00233-x.

[28] Goedhart, M., Van Kampeny, E.-J., Armaniniz, S., De Visser, C., and Chu, Q., "Machine learning for flapping wing flight control," *AIAA Infotech at Aerospace*, , No. 209989, 2018. https://doi.org/10.2514/6.2018-2135.

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

[30] Dwivedi, Y., Hughes, L., Ismagilova, E., Aarts, G., Coombs, C., Crick, T., Duan, Y., Dwivedi, R., Edwards, J., Eirug, A., Galanos, V., Ilavarasan, P., Janssen, M., Jones, P., Kar, A., Kizgin, H., Kronemann, B., Lal, B., Lucini, B., Medaglia, R., Le Meunier-FitzHugh, K., Le Meunier-FitzHugh, L., Misra, S., Mogaji, E., Sharma, S., Singh, J., Raghavan, V., Raman, R., Rana, N., Samothrakis, S., Spencer, J., Tamilmani, K., Tubadji, A., Walton, P., and Williams, M., "Artificial Intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy," *International Journal of Information Management*, Vol. 57, 2021. https://doi.org/10.1016/j.ijinfomgt.2019.08.002.

[31] Prokhorov, D., and Wunsch II, D., "Adaptive critic designs," *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, 1997, pp. 997–1007. https://doi.org/10.1109/72.623201.

[32] Prokhorov, D. V., Santiago, R. A., and Wunsch, D. C., "Adaptive critic designs: A case study for neurocontrol," *Neural Networks*, Vol. 8, No. 9, 1995, pp. 1367–1372. https://doi.org/https://doi.org/10.1016/0893-6080(95)00042-9.

[33] Ferrari, S., and Stengel, R., "Online adaptive critic flight control," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 777–786. https://doi.org/10.2514/1.12597.

[34] Van Kampen, E., Chu, Q., and Mulder, J., "Continuous adaptive critic flight control aided with approximated plant dynamics," *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2006*, Vol. 5, 2006, pp. 2989–3016. https://doi.org/10.2514/6.2006-6429.

[35] Heyer, S., Kroezen, D., and van Kampen, E., "Online adaptive incremental reinforcement learning flight control for a cs-25 class aircraft," *AIAA Scitech 2020 Forum*, Vol. 1 Part F, 2020. https://doi.org/10.2514/6.2020-1844.

[36] Sieberling, S., Chu, Q., and Mulder, J., "Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 6, 2010, pp. 1732–1742. https://doi.org/10.2514/1.49978.

[37] Haykin, S. S., *Adaptive filter theory*, Pearson Education India, 2008.

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

[39] Juozapaitis, Z., Koul, A., Fern, A., Erwig, M., and Doshi-Velez, F., "Explainable reinforcement learning via reward decomposition," *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*, 2019.

[40] Bautista-Montesano, R., Bustamante-Bello, R., and Ramirez-Mendoza, R. A., "Explainable navigation system using fuzzy reinforcement learning," *International Journal on Interactive Design and Manufacturing*, Vol. 14, No. 4, 2020, pp. 1411–1428. https://doi.org/10.1007/s12008-020-00717-1.

[41] Selvaraju, R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D., "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," *Proceedings of the IEEE International Conference on Computer Vision*, Vol. 2017-October, 2017, pp. 618–626. https://doi.org/10.1109/ICCV.2017.74.

[42] Ribeiro, M., Singh, S., and Guestrin, C., ""Why should i trust you?" Explaining the predictions of any classifier," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Vol. 13-17-August-2016, 2016, pp. 1135–1144. https://doi.org/10.1145/2939672.2939778.

[43] Lundberg, S. M., and Lee, S.-I., "A Unified Approach to Interpreting Model Predictions," *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Curran Associates, Inc., 2017.

[44] Liessner, R., Dohmen, J., and Wiering, M., "Explainable reinforcement learning for longitudinal control," *ICAART 2021 - Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, Vol. 2, 2021, pp. 874–881.

[45] Chen, H., Lundberg, S., and Lee, S.-I., "Explaining Models by Propagating Shapley Values of Local Components," *Studies in Computational Intelligence*, Vol. 914, 2021, pp. 261–270. https://doi.org/10.1007/978-3-030-53352-6_24.

[46] Shrikumar, A., Greenside, P., and Kundaje, A., "Learning important features through propagating activation differences," *34th International Conference on Machine Learning, ICML 2017*, Vol. 7, 2017, pp. 4844–4866.

II

Literature Review and Preliminary Analysis

2 | Reinforcement Learning Fundamentals

Reinforcement learning is one of the three traditionally described forms of machine learning problems, the other two being *supervised learning* and *unsupervised learning* [21]. Where supervised learning focuses on classification and regression of data, learned from a set of labeled examples supplied by an external supervisor, unsupervised learning is about discovering regularities or patterns in unlabeled data. Reinforcement learning differentiates itself from these two types of machine learning by making decisions, learned from *experience* [9]. By *interaction* with the environment, the learner discovers which action is the most preferable to take, given a certain circumstance, to maximize its *reward* signal over time.

As the optimal action is not specified by an external supervisor, a learner applying RL can find itself adapting to new and unfamiliar circumstances, distinguishing RL from supervised learning. Furthermore, as the goal in RL is to maximize a reward signal over time as opposed to identifying patterns, RL also distinguishes itself from unsupervised learning. With this reward signal maximization, a learner applying RL might choose not to pursue a short-term small reward but focus on a larger reward in the future. The competence to adapt to changing circumstances and the ability to make a trade-off between near and future rewards are the two most important distinguishing features of RL [9]. Furthermore, as the agent adjusts its behavior based on its experience through the reward signal, a well-designed reward system is essential to achieve the desired results.

This chapter introduces the basic principles of reinforcement learning, by first explaining its key concepts in Section 2.1, followed by an a division of RL methods into dynamic programming methods introduced in Section 2.2 and model-free methods introduced in Section 2.3. Then Section 2.4 introduces the field of approximate RL, including the basics of neural networks for function approximation. Finally, Section 2.5 introduces actor-critic algorithms, used in many state-of-the-art RL applications.

2.1. Key Concepts

The entity in RL responsible for the decision-making and learning is defined as the *agent*. In a sequential decision-making process, the agent interacts with the *environment*, which comprises everything outside of the agent. This process is illustrated schematically in Figure 2.1.

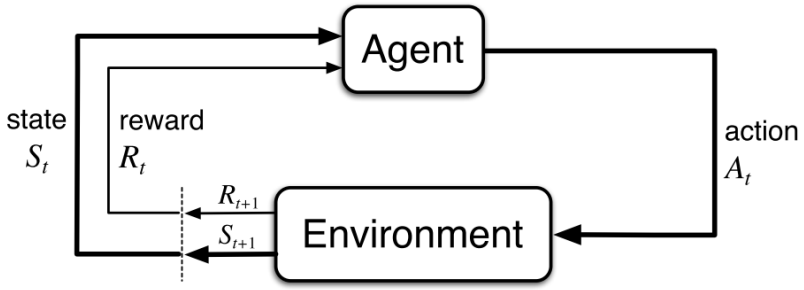


Figure 2.1: Interaction between the agent and its environment, displayed as a Markov Decision Process [9].

2.1.1. Markov Decision Process

The interaction process is often specified in discrete-time where $t = 0, 1, 2, ...$, but the principles can be extended to continuous time as well [9][22]. For the discrete-time scenario, the sequence of decision-making for RL can be formalized using a Markov Decision Process. The interaction process consists of two repeating steps: the agent chooses an *action* $A_t \in \mathcal{A} \subset \mathbb{R}^m$ to perform based on the observed state of the environment $S_t \in \mathcal{S} \subset \mathbb{R}^n$. Due to this action and possibly other dynamics the environment changes to another state S_{t+1},

and the agent receives a corresponding scalar *reward* $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. The notation for the reward corresponding to action A_t differs between R_t and R_{t+1} in literature. In this report R_{t+1} will be used as this reward is received at the same time step as the environment transitions to S_{t+1}. The sequence or *trajectory* of states, actions, and rewards is typical for RL processes and can be noted as $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 ...$

When the decision-making process is described as a *finite* MDP, the state, action, and reward spaces $\mathcal{S}, \mathcal{A}, \& \mathcal{R}$ are limited. This causes the random variables S_t and R_t to have a well-defined discrete probability distribution, based on exclusively the previous state and action pair. The dynamics of the finite MDP for all $s' \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}$ can then be described using Equation (2.1.1).

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2.1.1}$$

One of the characteristics of a MDP is the *Markov property*, which is also reflected in Equation (2.1.1). This property states that the Markov process is fully characterized in each time step, by completely capturing all relevant information, which might influence the future, in every state. This property is reflected in Equation (2.1.1) as the probabilities for each S_t and R_t are exclusively dependent on the preceding state and action S_{t-1} and A_{t-1}. In other words, for Markov decision processes knowing the state and action at a certain time step should be enough information to compute the probabilities for all possible states and rewards in the next time step.

2.1.2. RETURN

A RL agent learns and adjusts its behavior to maximize the reward signal over time. Whenever an agent only pursues the immediate reward, the agent is considered *myopic*. More often, not only immediate rewards are pursued, but the agent tries to maximize the expected value of the cumulative sum of the reward signal. In a more formal way, the agent tries to maximize the *expected return*, where the return G_t is some function of the accumulation of rewards as shown in Equation (2.1.2).

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.1.2}$$

For continuous processes however, where the assumption of $T = \infty$ is made, the return could become infinite as well. To solve this problem, the return as shown in Equation (2.1.2) is *discounted* using a *discount factor* $0 \leq \gamma \leq 1$ as shown in Equation (2.1.3). The discount factor determines how far-sighted the agent is, and is used when tuning RL algorithms to balance immediate and future rewards.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1.3}$$

The relation between returns at subsequent time steps shows an important characteristic of RL problems, as illustrated in Equation (2.1.4). This characteristic makes it possible to easily determine the return throughout the decision making process, based upon the sequence of rewards.

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{2.1.4}$$

2.1.3. POLICY AND VALUE

As mentioned before, the agent tries to maximize the expected return by, at every time step, selecting an action most likely to lead to an optimal trajectory. The probability distribution for choosing actions, based upon the state is defined as the *policy* denoted as π. If an agent follows this policy at time step t, then $\pi(a|s)$ is the chance of choosing action A_t given state S_t. By updating the policy, the agent will show different behavior. The agent's goal is to create an optimal policy, which when followed will cause the highest return.

To formalize this optimization process, the *value function* is defined. This function $v_\pi(s)$ specifies the expected return when in s, while following policy π. The formal definition of the value function is shown in Equation (2.1.5). Likewise, an *action-value* function can be given to a state s, when choosing an action a, and afterwards following policy π as shown in Equation (2.1.6).

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S} \tag{2.1.5}$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \tag{2.1.6}$$

In the same fashion as the return, the relation between the value functions at state s and its successive states can be derived using the recursive relationship as shown in Equation (2.1.7). This recursive property is used by many RL methods.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \end{aligned} \tag{2.1.7}$$

When the state and action spaces are discrete, the *Bellman optimality equation* can be deduced from Equation (2.1.7) by rewriting the expectation of the reward as a multiplication of the policy and the MDP dynamics, as shown in Equation (2.1.8). An example of a *backup diagram*, a schematic drawing exemplifying a RL process, corresponding to a non-deterministic MDP is shown in Figure 2.2. The value function for s, as calculated with Equation (2.1.7), is the expected return starting from the open node s at the top.

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right] \quad \text{for all } s \in \mathcal{S} \end{aligned} \tag{2.1.8}$$

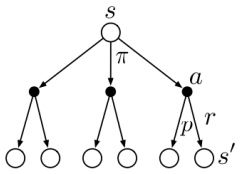


Figure 2.2: Example backup diagram for a non-deterministic MDP [9].

2.1.4. OPTIMAL POLICY

If the value functions are perfectly known and accurate for a decision-making process, the optimal policy is easily found by choosing the action sequence leading to the highest return. Determining the value functions is not always straightforward though, especially when the process is stochastic, when the state-action space is large or when the system dynamics are unknown. The value function and the action-value function can be determined either using knowledge of the model, or estimated from experience.

Policies can be compared, and hence a policy can be upgraded, by evaluation of their corresponding value functions for every state. A policy π is better than another policy π' when its expected return is higher for all states. Hence $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. For every RL problem, there is always one or more policies which are using the mathematical statement above, better than or equal to all other policies. This policy is called the *optimal policy*. There is always at least one optimal policy, but it is possible there are multiple policies for which this optimality holds. An optimal policy is denoted by π_*, and all optimal policies have the same state-value function, the *optimal state-value function* v_*, as shown in Equation (2.1.9).

$$v_*(s) \doteq \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S} \tag{2.1.9}$$

Likewise, the *optimal action-value function* is also shared by optimal policies, as shown in Equation (2.1.10).

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \tag{2.1.10}$$

2.2. DYNAMIC PROGRAMMING

As mentioned in Section 2.1, in the special case when the dynamics of an MDP are completely known, the state-value function can be determined without any agent-environment interaction experience. To find the optimal policy of such problems, *Dynamic Programming* (DP) can be applied. The collection of methods depending on a model of the MDP are known as *model-based* approaches. As this collection of methods, known as DP, requires a perfect model of the environment and has a large computational complexity, its use is often limited for practical applications. However, as the methods required for solving DP problems are often not complex, DP methods are a good starting point for showing the essentials of solving RL problems. An example of this is *Generalized Policy Iteration* (GPI), a method often applied for DP problems, but also occurring in other RL approaches shown in the subsequent sections. In this section, finite MDPs are assumed, but the ideas can be extrapolated to continuous problems as well.

2.2.1. POLICY EVALUATION

The essence of DP and other RL methods is to structure the search towards an optimal policy π_* using value functions. In the case of a finite MDP, where the dynamics are completely known, the Bellman optimality equation as shown in Equation (2.1.8) can be written as a system of linear equations. Through iteration, these equations are approximated. For this iteration, a recursive equation of the state-value function is required, which is shown in Equation (2.2.1). This equation is used in *policy iteration algorithms*, where the approximated value function becomes the true value function when the amount of iterations goes to ∞. This algorithm can be used to extract a state-value function from a given policy. The value-iteration algorithm is shown in Algorithm 2.1. Note that in this algorithm a *threshold* θ is inserted, to prevent the algorithm from running infinitely long.

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_k\left(s'\right) \right] \end{aligned} \tag{2.2.1}$$

Algorithm 2.1: Iterative Policy Evaluation [9]

input : State space $\mathcal{S}$, action space $\mathcal{A}$, state transition dynamics p, reward function r, discount rate γ, iteration threshold θ
output: Near-optimal state value function $V_*(s)$
initialize $V_0(s)$ arbitrarily for all $s \in \mathcal{S}$, except V($terminal$)=0 ;
while $\Delta \geq \theta$ **do**
    $\Delta \leftarrow 0$;
    for $s \in \mathcal{S}$ **do**
        $V_{j+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_j(s') \right]$;
        $\Delta \leftarrow \max(\Delta, \left| V_{j+1}(s) - V_j(s) \right|)$;
    end
end

2.2.2. POLICY IMPROVEMENT

Just like a value function can be created based on a policy, a policy can be generated based on a value function. By selecting the action leading to the highest expected return, a policy can be improved, a process known as *policy improvement*. This leads to a *greedy policy* π', as shown in Equation (2.2.2).

$$\begin{aligned} \pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a \right] \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right] \end{aligned} \tag{2.2.2}$$

2.2.3. GENERALIZED POLICY ITERATION

If the two procedures mentioned above, policy evaluation and policy improvement, are successively repeated after one another, the policy will continue to improve until it converges to the optimal policy. This process is known as *Generalized Policy Iteration* (GPI), and it is applied in many RL algorithms. The optimal policy has been found when the policy is greedy with respect to its own value function found using policy evaluation. This process is shown schematically in Figure 2.3. The pseudocode for dynamic programming policy iteration is shown in algorithm 2.2. As can be seen in this algorithm, the method requires a complete sweep through the state space $\mathcal{S}$ and the action space $\mathcal{A}$ for every iteration. This is feasible for small state-action spaces, but for larger problems DP suffers from the *curse of dimensionality*, where the number of computations grows *exponentially* with the number of state variables. This curse of dimensionality renders DP virtually unusable for some problems.

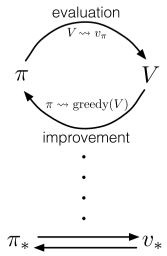


Figure 2.3: Example backup diagram for a non-deterministic MDP [9].

Algorithm 2.2: Policy Iteration Algorithm for Estimating $\pi \approx \pi_*$ [9]

input : State space $\mathcal{S}$, action space $\mathcal{A}$, state transition dynamics p, reward function r, discount rate γ, iteration threshold θ

output: Near-optimal state value function $V_*(s)$ and near-optimal policy π_*

initialize $V_0(s)$ and π_0(s) arbitrarily for all $s \in \mathcal{S}$;

repeat

  repeat

    $\Delta \leftarrow 0$;

    for $s \in \mathcal{S}$ **do**

      $V_{k+1}(s) \leftarrow \sum_a \pi_j(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_k(s') \right]$

      $\Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|)$

    end

  until $\Delta \geq \theta$;

  $\pi_{j+1} \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_{\pi_j}(s') \right]$

  j $\leftarrow j + 1$**until** $\pi_{j+1}(s) = \pi_j(s)$ *for all* $s \in \mathcal{S}$;

2.3. MODEL-FREE ALGORITHMS

In many RL problems complete knowledge of the underlying MDP is unavailable. For these situations the dynamics of the transition probability distribution can be estimated using *experience* from agent-environment interaction. This estimation becomes more accurate as every state-action pair is visited more times, by averaging their value functions. The used experience can either be *actual* or *simulated* experience. One subset of model-free algorithms is the collection of *Monte Carlo* (MC) methods, which uses experience from separate *episodes* to average the returns from the same state-action pair to estimate its value function. An episode is defined as the interaction between the starting and end state. MC algorithms are explained in more detail in Section 2.3.1. Another subset of methods is *Temporal-Difference Learning* (TD), which uses subsequent samples to estimate the value function, as will be explained in more detail in Section 2.3.2. MC methods and TD are two non-exhaustive subsets of model-free algorithms. There are also methods in-between MC and TD, where longer sequences of samples are used to estimate the value function, such as *n-step Bootstrapping* methods.

2.3.1. MONTE CARLO METHODS

Monte Carlo (MC) methods learn the state-value function for a given policy, by applying this policy during episodes and saving the samples generated from the agent-environment interaction. With the knowledge of the return signal during the episode, the return can be determined at every time step, giving insight into the value function for the encountered state-action pairs. Due to this return calculation method, MC methods are only suitable for *episodic* tasks. With each episode, return values for state-action pairs are generated. Thanks to the *law of large numbers*, simulating sufficient episodes allows the value functions to be estimated accurately, by averaging the encountered return values for every state-action pair. The same method can be applied to state-value functions, rather than state-action pairs. For the sample generation, a certain policy is followed. This policy can then be upgraded by analysis of the value functions as estimated with the samples. This sequence policy evaluation and policy improvement is another example of *generalized policy generation* as described for the DP methods.

A distinction can be between *first-visit* MC methods, where v_π is estimated by averaging the returns from every *first visit* to a state-action pair (s,a), and *every-visit* MC methods where the returns of every visit to the state-action pair are included in the calculation. Another distinction that can be made between MC methods, but also for other RL algorithms like TD, is which policy is being evaluated and improved. *On-policy* methods evaluate and improve the policy which is being used to generate the samples, while *off-policy* methods evaluate and improve a policy separate from the one producing the agent-environment experience. Off-policy methods can therefore also use experience from other sources, such as human experts, to evaluate and optimize value functions. Furthermore MC methods, as opposed to DP methods, do not use *bootstrapping*. Bootstrapping is using estimated values in the update step for the same variable, which can be seen for DP in Equation (2.2.1).

Due to the reliance on generated data rather than knowledge of the model, MC methods are the first collection of methods who suffer from the *exploration-exploitation problem*. This problem refers to the complex balance between *exploration* of unknown or rarely-visited state-action pairs, and *exploitation* of gained experience to generate a high return using the currently optimal policy. Due to the lack of experience, every untrained agent starts of with exploration.

The balance between exploration and exploitation is problem-specific, and currently no golden standard to solve this problem exists.

2.3.2. TEMPORAL-DIFFERENCE LEARNING

Like MC methods, *Temporal-Difference Learning* (TD) methods learn from experience rather than knowledge of the model's dynamics. However, where MC methods analyze the agent-environment experience per episode, *Temporal-Difference Learning* use sequential data points and bootstrapping to estimate the value functions. As these sequential samples can be any experienced transitions, TD methods can also be used for *continuous* tasks, rather than only episodic tasks. Furthermore, as TD methods do not have to wait until the end of an episode before learning takes place, learning rates can be higher for tasks with long episodes. The simplest example of TD learning is *one-step TD*, or $TD(0)$, which uses two consecutive samples for the update rule as shown in Equation (2.3.1).

$$V_{j+1}(S_t) \leftarrow V_j(S_t) + \alpha \left[R_{t+1} + \gamma V_j(S_{t+1}) - V_j(S_t) \right] \tag{2.3.1}$$

From this Equation, it can be observed that TD learning make use of bootstrapping just like DP methods, as the update rule for $V(S_t)$ makes use of the estimates of this same variable. Hence, TD learning combines the bootstrapping of DP methods with the *sample updates* of MC methods. The part between brackets in Equation (2.3.1), isolated for clarification in Equation (2.3.2), is called the *TD error*, an important concept in RL:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.3.2}$$

By setting the *learning rate* parameter α in Equation (2.3.1) it can be decided how aggressively the estimate of $V(S_t)$ can be updated with every time step. The learning rate is one of the tools that can be used for setting the balance between speed of learning and stability. This stability is an important issue for TD learning due to

the use of bootstrapping. However, using a small enough α, and using *batch updating* where all the training experience is presented repeatedly, often allows $TD(0)$ methods to converge. Convergence is even proven deterministically when all state-action pairs are visited an infinite number of times [9].

2.4. APPROXIMATE REINFORCEMENT LEARNING

The classical RL algorithms mentioned above are tabular methods, requiring exact representations of the value functions and/or policies. However, for very large state spaces $\mathcal{S}$ or action spaces $\mathcal{A}$, this exact representation becomes problematic. When $\mathcal{S}$ consists of n states, computational operations such as n-dimensional inner products can become prohibitively time-consuming or even impossible when the n-vector cannot be stored in the computer memory [23]. Not only is the necessary time and memory for this representation troublesome, also the data required to fill the tables becomes problematic for large $\mathcal{S}$ and $\mathcal{A}$ [9]. The chance of visiting a state-action pair once or multiple times reduces when the state-action space grows, causing biased or even missing estimates for a large portion of the state-action pairs. As an example, consider a simple discrete system with 10 binary variables, resulting in 2^{10} state combinations. Now if these states can have 10 possible values, there are already 10^{10} state combinations. Considering that continuous variables can have more than 10 possible values, continuous problems require function approximators.

To make well-considered choices in the whole state-action space based upon experience in a limited part of the state-action space requires *generalization* from previous encounters [9]. Generalization from examples is a common study field, allowing integration of existing knowledge into the field of RL. *Function approximation* is a very suitable form of generalization to be used for large state-action spaces. This method allows a function to be approximated by generalization by sampling the function in only a part of the domain. As a consequence of this generalization, an update originating from a single state visit influences the approximated value of many states. This generalization makes the learning more powerful, but often makes the policy less *explainable* [9]. Providing more insight into the function approximation in RL is one of the key goals of this work.

Applying function approximation to RL problems leads to *approximate reinforcement learning* (ARL). A common classification of approximators is the use of parameters for the function approximation. *Nonparametric* approximators are very flexible, but they become less computationally efficient when the amount of data grows [24]. In particular, for online RL algorithms, the growing amount of data is troublesome for nonparametric approximators. On the other hand, *parametric* approximators are more common in the field of ARL. These approximators use a set of tuneable parameters to approximate a function. As an example of a parametric approximator, a Q-function approximated with parameter vector $\mathbf{w} \in \mathbb{R}^n$ is denoted by an *approximation mapping* F: $\mathbb{R}^n \rightarrow \mathcal{Q}$, in which $\mathbb{R}^n$ is the parameter space, while $\mathcal{Q}$ is the space of the *approximated* Q-function [24]. Adjusting the parameter vector $\mathbf{w}$ results in different approximations of the Q-function as displayed in Equation (2.4.1).

$$\hat{Q} = F(\mathbf{w}) \tag{2.4.1}$$

Parametrized approximations are often classified as *linear* or *non-linear*. An example of a linearly parametrized approximation, which is linear in the parameters, is shown for the action-value function. This function is approximated using n basis functions $\phi_1, \phi_2, ..., \phi_n$, and a parameter vector $\mathbf{w} \in \mathbb{R}^n$, as shown in Equation (2.4.2) [24]. $\phi(x,u)$ in this Equation is the vector of basis functions $\left[\phi_1(x,u), \phi_2(x,u), ..., \phi_n(x,u)\right]$. The basis functions are also known as *features*, as they generalize a function based upon smaller features.

$$\hat{Q}(s,a;\mathbf{w}) = \sum_{i=1}^{n} \phi_i(s,a)\mathbf{w}_i = \phi^T(s,a)\mathbf{w} \tag{2.4.2}$$

This is an example of a linearly parametrized approximation, but the approximator can also be non-linear such as a multilayer *artificial neural networks* (NN). The field of ARL applying NNs for function approximation is known as *Deep Learning* (DL), which is currently among the most research-intensive domains of RL [12]. Deep learning is an often preferred approach for RL problems, thanks to the flexibility and high representation power of NNs. Nonlinear parametrized approximators often have a higher approximation accuracy and are hence better at representation of the function, but cause the performance of the approximate RL algorithm to be more difficult to analyze. Furthermore, linear methods can be very efficient concerning the

amount of computation and data, but they require specific domain knowledge to be effective [9]. The chosen linear approximator requires specific features of the function which is to be approximated for the algorithm to converge.

To evaluate an approximation of a function, for example the value function, a numeric assessment of the approximation compared with the actual function is required. For this the *Mean Squared Value Error* can be used, as presented in Equation (2.4.3). $\mu(s)$ is often defined as the fraction of time spent in s compared to the other states in $\mathcal{S}$. By reducing the Mean Squared Value Error, the approximation is improved. An approximated value function with a minimized $\overline{\text{VE}}$ is however not necessarily the optimal value function, as this measure does not asses the optimality of the value function.

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2 \tag{2.4.3}$$

For updating the parameter vector $\mathbf{w}$ multiple methods exist, with *Gradient Descent* being among the most widely used methods. This parameter update method is used in common classes of ARL such as *policy gradient* [25] and is used as the foundation of neural network optimization methods such as *Adam* [26]. The general form of gradient descent methods is displayed in Equation (2.4.4) and will be explained further when RL algorithms are discussed. $J(\mathbf{w})$ in this equation is a cost function, which is minimized by updating the parameter vector $\mathbf{w}$.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \nabla J(\mathbf{w_t}) \tag{2.4.4}$$

Generally speaking, approximate RL methods approximate the value function, the policy, or both. Value-based approximation methods are addressed in more detail in Section 2.4.2, policy-based approximation methods in Section 2.4.3, and actor-critic methods, combining both value-based and policy-based approximation, in Section 2.5.

2.4.1. ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (NNs) are one of the most widely-used, nonlinear function approximators thanks to their accuracy and flexibility. Furthermore, advancements in computing power have enabled faster convergence of these function approximators. Using NNs for function approximation, little to no knowledge is required of the function which is approximated. The first steps to NNs were made when the logic of biological neuron firings were applied to electronical circuits [27]. Currently many types of NNs exist, ranging from simple single-layer perceptrons to more complex neural networks capable of analyzing time series such as *recurrent neural networks*. For ARL, *feedforward* (FF) neural networks are the most common function approximators. This subchapter will therefore focus on the working principles of feed forward neural networks.

A feedforward neural network maps its input $\mathbf{x}$ to an output $\mathbf{y}$ based on the network's parameters $\boldsymbol{\theta}$ as $\mathbf{y} = F(\mathbf{x}, \boldsymbol{\theta})$. This mapping is carried out by passing information through simple, connected processors defined as *neurons* [28]. As illustrated in Figure 2.4, these neurons are structured in at least three layers: the input layer, the output layer, and one or more hidden layers. In the case of more than one hidden layer, the NN is considered as a *deep* neural network. Inspired by biological neurons in the brain, each artificial neuron produces an output signal based on the input to the neuron. This input-output mapping of an individual neuron is defined as *activation*. Input neurons are activated based on the inputs of the neural network $\mathbf{x}$, while the other neurons are activated based on the output of the connected neurons in the preceding layer.

The activation of a neuron depends on the chosen *activation function*, which specifies the output of a neuron based on the input. Many variations of activation functions exist, ranging from simple binary activation where the neuron fires at a constant value once the threshold is reached, to more complex sigmoid functions. Two examples of common activation functions are illustrated in Figure 2.5, these are the *Rectified Linear Unit* (ReLU) and *hyperbolic tangent* (tanh) functions.

An example of the internal processing of a neuron, connected to three preceding neurons, is shown in Figure 2.6. The output of the preceding neurons are multiplied with the corresponding weights of the connections, summed together with a bias during *propagation*, and then passed through the activation function as

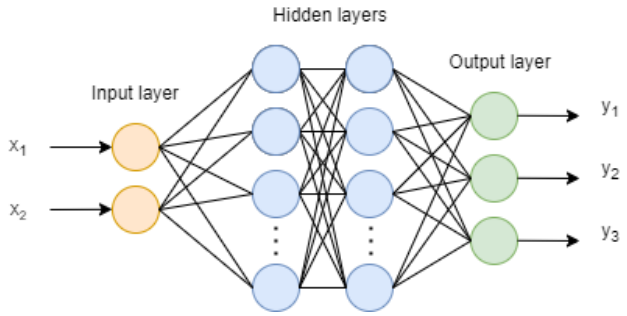


Figure 2.4: General structure of a feedforward neural network.

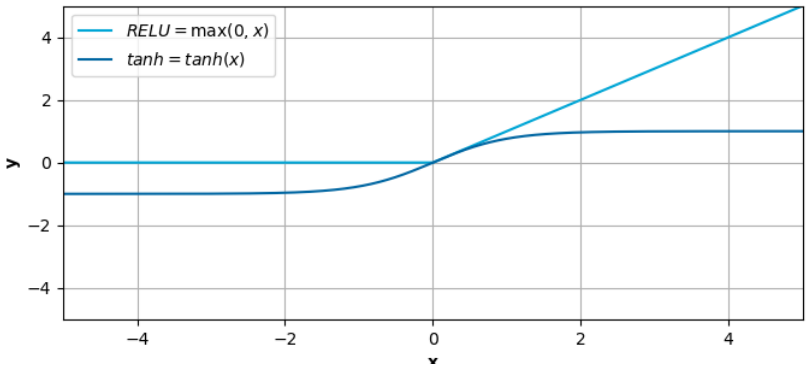


Figure 2.5: Two examples of activation functions, the Rectified Linear Unit and hyperbolic tangent.

shown in Equation (2.4.5). The biases of all neurons, and the weights of all connections in the NN together comprise the parameter vector $\boldsymbol{\theta}$ of the neural network, which is adjusted during training to cause the desired input-output mapping. The type of activation function can differ from layer to layer, but is not adjusted during training of the neural network.

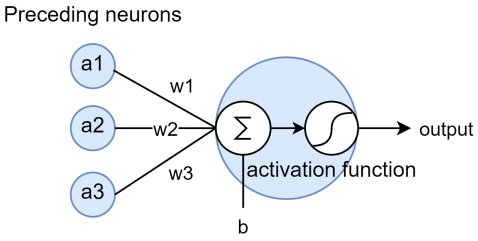


Figure 2.6: Internal processing of a single neuron connected with 3 other neurons.

$$n_{out} = F\left(b + \sum_{i=1}^{n} w_n \cdot a_n\right) \tag{2.4.5}$$

Through successive training of the neural network, where the NNs parameters $\boldsymbol{\theta}$ are adjusted, the neural network will aim to map the input to the desired output. This convergence is however not guaranteed, and can only happen when training is functioning properly. And even when training is set up properly, convergence is not guaranteed, depending on the initial NN parameters. One of the requirements for functional training is having a proper setting of *hyperparameters*. Furthermore, even when training is functional, there is no guarantee of the NN being 100% accurate of the desired value function or policy. In the sense of mapping input to desired output, training a neural network based on desired output is a form of *supervised learning*. Training of the neural network requires examples of input-output mapping, known as the *target* or *training*

data. During training, the prediction of the neural network is compared with the desired target data using a *loss function*, such as the *mean squared error* (MSE). Multiple methods exist for tuning the NNs parameters, of which most are based on the gradient descent as shown in Equation (2.4.4).

As each neuron has a bias and a weight for every connecting neuron, a complex modern NN requires tuning many parameters, as can be seen for the following example: consider an NN aiming to classify an image of 100 by 100 pixels. This results in $100 * 100$ inputs, and assuming there are 2 hidden layers of 128 neurons, this NN already requires tuning of 1.33 million weights and 258 biases. Efficient learning of complex NNs has been made possible through the use of *backpropagation*, which is currently used by the most popular update algorithms [28]. Through backpropagation, the gradients of the loss function with respect to the parameters $\boldsymbol{\theta}$ are calculated. The parameters can then be updated in the direction resulting in the largest loss reduction. The degree in which the parameters are updated in the direction of the gradients is determined by the *learning rate*. This is an important parameter for using NNs, and tuning this hyperparameter can be a time-consuming process, as low values result in slow learning, while a high learning rate may result in unstable behavior. Furthermore, a low learning rate might cause the NN to be optimized for a local minimum of the complex loss function, rather than the global minimum. This can be visualized using the analogy of a ball rolling over a field with small ditches representing the local minima. With a sufficiently high learning rate, the ball has sufficient momentum to roll over these local minima, potentially ending in the global minimum. When the learning rate would be low however, the ball would stop in a local minimum. State-of-the-art NN optimizers typically use a variable learning rate aim to combine the best of both worlds: stability and fast convergence to a global minimum.

2.4.2. VALUE-BASED APPROXIMATION

Value-based approximation methods approximate the optimal value function using agent-environment interaction data. One of the key methods in this class is *Deep Q Learning* (DQN), first introduced by [10]. This method applies classical Q-learning combined with a deep neural network, hence the adjective *deep*, for the value function approximation. This DQN algorithm in [10] is able to outperform humans for 49 Atari games. A popular successor of DQN is *Double Q-learning*, introduced by [29], reducing the problem of over-estimation from which DQN is suffering. Other approximators than NNs are also used for value-based approximation, such as the *Fourier basis* as presented in [30].

The gradient descent method introduced in Equation (2.4.4) can also be applied to classical value-based methods. These gradient-descent methods alter $\mathbf{w}$ in the direction that causes the most error reduction, as shown in Equation (2.4.6).

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t]^2 \\ &= \mathbf{w}_t + \alpha\,[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)]\,\nabla\hat{v}(S_t,\mathbf{w}_t)\end{aligned} \tag{2.4.6}$$

The true value function $v_\pi(S_t)$ in Equation (2.4.6) is generally unknown however, and must be replaced by an unbiased estimate of the value function U_t. In the case of *Gradient MC* methods for example, the value function is estimated using the return G_t, resulting in Equation (2.4.7). In the case of *Semi-gradient TD(0)* the return uses the Bellman equation, as shown in Equation (2.4.8).

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\,[G_t - \hat{v}(S_t,\mathbf{w}_t)]\,\nabla\hat{v}(S_t,\mathbf{w}_t) \tag{2.4.7}$$

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\left[R_t + \gamma\,\hat{v}_t(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}_t)\right]\nabla\hat{v}(S_t,\mathbf{w}_t) \tag{2.4.8}$$

The Q-learning algorithm, used to achieve super-human performance in the Atari games [10], is presented in Algorithm 2.3. The value-based approximation methods described here estimate the value function, but do not yet specify the optimal action. When $|\mathcal{A}|$ is small, finding the action maximizing $\hat{Q}$ is trivial, but this does not hold for large and continuous action spaces. Due to this curse of dimensionality for large action spaces, the mentioned algorithms are generally not suited for continuous action spaces. For applications with large or continuous action spaces, policy-based approximation methods are required.

Algorithm 2.3: Q-learning algorithm for estimating π_*, a value-based approximation technique [9]

input : learning rate α
output: Near-optimal value function $Q*$
initialize $Q(S,A)$ randomly;
for *each episode* **do**
  initialize episode, observe s_0;
  for *every time step* **do**
    sample A based upon policy from Q given S ;
    simulate next time step using A, observe R and S' ;
    update value function $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$;
    $S \leftarrow S'$
  end
end

2.4.3. POLICY-BASED APPROXIMATION

Policy-based approximation methods, such as REINFORCE [31], directly learn a policy for a RL problem, selecting actions without the use of a value function allowing these methods to cope with large or continuous action spaces [9]. Furthermore, a useful characteristic of policy-based approximation methods is their ability to cope with stochastic environments, thanks to the structure of these methods. Another advantage of policy-based methods is that some RL problems are easier to solve, when the value function would be hard to approximate, while the policy would be less complex. Methods applying policy approximation without value function estimation are known as *actor-only* methods. The methods described in Section 2.4.2, which only approximate the value function and then choose actions accordingly, are defined as *critic-only* methods. Whenever both the value function and the policy are approximated, the method is regarded as *actor-critic* [32]. Algorithms in the latter class are also known as *adaptive-critic designs* (ACDs) [33].

Some policy-based approximation methods are *non-parametric* [34], but generally the learned policy is *parametric*. Parametric policies are described using the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{d'}$. Many policy-based approximation methods are based on the idea of *policy-gradient* learning [35], which is derived from gradient-descent as shown in Equation (2.4.4). In policy-gradient methods, the policy is represented using a parametric and differentiable function [36], which is optimized for (locally) maximum returns using gradient updates. The gradient is the direction causing reduction in cost $J(\boldsymbol{\theta})$, where the cost is defined differently for episodic and continuous tasks. For episodic tasks the cost is defined as the value of the starting state s_0, shown in Equation (2.4.9). For continuing tasks, the average rate of reward per time step is used, shown in Equation (2.4.10).

$$J_1(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0) \tag{2.4.9}$$

$$J_{avR}(\boldsymbol{\theta}) \doteq \sum_s d_\pi(s) \sum_a \pi_{\boldsymbol{\theta}}(s,a) R_s^a \tag{2.4.10}$$

Using the *policy-gradient theorem*, presented in Equation (2.4.11), update rules for policy approximation methods can be constructed. The first algorithm applying gradient-descent for policy approximation was REINFORCE [37], which can be regarded as classical MC combined with gradient-descent used for policy approximation. For this algorithm, the gradient is defined as shown in Equation (2.4.12).

$$\nabla J(\boldsymbol{\theta}) \propto \sum_x \mu(\mathbf{x}) \sum_u q_\pi(\mathbf{x},\mathbf{u}) \nabla \pi(\mathbf{u}|\mathbf{x},\boldsymbol{\theta}) \tag{2.4.11}$$

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] \tag{2.4.12}$$

The expectation in Equation (2.4.12) is estimated in the REINFORCE method by sampling multiple episodes, similar to classical MC learning. Then using the gradient descent equation, previously presented in Equa-

tion (2.4.4), the parameters of the approximated policy can be changed using the update equation for REINFORCE, as shown in Equation (2.4.13). The REINFORCE algorithm is presented in Algorithm 2.4.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \tag{2.4.13}$$

Algorithm 2.4: REINFORCE algorithm for estimating π_*, a policy-based approximating technique using episodic data [9]

input : Policy approximation function $\pi($, learning rate α
output: Near-optimal policy π_*
initialize $\boldsymbol{\theta}$ randomly;
for *each episode* **do**
  Simulate episode $S_0, A_0, R_0, ..., S_{T-1}, A_{T-1}, R_T$ using $\pi(\cdot|\cdot,\boldsymbol{\theta})$;
  for $t = 0, 1, 2, ..., T-1$ **do**
    determine return $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
    update policy $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$
  end
end

While the update rule for REINFORCE shown in Equation (2.4.13) is simple to implement, the substantial variance of this algorithm makes it infeasible for most applications. One way of reducing this variance is by not only estimating the policy, but also the value function. This class of actor-critic algorithms will be introduced next in Section 2.5.

2.5. ACTOR-CRITIC ALGORITHMS

Actor-critic algorithms approximate both the policy and value function to combine the best of both worlds. Thanks to properties of policy gradient, actor-critic algorithms can provide continuous action output. The entity in actor-critic algorithms responsible for the action output is defined as the actor. The other entity, defined as the critic, evaluates the current policy of the actor by approximating the value function. Using this evaluation, the actor's approximation parameters can be adjusted in the direction causing the highest performance increase. This cooperation between the actor and critic significantly reduces the variance and increases sample-efficiency compared to actor-only methods [38]. The process of critic, actor, and environment interaction is illustrated in Section 2.5, where the dashed lines illustrate *parameter updates* caused by the critic.

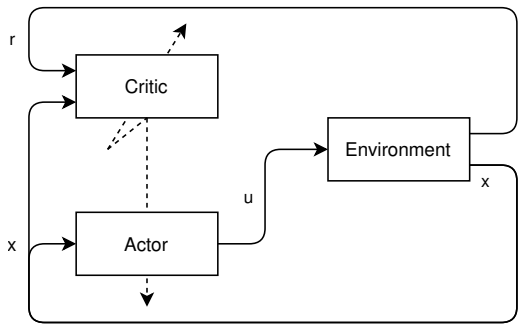


Figure 2.7: Actor-critic algorithm schematic, with the dashed lines illustrating the critic updating the parameters of the actor and critic itself.

The characteristics of an actor-critic algorithm are conveniently described by comparison with the REINFORCE algorithm as presented in Section 2.4.3. The policy for an actor-critic algorithm is updated using the gradient of the approximated value function rather than the gradient of the return. The approximated value function can be the action-value function such as in *Q actor-critic*, but other variations such as *TD actor-critic* exist. The comparison between these three algorithms is shown in Equation (2.5.1).

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(\mathbf{x},\mathbf{u}) G_t\right] && \text{for REINFORCE} \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(\mathbf{x},\mathbf{u}) Q^{\mathbf{w}}(\mathbf{x},\mathbf{u})\right] && \text{for Q actor-critic} \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(\mathbf{x},\mathbf{u}) \delta\right] && \text{for TD actor-critic}\end{aligned} \tag{2.5.1}$$

Equation (2.5.1) shows the core principles of 2 actor-critic algorithms, but many more variations of actor-critic algorithms exist. *Q actor-critic* is an example of actor-critic learning in its most basic form, which will therefore be used as an example to illustrate the general working principles of actor-critic algorithms.

Critic
The critic in *Q actor-critic* learning is updated using the TD-error, as presented in Equation (2.3.2) and repeated here for convenience:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.5.2}$$

As the true value function is now approximated with a parametric function, Equation (2.5.3) is used to calculate the TD-error for *Q actor-critic*.

$$\delta_t \doteq R_{t+1} + \gamma Q_{\mathbf{w}}(\mathbf{x}', \mathbf{u}') - Q_{\mathbf{w}}(\mathbf{x},\mathbf{u}) \tag{2.5.3}$$

Using this TD-error, the parameters for the approximation of the Q-function are updated using the update rule shown in Equation (2.5.4).

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_w \delta_t \nabla_{\mathbf{w}} Q_{\mathbf{w}}(\mathbf{x},\mathbf{u}) \tag{2.5.4}$$

Actor
Using the approximated Q-function, the policy approximation for the *Q actor-critic* algorithm is updated using gradient-descent, as shown in Equation (2.5.5). How these update steps for the critic and the actor can be used for control of a RL problem is shown in Algorithm 2.5.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_\theta Q_{\mathbf{w}}(\mathbf{x},\mathbf{u}) \nabla_{\boldsymbol{\theta}} \ln \pi_\theta(\mathbf{u}|\mathbf{x}) \tag{2.5.5}$$

Algorithm 2.5: Pseudocode for the *Q actor-critic* algorithm

initialize $\mathbf{x}, \boldsymbol{\theta}, \mathbf{w}$ at random; sample $\mathbf{u} \sim \pi_\theta(\mathbf{u},\mathbf{x})$;
for $t = 1, 2, ..., T$ **do**
  apply $\mathbf{u}_t$, observe $\mathbf{x}_{t+1}$ and r_{t+1} ;
  sample next action $\mathbf{u}_{t+1} \sim \pi_\theta(\cdot|\mathbf{x}_{t+1})$;
  calculate the TD-error: $\delta_t \leftarrow r_{t+1} + \gamma Q_{\mathbf{w}}(\mathbf{x}_{t+1},\mathbf{u}_{t+1}) - Q_{\mathbf{w}}(\mathbf{x},\mathbf{u})$;
  update $\mathbf{w}$: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_w \delta_t \nabla_{\mathbf{w}} Q_{\mathbf{w}}(\mathbf{x}_t,\mathbf{u}_t)$;
  update $\boldsymbol{\theta}$: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_\theta Q_{\mathbf{w}}(\mathbf{x}_t,\mathbf{u}_t) \nabla_{\boldsymbol{\theta}} \ln \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$;
end

SYNOPSIS

This Chapter started by covering the key elements in RL, and ended with a type of ARL commonly found in state-of-the art literature: actor-critic algorithms [12]. Chapter 3 will continue on this topic, elaborating how these algorithms are used for aerospace control. Furthermore, the different applications of RL for flight control will be presented.

3 | Reinforcement Learning for Flight Control

This chapter addresses the use of RL for flight control. First, the applications of RL in flight control are classified in Section 3.1. Secondly, adaptive critic designs are introduced in Section 3.2. Thirdly, applications of these adaptive critic designs in flight control are introduced and compared in Section 3.3. Then, one of the adaptive critic designs, Dual Heuristic Programming is featured in more detail in Section 3.4, followed by its extension Incremental Dual Heuristic Programming in Section 3.5.

3.1. RL for Flight Control Classification

The classification is shown in Figure 3.1. The classification is further elaborated on the next page, including examples of the cited publications.

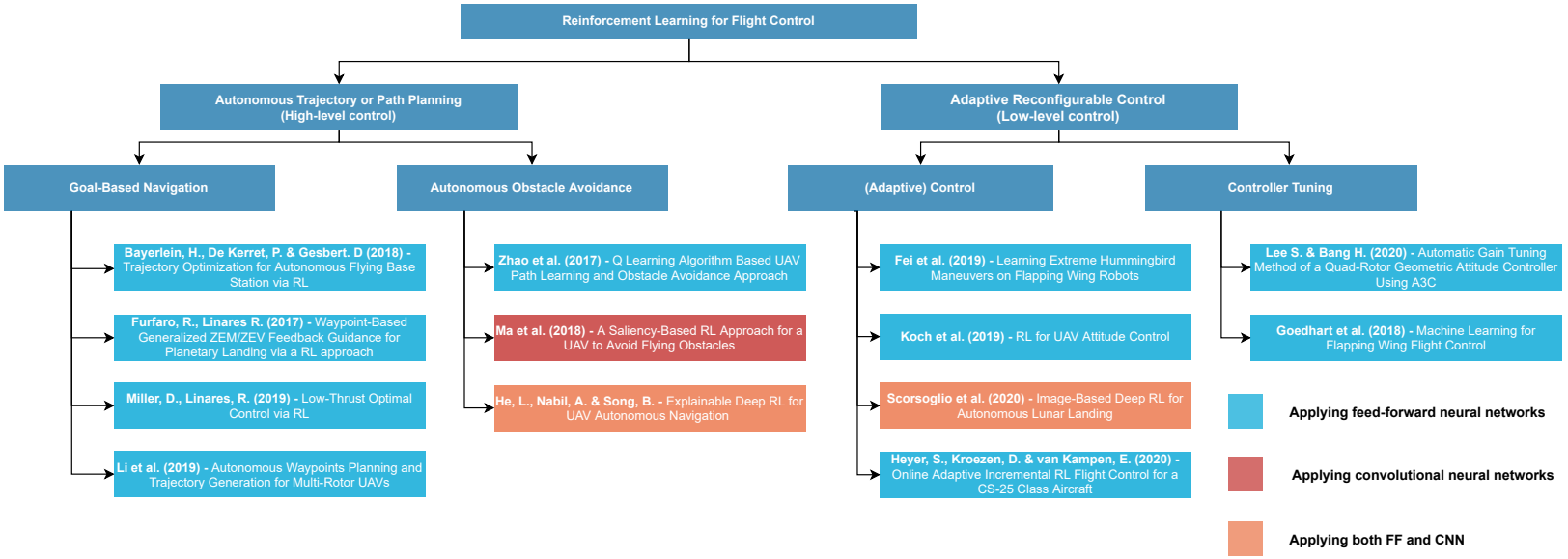


Figure 3.1: Non-exhaustive classification of RL for flight control, based on academic research available through Scopus.

The non-exhaustive classification presented in Figure 3.1 is inspired by all TU Delft research into RL for flight control. As of July 2021, all research into RL for flight control published through TU Delft can be classified into one of the presented categories. The upper branch of the classification divides the research into either high-level or low-level control. In literature, multiple methods of classifying autonomous control exist, such as the following structure of a hierarchical, ordered from high-level to low-level control: [39]

- Autonomous decision making - top-level decisions, such as trade-off between mission success and vehicle survivability, also includes obstacle avoidance
- Autonomous path planning - generation of the waypoints which are to be followed by the autonomous application
- Autonomous trajectory generation - generation of the trajectory through the specified waypoints while taking into consideration the physical limits of the application
- Adaptive reconfigurable controller - responsible for the trajectory following using sensors and actuators

Another control hierarchy for unmanned aerial vehicles is illustrated in Figure 3.2.

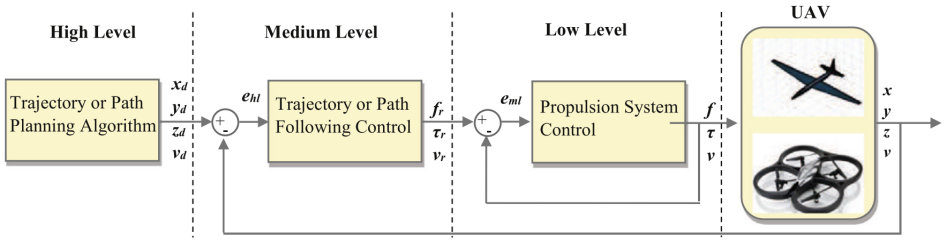


Figure 3.2: Control hierarchy for unmanned aerial vehicles, also applicable to other aerospace applications [40].

Based on these levels of autonomy, the analyzed RL papers for flight control presented in Figure 3.1 are classified into either high-level or low-level control. Referencing the first classification, high-level control covers autonomous decision making, autonomous path planning, and autonomous trajectory generation, as these tasks are ultimately responsible for creating the trajectory. Low-level control covers the trajectory following using sensors and actuators, including both trajectory-following and propulsion system control mentioned in Figure 3.2. Based on the available RL literature for flight control, the research is further divided into the following categories:

- **Goal-based navigation** - research utilizing RL for waypoint and/or trajectory generation [41] [42] [43] [44]. Research in this category typically focuses on generating a trajectory while minimizing time flown or energy spent. The included research in this category all used feed-forward neural networks.
- **Autonomous obstacle avoidance** - research applying RL methods for active obstacle avoidance, where the computed trajectory is infeasible due to an obstruction [45] [46] [20]. Typically works in this category utilize convolutional neural networks for image analysis. Currently, this class of RL applications includes the only work applying explainable RL for flight control [20].
- **Adaptive control** - low-level control of the aerospace application, using RL for trajectory-following and/or propulsion system control [47] [15] [48] [49]. In terms of number of publications, this class of RL research is currently the largest.
- **Controller tuning** - research where RL is used for controller adjustments, such as gain tuning [50] [51].

SYNOPSIS

From the classification, adaptive control is the largest of the four categories when considering the number of publications. This extensive research is likely due to the potential of RL for adaptive control. The flexible characteristics of function approximators, especially NNs, allow both good tracking and robust control of aerospace applications. However, these same complex function approximators lack transparency due to the extensive number of parameters. Currently, only one publication aims at improving the explainability of reinforcement learning for flight control [20]. As adaptive control is likely the most powerful flight control

application for RL, and because these methods still significantly lack transparency due to the limited amount of research, improving the explainability of RL flight control will be the main focus of this research. More details about explainability, including terminology, is presented in Chapter 4.

3.2. ADAPTIVE CRITIC DESIGNS

The application of neural networks for the function and policy approximation in actor-critic algorithms, and the development of the backpropagation algorithm, led to the development of *adaptive critic designs* (ACDs) [52]. This class of RL algorithms is first introduced as an approach for approximate/adaptive dynamic programming (ADP), by using neural networks for approximation of the value and policy function. Later this approach is referred to as adaptive critic designs. These methods, specifically developed for engineering control, typically utilize neural networks for the actor and the critic, and potentially a neural network for approximation of the global system dynamics [16]. These neural networks for the global system model, the actor, and the critic have respectively weights $\mathbf{w}_m, \mathbf{w}_a$, and $\mathbf{w}_c$. Adaptive critic designs apply temporal-difference learning, allowing a problem to be solved forward-in-time [53], therefore enabling online control just like the methods introduced in Section 2.3.2. A schematic representation of Dual Heuristic Programming, one of the ACD variants is shown in Figure 3.3. In this example, NNs are used for the actor, critic, and the global system model.

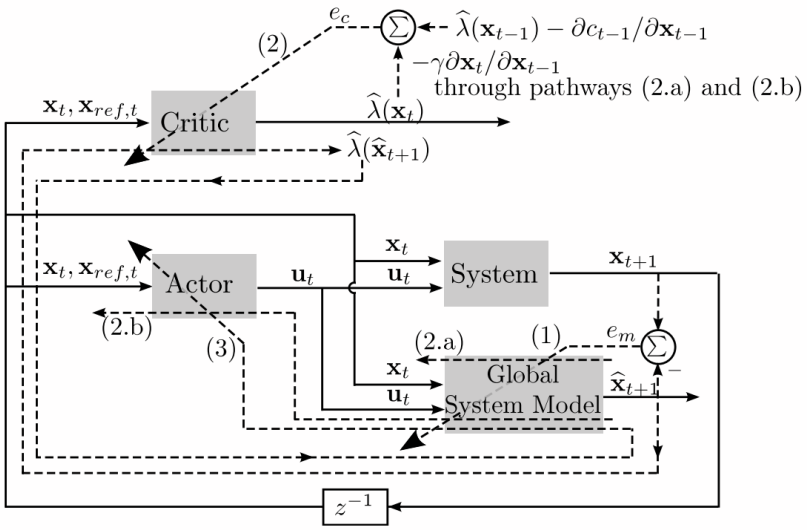


Figure 3.3: Example of an adaptive critic design structure: Dual Heuristic Programming [17].

Adaptive critic designs can be categorized into *heuristic dynamic programming* (HDP), *dual heuristic programming* (DHP), and *globalized dual heuristic programming* (GDHP). Additionally, an ACD can be action-dependent when the actor's output is used for the value function approximation, resulting the prefix AD-. The three main categories and their action-dependent variants are illustrated in Figure 3.4.

The characteristics of the six variants shown in Figure 3.4 are described below: [53]

- **HDP:** All ACD methods alternate between policy evaluation and policy improvement, like the DP methods explained in Section 2.2. In heuristic dynamic programming, the critic outputs $\hat{V}_t$ as the estimate of V_t based on the current state $\mathbf{x}_t$. By sampling the states $\mathbf{x}_t$ and reward or cost c_t encountered during control, the error used for updating the critic network can be computed. As therefore all knowledge required for updating the critic network is available, HDP does not require a plant model for updating $\mathbf{w}_c$. Training of the actor network however requires the derivative $\frac{\partial V(t)}{\partial \mathbf{u}(t)}$, which is dependent on the derivatives $\frac{\partial V(t)}{\partial \mathbf{x}(t)}$ and $\frac{\partial \mathbf{x}(t)}{\partial \mathbf{u}(t)}$ through the chain rule. This first derivative is found through backpropagation of the critic NN, but to compute the latter derivative a model of the system dynamics is required. Therefore HDP does not require a model for critic training, but does require a plant model for training of the actor network. This plant model is also updated during training, for example with gradient-descent using the error between the prediction of the next state and the actual measured state in the following time step.

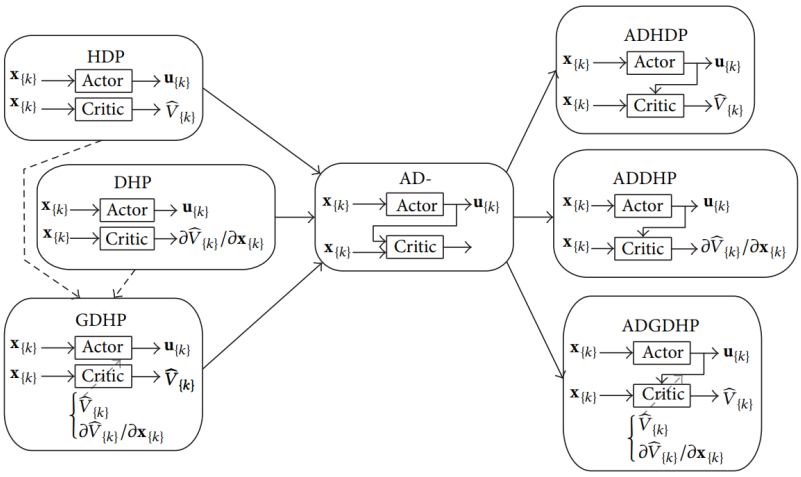


Figure 3.4: Overview of the three adaptive critic design variants, and their action-dependent alternatives [54].

- **ADHDP:** Action-dependent heuristic dynamic programming (ADHDP) uses the same critic structure and training as HDP. By also using the action output of the actor as an input of the critic, the derivative $\frac{\partial \mathbf{x}(t)}{\partial \mathbf{u}(t)}$ can be determined directly through backpropagation of the critic network. Therefore the need for a plant model for updating $\mathbf{w}_a$ is removed. As a result, ADHDP does not require a plant model for updating any parameters.
- **DHP:** In dual heuristic programming, the critic does not estimate the value function V_t, but it estimates the gradient of the value function with respect to the state: $\hat{\lambda}(\mathbf{x}_t) = \frac{\partial \hat{V}(t)}{\partial \mathbf{x}(t)}$. As a result, updating the critic requires the term $\frac{\partial \mathbf{x}(t+1)}{\partial \mathbf{x}(t)}$ which is obtained through backpropagation of the estimated model dynamics. Training of the actor for DHP, like the case of HDP, requires the derivatives $\frac{\partial V(t)}{\partial \mathbf{x}(t)}$ and $\frac{\partial \mathbf{x}(t)}{\partial \mathbf{u}(t)}$. This first derivative, $\lambda(t)$ is directly available for DHP through the critic network, while the second is obtained through the global system model. In conclusion, DHP uses the global system model for both the critic and actor updates.
- **ADDHP:** Like ADHDP, action-dependent dynamic heuristic (ADDHP) programming uses the measured state and output of the actor as inputs for the critic. The derivative required for updating the actor network is now available through the critic, removing the need for a global system model for updating $\mathbf{w}_a$. Therefore ADDHP only requires the approximation of the global system model for updating the critic.
- **GDHP:** Global dynamic heuristic programming can be regarded as a combination of HDP and DHP, as the critic outputs both $\hat{V}(\mathbf{x}_t)$ and $\hat{\lambda}(\mathbf{x}_t)$. This first term is computed as in HDP, without using a global system model. Due to the latter term however, like DHP, updating the critic does require a global system model. Actor training is similar to HDP, and DHP, and therefore also requires the global system model.
- **ADGDHP:** Similar to the preceding action-dependent ACDs, ADGDHP uses not only the measured states as input for the critic, but also the output of the actor. This removes the need for a global system model for updating the actor. The plant model is still required however for computation of the $\hat{\lambda}(\mathbf{x}_t)$ output of the critic.

The requirements for having a global system model for training of the critic or actor of all six adaptive critic design categories are summarized in Table 3.1.

Table 3.1: Overview of the six adaptive critic design structures

ACD Structure	**Critic structure**		**System model requirement:**	
	Input	Output	Critic	Actor
HDP	[**x**]	V		✓
ADHDP	[**x** **u**]	Q		
DHP	[**x**]	$\frac{\partial V}{\partial \mathbf{x}}$	✓	✓
ADDHP	[**x** **u**]	$\left[\frac{\partial Q}{\partial \mathbf{x}} \quad \frac{\partial Q}{\partial \mathbf{u}}\right]$	✓	
GDHP	[**x**]	$\left[V \quad \frac{\partial V}{\partial \mathbf{x}}\right]$	✓	✓
ADGDHP	[**x** **u**]	$\left[Q \quad \frac{\partial Q}{\partial \mathbf{x}} \quad \frac{\partial Q}{\partial \mathbf{u}}\right]$	✓	

3.3. ADAPTIVE CRITIC DESIGN APPLICATIONS IN FLIGHT CONTROL

One of the earliest applications of adaptive critic designs for flight control is presented in a comparison between a classical PID controller with HDP and DHP controllers for a non-linear auto-landing control problem with disturbance. In this comparison, DHP has been proven to show the best tracking behavior [16]. The HDP controller in this research is also able to show sufficient tracking, but is less consistent than the DHP and PID controller. Additionally, in other comparative research DHP has been shown to outperform HDP for non-linear voltage regulation of a turbo generator [55]. These two researches show that both HDP and DHP can be successful in non-linear tracking tasks, but DHP shows overall more consistent and accurate control behavior. The performance advantage of DHP above HDP is attributed to the use of the $\hat{\lambda}(\mathbf{x}_t)$ terms in DHP [56]. These terms reduce the chance of introducing errors in the trained parameters due to backpropagation. As GDHP can be regarded as a combination of HDP and DHP, one might expect this ACD to have superior performance when compared to the other two. The performance advantage is minor however, while the increased computational complexity of GDHP is significantly higher than that of DHP [56].

In 2004 DHP was successfully applied to a six-degree-of-freedom flight control problem, using a simulation of a business jet [57]. This research shows that DHP can still provide adequate tracking under unforeseen circumstances, such as control failures and unmodelled dynamics. Before this controller can be used for online control however, the DHP controller's critic and actor neural networks require an offline training phase.

In other research, HDP and ADHDP are implemented successfully for flight control of a non-linear F-16 simulation [58]. The ADDHP controller uses the ADDHP framework above, with a neural network modelling the plant dynamics used for updating the critic's weights. After an offline training phase, both ADDHP and ADHDP successfully complete a pitch angle tracking task. Offline in this sense does not imply that the agents are learning from samples generated by external agents, but that the simulation used for learning can be reset whenever unstable behavior is detected. During offline training, the ADDHP agent's chance of a satisfactory run is twice as high as the ADHDP agent. As both ADDHP and ADHDP are implemented in this research, both methods can be compared to assess their adaptability. In this research, ADDHP adapts better to different plant dynamics and flight conditions than ADHDP agent, but the ADHDP agent is more robust to measurement noise.

In 2018, the DHP framework was extended using a recursive least squares method for online incremental model identification [17]. This method, called *Incremental Dual Heuristic Programming* (IDHP) utilizes an incremental model, removing the need for an offline training phase. The IDHP framework allows near-optimal control without a prior learning phase, and without any knowledge of the system dynamics. Figure 3.5 shows a comparison between the tracking performance of the DHP and IDHP framework, for control of a non-linear missile model.

In the example shown in Figure 3.5, IDHP has a significantly shorter settling time, and the tracking is clearly more accurate. Furthermore, the online learning ability of IDHP allows adaptation to changing system dynamics during flight. Due to the superior tracking performance, settling time, and fault-tolerance of IDHP compared to other RL flight control frameworks, this algorithm is considered the state-of-the-art of RL for flight control and will therefore be the framework of choice for this thesis.

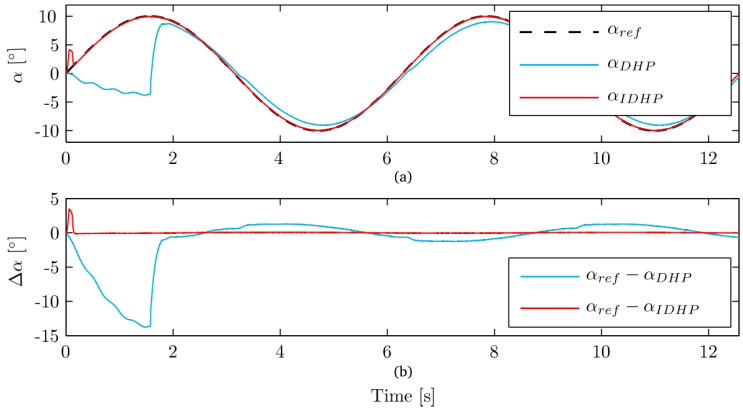


Figure 3.5: Online tracking task comparison between DHP and IDHP, for a non-linear missile model [17]. IDHP here shows superior tracking performance and more rapid convergence to α_{ref}.

3.4. DUAL HEURISTIC PROGRAMMING

Before presenting the IDHP framework in detail, DHP will be introduced first. As IDHP is an extension of the DHP framework, this non-incremental framework will be presented first in detail. Dual heuristic programming requires three non-linear function approximators, for approximation of the value function, the optimal policy, and for the system dynamics with respectively weights $\mathbf{w}_c$, $\mathbf{w}_a$, and $\mathbf{w}_m$. The critic in DHP estimates the derivatives of the value function $V(\mathbf{x}_t)$ with respect to the system's state $\mathbf{x}_t$, defined as $\hat{\lambda}(\mathbf{x}_t) = \frac{\partial \hat{V}(t)}{\partial \mathbf{x}(t)}$.

System model:
This global system approximation in DHP is used to backpropagate error signals through to update the critic and actor approximators. The system model approximates the dynamics of the global system, using the observed state of the system $\mathbf{x}_t \in \mathbb{R}^n$ and the action chosen by the actor $\mathbf{u}_t \in \mathbb{R}^m$ as inputs. Using these inputs, and the system model based on the weights $\mathbf{w}_m(t)$, the next state of the system is estimated as $\hat{\mathbf{x}}_{t+1} \in \mathbb{R}^n$. At every time step during training, the system model can be optimized based upon the measured new state. For this optimization the model error $E_m(t)$, shown in Equation (3.4.1), is continuously minimized by updating the system weights. $\mathbf{e}_m$ is defined as the error between the measured and estimated state, as shown in Equation (3.4.2).

$$E_m(t) = \frac{1}{2}\mathbf{e}_m^2(t) \tag{3.4.1}$$

$$\mathbf{e}_m = \mathbf{x}_t - \hat{\mathbf{x}}_t \tag{3.4.2}$$

The system model weights can, for example be updated using gradient descent, as described in Section 2.4, resulting in Equation (3.4.3). The gradient $\Delta\mathbf{w}_m(t)$ is scaled with the *model learning rate* $0 < \eta_m < 1$ as displayed in Equation (3.4.4).

$$\mathbf{w}_m(t+1) = \mathbf{w}_m(t) + \Delta\mathbf{w}_m(t) \tag{3.4.3}$$

$$\begin{aligned}\Delta\mathbf{w}_m(t) &= -\eta_m \cdot \frac{\partial E_m(t+1)}{\partial \mathbf{w}_m(t)} \\ &= -\eta_m \cdot \frac{\partial E_m(t+1)}{\partial \hat{\mathbf{x}}_{t+1}} \frac{\partial \hat{\mathbf{x}}_{t+1}}{\partial \mathbf{w}_m(t)}\end{aligned} \tag{3.4.4}$$

Critic:
In DHP, the critic approximates the gradient of the true value function $V(\mathbf{x}_t)$ with respect to the state vector $\mathbf{x}_t$. In ADP, the value function is defined as the discounted cumulative cost over time, also defined in ADP

literature as the *cost-to-go*:

$$V(\mathbf{x}_t) = \sum_{l=t}^{\infty} \gamma^{l-t} c_l \tag{3.4.5}$$

Where c is defined as the non-negative cost due to transitioning from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$, comparable with a penalty for classical RL problems. This cost is part of the design of the problem, and can for example be based on the state, chosen action, and reference state. Similar to the *Q actor-critic* example in Section 2.5, the critic network is updated using the TD-error. The recursive property of the value function, repeated here in Equation (3.4.6), can be rewritten for $\lambda(\mathbf{x}_t)$ as shown in Equation (3.4.7).

$$V(\mathbf{x}_t) = c_{t+1} + \gamma V(\mathbf{x}_{t+1}) \tag{3.4.6}$$

$$\begin{aligned} \frac{\partial V(\mathbf{x}_t)}{\partial \mathbf{x}_t} &= \frac{\partial c_{t+1}}{\partial \mathbf{x}_t} + \gamma \frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{x}_t} \\ \rightarrow \lambda(\mathbf{x}_t) &= \frac{\partial c_{t+1}}{\partial \mathbf{x}_t} + \gamma \lambda(\mathbf{x}_{t+1}) \end{aligned} \tag{3.4.7}$$

The difference between the left- and right-hand side of Equation (3.4.7) defines the TD-error for the critic. This TD-error $\mathbf{e}_c(t)$ is hence defined as:

$$\begin{aligned} \mathbf{e}_c(t) &= \frac{\partial c_{t+1}}{\partial \mathbf{x}_t} + \gamma \lambda(\mathbf{x}_{t+1}) - \lambda(\mathbf{x}_t) \\ &= \frac{\partial V(\mathbf{x}_t)}{\partial \mathbf{x}_t} - \gamma \frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{x}_t} - \frac{\partial c_t}{\partial \mathbf{x}_t} \end{aligned} \tag{3.4.8}$$

The term $\frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{x}_t}$ in Equation (3.4.8) can be expanded using the chain rule, as follows:

$$\frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{x}_t} = \frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{x}_{t+1}} \cdot \frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} \tag{3.4.9}$$

This expression of $\frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{x}_t}$, and the definition of $\lambda(\mathbf{x}_t)$ can then be used to write the TD-error for the critic using gradient functions, as presented in Equation (3.4.10).

$$e_c(t) = \lambda(\mathbf{x}_t) - \gamma \lambda(\mathbf{x}_{t+1}) \frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} - \frac{\partial c_t}{\partial \mathbf{x}_t} \tag{3.4.10}$$

Using this TD-error, the loss function for the critic can be defined, as shown in Equation (3.4.11). By minization of this error measure, the parameters of the critic are tuned.

$$E_c(t) = \frac{1}{2} \mathbf{e}_c(t)^T \mathbf{e}_c(t) \tag{3.4.11}$$

The terms $\lambda(\mathbf{x}_t)$ and $\lambda(\mathbf{x}_{t+1})$ in Equation (3.4.8) are computed by respectively passing $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ through the critic network, while $\frac{\partial c_t}{\partial \mathbf{x}_t}$ is computed with the chosen cost function. The $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$ term, however, is less easily computed. The global system model gives $\hat{\mathbf{x}}_{t+1}$ as an output based on the measured input $\mathbf{x}_t$, therefore the derivative $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$ is available through backpropagation of the system model. However, due to a change in $\mathbf{x}_t$, the actor will also give another output $\mathbf{u}_t$, which also influences the system model. Therefore the term $\frac{\partial \mathbf{u}_t}{\partial \mathbf{x}_t}$ due to the actor and the term $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{u}_t}$ due to the system model should also be incorporated in the backpropagation path for determining $\mathbf{e}_c(t)$. Equation (3.4.12) shows this complete calculation for $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$ [59].

$$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} = \underbrace{\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}}_{\text{System model}} + \underbrace{\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{u}_t}}_{\text{System model}} \cdot \underbrace{\frac{\partial \mathbf{u}_t}{\partial \mathbf{x}_t}}_{\text{Actor}} \tag{3.4.12}$$

Similar to the global system model, the critic's parameters can be updated using gradient-descent. The update step for this training is shown in Equation (3.4.13), where the increment $\Delta \mathbf{w}_c(t)$ is extended in Equation (3.4.14).

$$\mathbf{w}_c(t+1) = \mathbf{w}_c(t) + \Delta \mathbf{w}_c(t) \tag{3.4.13}$$

$$\begin{aligned} \Delta \mathbf{w}_c(t) &= -\eta_c \cdot \frac{\partial E_c(t)}{\partial \hat{\lambda}(\mathbf{x}_t)} \cdot \frac{\partial \hat{\lambda}(\mathbf{x}_t)}{\partial \mathbf{w}_c(t)} \\ &= -\eta_c \cdot \mathbf{e}_c(t)^T \cdot \frac{\partial \hat{\lambda}(\mathbf{x}_t)}{\partial \mathbf{w}_c(t)} \end{aligned} \tag{3.4.14}$$

Actor:
The optimal control action at every time step is the one minimizing the value function, as presented in Equation (3.4.15) [59].

$$\mathbf{u}_t^* = \arg\min_{\mathbf{u}_t} V(\mathbf{x}_t) \tag{3.4.15}$$

The loss function is therefore defined as in Equation (3.4.16). Then using the recursive expression of the value function, shown in Equation (3.4.6), the gradient of $E_a(t)$ with respect to the actor's parameters $\mathbf{w}_a(t)$ can be expanded as in Equation (3.4.17).

$$E_a(t) = V(\mathbf{x}_t) \tag{3.4.16}$$

$$\begin{aligned} \frac{\partial E_a(t)}{\partial \mathbf{w}_a(t)} &= \frac{\partial V(\mathbf{x}_t)}{\partial \mathbf{u}_t} \cdot \frac{\partial \mathbf{u}_t}{\partial \mathbf{w}_a(t)} \\ &= \left[\frac{\partial c_t}{\partial \mathbf{u}_t} + \gamma \frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{u}_t} \right] \cdot \frac{\partial \mathbf{u}_t}{\partial \mathbf{w}_a(t)} \\ &= \left[\frac{\partial c_t}{\partial \mathbf{u}_t} + \gamma \frac{\partial V(\mathbf{x}_{t+1})}{\partial \mathbf{x}_{t+1}} \frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{u}_t} \right] \cdot \frac{\partial \mathbf{u}_t}{\partial \mathbf{w}_a(t)} \\ &= \left[\frac{\partial c_t}{\partial \mathbf{u}_t} + \gamma \lambda(\mathbf{x}_{t+1}) \frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{u}_t} \right] \cdot \frac{\partial \mathbf{u}_t}{\partial \mathbf{w}_a(t)} \end{aligned} \tag{3.4.17}$$

The gradient shown in Equation (3.4.17) is used to calculate $\Delta \mathbf{w}_a(t)$, as shown in Equation (3.4.18), which also displays the entities responsible for the different elements in the equation. Finally, Equation (3.4.19) shows the update equation for the actor's parameters.

$$\begin{aligned} \Delta \mathbf{w}_a(t) &= -\eta_a \cdot \frac{\partial E_a(t)}{\partial \mathbf{w}_a(t)} \\ &= -\eta_a \cdot \Big[\underbrace{\frac{\partial c_t}{\partial \mathbf{u}_t}}_{\substack{\text{Reward} \\ \text{signal}}} + \underbrace{\gamma \lambda(\mathbf{x}_{t+1})}_{\text{Critic}} \underbrace{\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{u}_t}}_{\substack{\text{System} \\ \text{model}}} \Big] \cdot \underbrace{\frac{\partial \mathbf{u}_t}{\partial \mathbf{w}_a(t)}}_{\text{Actor}} \end{aligned} \tag{3.4.18}$$

$$\mathbf{w}_a(t+1) = \mathbf{w}_a(t) + \Delta \mathbf{w}_a(t) \tag{3.4.19}$$

3.5. INCREMENTAL DUAL HEURISTIC PROGRAMMING

The IDHP framework is an extension of the DHP framework, using incremental control techniques [17]. The main difference compared to DHP is the use of incremental model identification, as opposed to DHP's global system model. Using only a few state and input measurements, the critic and actor can be updated resulting in near-optimal control after a short settling-time. The requirements for this incremental model identification are high frequency state and input measurements, and relatively slow-varying system states.

Incremental system model:
Rather than using a function approximator for estimation of the global system dynamics, IDHP utilizes high-frequency system measurements and the Recursive Least Squares approach to estimate the system transition and input distribution matrices. The dynamics of a non-linear system can in general be modelled as in Equation (3.5.1).

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f[\mathbf{x}(t), \mathbf{u}(t)] \\ \mathbf{y}(t) &= h[\mathbf{x}(t)] \end{aligned} \tag{3.5.1}$$

The form shown in Equation (3.5.1) is continuous, but measurements in flight control are discrete. Rewriting this general description of a non-linear system to a discrete form results in:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{3.5.2}$$

Using the requirement for high-frequency state and input measurements described above, Equation (3.5.2) can be linearized to:

$$\mathbf{x}_{t+1} \approx \mathbf{x}_t + \mathbf{F}_{t-1} \cdot (\mathbf{x}_t - \mathbf{x}_{t-1}) + \mathbf{G}_{t-1} \cdot (\mathbf{u}_t - \mathbf{u}_{t-1}) \tag{3.5.3}$$

Then, Equation (3.5.3) is rewritten into the incremental control form as shown in Equation (3.5.4). In this incremental form $\mathbf{F}_{t-1}$ is the system transition matrix at $t-1$, and $\mathbf{G}_{t-1}$ is the input distribution matrix at $t-1$.

$$\Delta\mathbf{x}_{t+1} \approx \mathbf{F}_{t-1}\Delta\mathbf{x}_t + \mathbf{G}_{t-1}\Delta\mathbf{u}_t \tag{3.5.4}$$

By measuring $\Delta\mathbf{x}_t$ and $\Delta\mathbf{u}_t$ with high frequency, $\mathbf{F}_{t-1}$ and $\mathbf{G}_{t-1}$ are estimated. In IDHP, Recursive Least Squares is used for this parameter estimation.

Critic:
Updating of the critic weights is similar to DHP:

$$\Delta\mathbf{w}_c(t) = -\eta_c \cdot \mathbf{e}_c(t)^T \cdot \frac{\partial\hat{\lambda}(\mathbf{x}_t)}{\partial\mathbf{w}_c(t)} \tag{3.5.5}$$

Where $\mathbf{e}_c(t)$ is similar to in the DHP framework:

$$e_c(t) = \lambda(\mathbf{x}_t) - \gamma\lambda(\mathbf{x}_{t+1})\frac{\partial\mathbf{x}_{t+1}}{\partial\mathbf{x}_t} - \frac{\partial c_t}{\partial\mathbf{x}_t} \tag{3.5.6}$$

Using the incremental system identification model, the calculation of $\frac{\partial\mathbf{x}_{t+1}}{\partial\mathbf{x}_t}$ is simplified to:

$$\frac{\partial\mathbf{x}_{t+1}}{\partial\mathbf{x}_t} \approx \hat{\mathbf{F}}_{t-1} + \hat{\mathbf{G}}_{t-1} \cdot \frac{\partial\mathbf{u}_t}{\partial\mathbf{x}_t}|_a \tag{3.5.7}$$

Actor:
In DHP, the actor weights are updated using Equation (3.4.18). For IDHP, the $\frac{\partial\mathbf{x}_{t+1}}{\partial\mathbf{u}_t}$ term is approximated using the incremental model:

$$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{u}_t} \approx \mathbf{G}_{t-1} \tag{3.5.8}$$

Resulting in the simplified equation for updating the actor weights in IDHP:

$$\Delta \mathbf{w}_a(t) = -\eta_a \cdot \left[\frac{\partial c_t}{\partial \mathbf{u}_t} + \gamma \lambda \left(\mathbf{x}_{t+1} \right) \mathbf{G}_{t-1} \right] \frac{\partial \mathbf{u}_t}{\partial \mathbf{w}_a(t)} \tag{3.5.9}$$

The schematic outline of the IDHP is shown in Figure 3.6, illustrating the backpropagation paths and feed-forward connections between the critic, actor, incremental model, and system.

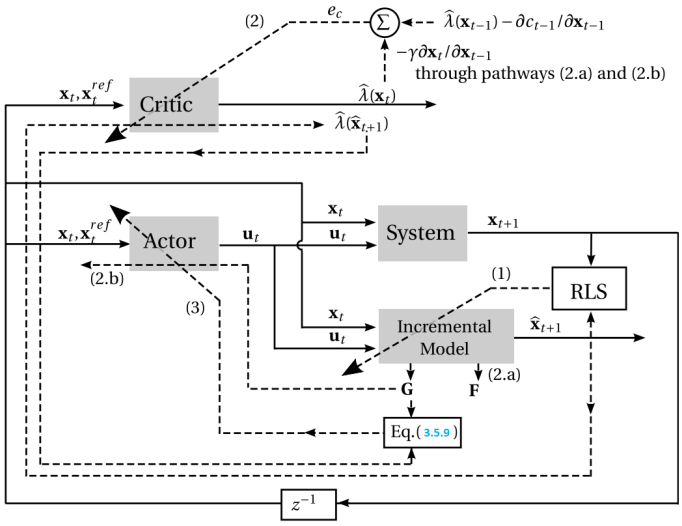


Figure 3.6: Schematic overview of the IDHP algorithm, where the dashed lines indicate backpropagation paths [17].

SYNOPSIS

RL is used for several purposes in flight control research, including goal-based navigation, autonomous obstacle avoidance, adaptive control and controller tuning. Based on the literature classification of RL for flight control, adaptive control is currently the most active research field. Furthermore, DRL shows great potential for this application due to flexibility of the NNs. From the reviewed RL for adaptive control publications, IDHP is considered the state-of-the-art due to its superior performance, adaptability, and fault-tolerance. This model, like many DRL techniques, is considered a block-box model due to the opaqueness of the NNs used for the actor and critic modules. As the actor is ultimately responsible for mapping the observable states of the aerospace vehicle to actuator outputs, explaining the inner workings of this module will provide useful insights into the mechanics of IDHP. In an effort to create these explanations, the following Chapter will review existing XAI techniques, and make an selection of potentially useful XAI methods to explain IDHP.

4 | Explainable AI Techniques

As illustrated in the preceding chapter, artificial intelligence methods such as deep RL can show exceptional performance for complex tasks. A major drawback of many AI methods however is the *black box* property, meaning its lack of transparency in its decision-making. The input and output of the AI model can be clearly observed, but what happens inside the black box remains a mystery. Especially state-of-the art AI techniques utilize millions of parameters, hindering the ability to easily interpret these methods. In recent years there has been an extensive effort into the development of both interpretable AI and explainable AI techniques. First of all, this chapter introduces the topic of both interpretable and explainable AI in Section 4.1. Secondly, in Section 4.2 the methodology for selecting the XAI literature for this thesis is presented. Then the selected recent XAI techniques which could be relevant for explainable RL for flight control are introduced, by first addressing transparent design techniques in Section 4.3, and then post-hoc explainability techniques in Section 4.4. These selected AI techniques are featured by first explaining their methodology for explanation, then their strengths and weaknesses are highlighted, and finally these are assessed based on their applicability to explainable RL for flight control. Finally the chapter is concluded in Section 4.5 where the most promising AI technique is selected.

4.1. Motivation, Challenges, and Terminology

While the origin of AI has been established in the 50s, the ability to handle complex tasks such as image recognition, audio processing, and data analysis has only emerged in the current millennium [6]. The forms of AI designed in the 20th century are of limited complexity compared to the wide range of AI applications used today. The acceleration of AI performance since the 2000s, and especially in the last decade, is mainly achieved by improved optimization algorithms, the emergence of open-source AI libraries, the rise of Big Data and advancements in the computing power of hardware [6] [7]. However, where the first AI systems were easy to interpret due to their limited complexity, state-of-the art AI systems such as deep learning are considerably more opaque [18]. *Interpretability* in this thesis is defined as the ability to explain something in terms understandable to a human. The parameter space of deep learning methods, as explained in Section 2.4, grows exponentially when increasing the number of neurons, making the method less interpretable. When the inner workings of a model are hidden, the model is often defined as a black box model [19]. If the inner workings are completely clear on the other hand, the model is regarded as *transparent*. However, the interpretability of many ML models cannot be classified using these two extremes, but is best described on the spectrum between transparent and black box.

The opaqueness is one of the major drawbacks of AI methods, and can be a severe limitation in applying AI techniques to solve daily problems. The danger of opaqueness is the possibility of a decision not being justifiable or legitimate [18]. Additionally, it is often desirable to understand the reasoning behind a certain action, to trust the decision being made. Furthermore, the need for transparency grows when the stakes involved in the decision-making process are higher. For some fields, such as self-driving cars where lives are involved, explanations of the prediction of a model are crucial before AI methods can be used.

Another disadvantage of opaque AI models can be explained using the *irony of automation* [60]. This theory states that as processes are automated, as AI is currently doing on a large scale, human operators are charged with the following two tasks: monitoring the operation of the autonomous agent, and taking over control manually when the task is too complex for the automated machine. Both cases can be described using a human pilot, monitoring the flight of the autopilot, and taking over control in the case of for example an engine failure. Monitoring an autonomous system can only be done while understanding the mechanics of the controller and the controlled system. The pilot must understand how the aircraft and the autopilot operate nominally to monitor its operation. For the second task, the irony of automation states that when simple tasks are automated, the human expert loses skill [60]. When manual take-over is required though, the control task is likely difficult as the automated machine is not able to fulfill it. The human expert is however less skilled to complete this task due to the simple tasks being automated. Using the pilot again as

an example: when the aircraft is flown using the autopilot, the pilot will slowly lose his proficiency of manual flight. Whenever manual take-over is necessary, the flight control task likely requires a lot of skill, due to for example the event of a failure. For both monitoring and intervention, transparency of AI models is essential. For monitoring, the human expert requires understanding of the mechanics of the AI model. In addition, explainability of AI models can mitigate skill loss due to automation. If the mechanics of the AI model are properly explained, experts can learn from what the algorithm is doing, reducing skill loss compared to the case where the black box makes all decisions without any justification.

In [18], the following three reasons are mentioned for using interpretability as a design driver for a ML model [18]:

1. Interpretability helps ensure impartiality in decision-making, i.e. to detect, and consequently, correct from bias in the training dataset.
2. Interpretability facilitates the provision of robustness by highlighting potential adversarial perturbations that could change the prediction.
3. Interpretability can act as an insurance that only meaningful variables infer the output, i.e., guaranteeing that an underlying truthful causality exists in the model reasoning.

These three reasons are mainly focused on shortcomings in AI for classification and prediction, mainly through supervised and unsupervised learning. However, RL also suffers from the same shortcomings. In RL, the first reason helps to identify decision-making based on unrealistic training data. As RL often requires a lot of sampling data before an agent is successful, learning which data is responsible for the actions of the agent can help in identifying training bias. The second reason is also applicable to RL, as having insights into the sensitivity of a decision with respect to the input variables improves the robustness. Finally, the third reason is likely the most important for RL applications, as proof that the input-output mapping follows logical reasoning will greatly increase the trustworthiness.

In recent years the opaqueness problem is tackled by the development of both *interpretable AI* and *eXplainable AI* (XAI) methods. Interpretable AI is defined as AI methods which are inherently understandable by a human without any prior knowledge. For these methods it is clear how a change in input alters the output of the model. Explainable AI methods on the other hand aim to explain the inner workings of ML models which are not inherently interpretable. The suite of XAI methods can be defined as follows: *Given an audience, an explainable Artificial Intelligence is one that produces details or reasons to make its functioning clear or easy to under-stand* [18].

The audience is an important term in the XAI definition, as explainability can be very subjective according to the audience to which the explanation is given. Possible target audiences are for example regulatory agencies responsible for the legislation of a self-driving car utilizing AI for its navigation, but also users affected by the decisions of a model such as bank customers applying for a loan which is judged using a ML model. The main target audience for this thesis is RL researchers. The choice of this audience stems from the current state of RL for flight control. While the current state-of-the-art RL methods like IDHP show great performance, their technology readiness level is not high enough for these methods to be explained to for example regulatory instances. For RL researchers, having insights into the working principles of the RL algorithms can aid development as potential bugs or biases are easier to detect, significantly accelerating development. An example of how XAI techniques could be used in a design cycle for RL development is shown in Figure 4.1.

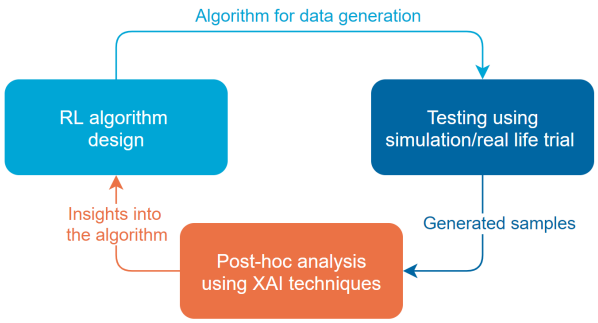


Figure 4.1: Schematic representation of how XAI techniques can be used for RL researchers to adjust their algorithm in a design cycle.

In the example shown in Figure 4.1, a newly designed RL algorithm is tested in a simulation or real life to generate test data. These test samples are then used in the *post-hoc*, or after data generation, design analysis to generate explanations of the algorithm. These insights can then be used to update the RL algorithm after which the design cycle is repeated until the goal is fulfilled. Where typically only the environment reward, the output of the RL agent, and the time traces of the states can be used for analysis, post-hoc methods allow insights into the input-output mapping of the agent. Potentially, this allows for detection of flaws in the design of the reward function or the agent that are otherwise unobservable.

The major benefit of XAI techniques above interpretable AI is that no compromises are required in terms of reduced model complexity for increased interpretability. Where interpretable AI has its limitations in terms of complexity to be understood, XAI techniques aim to make complex AI methods more understandable, or more formally: "*The XAI program's goal is to create a suite of new or modified ML techniques that produce explainable models that, when combined with effective explanation techniques, enable end users to understand, appropriately trust, and effectively manage the emerging generation of AI systems*" [61]. Figure 4.2 shows the tradeoff between model accuracy and model interpretability mentioned before, and how XAI aims to improve both these characteristics. While this tradeoff mentions accuracy, this does not imply that simple models cannot be accurate, or that complex models are inherently more accurate. For example, for a simple task a rule-based learning model can be perfectly accurate, while for a complex non-linear control task a deep learning model is not necessarily accurate. Complex models however offer more flexibility, allowing higher accuracy for complex tasks.

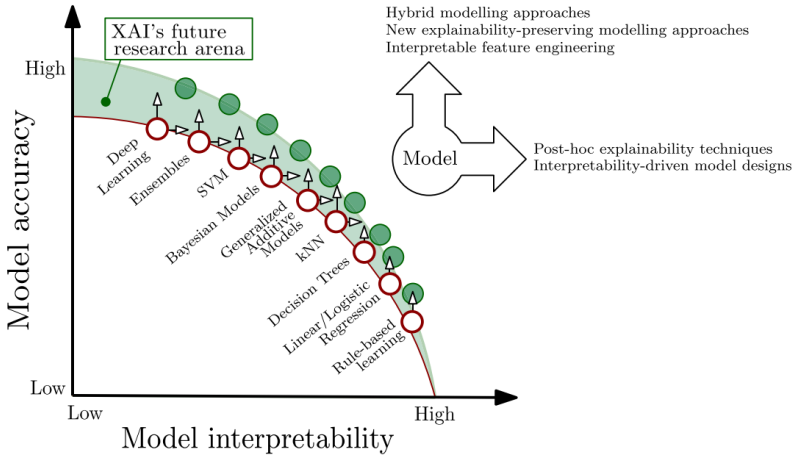


Figure 4.2: Trade-off between model interpretability and performance, and a representation of the area of improvement where the potential of XAI (eXplainable AI) techniques and tools resides [18].

In the literature survey of [18], a taxonomy is used where the analyzed XAI works are classified as *transparent model*, or *post-hoc explainability* techniques. Transparent models follow the definition of interpretable AI, however the model is not required to be inherently interpretable, as long as some form of interpretability is used in the design of the model. Following this definition, multiple levels of transparency are possible ranging from simple decision trees to more complex Bayesian models. Post-hoc explainability techniques on the other hand aim at explaining already designed models. According to the taxonomy, post-hoc explanation methods generally use three techniques: *model simplification*, *feature relevance*, and *visualization*. More details of post-hoc explainability methods are featured in Section 4.4, and more extensive in [18].

While the latest years have shown a tremendous growth in the development of explainability techniques for ML models in supervised and unsupervised learning, the RL domain still has many intricacies to be better understood [62]. This difference can be observed in the amount of publications into XAI and *eXplainable reinforcement learning* (XRL), shown in Figure 4.3. While giving some indication, the number of publications presented in Figure 4.3 is likely not exhaustive. The selected XAI works applying transparent design presented in Section 4.3 are all XRL methods, while the post-hoc techniques presented in Section 4.4 are developed with supervised learning in mind, but also useful for RL. A more extensive collection of XRL methods is available in the survey for XRL methods [63] and in the survey of explainable DRL methods [62].

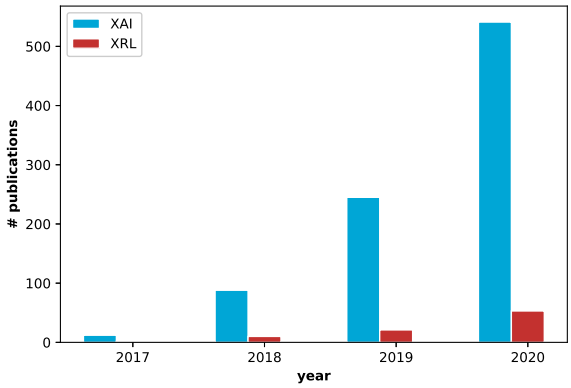


Figure 4.3: Comparison of the research into eXplainable AI and eXplainable RL, retrieved from Scopus in June 2021.

4.2. LITERATURE SELECTION METHODOLOGY

The XAI research covered in this literature review follows from a literature elimination process, where a large selection of the available XAI literature is narrowed down to potentially useful publications for explainability of flight control using RL. This process is shown schematically in Figure 4.4.

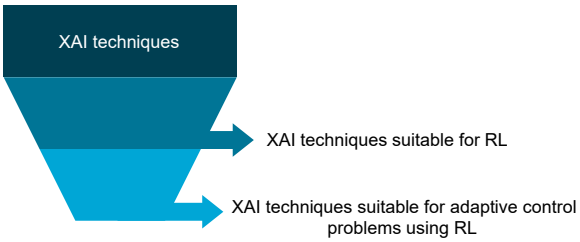


Figure 4.4: Selection funnel of the XAI research for this literature review.

The available XAI techniques, shown at the top in Figure 4.4 are explored through literature reviews and through Scopus [18] [62] [63]. Then, these works are filtered based on suitability for providing useful explanations for RL. Finally, another selection takes place based on the potential for explaining RL used for control problems, resulting in the provided literature. This latter selection is required as many XRL techniques focus on high-level control, as defined in the RL for flight control classification presented in Section 3.1. Other unsuitable works include XAI techniques utilizing less complex feature approximation methods, such as decision trees. The selected XAI works are divided according to the *transparent design* and *post-hoc explainability* classifications described above, and analyzed according to the following structure:

- **Methodology** - explaining how this XAI aims to provide explanations
- **Strengths and weaknesses** - description of the opportunities and limitations of these methods
- **Synopsis** - concluding analysis of how the selected work could be useful for creating explanations for RL flight control

4.3. TRANSPARENT DESIGN

This section highlights some explainable AI methods applying transparent design, in which explainability is one of the key drivers in the design of the AI method. First in Section 4.3.1 a navigation system will be introduced, utilizing fuzzy RL for explainability. Then in Section 4.3.2 RL explainability using reward system decomposition will be featured.

4.3.1. EXPLAINABLE NAVIGATION USING FUZZY REINFORCEMENT LEARNING

METHODOLOGY

Fuzzy control is one of the intelligent control methods, applying fuzzy logic to control systems. Fuzzy logic, as opposed to Boolean logic, allows logical values to be partially true by ranging between false and true. By applying fuzzy logic to control systems, control engineers can introduce knowledge about the system into the control laws. This knowledge is introduced through linguistic variables, such as cold-warm-hot for control involving temperature, and through logic IF-THEN rules [64]. The input variables of a fuzzy control system have numerical or *crisp* values. These crisp signals are transformed using the linguistic variables into a fuzzy state of the system through a process defined as *fuzzification*. Then utilizing the system knowledge of the designer, the fuzzified variables are used to produce a fuzzy decision, following the IF-THEN rules in a process defined as *inference*. However, as the actual system requires a numerical input, the fuzzy decisions are *defuzzified* into crisp values which can are used as system input. This process is illustrated schematically in Figure 4.6.

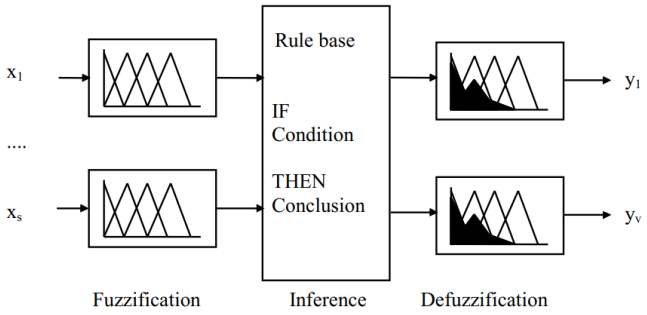


Figure 4.5: Schematic representation of the fuzzy decision-making process, starting and ending with crisp values[64]. Through design of the (de)fuzzification and the rule-base, expert knowledge is utilized in the process.

In one example of transparent design, fuzzy control is combined with DRL to create an explainable navigation system for an autonomous racing car [65]. The environment of the system is transformed into linguistic variables. The position on the track is for example defined as [middle, center, edge], while steering is defined as [left, center, right], and speed as [slow, fast]. Using these linguistic terms the inference rules are used to create the reward function. An example of this is the following rule: *Right lane:* IF *Speed* is *Slow* AND *Steering* is *Left* AND *Distance to center* is *Center* THEN *Reward* is *High*. Through these inference rules, human knowledge about driving a car is inserted into the reward signal for the RL problem. Subsequently, the reward signal is passed through an exponential function, to punish low rewards and reinforce high rewards. The agent is trained with the Proximal Policy Optimization actor critic algorithm, resulting in stable performance with the car driving along the centre of the racetrack.

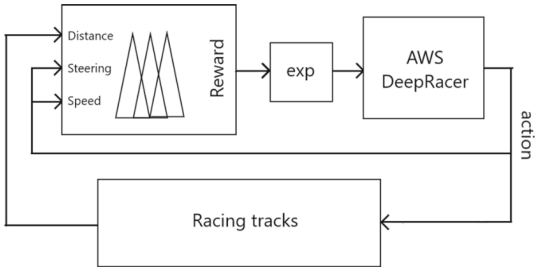


Figure 4.6: Diagram of the Fuzzy Reinforcement Learning system [65]. The "exp" block resembles an $e^{\mathbf{r}}$ function, where $\mathbf{r}$ is the reward vector.

SYNOPSIS

Due to the relatively simple IF-THEN inference rules, fuzzy control is often not as well-performing as other RL methods and has limited potential for complex and non-linear tracking tasks. However, Fuzzy RL as defined in the paper is not classic fuzzy control but rather a combination of fuzzy logic and DRL. According to the XAI classification presented in Section 4.1, Fuzzy RL is a combination of two learning methods: rule-based learning and deep learning. The fuzzy inference module for generating the reward system is rule-based learning, due to the IF-THEN rules. Applying fuzzy logic to rule-based learning improves both the model explainability

and accuracy [18]. Rule-based models applying fuzzy logic are more understandable thanks to the linguistic terms, and are more accurate in domains where the truth values are best described using fuzzy rather than Boolean logic. In the paper of Fuzzy RL, deep learning is applied for training of the autonomous racing car, mapping the input (velocity, steering angle, and distance to centre of lane) to the actions (velocity and steering angle) using the fuzzy rule-based reward system. Due to this combination of two learning systems, the reward signal is easy to understand while the steering system allows complex behavior thanks to complexity of the actor's neural network used for steering and velocity control.

Fuzzy control has been used for flight control in simulations for an unmanned aerial vehicle (UAV) [66], quadrotor [67] and helicopter [68], and has also been proven to be able to control a real-life helicopter during a flight test [69]. These simulations and flight tests show promising results, although stability and robustness is not always guaranteed. In comparison to DRL, fuzzy control does not possess the same level of adaptability, but fuzzy logic can be an adequate approach for inserting human knowledge into the reward signal [65].

4.3.2. REWARD DECOMPOSITION

METHODOLOGY

As explained in Chapter 2, reward signal design is vital to successfully solving a RL problem, as the desired behavior is reflected through this function. A RL problem can appear to be solved when an agent is successively achieving high rewards, however achieving a high score is no guarantee for desired decision making of the agent. Only when the reward function is designed properly will the agent show the wanted behavior. This reward function consists of either positive rewards for reinforcing desired behaviors, negative rewards or penalties for preventing undesired behaviors, or a mix of both positive and negative rewards. A commonly applied approach for reward function design is reward shaping, where the agent is given rewards based on prior knowledge of the domain. Through this reward shaping, the agent receives the highest return by transitioning into the desired state, allowing the agent to receive continuous feedback rather than a *sparse* reward [70].

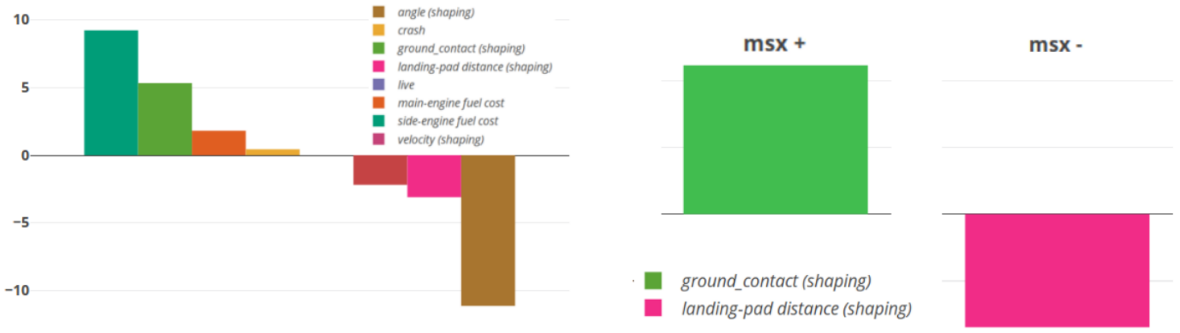
The reward signal can comprise multiple sources of feedback. An example of this for an engineering problem could be gaining positive reward for steering through a target, but simultaneously receiving negative reward for consuming fuel. The authors in [71] make use of reward signals comprising of multiple sources to explain the decisions of the RL agent through their approach called *reward decomposition*. Typically, the rewards originating from the multiple sources are summed to a scalar value, which is passed to the agent for learning. However, as only the sum of the components is known to the agent and not the individual contributions, information about the decision-making is lost. For example, in Q-learning it can only be observed which action is preferred according to policy, but not what are positive or negative motives for awarding a certain Q-value to an action.

Through reward decomposition the decomposed reward signal is exposed in the form of a vector to the agent for improved explainability. This decomposition has been studied more often for the benefit of improved learning speed, but in [71] the decomposition is applied purely for improved explainability. The agent is still trained to maximize the total reward signal over time, but by conveying the reward signal as a vector, a Q-function can be determined for every reward signal component R_c. These component Q-functions $Q_c^\pi(s,a)$ are summed together for the overall Q-function, as illustrated in Equation (4.3.1). Using the component Q-values, the Q-function can be defined as vector $\overrightarrow{Q}^\pi(s,a)$

$$Q^\pi(s,a) = \sum_c Q_c^\pi(s,a) \tag{4.3.1}$$

In [71], two methods are proposed for using the component Q-values for explaining the agents behavior: *Reward Difference eXplanations* (RDX) and *Minimal Sufficient eXplanations* (MSX). To investigate why a certain action is preferred over another, their vector Q-functions $\overrightarrow{Q}^\pi(s,a_1)$ and $\overrightarrow{Q}^\pi(s,a_2)$ can be compared, resulting in the RDX. Each component of the RDX is either a positive or negative motivation for choosing a_1 above a_2, as illustrated for the Lunar Lander environment in Figure 4.7a. The purpose of MSX on the other hand is to deal with environments containing many reward sources. For these environments it can be hard to distinguish which individual components are the most important reason for choosing one action over the other. The MSX aims to identify a small set of the most critical reasons why a_1 is preferred above a_2, illustrated in

Figure 4.7b. The calculation for choosing this set of components is given in [71].



(a) Reward Difference eXplanations (RDX)

(b) Minimal Sufficient eXplanations (MSX)

Figure 4.7: Two methods for explaining the decisions of the agent using reward decomposition in the Lunar Lander environment [71]

SYNOPSIS

Reward decomposition aids in explaining the actions of an agent by making use of information already present in the environment. Insight into the decision-making based on the reward signals can help in spotting errors relating to the reward function, which is essential for the agent to show the desired behavior. A prerequisite for using reward decomposition for RL explainability is the presence of multiple components in the reward signal, or the ability to split the single channel into multiple components. For example, in [71] the environment is tweaked to extend the single shaping reward into their separate components, such as the angle of the spacecraft and its distance to the landing pad. Having more sources allows for more detailed insights into the decision-making, but at least 2 components are required for using reward decomposition. Furthermore, reward decomposition for explainability has been proven to work for Q-learning, but not for other approaches of RL such as actor-critic.

4.4. POST-HOC EXPLAINABILITY

Post-hoc explainability methods aim to explain AI models that are not inherently interpretable. All selected post-hoc explainability methods are not RL-specific, and originate from SL and/or USL.

Simple models such as linear regression models are easily interpreted by investigation of the model itself. Due to the lack of complexity in these models, it is easy to interpret why a certain output is given by inspection of the feature values and the weights of the models. Additionally, these models can be interpreted by investigation of the model itself, they do not require simplifications or assumptions to explain the decision-making. On the other hand, more complex models such as Bayesian models and especially deep learning models are not interpretable by the model itself, these require an *explanation model*. Ideally this explanation model, also defined as *surrogate model*, allows to give insight into the actual AI model, while still accurately representing the predictions of this AI model. As the machine learning model is not adjusted for explainability but a separate explanation model is created for this goal, methods using explanation models are *model-agnostic*. A schematic representation of the surrogate model structure is presented in Figure 4.8. As indicated in this Figure, the actual model is represented by $f(x)$, and the explanation model using $g(x)$.

An explanation for a simple univariate linear regression model, shown in Figure 4.9a, would explain the whole model. An explanation valid for the whole model is defined as a *global* explanation. An explanation valid around an instance of a model, is defined as a *local* explanation. For non-linear models, as the example shown in Figure 4.9b, a local explanation is not enough to globally describe the model.

ADDITIVE FEATURE ATTRIBUTION METHODS

Explanation methods known as *additive feature attribution methods* apply an explanation model which can be represented as a linear function with binary variables:

$$f(x) \approx g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{4.4.1}$$

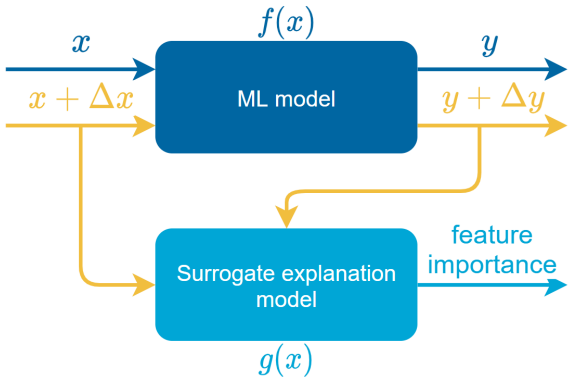
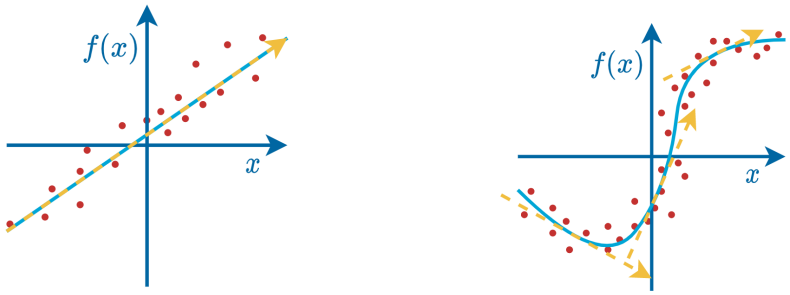


Figure 4.8: Schematic representation of the surrogate model, used in both LIME and SHAP.



(a) A univariate linear model is fully explainable by one local explanation.

(b) For non-linear functions, multiple different local explanations exist.

Figure 4.9: Global and local explanations, where the red dots represent feature data, the blue line is the approximated model $f(x)$, and the yellow dashed lines represent local and/or global explanations $g(x)$.

Where $f(x)$ is the true function, $g(x')$ is the approximation function, $\phi_i \in \mathbb{R}$ is the contribution of feature i, and $x' \in \{0,1\}$ is a binary vector with M elements, equal to the number of features.

For additive feature attribution methods, three desirable conditions can be used for assessing their approximation capabilities [72]:

1. Local accuracy:
The approximation model is locally accurate when the following holds:

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{4.4.2}$$

When compared with Equation (4.4.1) it can be observed that for local accuracy approximation is not enough, $f(x)$ and $g(x')$ should locally be the same function.

2. Missingness:
The missingness property implies the following:

$$x'_i = 0 \;\Rightarrow\; \phi_i = 0 \tag{4.4.3}$$

Hence, when a feature is missing there should be no consequence for the model output. Note that following the definition for local accuracy given in Equation (4.4.2), the contribution to the approximation model would be zero due to the multiplication with x'_i. The definition forces only one solution to the three desired properties, as without the missingness property infinite combinations of ϕ would be possible.

3. Consistency:

Finally, let $f_x(z') = f(h_x(z'))$ and let $z'\ i$ indicate that $z'_i = 0$, then when comparing two models f and f':

$$f_x(z') - f'_x(z'\ i) \geq f_x(z') - f_x(z'\ i) \tag{4.4.4}$$

For all inputs $z' \in \{0,1\}^M$, then the following should hold:

$$\phi_i(f', x) \geq \phi_i(f, x) \tag{4.4.5}$$

This consistency property states that when the model changes such that the contribution of an input increases or stays the same while other contribution decrease, its attribution should not decrease with respect to the attribution of the other features. When this property would be violated, the feature orderings cannot be trusted, even within the same model.

4.4.1. LOCAL INTERPRETABLE MODEL-AGNOSTIC EXPLANATIONS (LIME)

Local Interpretable Model-Agnostic Explanations (LIME), published in 2016, is an explanation technique capable of interpreting any classification model [73]. Because no knowledge of the original model is required and any type of ML model can be explained through LIME, the method is model-agnostic. As stated in the name of this explanation method, LIME interprets the original model locally around a prediction, through creation of a surrogate model. A complex ML model such as a linear model with 100 terms is difficult to interpret globally, but upon inspection of a single prediction it might become apparent that only a few terms are dominant for this prediction.

METHODOLOGY

LIME applies a local explanation model, using the additive feature attribution form presented in Equation (4.4.1). The feature attribution vector ϕ is obtained by solving Equation (4.4.6) [73].

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}\left(f, g, \pi_x\right) + \Omega(g) \tag{4.4.6}$$

Where g is the explanation model out of G possible explanation model types, $\mathcal{L}$ is the fidelity function assessing the accuracy of the fit, and π_x is a measure for the proximity around the prediction instance. Furthermore $\Omega(g)$ is added to the loss function to penalize $g(z')$ and therefore discourage the complexity of the explanation model. The loss function is therefore a balance between the approximation power and the complexity of the explanation model. The complexity of a decision tree model can for example be the amount of branches. LIME makes use of *sparse linear* models for the local explanations, where $g(z') = w_g \cdot z'$. The used fidelity function $\mathcal{L}$ is presented in Equation (4.4.7). In practice, the $\Omega(g)$ function is often applied by choosing the number of features for an explanation.

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z)(f(z) - g(z'))^2 \tag{4.4.7}$$

The weights w_g are determined by sampling around the desired prediction, and minimizing the loss defined in Equation (4.4.6). Through these sampling deviations Δx from the instance x, the surrogate model is formed, as previously illustrated in Figure 4.8. A graphical example of forming this local model for a complex classification model is shown in Figure 4.10, where the blue-pink background represents the complex function f. This function is explained around one instance, illustrated with the bold plus symbol. The local explanation model g is represented by the dashed line. Through the proximity function π_x, samples are weighed based on their distance from the prediction x, illustrated through the difference in size.

STRENGTHS AND WEAKNESSES

As LIME utilizes a surrogate model, any ML model ranging from simple linear models to deep neural networks can be interpreted using this method. This offers great flexibility in the number of applications where LIME can be applied. Another advantage of LIME offering flexibility is that no prior knowledge of the model is required. Furthermore, the fidelity function $\mathcal{L}$ produces a measure indicating the fit of the interpretation with respect to the original function. This allows to assess the reliability of the explanation. Finally, as only

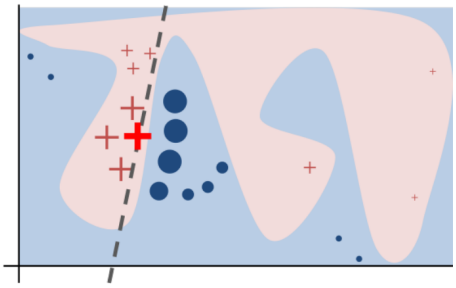


Figure 4.10: Example to explain the local-global difference for LIME [73]. The blue-pink background is the non-linear classification function f, which is approximated using a linear explanation model g around the dashed line.

local samples are used for computation of the weights, LIME is a fast algorithm compared to other post-hoc XAI techniques.

However, using linear models for model approximation has drawbacks. For instance, where the local model represents linearity, such as the instance explained in Figure 4.10, this approximation is valid for a significant part of the complex model. For more complex models however, the linearity assumption will only be correct for a small region around the chosen instance. Additionally, regarding the three desirable properties for feature attribution methods, according to the decision of the fidelity function $\mathcal{L}$, weighing function π_x, and complexity function ω, a choice has to be made between either local accuracy or consistency. Therefore, a model interpretation produced by LIME will never adhere to all three desirable properties. Furthermore, tuning of the proximity function for defining the locality of the explanation is often difficult in practice. Additionally, the perturbations for sampling are chosen randomly following a Gaussian distribution, not taking into account the actual variance of the features. Another major drawback of LIME is its stability. Testing explanations for random forest and gradient boosting trees has exposed three types of uncertainty: the (Gaussian) randomness in selecting the samples, variations due to sampling proximity, and variation in model credibility for different data points [74]. Finally, it has been shown that ML models can be manipulated to be biased, while appearing to have no bias when interpreting this model using LIME [75].

SYNOPSIS

The ability for post-hoc interpretation of any ML model shows potential for using LIME for explainable RL for flight control. As a surrogate model is utilized for the explanations, the original model requires no compromises of complexity for explainability. However, LIME is lacking both accuracy and stability, which are highly desired to accurately explain the inner workings of a RL algorithm. Regarding the three desirable properties for additive feature attribution methods, a choice is required between either local accuracy or consistency. For a compromise of accuracy LIME does however offer low computation times for explanations. This creates interesting opportunities for real-time analysis, such as live explanation of a RL agent controlling an aircraft giving the pilots insight into the decision-making process, or live feedback of an unmanned UAV where the operators can interpret the choices being made by the RL algorithm.

4.4.2. SHAPLEY ADDITIVE EXPLANATIONS (SHAP)

METHODOLOGY

Shapley values, originating from game theory in 1951, are the only set of values adhering to a set of axioms similar to the three desirable conditions for additive feature attribution methods [72]. (*Lundberg & Lee*) prove that SHapley Additive exPlanations (SHAP), utilizing Shapley game theory for determination of the feature contributions, is the only additive feature attribution method capable of producing explanations which simultaneously adhering to all three desired properties for additive feature attribution methods: local accuracy, missingness, and consistency.

These Shapley values can be used to allocate value to individual players in a collaborative turn-based game. The Shapley value is defined as the average expected marginal contribution of a player during a game, when considering all possible combinations of how the game could be played. Table 4.1 shows an example of this calculation, for a game with two players. The total contribution of the two players is 20 for both possibilities of games, but their contribution depends on the order in which the game is played. By taking the average of the marginal contributions for both players, the fair contribution of each player is determined.

Table 4.1: Example of Shapley value calculation for a game with 2 players.

Order	Probability	Total contribution	Marginal contribution A	Marginal contribution B
AB	$\frac{1}{2}$	20	$v(\{A\}) = 8$	$v(\{A\}) = v(\{AB\}) - v(\{A\}) = 12$
BA	$\frac{1}{2}$	20	$v(\{B\}) = v(\{AB\}) - v(\{B\}) = 6$	$v(\{B\}) = 14$
Shapley Value			$\phi_A = \frac{1}{2} \cdot 8 + \frac{1}{2} \cdot 6 = 7$	$\phi_B = \frac{1}{2} \cdot 12 + \frac{1}{2} \cdot 14 = 13$

As SHAP is a feature attribution method, the following equation is used to create a single explanation:

$$f(x) \approx g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{4.4.8}$$

In this equation, ϕ_0 is the expected model output when one would have no knowledge about the feature values, defined as $E\left[f(x)\right]$. In SHAP, this value is calculated as the average of all predictions. This value is defined as the *base rate* or *baseline*. The *feature values*, or *SHAP* values, ϕ_i for $i = \{1, 2, ..., M\}$ are added to this base rate, resulting in the final model output. As an analogy, the SHAP values can be seen as forces summing towards a net force.

Waterfall plot:
Figure 4.11 shows an example of a SHAP waterfall plot. The waterfall plot can be used to illustrate the explanation for an individual prediction, also defined as an *instance*. The plot shows how the actual model output is obtained by adding the feature effects at the specified instance to the base rate. This allows to observe the magnitude of the feature effects, and whether they are positive or negative. The explanation of the prediction is illustrated by starting at the base rate $E\left[f(x)\right]$ at the bottom. From this baseline, the marginal contributions from all features are added, resulting in the actual model output. The waterfall plot shown in Figure 4.11 originates from a RL control example, where the model output is used to control the longitudinal acceleration of a car[76]. Features having a positive contribution to the model output are colored red, while negative contributions are colored blue. Waterfall plots are useful to illustrate a single prediction, but it is often hard to extract global explanation information from this plot. For a more global understanding of the feature effects, summary plots can be used.

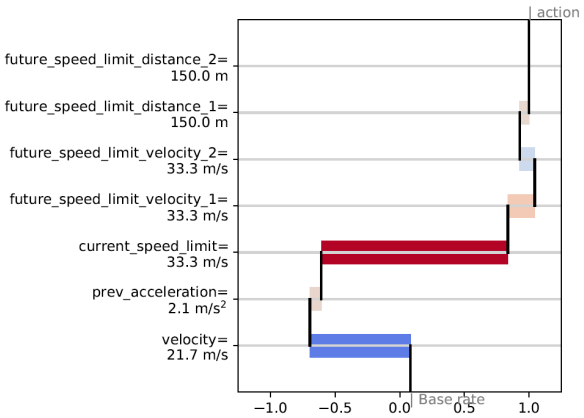


Figure 4.11: Example of a SHAP waterfall plot, where the model output is explained using the baseline and feature effects for a given action. This example illustrates an explanation for longitudinal acceleration of a car [76].

Summary plot:
The SHAP summary plot shows the explanations for all predictions simultaneously. These plots allow investigation of the relation between feature value and SHAP value, and of feature importance. These plots are generated for every output of the specified ML model. Hence, for a NN with 4 outputs, 4 separate SHAP summary plots are generated. For every instance $\mathbf{x}$, a dot is plotted for every feature, along the x-axis corresponding to its SHAP value. These dots are colored according to their feature value, from low to high. Overlapping points in the summary plot are stacked in the y-direction, giving an idea of the distribution of SHAP values for a given feature. The order of the features is determined by the sum of the absolute feature effect, shown

in Equation (4.4.9), ordered from high to low.

$$I_j = \sum_{i=1}^{n} \left| \phi_j^{(i)} \right| \tag{4.4.9}$$

By comparison of a feature's color gradient and the SHAP values, the relation between feature value and SHAP value can be observed. Furthermore, by investigation of the range of the SHAP values for a given feature, and the distribution of these SHAP values, the importance of this feature for the specified action can be determined. Figure 4.12 shows an example of a SHAP summary plot for a supervised learning model, estimating a person's biological age. As the SHAP value is plotted along the x-axis, and the feature value is shown using the color scale, the relation between SHAP value and feature value is easily observed. Additionally, the feature importance is easily observed, not only by the ordering of the features based on Equation (4.4.9), ordered from high to low, but also by the range between the smallest and largest SHAP value for a given feature, and the distribution of SHAP values.

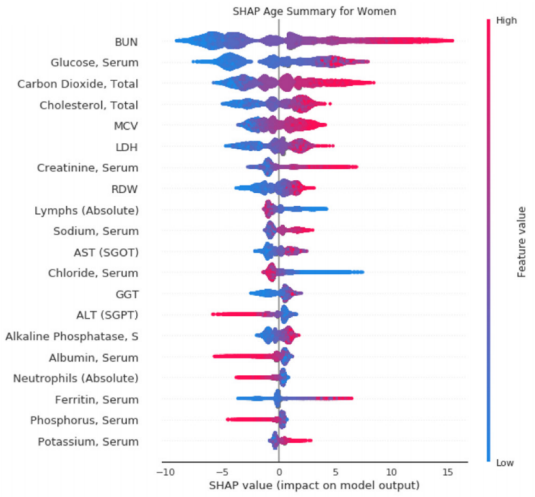
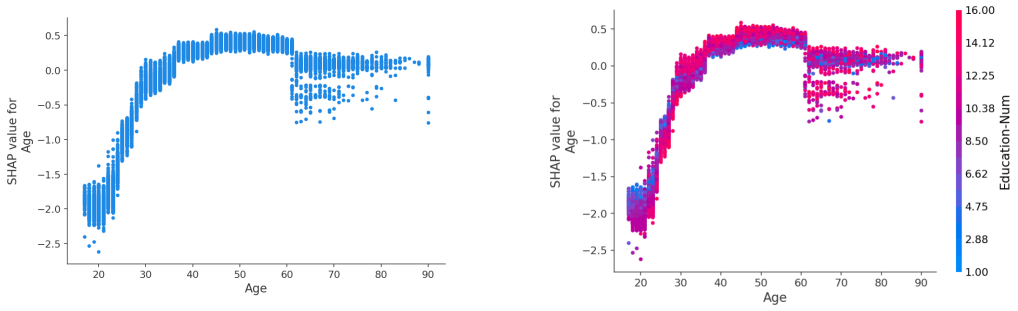


Figure 4.12: Example of a SHAP summary plot, where the x-axis indicates SHAP value and the color represents feature value. The model output of this example is a person's estimated biological age, based on concentrations of various chemical compounds present in the human body [77].

Dependence plot:
The relation between feature value and SHAP value can roughly be determined using the SHAP summary plot. However, the color scale indicating the feature value cannot be read quantitatively. To better observe the relation between feature value and SHAP value, the dependence plot can be used. Figure 4.13a shows an example of a simple SHAP dependence plot, with the feature value on the x-axis and the corresponding SHAP value on the y-axis. Figure 4.13b shows a dependence plot where a second feature is added, indicated using the color scale. This dual feature plotting allows for the interaction between 2 features to be investigated. If vertical coloring patterns show up, like in Figure 4.13b, an interaction effect is present between the 2 features.



(a) Simple dependence plot

(b) Dependence plot with second feature for analyzing interactivity

Figure 4.13: Example SHAP dependence plots, where the model output predicts the income of individuals in the 90s [78].

RL-SHAP diagram:

The SHAP plots shown so far are helpful for giving local and global explanations for a ML model, but lack insight into the RL-specific decision making process over time. For this, RL-SHAP diagrams can be used. Figure 4.14 is an example of a RL-SHAP diagram for the same longitudinal acceleration used in Figure 4.11 [76]. The black dashed line in the RL-SHAP diagram corresponds to the time step where the waterfall plot is generated, shown in Figure 4.11.

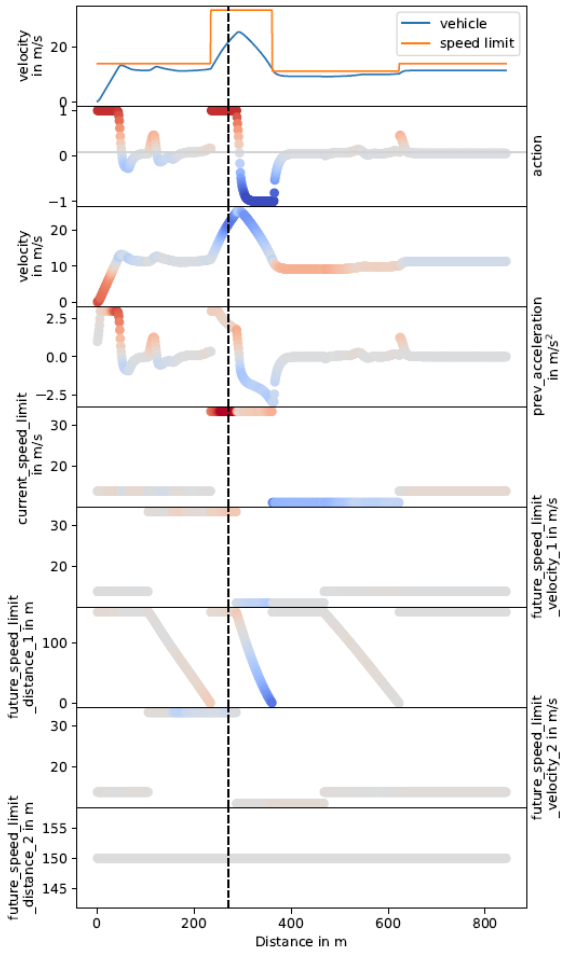


Figure 4.14: Example of a RL-SHAP diagram, for longitudinal acceleration control of a car [76]. The first subplot shows the velocity and speed limit over time, the second subplot is the actor NN output, and the other 7 subplots are the NN inputs. The horizontal grey line of the action is the SHAP baseline, and the vertical dashed line corresponds to the waterfall plot shown in Figure 4.11.

The RL-SHAP diagram consists of multiple subplots. The top subplot shows relevant information about the environment and the goal. In the case of Figure 4.14, the velocity of the car and the speed limit are plotted against the driven distance. The second subplot shows the actor's NN model output, the longitudinal accelerator, plotted against the driven distance. The other subplots show the feature values. Using the color scale, the feature's SHAP values over time are plotted, where blue represents negative SHAP values and red represents positive SHAP values. As indicated with the black dashed line in Figure 4.14, the RL-SHAP diagram is essentially a series of waterfall plots stacked together, to illustrate the SHAP values over time.

STRENGTHS AND WEAKNESSES

Explanations using SHAP follow the three desirable features for additive feature attribution methods. In contrast to LIME, no trade-off between local accuracy and consistency is required. Therefore, the quality of the explanations provided by SHAP are of higher accuracy. Furthermore, SHAP can not only be used for local explanations, but also provides global insights of how the model output is influenced by the input features. Using the SHAP summary plot for example, feature importance and SHAP correlation can be investigated. Additionally, like LIME, SHAP provides explanations for any kind of ML model, increasing flexibility and removing the need for model knowledge. The advantage of SHAP over LIME is also reflected in the amount of scientific research performed into using these methods for explainable AI. Figure 4.15 shows the number of publications for *LIME + "Machine Learning"* and *SHAP + "Machine Learning"* retrieved from Scopus on June 29 2021. Although SHAP has only been around for the last 3 years, the amount of research using this method has already surpassed that of LIME, which has been around for 5 years.

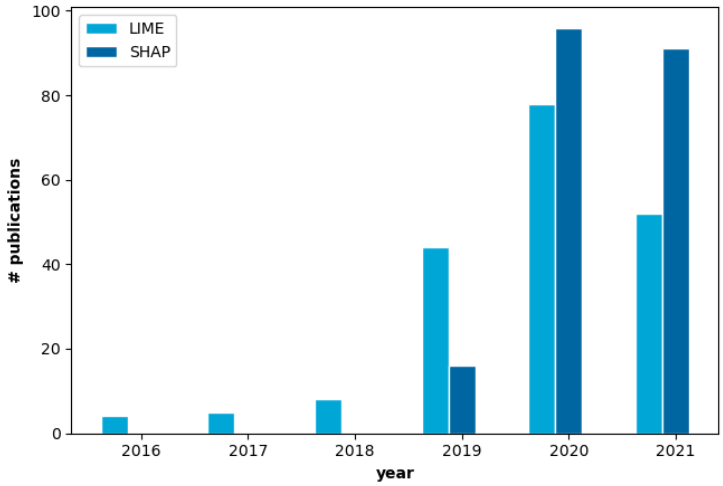


Figure 4.15: Number of publications for *LIME + "Machine Learning"* and *SHAP + "Machine Learning"*. Data retrieved from Scopus on June 29 2021.

The main disadvantage of SHAP is the computational expense. Compared to LIME, creating explanations using SHAP can be time-consuming. Another disadvantage of SHAP is that feature dependence is ignored for sampling the coalition vector z'. Random combinations of feature presence and absence are required for sampling to determine the SHAP values, but these coalitions may be unrealistic. The consequence of this is that when features are dependent, unlikely coalitions will contribute too much to the estimation of the SHAP values. This problem is inherent to statistical methods using permutations for sampling. Finally, SHAP explanations can be misinterpreted. It can be easy to assume that the SHAP value is equal to the difference of the model output when the feature is removed. The SHAP value is however the difference between the average model prediction and the actual prediction. This problem can, however, easily be mitigated by clearly stating how these values should be interpreted.

SYNOPSIS

SHAP shows great potential for creating explanations for flight control using RL, as it is the state-of-the-art in ML explainability and because of its post-hoc explanation ability, the local and global explanation methods, and the high explanation accuracy. Like LIME, SHAP allows post-hoc interpretation of any ML model due

to the use of a surrogate explanation model. Therefore, no compromises in complexity of the model are required. Furthermore no prior model knowledge is required and the to-be explained model can be easily changed, allowing flexibility for researchers testing different RL models or model settings for flight control. The advantages mentioned so far also apply to LIME, however the accuracy of the explanations provided by SHAP are of higher quality. This high quality does however require longer computation times than LIME. For the purpose of this research however, high accuracy is deemed more important than computation time, as for researchers an in-depth analysis is often preferred in order to assess the quality of RL models for flight control. Additionally, this research does not consider real-time applications. Therefore, computation time is not as much of a concern.

Finally, SHAP has been proven to provide useful explanations for not only supervised and unsupervised learning, but also for RL applications. On the other hand, LIME has not been applied for RL problems in the academic world [1]. SHAP on the other hand has been used for at least 5 RL applications, such as the explanation of a learned strategy for traffic light control [79] [2].

One recent example of how SHAP is used for explaining the decision making process of a RL agent for a control problem is the explanation of longitudinal acceleration of a self-driving car, where the RL-SHAP plots are introduced [76]. Figure 4.11 and Figure 4.14 are from this paper. In another research where SHAP is used for explaining RL for control, a flight application is used. In this research, RL is used to control the flight of an unmanned aerial vehicle through an obstacle course.

4.5. CONCLUSIONS

Target audience is an important factor in the subjective field of explainability, as the type of audience influences the type of explanation and the desired level of explanation detail. Due to low technology readiness level of current RL control applications for flight control, the main target audience of this research is RL researchers. Explainable AI methods can aid RL researchers in better understanding their algorithms and notice possible flaws, reducing the overall development time and potentially resulting in improved algorithms. RL researchers can use these XAI methods in their design cycle, to assess their algorithms and update them using the generated insights.

Accuracy is typically one of the key design drivers for RL applications in flight control due to the complex and non-linear dynamics encountered in aerospace. This demand for accuracy limits the possibilities for transparent design techniques, such as reward decomposition and explainable fuzzy RL, as these require compromises in accuracy for increased interpretability. However, the other class of XAI, post-hoc explainability techniques, are developed specifically to interpret already designed ML models. Therefore post-hoc explainability methods will be used in this research, as these methods allow maximum accuracy of the to-be explained ML model. Another advantage of this suite of explainability techniques is that RL researchers can use these methods for already existing RL flight control algorithms, requiring no adaptations to their work.

From the analyzed post-hoc explainability techniques, SHAP shows the most potential for explaining RL applications in aerospace. SHAP uses Shapley values from game theory to calculate feature importance, offering insightful local and global explanations. As the target audience for this research is researchers, accuracy of explanations is deemed more important than computation time, resulting in the choice of SHAP for this research. For other research subjects, such as real-time explanations, LIME could be more suitable. An example of this is live explanations of a RL agent controlling an aircraft, to allow the pilot to monitor the decisions being made during flight. In the preliminary analysis, SHAP will be used to create explanations for classic and relatively simple RL problem, the lunar lander. This is a commonly studied RL problem encountered in multiple academic publications, and allows both discrete and continuous control.

[1] Based on academic research available through Scopus - Query: *LIME + reinforcement* in June 2021
[2] Based on academic research available through Scopus - Query: *SHAP + reinforcement* in June 2021

5 | Preliminary Analysis

This chapter presents the preliminary analysis of this thesis. The goal of this preliminary analysis is to explore the possibilities and limitations of the SHapley Additive exPlanations methodology to increase the transparency of RL for flight control. In this preliminary analysis, a commonly used environment is applied to train both discrete and continuous control RL agents. The decision-making processes of these agents are then explained using the techniques available through SHAP. First, the methodology of this analysis is presented in Section 5.1, including the used environment and the discrete and continuous control algorithms. The results of the discrete control analysis are then presented in Section 5.2. Finally, the results of the continuous control analysis are shown in Section 5.3.

5.1. Preliminary Analysis Setup

5.1.1. Environment

The lunar lander environment, part of Open AI's Gym [80] RL testing toolbox, is inspired by the set of classic arcade lunar landing games from the 70s. In this environment the goal is to land the lunar spacecraft on the randomly generated lunar surface on the landing pad between the two yellow flags, located at coordinates $(0,0)$, as illustrated in Figure 5.1.

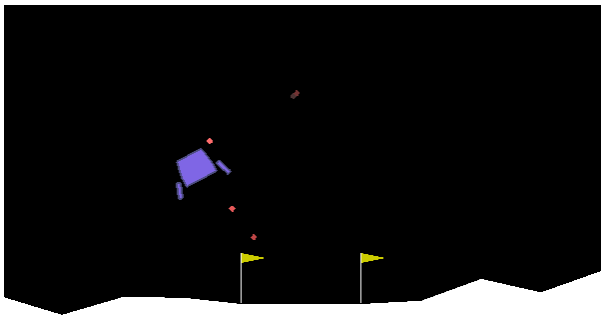


Figure 5.1: Screenshot from Open AI's lunar lander environment, including the purple spacecraft and the landing pad between the two yellow flags [80]. The terrain is generated randomly for every episode.

State space:
The following continuous states are available: the spacecraft's position (x, y), the spacecraft's velocity in the inertial frame (V_x, V_y), the spacecraft's angle with respect to the positive vertical axis θ, the spacecraft's rotational velocity ω. Furthermore, 2 discrete states are used to indicate ground contact of the 2 landing legs $(\text{leg}_L, \text{leg}_R)$. The 2 discrete variables are binary, where a 0 indicates no ground contact while a 1 indicates contact between the leg and the lunar surface. At the start of each episode, the agent is initialized at exactly the same position near the top and centre of the screen. However, to discourage trivial solutions a force with a random magnitude and direction is applied to the main body of the spacecraft during initialization. The state space, including upper and lower limits, is summarized in Table 5.1.

Action space:
The action space of the lunar lander environment can be either discrete or continuous. In the discrete action space, the following four actions are possible:

- a_0: do nothing
- a_1: fire left thruster, causing clockwise rotation
- a_2: fire main thruster

Table 5.1: State space of the lunar lander environment.

State	Description	Unit	Range
x	Horizontal position	m	[-1.0, 1.0]
y	Vertical position	m	[-0.2, 2.0]
V_x	Horizontal velocity	m s^{-1}	[-∞, ∞]
V_y	Vertical velocity	m s^{-1}	[-∞, ∞]
θ	Angle with respect to positive y-axis	rad	[-π, π]
ω	Angular velocity	rad s^{-1}	[-∞, ∞]
leg_L	Left leg contact	[-]	[0,1]
leg_R	Right leg contact	[-]	[0,1]

- a_3: fire right thruster, causing counter-clockwise rotation

For the continuous lunar lander, the action space consists of the following:

- a_0: main thruster $\in [-1.0, 1.0]$, where the thruster is only operational in the $[0.0, 1.0]$ domain. For the other values the thruster is idling.
- a_1: left-right thrusters $\in [-1.0, 1.0]$, where the right thruster (counter-clockwise rotation) is operational for $a_1 \in [-1.0, -0.5]$, the left thruster (clockwise rotation) for $a_1 \in [0.5, 1.0]$. Both engines are idling for $a_1 \in < -0.5, 0.5 >$.

The action space of the continuous agent is illustrated in Figure 5.2.

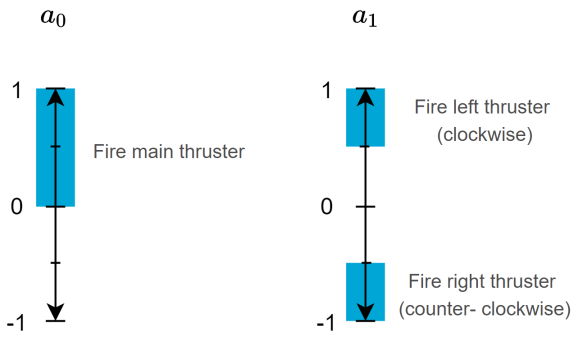


Figure 5.2: Action space for the continuous lunar lander environment. The thrusters are only active on the blue domains.

Reward Function:
The reward signal is designed using reward shaping, the reward design method described in Section 4.3.2. Through this function, positive reward is awarded by moving towards the goal at coordinate $(0,0)$. Additionally, minimizing the spacecraft's velocity results in positive reward. Minimizing both the distance to the goal and the spacecraft's velocity results in 100 to 140 points, depending on the random initial condition. Furthermore, through the reward shaping function the agent is encouraged to minimize θ. Additionally, a reward of 10 points is given for each landing leg making contact with the lunar surface, to discourage flying away after landing. The episode is terminated by either a successful landing where the spacecraft is stable on the surface, or when crashed. A crash is defined as the scenario where the main body of the spacecraft makes contact with the lunar surface, or when the spacecraft flies out of the horizontal coordinate bounds ($[-1.0, 1.0]$). For a successful landing 100 points are awarded, while for a crash -100 points are given. Finally, 0.3 points are deducted for every time step where the main thruster is turned on to discourage fuel usage. The reward function is summarized in Equation (5.1.1)

$$r_{total} = r_{shaping_{xy}} + r_{shaping_V} + r_{shaping_\theta} + r_{leg_L} + r_{leg_R} + r_{crash} \tag{5.1.1}$$

5.1.2. DISCRETE CONTROL - ADVANTAGE ACTOR CRITIC (A2C)

For discrete control of the lunar lander environment, the *Advantage Actor Critic* (A2C) algorithm is applied. This algorithm shows many similarities with the Q actor-critic, presented as Algorithm 2.5 in Section 2.5. A schematic overview of this algorithm is shown in Figure 5.3. As A2C is an actor critic algorithm, the actor is responsible for determining the action, while the critic estimates the value function. As shown in Figure 5.3, the actor's parameters θ and the critic's parameters $\mathbf{w}$ are updated by backpropagation, using the value function for both updates.

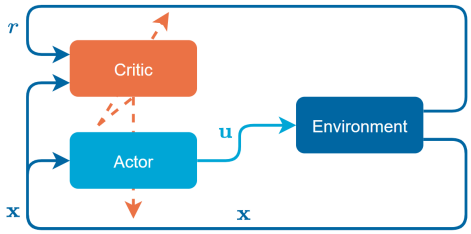


Figure 5.3: Schematic overview of the (advantage) actor critic structure. The dashed lines indicate the backpropagation paths for updating both the critic and actor.

Instead of only using the estimated value $V(\mathbf{x};\mathbf{w})$ for this update step, the *advantage* is utilized. The advantage in A2C is defined as in Equation (5.1.2). By using the difference between the return and the estimated value function, the gradients of the actor and the critic is either enhanced for positive advantage values, or reversed for negative values, resulting in expedited learning and reduced variance.

$$A(\mathbf{x}_i;\mathbf{w}) = R - V(\mathbf{x}_i;\mathbf{w}) \tag{5.1.2}$$

As a Monte Carlo method, A2C calculates the return for every time step at the end of each episode. Then, the gradients for the actor $d\boldsymbol{\theta}$ and critic $d\mathbf{w}$ are accumulated by stepping through every time step. Finally, using the accumulated gradients, the parameter vectors $\boldsymbol{\theta}$ and $\mathbf{x}$ are updated. This sequence of steps for training the agent is shown in more detail in the pseudocode of Algorithm 5.1.

Algorithm 5.1: Pseudocode for training of the Advantage Actor Critic (A2C) algorithm

initialize $\mathbf{x}, \boldsymbol{\theta}, \mathbf{w}$ at random; where $\boldsymbol{\theta}$ is for the policy, and $\mathbf{w}$ for the value function ;
reset gradients: $\boldsymbol{\theta} \leftarrow 0$ and $\mathbf{w} \leftarrow 0$;
for *each episode* **do**
  repeat
    sample action $\mathbf{u}_t \sim \pi_\theta(\cdot|\mathbf{x}_t)$;
    apply $\mathbf{u}_t$, observe $\mathbf{x}_{t+1}$ and r_{t+1} ;
    set $t \leftarrow t+1$;
  until $s_{terminal}$ *or* $t > t_{max}$;
  for $i \in \{t-1, t-2, ..., 0\}$ **do**
    calculate return $R \leftarrow r_i + \gamma R$;
    accumulate $\boldsymbol{\theta}$ gradient: $d\boldsymbol{\theta} \leftarrow d\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{u}_i|\mathbf{x}_i;\theta) \cdot (R - V(\mathbf{x}_i;\mathbf{w}))$;
    accumulate $\mathbf{w}$ gradient: $d\mathbf{w} \leftarrow d\mathbf{w} + \frac{\delta V(\mathbf{x}_i;\mathbf{w})}{\delta \mathbf{w}} \cdot (R - V(\mathbf{x}_i;\mathbf{w}))$
  end
  update $\boldsymbol{\theta}$ using $d\boldsymbol{\theta}$;
  update $\mathbf{w}$ using $d\mathbf{w}$;
end

For the preliminary analysis, NNs are used for approximation of both the policy and value function. The neural structure is illustrated in Figure 5.4. The actor and critic network both have one hidden layer, and share these neurons together for expedited learning. This means that the biases of these neurons are shared, and that weights between the input layer and the hidden layer are similar for the actor and critic. The actor network has 4 output neurons, one for every action, while the critic has one output neuron for estimation of $V(\mathbf{x})$

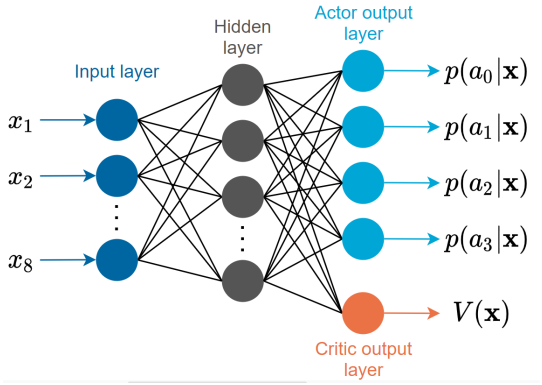


Figure 5.4: Neural network structure of both the actor and the critic for A2C. The critic and the actor share the same neurons in their hidden layer.

As the A2C algorithm is used for the discrete lunar lander environment, the output of the actor should be used to choose 1 of the 4 possible actions at every time step. Therefore, the output of the actor is the probability distribution for the 4 actions, using the *softmax* function as the activation function of the actor output layer. This function, shown in Equation (5.1.3) transforms a vector of n elements into a vector summing up to 1, resulting in a set of probabilities. Negative or small values receive a low probability while large values will receive a large probability. Finally, the used hyperparameters for the A2C agent are presented in Table 5.2, and the corresponding training convergence plot is presented in Appendix A.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{5.1.3}$$

Table 5.2: Hyperparameters of the A2C agent resulting in solving the discrete lunar lander environment.

			Hidden layer		**Output layer**	
	α	γ	n neurons	Activation function	n neurons	Activation function
Critic	0.001	0.99	256	ReLU	4	softmax
Actor	0.001	0.99			1	-

5.1.3. CONTINUOUS CONTROL - DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

The applied algorithm for the continuous control lunar lander environment is the *Deep Deterministic Policy Gradient* (DDPG) algorithm. This algorithm provides continuous control and applies two strategies from previously developed algorithms: *target networks* from Deep Q-learning, and *experience replay*. Target networks are developed to prevent *catastrophic forgetting*, defined as loss of performance over time of a RL agent. RL algorithms applying bootstrapping are prone to instability, due to the use of estimations in the update equations. By applying one or more target networks, this instability can be reduced. In DDPG, both the actor and the critic have a target network. The second adopted concept, experience replay, allows to store interaction data in the *replay buffer*, in order to not only learn from all past experience rather than only the latest time step. The general structure of the DDPG algorithm is visualized in Figure 5.5.

In DDPG, both the actor target network and the critic target network are updated every time step, using a soft-update where the parameters of the target networks are pushed slightly in the direction of the local actor and critic network using the hyperparameter τ. These two update steps are shown in Equation (5.1.4), where $\boldsymbol{\theta}$ represents the parameter vector of the actor NN, $\boldsymbol{\theta}'$ the parameter vector of the actor target NN, $\mathbf{w}$ the parameter vector of the critic NN, and $\mathbf{w}'$ the parameter vector of the critic target NN.

$$\begin{aligned} \mathbf{w}'_{t+1} &= \tau \cdot \mathbf{w}_t + (1-\tau)\mathbf{w}'_t \\ \boldsymbol{\theta}'_{t+1} &= \tau \cdot \boldsymbol{\theta}_t + (1-\tau)\boldsymbol{\theta}'_t \end{aligned} \tag{5.1.4}$$

To stimulate exploration, DDPG agents add noise to the action output. Due to this noise, extra large regions

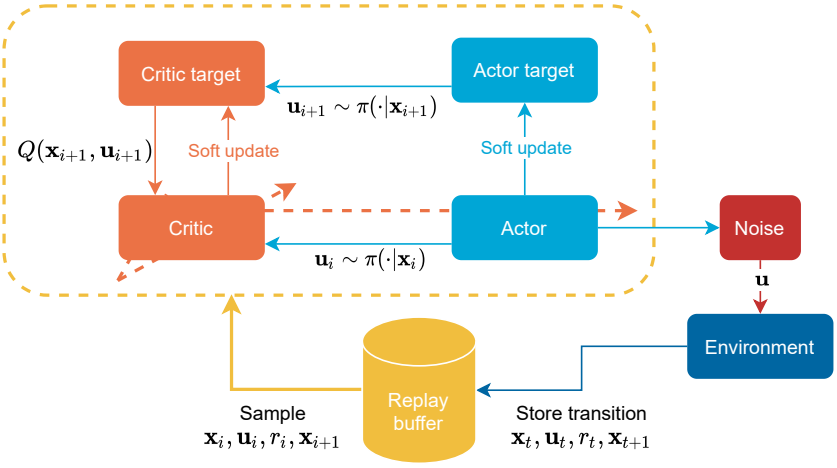


Figure 5.5: Global structure of the DDPG algorithm, where the dashed lines indicate the backpropagation paths.

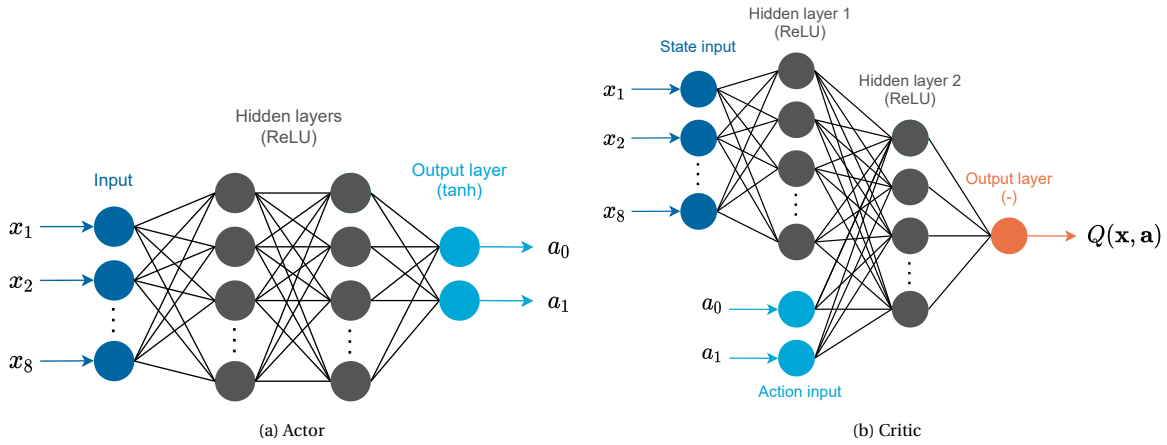
of the state space are explored. Typically for DDPG, the noise is sampled from a *Ornstein–Uhlenbeck process* [81], described in Equation (5.1.5). In this process, $\eta(t)$ is commonly white noise with $\mu = 0$ and $\sigma = 1$.

$$\frac{\partial x_t}{\partial t} = -\theta \cdot x_t + \sigma \cdot \eta(t) \tag{5.1.5}$$

To stimulate state space exploration at the start of training, but stimulate exploitation later in training when the agent has become more intelligent, epsilon decay is used for the noise addition. At start of training, $\epsilon = 1$, and every time step ϵ is reduced with ϵ_{decay} as shown in Equation (5.1.6). The added noise from the Ornstein–Uhlenbeck process is multiplied with this ϵ, resulting in gradually decreasing influence from this noise during training.

$$\epsilon_{t+1} = \epsilon_t - \epsilon_{\text{decay}} \tag{5.1.6}$$

The NN structures of the actor and critic are illustrated in Figure 5.6. The structures of the target NNs are equal to what is displayed in Figure 5.6. The hyperparameters for these NNs are summarized in Table 5.3.



(a) Actor (b) Critic

Figure 5.6: Neural network structures for the DDPG agent. The target networks follow the exact same structure.

Table 5.3: Hyperparameters of the DDPG agent networks resulting in solving the continuous lunar lander environment.

		Hidden layers			
Output layer					
	α	n_{neurons}	Activation function	n_{neurons}	Activation function
Actor	10^{-4}	128	ReLU	2	tanh
Critic	10^{-3}	128	ReLU	1	-

The pseudocode for training the DDPG agent is presented in Algorithm 5.2. The n_{update} hyperparameter in the pseudocode defines the number of samples drawn from the replay buffer for learning. The hyperparameters mentioned in this pseudocode and other general parameters are summarized in Table 5.4, and the training convergence plot is presented in Appendix A.

Table 5.4: General hyperparameters for the DDPG agent, used to solve the continuous lunar lander environment.

Hyperparameter	**Value**
batch size	256
buffer size	10^6
ϵ	1
ϵ_{decay}	10^{-6}
γ	0.99
T_{learning}	20
noise μ	0
noise σ	0.2
noise θ	0.15
optimizer	Adam[26]
τ	10^{-3}
n_{update}	10

5.2. RESULTS OF DISCRETE CONTROL PRELIMINARY ANALYSIS

In this section the results of the discrete control environment preliminary analysis are presented, by explaining the trained strategy of the discrete agents. The main tool for explaining this strategy is SHAP. However, before interpreting the input-output mapping using SHAP values it is useful to understand the input space and outputs of the NN. Therefore, in Section 5.2.1 the input space is analyzed for both an untrained and trained agent. Then, in Section 5.2.2 the NN output is presented, including an explanation of the agent's strategy using these outputs. Finally, in Section 5.2.3 the input-output mapping is explained using SHAP values and multiple types of SHAP plots.

5.2.1. INPUT ANALYSIS

As the main focus of this preliminary analysis for increasing RL explainability is the input-output mapping of the actor's NN, insight into the encountered inputs during the simulated episodes helps understanding the agent's choices. Figure 5.7 shows the encountered continuous states during 10 episodes for an untrained and trained agent.

From Figure 5.7 several conclusions can be drawn:

- Both the trained and untrained agent show similar minimum and maximum values for the vertical coordinate. As each episode is initialized near the maximum y-coordinate 1.5, and the ground is at 0, this makes sense. The slightly negative values are explained due to variations in the terrain. At some instances the spacecraft lands outside of the landing pad, causing it to slide down into a moon crater.
- The trained agent encounters lower vertical velocities. As the landing is better controlled, the agent is not falling uncontrollably towards the ground.
- Significant differences are also observed in V_x. As the trained agent performs controlled landings, the horizontal movement is minimized.

Algorithm 5.2: Pseudocode for training of the Deep Deterministic Policy Gradient (DDPG) algorithm

Initialize actor NN using randomly generated $\boldsymbol{\theta}$ and critic network using randomly generated $\mathbf{w}$;
Initialize actor target network setting $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$ and critic target network $\mathbf{w}' \leftarrow \mathbf{w}$;
Initialize replay buffer $\mathcal{R}$;
Initialize random process $\mathcal{N}$ for exploration ;
for *episode* $= 1, 2, ..., M$ **do**
  Initialize episode and get $\mathbf{x}_0$;
  Initialize $t \leftarrow 0$;
  for $t = 0, 1, ..., T$ **do**
    Sample action with noise $\mathbf{u}_t \sim \pi_\theta(\cdot|\mathbf{x}_t) + \epsilon \cdot \mathbf{N}_t$;
    Apply $\mathbf{u}_t$, observe $\mathbf{x}_{t+1}$ and r_{t+1} ;
    Store $\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1}$ in $\mathcal{R}$;
    if t % $T_{learning} = 0$ **then**
      for $n = 0, 1, ..., n_{update}$ **do**
        Sample N transitions $\mathbf{x}_i, \mathbf{u}_i, r_i, \mathbf{x}_{i+1}$ from $\mathcal{R}$;
        Use actor target network for sampling next action: $\mathbf{u}_{i+1} \sim \pi_{\boldsymbol{\theta}'}(\cdot|\mathbf{x}_{i+1})$;
        Use critic target network for sampling $Q^{\mathbf{w}'}(\mathbf{x}_{i+1}, \mathbf{u}_{i+1})$;
        Compute target $Q_{\text{tar}} \leftarrow r_i + \gamma \cdot Q^{\mathbf{w}'}(\mathbf{x}_{i+1}, \mathbf{u}_{i+1})$;
        Update critic using MSE loss $L \leftarrow (Q_{\text{tar}} - Q^{\mathbf{w}}(\mathbf{x}_i, \mathbf{u}_i))^2$;
        Sample action from actor network $\mathbf{u}_i \sim \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x}_i)$;
        Update actor using loss $L \leftarrow -Q^{\mathbf{w}}(\mathbf{x}_i, \mathbf{u}_i)$;
        Update actor target network $\boldsymbol{\theta}' \leftarrow \tau \cdot \boldsymbol{\theta} + (1 - \tau) \cdot \boldsymbol{\theta}'$;
        Update critic target network $\mathbf{w}' \leftarrow \tau \cdot \mathbf{w} + (1 - \tau) \cdot \mathbf{w}'$
      end
    end
  end
end

- Similarly, the encountered angle θ and rotational velocity ω of the spacecraft controlled by the trained agent are significantly lower. The amount of outliers for ω is remarkable compared to the rest of the encountered states, for both agents. These outliers of ω are encountered when the spacecraft lands outside of the landing pad against the slope of a lunar hill, or in a trench. This rare phenomena causes the spacecraft to quickly tip over, inducing a high $|\omega|$.

5.2.2. OUTPUT ANALYSIS

Similar to understanding the encountered input space as presented in Section 5.2.1, knowledge of the actual model output helps to explain the decisions of the agents and benefits interpretation of the SHAP analysis. Figure 5.8 shows the simulation of one episode for both the untrained and trained agent. As illustrated in Figure 5.8a, the 4 action probabilities for the untrained agent are equal and about constant throughout the episode. Even though the 8 inputs changes considerably during the episode, their contribution to the model output is negligible due to the lack of optimized weights and biases.

Figure 5.8b shows some insights into the learned behavior of the trained agent. At the start of the episode, the highest probability is for a_1: firing of the left thruster, causing clockwise rotation. Subsequently, after about 10 time steps a_3 becomes the most dominant action, halting the rotation of the spacecraft. Then at about $t = 25$ a_2 receives the largest probability, for firing the main thruster slowing the spacecraft. Finally, when the spacecraft touches the lunar surface around $t = 140$, a_0 (do nothing) becomes the preferred action. This forces the spacecraft to a halt, terminating the episode.

5.2.3. INPUT-OUTPUT ANALYSIS USING SHAP

This Section covers the SHAP analysis, the main explainable AI technique of this preliminary analysis, for the discrete lunar lander environment. First, the waterfall plot will be presented, to provide an explanation for a single time step. The waterfall plot offers a simple introduction to the realm of SHAP values for RL. Secondly, SHAP summary plots are presented to provide global explanations. Using these plots, feature importance can

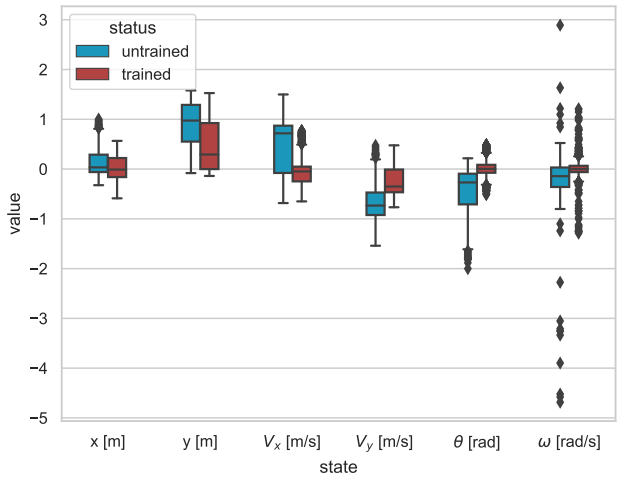
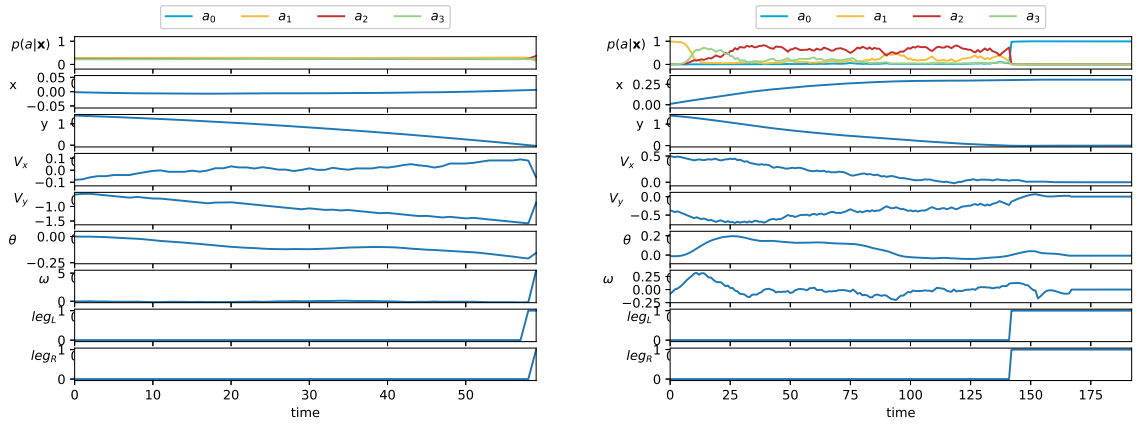


Figure 5.7: Boxplot for the encountered continuous states during simulation of 10 episodes for an untrained and trained discrete agent.



(a) Untrained agent (average score = -156.2).

(b) Trained agent (average score = 212.13).

Figure 5.8: Model output and observed states, for an untrained and trained agent.

be analyzed, and the relation between feature and SHAP value can be roughly determined. Thirdly, the RL-SHAP plots are introduced, where all time steps of an episode are explained by combining feature time traces and SHAP values. Finally, SHAP values are used to visualize the training process of the discrete agent.

WATERFALL PLOT

Due to the limited amount of information presented in one waterfall plot, the waterfall plot is the easiest SHAP plot to fundamentally explain SHAP values. Therefore, the waterfall plot is the first type of SHAP plots to be presented, before more complex SHAP plots are introduced. As explained in Section 4.4.2, the waterfall plot shows the explanation of one instance. The waterfall plot illustrates how every feature contributes to one prediction. Starting from the base value, defined as the expected model output $E\left|f(x)\right|$, the SHAP values are added, resulting in the actual model output $f(x)$. Red bars indicate positive SHAP values, while blue bars indicate negative contributions. Figure 5.9 shows the waterfall plot for a_2, "fire main thruster", during flight.

Without the feature values Figure 5.9 is however difficult to interpret. Later this chapter this will be easier, when RL-SHAP diagrams are presented where the SHAP values are shown combined with the feature values. The waterfall plot is however a good introduction for using SHAP values for analyzing the agent's dynamics during simulations in the lunar lander environment. The baseline or base rate is shown at the bottom, and equals 0.299 for a_2. This means that the average probability for the "fire main thruster" action during this

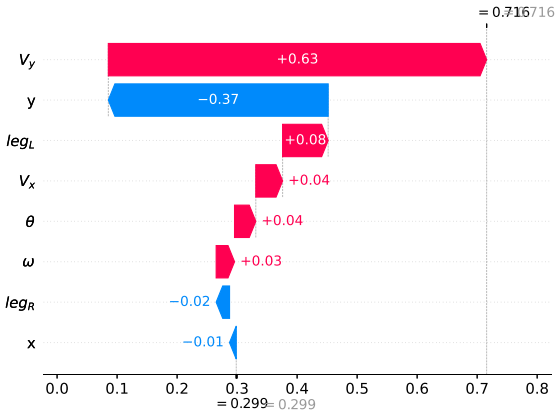


Figure 5.9: Waterfall plot for the "fire main thruster" action during flight, showing the baseline $E[f(x)]$ at the bottom, and the model output $f(x)$ at the top. The lack of feature values complicates the interpretability, but the waterfall plot helps to understand how a SHAP explanation works.

episode is 0.299. From this baseline, the feature SHAP values ϕ are added resulting in the final output $f(x)$, also described in Equation (5.2.1).

$$f(x) \approx \phi_0 + \sum_{i=1}^{8} \phi_i x_i' \tag{5.2.1}$$

For the instance depicted in Figure 5.9 it is clear that V_y and y are the dominant features. As the waterfall plot only illustrates a local explanation it is however impossible to determine from this plot if these features are always dominant, and to estimate the relation between these feature values and their SHAP values. For these analyses, the summary plot is a good option, which will be introduced next.

SUMMARY PLOT

The SHAP summary plots allow investigation of the relation between feature value and SHAP value, and of feature importance. These plots are generated for every output of the specified ML model. Hence, for an NN with 4 outputs, 4 separate SHAP summary plots are generated. For every instance $\mathbf{x}$, a dot is plotted for every feature, along the x-axis corresponding to its SHAP value. These dots are colored according to their feature value, from low to high. Overlapping points in the summary plot are stacked in the y-direction, giving an idea of the distribution of SHAP values for a given feature. The order of the features is determined by the sum of the absolute feature effect, shown in Equation (5.2.2), ordered from high to low. By comparison of a feature's color gradient and the SHAP values, the relation between feature value and SHAP value can be observed. Furthermore, by investigation of the range of the SHAP values for a given feature, and the distribution these SHAP values, the importance of this feature for the specified action can be determined.

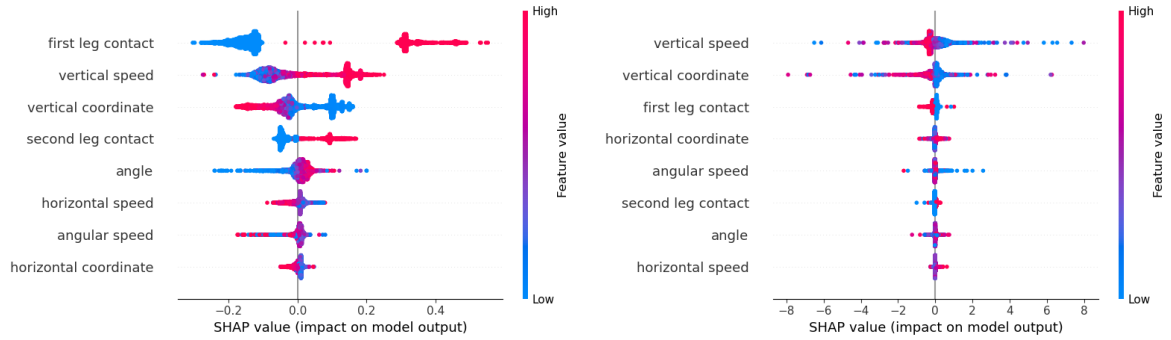
$$I_j = \sum_{i=1}^{n} \left| \phi_j^{(i)} \right| \tag{5.2.2}$$

Figure 5.10 shows the SHAP summary plot for the "do nothing" and "fire main thruster" actions, and Figure 5.11 shows the SHAP summary plot for the two rotational actions. These plots are generated using samples from 20 episodes, for extended state space coverage. Due to the random initial conditions of every episode, an agent will only cover a limited part of the state space during an episode. By increasing the number of episodes, the SHAP summary plots will illustrate a larger part of the SHAP value domain, and will therefore explain more about the dynamics of the spacecraft during simulation. Furthermore, the instances marked as outliers are not shown in these SHAP summary plots, as including these makes interpretation of the plots impossible. An instance is specified as an outlier when one of the 6 continuous features has a SHAP value not in the $\mu \pm 2\sigma$ range.

From Figure 5.10a it can be observed that leg_L is the most dominant state for determining $p(a_0|\mathbf{x})$, as the

range between the minimum and maximum value and the spread between the SHAP values is the largest. The SHAP value is positive when the landing legs make contact, and negative when they do not. Hence the agent is encouraged to "do nothing" when the landing legs make contact, and discouraged when the spacecraft is flying in the air. Figure 5.10b clearly shows that for firing of the main thruster, V_y and y are the most dominant features. These two features both show a negative correlation between the feature value and SHAP value. Hence when V_y or y is low, firing the main thruster will be encouraged. The other 6 features are significantly less important for a_2.

For the two rotational actions shown in Figure 5.11, the distinction between important and less relevant features is less clear. According to the sum of absolute SHAP effect, y and θ are the most important for both a_1 and a_3. The vertical coordinate shows a positive correlation between feature and SHAP value for both actions, indicating a larger chance of choosing these rotational actions at higher altitude. This relation for θ is mirrored between a_1 and a_3 however. Where $\theta > 0$ results in encouragement of a_3, the opposite is true for a_1. This makes sense from the control perspective: positive theta indicates that the spacecraft is tilted counter-clockwise. From this state, action 3 allows the spacecraft to level itself. For the rest of the features, the order of the importance differs significantly between both actions. Also, the scale of the SHAP values for a_1 in Figure 5.11a and a_3 in Figure 5.11b is different. Where the SHAP values for a_1 are in the $[-6.5, 4.8]$ domain, the SHAP values for a_3 are in the $[-2.0, 2.0]$ domain. The reason for this difference in the range of the SHAP values is unknown.



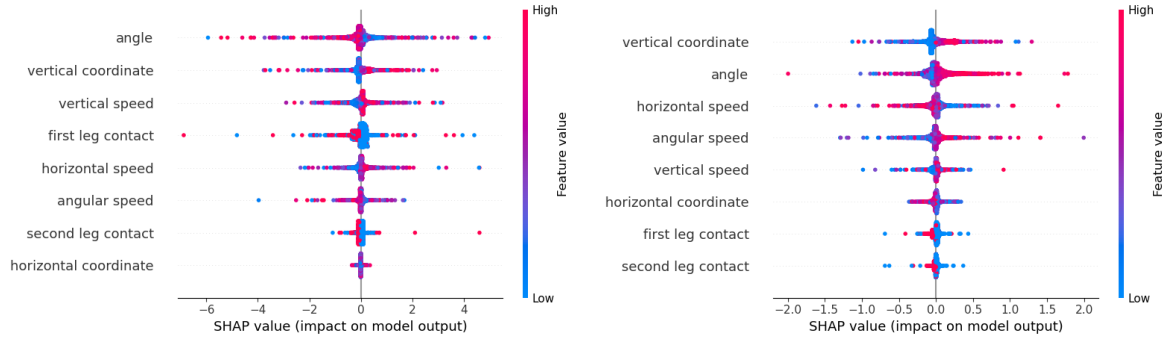
(a) Action 0: do nothing

(b) Action 2: fire main thruster

Figure 5.10: SHAP summary plots for the *do nothing* and *fire main thruster* actions, generated using 20 episodes for improved state-space coverage. Instances where one the continuous feature's SHAP value is not in the $\mu \pm 2\sigma$ range are excluded.

RL-SHAP DIAGRAM

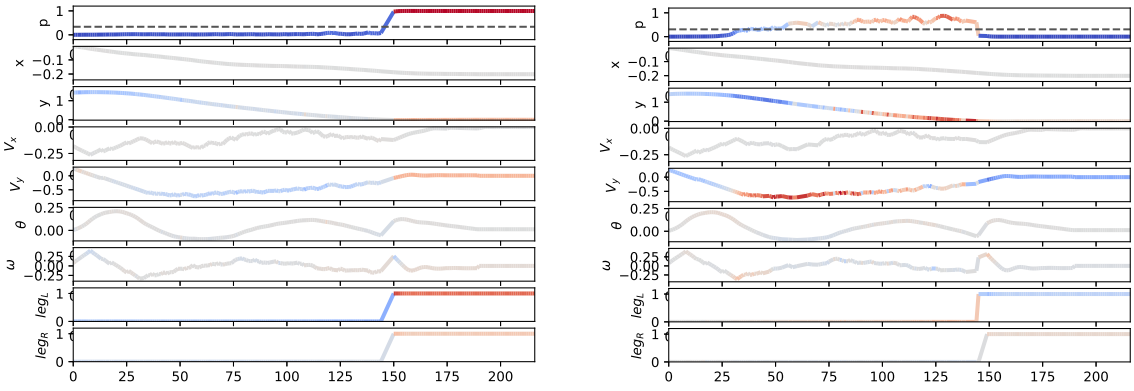
As explained in Section 4.4.2, the RL-SHAP diagram can be regarded as a sequence of waterfall plots. The RL-SHAP diagram shows the actual model output over time, the inputs over time, and most importantly the SHAP values for the inputs over time. Figure 5.12 shows the RL-SHAP diagrams for a_0 and a_2. These two are grouped together as these actions influence the longitudinal acceleration of the spacecraft. Figure 5.13 shows



(a) Action 1: fire left thruster (clockwise rotation)

(b) Action 3: fire right thruster (counter-clockwise rotation)

Figure 5.11: SHAP summary plots for the rotational actions, generated using 20 episodes for improved state-space coverage. Instances where one the continuous feature's SHAP value is not in the $\mu \pm 2\sigma$ range are excluded.



(a) Action 0: do nothing

(b) Action 2: fire main thruster

Figure 5.12: RL-SHAP diagrams for the *do nothing* and *fire main thruster* actions. The grey dashed lines indicate the base SHAP values of the action.

the RL-SHAP diagrams of a_1 and a_3, the clockwise and counter-clockwise rotation actions. The upper subplot in the RL-diagrams shows the model output over time, equal to the probability of choosing the action. The color scale of this subplot indicates the value of the model output, ranging from 0 to 1. The other subplots show the feature values over time, where the color scale indicates the SHAP value.

From Figure 5.12a, the following conclusions for a_0, "do nothing", can be made:

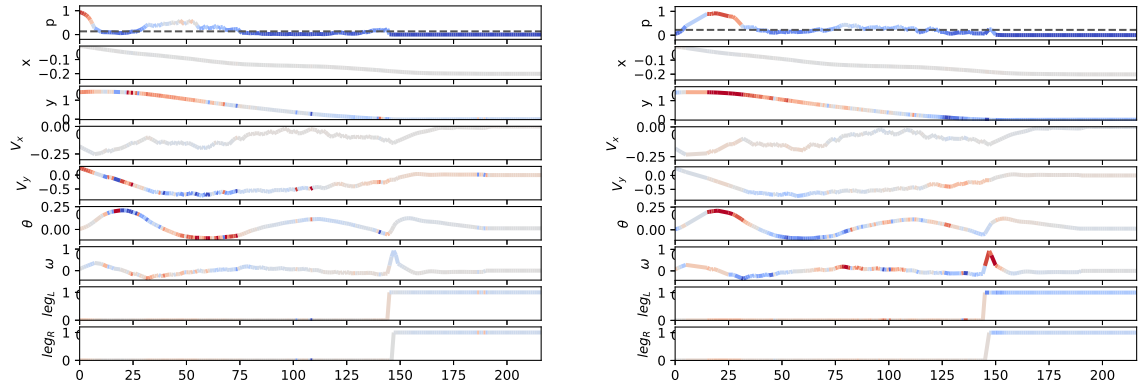
- The main contributor to choosing a_0 is clearly distinguishable due to the bright and constant colors. Once the landing legs make contact, and especially leg_L, the probability for doing nothing changes from about 0 to 1.
- The vertical coordinate y shows a minor negative correlation, where a high altitude discourages "doing nothing". As the landing pad is located at $y = 0$, the agent has good reasons to do so.
- Similarly, V_y is also used to judge the decision for doing nothing. This state on the other hand shows a positive correlation, as a negative vertical speed discourages "doing nothing", while a near-zero V_y results in positive SHAP values.
- The other states show no significant contribution for a_0.

Using Figure 5.12b, the following conclusions for a_2, "fire main thruster", can be made:

- The two most important states for a_2 are y and V_y. Both these variables have a negative correlation between their state value and the corresponding SHAP value. At high values of y, the agent is discouraged from choosing the main thruster, while at lower altitude this is encouraged. Similarly, when V_y is positive or near-zero, firing the main thruster is discouraged. Some interesting interaction between y and V_y can be observed. From $t = 30$, V_y encourages choosing a_2 From $t = 30$ to $t = 80$ y however it causes discouragement, resulting in a compromise where $p(a_2|\mathbf{x})$ is increased from 0 to its base value. Only when y causes positive SHAP values, from around $t = 100$ where $y = 0.2$, does $p(a_2|\mathbf{x})$ increase further. The late deceleration resembles optimal control theory for spacecrafts, where maximum deceleration at the latest moments results in the most efficient burns.
- The left landing leg, leg_L, shows negative SHAP values when making contact. This prevents the spacecraft from taking off when ground contact has been established, significantly reducing $p(a_2|\mathbf{x})$.

From Figure 5.13, the following conclusions for the rotational actions can be drawn:

- Based on the intensity of the colors, the most important features for both a_1 and a_3 are y, V_y, θ, and ω.
- The SHAP values of y, V_y, θ, and ω change with a high frequency, especially for $0 < t < 75$ in Figure 5.13a. Some changes in SHAP value are so large that the sign changes, which can be seen through the color changing from blue to red in only a few time steps. It appears that the when the SHAP value of one



(a) Action 1: fire left thruster (clockwise rotation)

(b) Action 3: fire right thruster (counter-clockwise rotation)

Figure 5.13: RL-SHAP diagrams for the rotational actions. The grey dashed lines indicate the base SHAP values of the action.

feature is suddenly reduced, the SHAP value of another feature is increased. Therefore the changes in SHAP values of the features are significant, but the model output is barely changed. This can be observed in Figure 5.13a for $10 < t < 25$, where $p(a_1|\mathbf{x})$ is about constant around the baseline, while the SHAP values change rapidly and with great magnitude. The high frequency changes in SHAP value might indicate feature interaction. This is an interesting topic for future research, as it is possible that the agent makes its decisions based on a feature consisting of a combination of states.

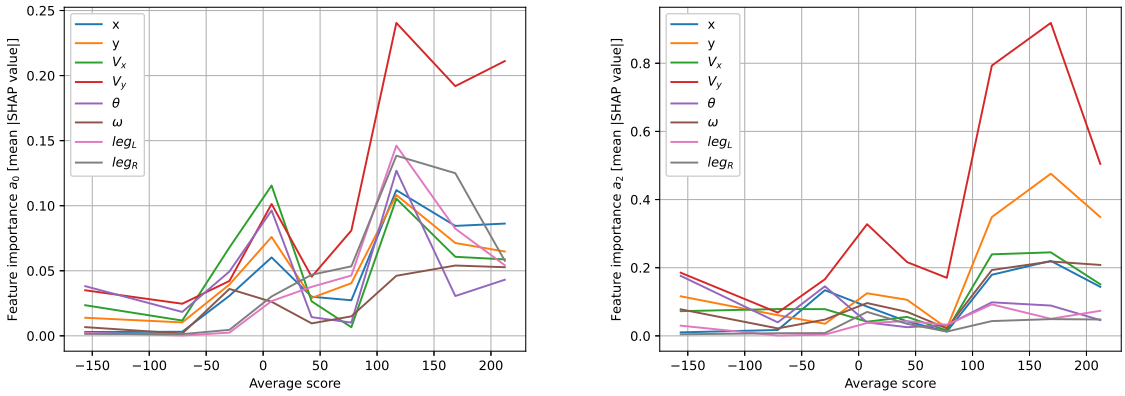
- Both a_1 and a_3 show a positive correlation with y: at greater altitude there is a higher chance of both actions. The effect of y on $p(a_1|\mathbf{x})$ and $p(a_3|\mathbf{x})$ can be observed through the gradually decreasing line in the upper subplots of Figure 5.13a and Figure 5.13b.
- V_y shows a similar positive correlation for both actions, however this effect is more pronounced for a_1 than for a_3.
- In the discussion of the waterfall plots, it is mentioned that the correlation between the feature value of θ and its SHAP value is mirrored between the two actions. This can also be observed in the RL-SHAP diagrams: when $\theta > 0$, this causes a positive SHAP values for a_3, but negative SHAP values for a_1. As can be seen in Figure 5.13b, the positive SHAP values when $\theta > 0$, around $t = 25$ causes $p(a_3|\mathbf{x})$ to increase significantly. Performing a_3 then results in leveling of the spacecraft.
- The same correlations discussed for θ also hold for ω. This can best be observed around $t = 150$, where a high positive ω is incited due to the spacecraft landing on a slope.

TRAINING VISUALIZATION

This subsection aims to investigate whether SHAP values can be used to visualize training and if these can be used to assess convergence during training. This analysis is carried out by calculating the SHAP values of the initial model, the final model, and intermediate models. The intermediate models are saved when the average score of the last 20 episodes is at least 25 points more than the latest saved model. Every model is used to simulate 50 episodes, after which the SHAP values are calculated using 100 background samples. Then, feature importance is determined using the mean absolute SHAP value method. The final and intermediate models created for this analysis are different from those used for the SHAP analysis above. Therefore, differences can be observed in feature dominance between the models displayed in this section and those shown above. Figure 5.14a shows the feature importance plotted against the the average score of the model, for the "do nothing" action, while Figure 5.14b addresses the fire main thruster action.

From the two training visualization plots shown in Figure 5.15 the following conclusions can be drawn:

- V_y is the most dominant feature of both a_0 and a_2, for not only for the final models but also for many intermediate models. For a_0, V_y is the most dominant when the model has reached an average score of about 80, and remains the most dominant until the end. Similarly for a_2, V_y is the most dominant



(a) Action 0: do nothing

(b) Action 2: fire main thruster

Figure 5.14: Feature dominance over time, quantified as the average absolute SHAP value measured during 50 episodes for every model.

feature starting from an average score of around -30.

- For a_0, the order of the features other than V_y is generally non-constant. When ranking the 8 features based on mean absolute SHAP value, only the rank of V_y remains constant after the model with average score 80. However, the rank of the other features varies extensively.
- For a_2, the rank of the second-most important feature y is constant for the last three models. The rank of the other features is however not constant.

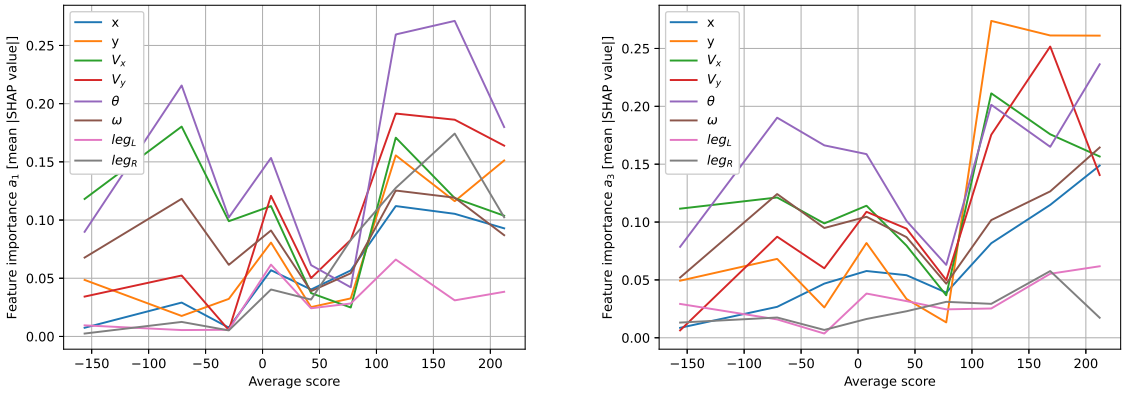
The training visualization for a_1 is shown in Figure 5.15a, and for a_3 in Figure 5.15b. From these plots the following conclusions are drawn:

- For a_1, θ emerges as the most important feature. Its rank is constant for the last three analyzed models, starting where the average score of the model is around 110. Also the rank of the second most important feature V_y is constant for these three models.
- Similarly, y emerges as the most important feature for a_3 during last three analyzed models.
- The rank of the other features for both a_1 and a_3 is not constant however. As an example, the rank of V_y for a_3 changes rapidly during the last 3 models. At average score 110 its rank is the four-most important feature. Then around the average score of 160 it is the second-most important feature, while for the final model it is the sixth-most important feature.

Based on these statements, it is concluded that SHAP could be used to assess training convergence, but more research is required to confirm whether this is also the case for training using different random initializations. In all 4 training convergence plots of the discrete agent, the most important feature is constant for at least the final 3 models. The other features show less consistency. For future research it would be interesting to investigate whether convergence of training for agents with other random initializations can also be assessed by observing the consistency of the most important feature.

5.3. RESULTS OF CONTINUOUS CONTROL PRELIMINARY ANALYSIS

In this section the results of the continuous control analysis are presented. Similar to the discrete control analysis. First, the encountered states are visualized and discussed in Section 5.3.1, to explain some of the agent's actions but most importantly to create some idea of the encountered states before interpreting the SHAP plots for the continuous agent. Then, the output of the actor NN is analyzed for an episode in Section 5.3.2. Finally, the SHAP analysis is presented in Section 5.3.3, where the decision-making process of the agent is explained using SHAP summary plots, RL-SHAP diagrams, and SHAP dependence plots.



(a) Action 1:

(b) Action 3:

Figure 5.15: Feature dominance over time, quantified as the average absolute SHAP value during 50 episodes for every model.

5.3.1. INPUT ANALYSIS

Figure 5.16 shows the encountered state space for the 6 continuous states during 50 episodes of both an untrained and trained DDPG agent. Similar to the encountered states of the discrete A2C agent illustrated in Figure 5.7, the DDPG agent has learned to minimize V_x, V_y, θ, and ω. Another similarity to the input analysis of the discrete agent is the great number of outliers for θ and ω. Like in the discrete environment these outliers are instances where the spacecraft lands under an angle, inducing high $|\omega|$.

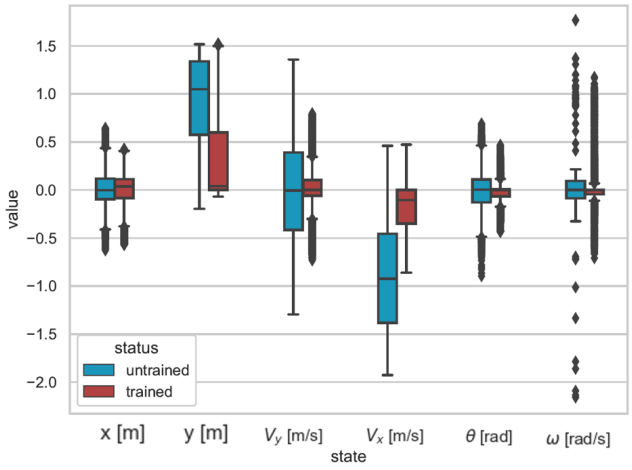
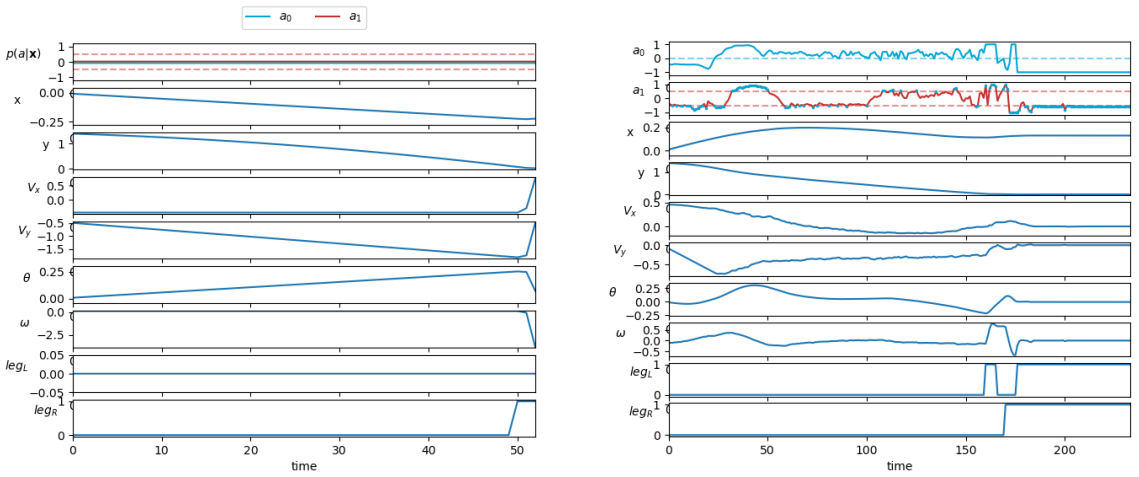


Figure 5.16: Continuous control environment - boxplots for the encountered continuous states during 50 episodes for an untrained and trained DDPG agent.

5.3.2. OUTPUT ANALYSIS

Figure 5.17 shows the time traces of both the model output and the 8 inputs for an untrained and trained agent, to better interpret the SHAP analysis in Section 5.3.3. The red dashed lines indicate the limits of the rotational thruster actions, previously illustrated in Figure 5.2. The left thruster (clockwise rotation) is active for $a_1 \in [0.5, 1.0]$, while the right thruster is active for $a_1 \in [-1.0, -0.5]$. Similar to the untrained discrete agent, the output of the DDPG agent is constant. However, where the 4 outputs of the untrained A2C agent are close to 0.25 due to the softmax function, the 2 outputs of the DDPG agent are 0. The output of the DDPG agent is bounded on $[-1.0, 1.0]$ due to the tanh activation function.

The output of the trained agent illustrated in Figure 5.17 will be explained by first discussing a_0, and then a_1. The main thruster setting a_0 is initially negative, but increases after about 25 time steps when the spacecraft's V_y has decreased. a_0 is then slightly positive until right before the landing around $t = 160$ when the thrust



(a) Untrained agent

(b) Trained agent. The blue dots for a_1 indicate where $|a_1| > 0.5$

Figure 5.17: Model output and observed states, for an untrained and trained DDPG agent in the continuous lunar lander environment. The red dashed lines indicate the limits of the rotational thrusters for a_1, while the main thruster is active when $a_0 > 0$.

increases before landing. Finally when the landing legs have made contact, a_0 is completely reduced. a_1 is initially negative, below the lower threshold, resulting in firing of the right thruster and hence counter-clockwise rotation. This causes θ to increase, after which the opposite thruster is fired to counter the rotation around $t = 40$. Then, from $t = 50$ to $t = 150$ both thrusters are slightly activated. Interestingly, the value of a_1 oscillates slightly around the cutoff values, first for the right thruster and then for the left thruster. Subsequently, the left landing legs makes contact with the lunar surface after which the left thruster is activated. When both landing legs make contact, the right thruster is slightly activated until the episode is terminated.

5.3.3. INPUT-OUTPUT ANALYSIS USING SHAP

SUMMARY PLOT

Figure 5.18 shows the SHAP summary plot for a_0 of the DDPG agent. The plot is generated using samples from 50 episodes. Unlike the SHAP values generated for the A2C agent, the SHAP value results for the DDPG agent do not require outlier filtering, as the range of the SHAP values is smaller. Like in Section 5.2.3, the features are ordered based on the feature importance measure shown in Equation (5.2.2).

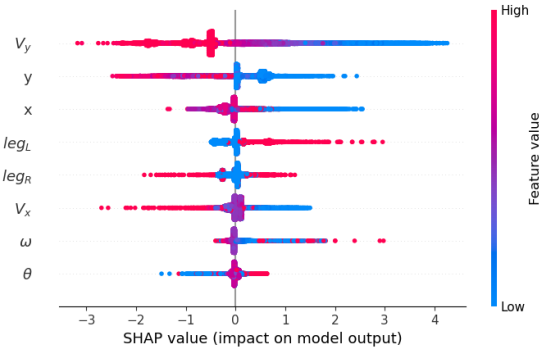
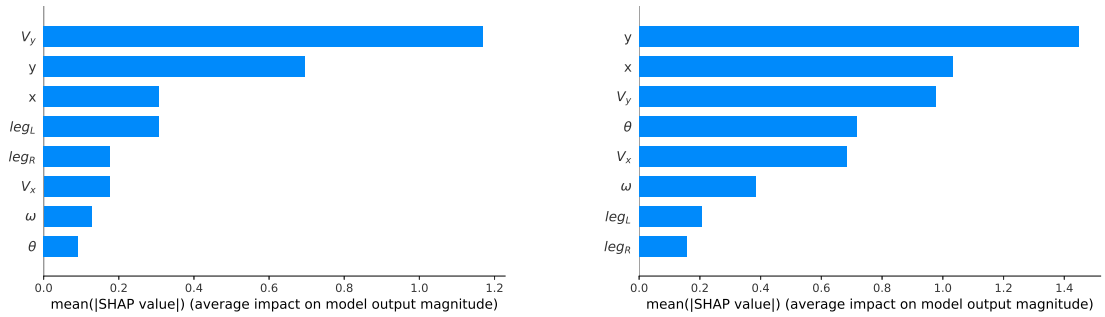


Figure 5.18: SHAP summary plot for a_0, firing the main thruster, of the continuous DDPG agent. The plot shows samples from 50 episodes, without outlier filtering.

From Figure 5.18 the following conclusions can be drawn about the learned strategy for a_0:

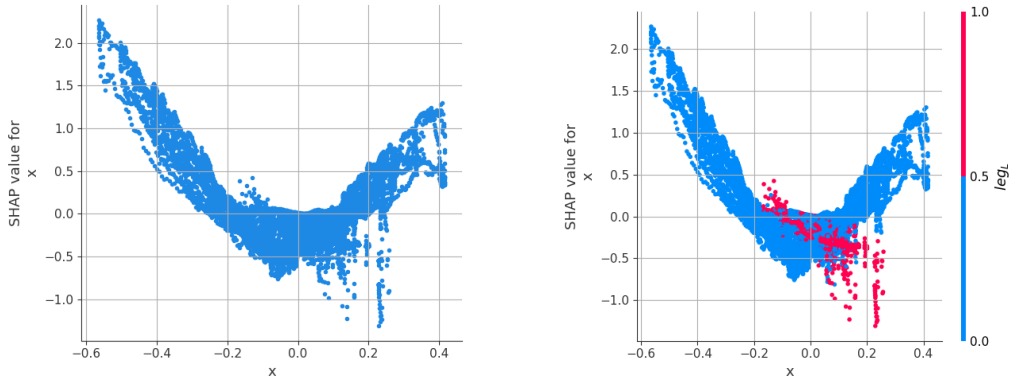
- Like the discrete A2C agent, V_y and y are the most important features for firing the main thruster. These two features show a negative correlation between feature value and SHAP value, like the discrete agent. The feature importance of V_y and y can also be observed in the feature importance plot Figure 5.19a.



(a) Action 0: main thruster

(b) Action 1: rotational thrusters

Figure 5.19: SHAP feature importance for the continuous DDPG agent, measured as the mean absolute SHAP value. The plots helps identifying the most important global features.



(a) Simple dependence

(b) Dual dependence showing the interaction with leg_L, explaining the deviation from the apparent quadratic function

Figure 5.20: Dependence plot of feature x for a_0 of the continuous DDPG agent, showing the increase in thrust when the spacecraft is not above the landing pad.

- The horizontal coordinate does not appear to show a linear correlation, as the purple and red dots overlay each other. This correlation is shown more clearly in Figure 5.20a, the feature dependence plot for a_0 and x. Here it can be observed that the correlation resembles a quadratic function, where ϕ_x is negative for small $|x|$ and positive otherwise. This indicates that the agent has learned to increase main engine thrust when the spacecraft is not above the landing pad. This could be learned in order to spend more time above the lunar surface, and prepare the landing by moving towards $x = 0$. The deviations from what appears to be a quadratic function are explained by the interaction with leg_L. This can be observed in the interaction dependence plot shown in Figure 5.20b. This interaction indicates that ϕ_x is lower when the left landing legs makes contact, forcing the spacecraft on the ground.
- Interestingly, leg_L causes a positive contribution to a_0 when making contact. Rather than preventing taking off right after landing, the agent has learned to increase thrust when the left landing leg touches the lunar surface. The SHAP values for leg_R when this leg makes contact on the other hand can be either positive or negative.

Figure 5.21 shows the SHAP summary plot for a_1. From this summary plot the following conclusions about a_1 for the DDPG agent can be drawn:

- The vertical coordinate y is one of the most important features for the rotational thrust. In the feature importance plot for a_1, Figure 5.19b, it becomes evident that y is the most important feature. For the discrete agent, y is the most dominant feature for a_3, and the second most dominant feature for a_1. For the discrete agent, the relation between y and ϕ_y is roughly linear for both a_1 and a_3, but this appears to be a more complex function for the DDPG agent, due to the overlay of purple and red dots in

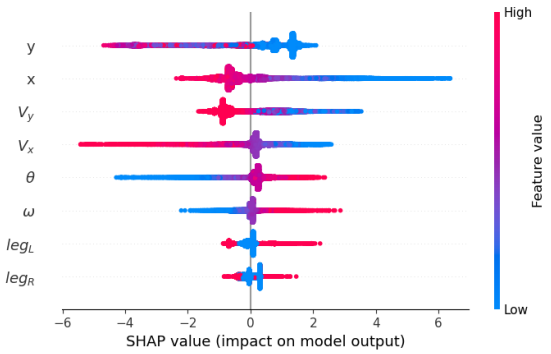
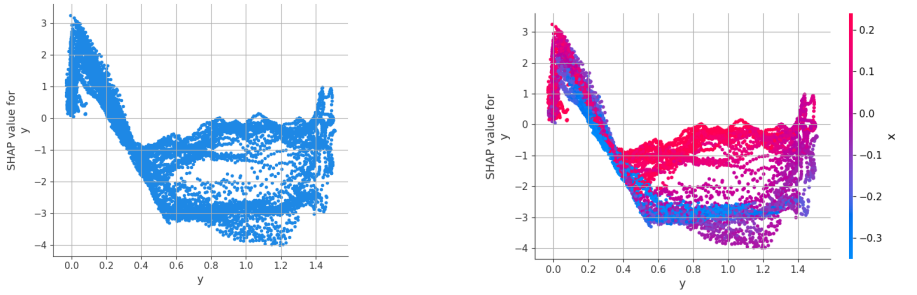


Figure 5.21: SHAP summary plot for a_1, rotational thrust, of the continuous DDPG agent. The plot shows samples from 50 episodes, without outlier filtering.

Figure 5.21. The complex function is illustrated in more detail in Figure 5.22a, the feature dependence plot for y. Interestingly, $\phi_y > 0$ for $y < 0.3$ indicating a preference for clockwise rotation when the altitude is low. However, for altitudes close to 0, $\phi_y \to 0$. The function of ϕ_y for $y > 0.3$ is more complex. This part of the function is better explained using x as interaction data, illustrated in Figure 5.22b. From this combined feature dependence plot it becomes evident that in the higher altitudes ϕ_y is slightly negative or positive when $x > 0$, and ϕ_y is very negative when $x < 0$. From a control perspective this seems counter-intuitive, as counter-clockwise rotation in the left region of the screen would result in movement further away from the landing pad. Not only is this unexpected phenomenon observed in the SHAP plots, this unexpected behavior is also observed during episodes in the continuous lunar lander environment. This can also be seen in the output analysis, shown in Figure 5.17, where $a_1 < 0$ at low altitude, and $a_1 > 0$ near the ground. It is unknown if this learned behavior is beneficial, but the SHAP plots permit identification of these anomalies.

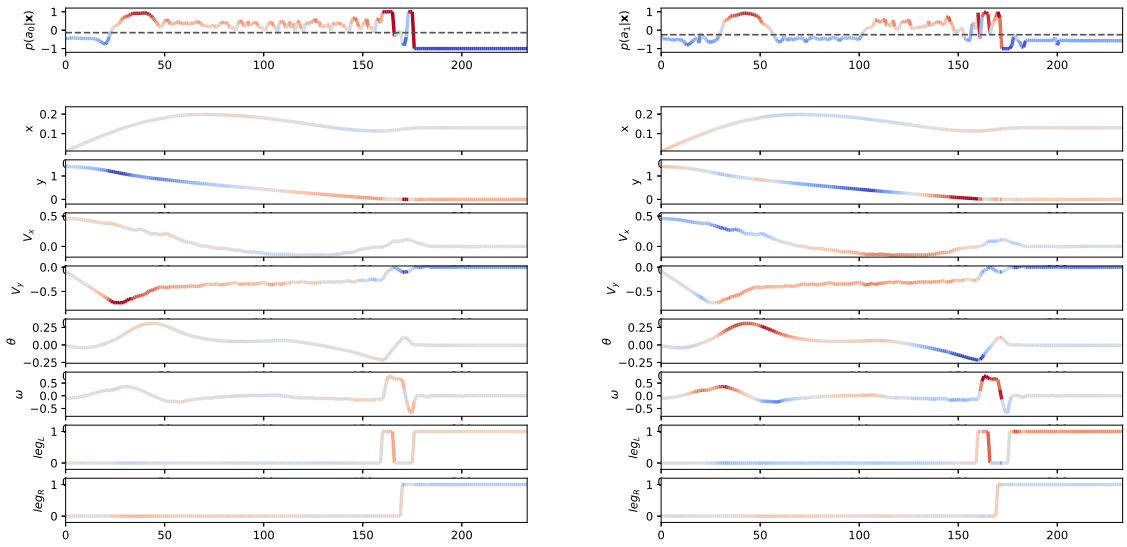
- The second most dominant feature is x, showing a non-complex negative correlation between x and ϕ_x. The correlation implies that ϕ_x causes counter-clockwise rotation when x is high, and clockwise rotation for low x. This makes sense from the control perspective, as rotation towards $x = 0$ combined with force from the main thruster results in movement towards the landing pad. V_x shows a similar correlation between V_x and ϕ_x.
- The correlation between V_y and ϕ_{V_y} shows similarities with the correlation between y and ϕ_y. Both are in the top three of the most dominant features, and show a complex negative correlation between the feature value and its SHAP value. This implies that ϕ_{V_y} favors counter-clockwise rotation at low vertical speeds, and favors clockwise rotation when the spacecraft is quickly moving towards the lunar surface.
- The correlations for θ and ω are similar to those found for the discrete agent. Positive θ and ω indicate counter-clockwise rotation, and result in positive ϕ_θ and ϕ_ω values. Hence, these SHAP values try to minimize the spacecraft's angle and rotational rate.



(a) Simple dependence

(b) Dual dependence showing the interaction with x

Figure 5.22: Dependence plot of feature y for a_1 of the continuous DDPG agent.



(a) Action 1: fire left thruster (clockwise rotation)

(b) Action 3: fire right thruster (counter-clockwise rotation)

Figure 5.23: RL-SHAP diagrams for the rotational actions. The grey dashed lines indicate the base SHAP values of the action.

RL-SHAP DIAGRAM

An example of a RL-SHAP diagram for a_0 is shown in Figure 5.23a and for a_1 in Figure 5.23b. These diagrams are created using sample data from one episode, the same used for the output analysis shown in Figure 5.17.

From Figure 5.23a the following conclusions about a_0 can be made:

- In the discussion about the SHAP summary plot of a_0 it is mentioned that y and V_y are the dominant features for firing the main thruster. This same observation can be made in Figure 5.23a where these two features have the brightest colors, indicating high $|\phi|$ values. The velocity-reducing effect of ϕ_{V_y} is clearly shown in Figure 5.23a where a_0 is increased due to V_y when this feature becomes more negative. Especially the sharp increase in a_0 around $t = 40$ can be clearly attributed to V_y.
- The rapid oscillations of a_0 after $t = 150$ are explained by the landing legs making and losing contact, and the positive contributions of ω. Apparently, high $|\omega|$ values during this run increase a_0. A possible reason for this could be to spend more time in the air to prepare the landing by leveling the spacecraft. Why leg_L induces a positive a_0 is unknown.
- Around $t = 170$, leg_R makes contact with the ground. The corresponding negative SHAP values significantly reduces a_0, forcing the spacecraft on the lunar surface.
- Although x is the third most important feature according to the feature importance plot Figure 5.19, $|\phi_x|$ is small during the run shown in Figure 5.23a. This is due to the limited range of x during this run. This can be observed in the dependence plot of x for a_0 in Figure 5.20a, where ϕ_x is near zero for $|x| < 0.2$.
- The features V_x and θ show no significant contribution to a_0. This can be confirmed using the feature importance plot Figure 5.19a as well, where these features are significantly less important than V_y and y according to the mean absolute SHAP value measure.

From Figure 5.23b the following conclusions about a_1 can be made:

- The initial low values of a_1 are mainly caused by V_x. The spacecraft in this episode is initialized with a positive V_x, indicating that the spacecraft moves towards the right of the screen. The resulting low values of a_1 cause counter-clockwise rotation. The learned strategy by doing so is to move towards the centre of the screen when $\theta > 0$ and then firing the main thruster. The direction of the force of the main

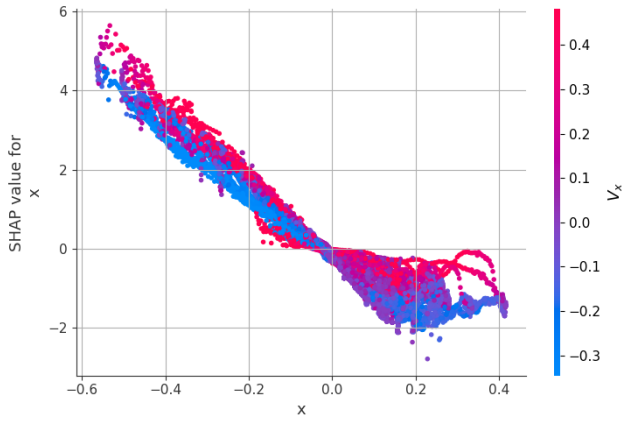


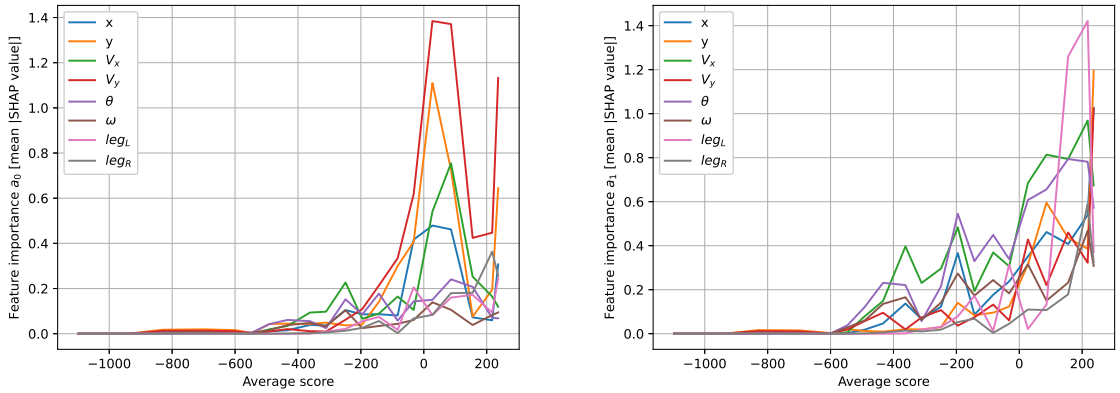
Figure 5.24: Dependence plot of x for a_1, with V_x interaction.

thruster will then cancel the positive V_x. This can be observed in the value of V_x from $t = 0$ to around $t = 100$ where V_x is reduced. Around $t = 75$ V_x is reduced to 0, and from $t = 75$ to $t = 150$ V_x is even negative to counteract the previously travelled horizontal distance.

- The negative ϕ_{V_x} values in the beginning, and the resulting negative a_1 results in counter-clockwise motion and hence positive ω. The corresponding positive ϕ_ω values have a dampening effect on the rotation, as a_1 is increased and even becomes positive.
- As ω is positive during $t = 0$ and $t = 50$, θ increases. The values of θ from $t = 30$ to $t = 60$ cause positive ϕ_θ values, significantly increasing a_1. Between $t = 40$ and $t = 60$ a_1 has reached a maximum due to ϕ_θ and ϕ_ω, causing clockwise rotation. The corresponding negative ω then reduces a_1 again around $t = 50$.
- As mentioned in the discussion of the SHAP summary plot, y is the most dominant feature for a_1. The effect of y on a_1 can be observed during $t = 60$ and $t = 150$. From $t = 60$ to around $t = 110$, $y > 0.3$ and ϕ_y is negative. This altitude of 0.3 is the switching point discussed in the SHAP summary plot using the dependence plot Figure 5.22b. Interestingly, according to the SHAP values of y, counter-clockwise rotation is favored at the higher altitudes before $t = 110$. Then when the altitude is reduced below $y = 0.3$, clockwise rotation is induced. As explained in the discussion of the SHAP summary plot, it is unknown why this strategy is learned, and whether it has any benefits. However, using the RL-SHAP diagram, it becomes evident what is causing the sudden increase in a_1 between $t = 60$ and $t = 150$. Even though the behavior is unexpected, and potentially unwanted, the RL-SHAP plot allows identification of this decision-making process.
- According to the feature importance plot for a_1, Figure 5.19b, x is the second most important feature for a_1. In the RL-SHAP diagram however, x shows almost no significance. This is due to the same reason presented in the discussion for a_0. From the dependence plot of x for a_1, shown in Figure 5.24, this can be observed. The SHAP values for x can be significant, especially for low x. In the depicted episode however, x ranges between 0 and 0.2. According to Figure 5.24 ϕ_x is small in this region, especially when V_x is positive like in the depicted episode.
- In the depicted episode, the landing legs making contact cause SHAP values to incite rotation such that the other landing leg also makes contact. The SHAP values for leg_L are positive when making contact, increasing a_1. Clockwise acceleration is triggered when $a_1 > 0.5$, hence the positive ϕ_{leg_L} tries to level the spacecraft after touchdown. The same effect occurs for the negative ϕ_{leg_R} encountered during the episode. For ϕ_{leg_L} and ϕ_{leg_R} it can also be observed that these values are not constant over time, even though their corresponding feature values are constant. ϕ_{leg_L} is for example grey during the first time steps, and then becomes blue indicating $\phi_{\text{leg}_L} < 0$. This shows that SHAP values are not exclusively dependent on the value of the corresponding feature, but on the coalition of the corresponding feature, taking into account the other contributing features.

TRAINING VISUALIZATION

Similar to the training visualization analysis, this section investigates whether SHAP values can be used for assessing training convergence. Figure 5.25a is the training visualization plot for a_0, and Figure 5.25b for a_1.



(a) Action 0

(b) Action 1

Figure 5.25: Feature dominance over time for DDPG, quantified as the mean absolute SHAP value during 20 episodes for every model.

From the training visualization plots shown in Figure 5.25 the following conclusions are drawn:

- Similar to the training visualization of a_0 and a_2 of the discrete agent, V_y emerges as the most important feature for a_0 of the continuous agent. The rank of V_y is constant for the last 7 models, and is hence constant for the second half of training.
- The rank of all other features is however not constant.

Similar to the training visualization plots of the discrete agent, the training visualization plot of a_0 of the continuous agent shown in Figure 5.25a shows consistency in the most important feature for the final models. This supports the statement of consistency of the most importance feature implying training convergence. Figure 5.25b on the other hand contradicts this statement. Therefore this training convergence analysis using SHAP requires more research before a conclusion can be drawn.

III

Additional Results and Discussions

6 | Validation of the Linear Representation Models

The paper presented in Part I introduces the linear representations of the learned strategy for δ_e, δ_a, and δ_r. In the paper, it is shown that only the slope between feature value and SHAP value is relevant for the linear models. The output of the three linear models is verified by comparison with the actual deflections as controlled by the longitudinal and lateral IDHP agents. For every segment, it is shown that the linear model matches the IDHP output under the assumption of constant weights. For this verification, the linear model coefficients are determined separately for every segment, using linear regression based on the relation between feature value and SHAP value. To validate the linear models and test their stability and accuracy, the linear representation models are used to actually control the simulated Cessna Citation aircraft. The difference between this experiment and the verification in Part I is that the linear models are now used to actually control the aircraft, rather than being a post-hoc representation of the learned strategy using IDHP.

The flight task for this experiment is similar to the mission profile used in the scientific paper, starting at $H_0 = 2000$ m and $V_0 = 80$ m/s. The longitudinal and lateral IDHP agent of the flight controller, as illustrated in Figure 5 in Part I, are replaced by the linear models given by Equation (6.0.1), Equation (6.0.2), and Equation (6.0.3). The slope coefficients of these linear models are constant during flight, to simulate non-adaptive linear control. The a-coefficients are based on the calculated SHAP values from the mission profile presented in the paper, and are calculated as the mean a-coefficient during all constant weight segments. As not every selected segment has the same length, the mean slope is calculated over all time samples within the selected segments. The resulting coefficients for the linear longitudinal controller for δ_e are summarized in Table 6.1, and for the lateral linear controllers in Table 6.2.

$$\delta_{e_{lin}} = a_q \cdot q + a_\alpha \cdot \alpha + a_\theta \cdot \theta + a_{err_q} \cdot err_q \tag{6.0.1}$$

$$\delta_{a_{lin}} = a_{p_{\delta_a}} \cdot p + a_{r_{\delta_a}} \cdot r + a_{\beta_{\delta_a}} \cdot \beta + a_{\phi_{\delta_a}} \cdot \phi + a_{err_{p\delta_a}} \cdot err_p + a_{err_{\beta\delta_a}} \cdot err_\beta \tag{6.0.2}$$

$$\delta_{r_{lin}} = a_{p_{\delta_r}} \cdot p + a_{r_{\delta_r}} \cdot r + a_{\beta_{\delta_r}} \cdot \beta + a_{\phi_{\delta_r}} \cdot \phi + a_{err_{p\delta_r}} \cdot err_p + a_{err_{\beta\delta_r}} \cdot err_\beta \tag{6.0.3}$$

Table 6.1: Coefficients used for the validation of the linear model for δ_e.

Feature	Slope for δ_e
q	-0.408
α	-0.137
θ	0.054
err_q	0.746

Table 6.2: Coefficients used for the validation of the linear model for δ_a and δ_r.

Feature	Slope for δ_a	Slope for δ_r
p	-0.219	-0.067
r	0.055	0.003
β	-0.026	-0.007
φ	0.003	-0.023
err_p	0.665	0.050
err_β	0.003	-0.001

The time traces of the simulated Cessna Citation controlled with the linear controllers are presented in Figure 6.1, plotted together with the time traces of the IDHP controllers for comparison. The time traces for q are shown separately in Figure 6.2 for easier interpretability. The following observations can be made regarding the linear model output, based on the time traces:

- Figure 6.1 shows that $\delta_{a_{lin}}$ is very similar to the aileron deflection commanded by the lateral IDHP controller. Consequently, the roll rate p is almost similar for both models. Furthermore, the roll rate time traces for both models closely match p_{ref}.
- The rudder deflection commanded by the linear model, $\delta_{r_{lin}}$, shows many similarities with δ_r from the lateral IDHP controller. The sideslip angle β of the linear model is slightly more than that of the IDHP controller, but does not exceed ± 0.2 deg.
- Larger differences can be observed for the elevator models in Figure 6.1 and Figure 6.2. Even though the linear elevator deflection $\delta_{e_{lin}}$ shows many similarities with the IDHP elevator deflection, the error between pitch rate and commanded pitch rate is significantly larger for the linear model. Figure 6.2 clearly shows this offset between q and q_{ref} for the linear controller. Apparently, adaptability of the longitudinal controller is required to accurately match q_{ref}. A possible explanation for the required adaptability is the air speed V_{tas} as this is an important state for longitudinal motion and because this state is not constant during the simulated mission profile. [49] shows that variations in V_{tas} cause the actor parameters $\mathbf{w}_a$ to change. Further research is required to verify if and how airspeed requires adaptability of the longitudinal controller.

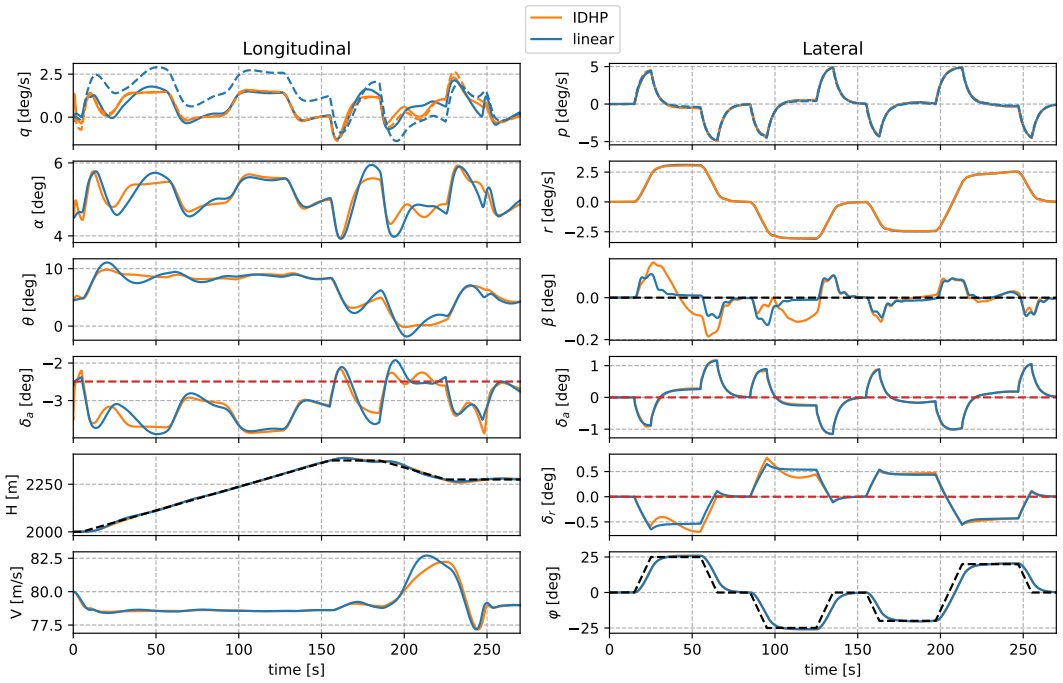


Figure 6.1: Validation of the linear models, during simulation of a mission profile starting at $H_0 = 2000$ m and $V_0 = 80$ m/s. The blue lines represent the state space and reference states of the linear models, and the orange lines for IDHP. The red dashed lines represent actuator trim values, and the black dashed lines represent the mission profile identical to both the IDHP and linear controllers. The time traces of q are shown separately in Figure 6.2 for better interpretability.

DISCUSSION

Though there are differences between the time traces of the linear and IDHP models, due to the fixed linear model not adapting to different flight conditions, the linear models allow stable tracking of the mission profile. Not only does this validate the linear models obtained through the SHAP analysis in Part I, it also opens the door to interesting opportunities regarding the extraction of control laws. The control architecture of aerospace applications with limited computational complexity may for example be designed using linear control laws, obtained through SHAP analysis.

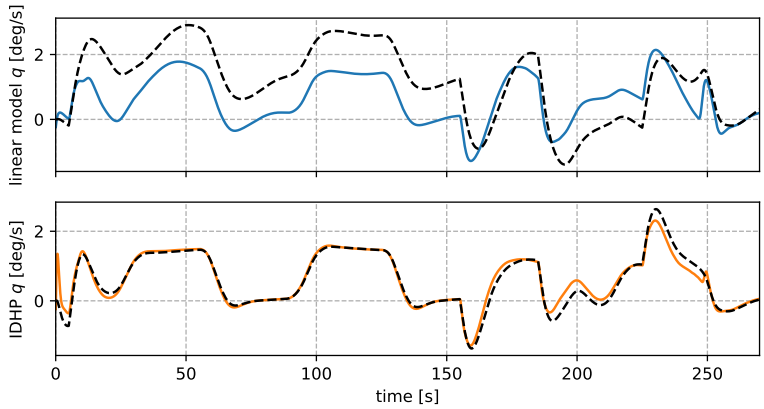


Figure 6.2: Time traces of q for the linear controller shown in blue, and the IDHP controller in orange.

7 | Illustrating and Explaining Adaptive Properties using SHAP

This chapter presents how the adaptive properties of RL for flight control can be visualized using SHAP and CWSD. First, the adaptation of the IDHP controller during the mission profile introduced in Part I is visualized by plotting the linear slopes over time and comparing the variation in this control behavior with the segments selected by CWSD. Subsequently, the fault-tolerance of IDHP to reduced aileron deflection IDHP is visualized using the combined dependence plots.

7.1. Showing Adaptation Using the Linear Slope

The combined dependence plots introduced in Part I explain how the learned strategy changes during flight, by illustrating the relation between feature value and SHAP value for all selected constant-weight segments. As these relations are linear for the selected application, and because only the slope of the linear relation is important, the changing slopes can be plotted over time. This allows to investigate one of the possible shortcomings of the CWSD algorithm, where sections with different weights do not necessarily imply different learned behavior as the input-output mapping can still be similar. As ultimately the slopes between feature value and SHAP value define the learned control behavior, inspecting how these relations change over time creates understanding of the control strategy adaptation.

To show how this control adaptation can be explained using the progression of the slopes, this section presents how the slopes of the linear representation models for δ_e, δ_a, and δ_r change during the mission profile introduced in Part I. The same starting conditions, $H_0 = 2000$ m and $V_0 = 80$ m/s, are used. To force the slopes to be computed throughout the whole profile, the mission profile is divided into segments with a length of 100 samples, in which the SHAP values are calculated. Therefore, the weights are assumed to be constant during every time window of 100 samples, equalling a period of 2 seconds as $f_s = 50$ Hz. Even though the CWSD algorithm would not specify many of these windows as constant weight segments, dividing the mission profile into windows of 100 samples allows the progression of the slopes to be determined as a smooth function. Within every window, the SHAP values are calculated using all 100 samples as background samples.

Figure 7.1 shows the linear coefficients for δ_e over time. Furthermore, the segments selected by the CWSD algorithm are included are included to compare how the selected segments compare with the changes in slopes. The changes in slope of features q and $err_q = q - q_{ref}$ are smaller than the changes in slope of α and θ. A possible explanation for the adaptation of these two features could be the changes in true airspeed V_{tas} as mentioned in Chapter 6. Figure 7.2 presents the time traces for a_α and V_{tas} in more detail, showing their possible relation. However, V_{tas} appears to lag the slope a_α, contradicting the statement of V_{tas} influencing a_α. Further research is required to investigate the influence of V_{tas} on the slopes of the linear representation for δ_e.

Furthermore, Figure 7.1 shows that the change in of the slopes is minimal during the time segments selected by CWSD. Except for segments 1 and 2, every segment appears to show different slopes. Apparently, even though the weights change enough between segments 1 and 2 to define these as segments with different weights, the resulting learned strategy does not change. This shows that the learned behavior can be more generalized than what the CWSD algorithm implies. This shortcoming of the CWSD can be overcome by interpreting the SHAP results after the constant-weight segments are selected, and combining segments with similar behavior.

The progression of the slopes for the linear representation of δ_a and δ_r is shown in Figure 7.2. The slope of φ for for δ_r right before t=50 s. The reason for this is that the SHAP values for φ are close to 0 and barely change in this region. Therefore it is hard to calculate the linear fit in this time window for φ. This problem could be overcome by choosing larger windows. However, this would limit the resolution of the slope plot. For this analysis, the jump to 0 will be considered as a defect of the slope determination methodology.

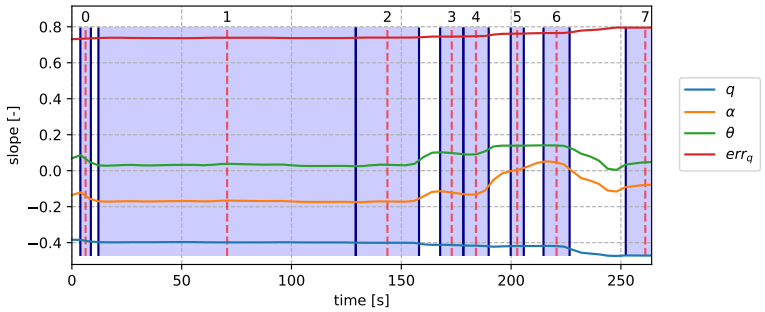


Figure 7.1: Linear explanation model slopes for δ_e during the mission profile introduced in Part I, starting at $H_0 = 2000$ m and $V_0 = 80$ m/s. The slopes are determined using SHAP, based on segments with 100 samples. The shown segments are not used to determine the slopes, but allow comparison between the decisions of CWSD and the learned strategy.

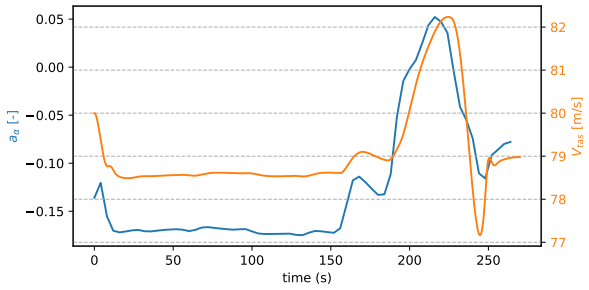


Figure 7.2: Time traces for the linear slope between α and its SHAP value, and the true airspeed V_{tas}.

Figure 7.3 shows that many slopes for δ_a and δ_r are consistent throughout the mission profile. The largest adaptation is observed for φ in the linear representation model for δ_r. The reason for the variance in the slope of this feature for δ_r could also be the air speed V_{tas}. Further research is required to confirm this potential influence. Figure 7.3 shows multiple segments with similar slopes, hence the behavior can be further generalized. Similar to the longitudinal controller explanation methodology, this generalization could be made after the SHAP analysis by comparing the learned behavior between segments, and combining segments when the strategy is consistent. Furthermore, there are multiple regions in the plot for δ_a where the slopes appear to be constant, yet these are not selected by the CWSD algorithm. The reason for this is that the complete $\mathbf{w}_a$ parameter vector is used for the selection of segments for the lateral controller. As the lateral IDHP has two outputs, there are regions where the weights corresponding with the δ_r output change, while the weights corresponding with δ_a are constant and vice versa. An example of this is the region between segments 3 and 4. Knowing this, higher sample coverage for both outputs can be obtained by only using the relevant weights for a certain output to the CWSD algorithm. For example, the weights between the hidden layer and the δ_r output are irrelevant for the δ_a analysis.

7.2. ILLUSTRATING FAULT-TOLERANT ADAPTATION

One of the key strengths of IDHP and other adaptive RL frameworks for flight control is their fault-tolerance. [49] shows that the IDHP framework, including a target critic NN, is able to quickly adapt to an aileron failure, resulting in stable and accurate flight. The failure used in this research is a 50% less effective aileron deflection. Hence, the commanded aileron deflection is halved before passing it to the plant.

SHAP, in combination with the CWSD algorithm, can be used to illustrate the adaptation to failures like these. As an example, the mission profile control experiment introduced in Part I is repeated, now with an aileron failure introduced at 67 seconds. After this moment, the control effectiveness is 25%, hence the commanded aileron output is multiplied with 0.25 before passing it to the simulated plant. The agent will have to adapt to these new dynamics by updating its weights. SHAP can illustrate how the control strategy has changed after the failure. Figure 7.4 shows the mission profile before and after the aileron failure at 67 seconds, indicated by the red dashed line. Around 50 seconds after the failure, the agent has successfully adapted its strategy regarding δ_a, as the roll rate p closely follows p_{ref} again. For adapting the control strategy of the rudder δ_r,

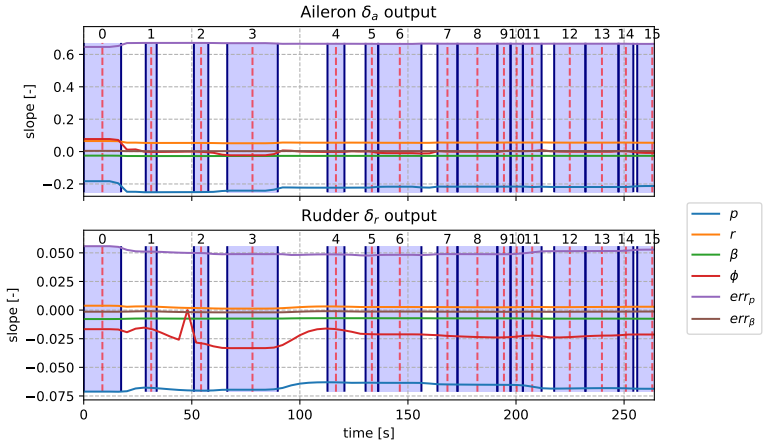


Figure 7.3: Linear explanation model slopes for δ_a and δ_r during the mission profile introduced in Part I, starting at $H_0 = 2000$ m and $V_0 = 80$ m/s. The slopes are determined using SHAP, based on segments with 100 samples. The shown segments are not used to determine the slopes, but allow comparison between the decisions of CWSD and the learned strategy.

the agent requires more time than provided in this mission profile, assuming it will improve at all. The peaks in sideslip in the last 100 seconds are in the order of ±0.5 deg, while the lateral controller presented in Part I was able to minimize β to ±0.1 deg.

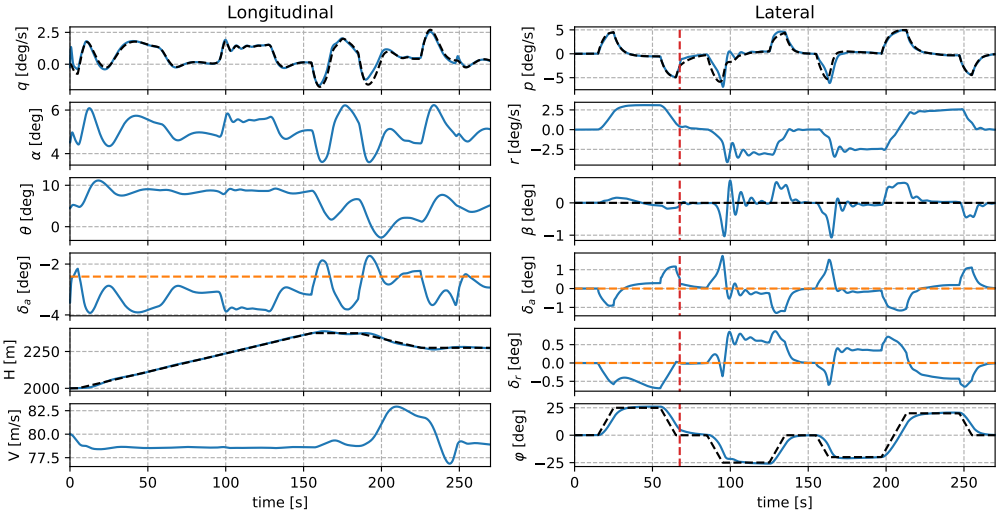
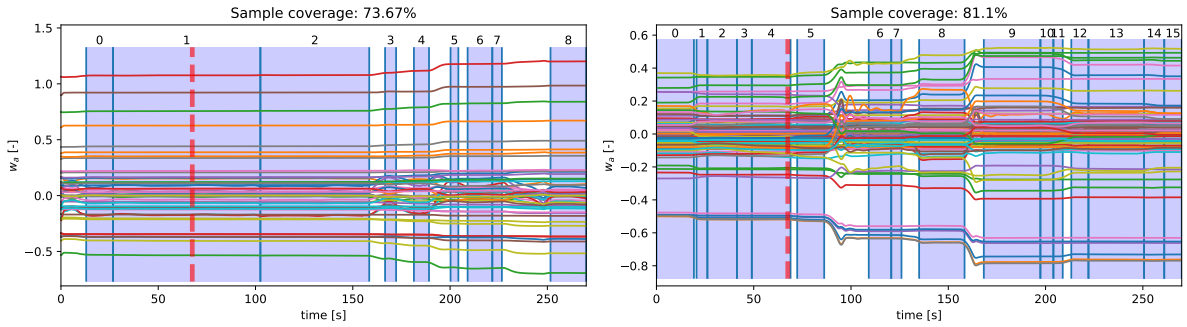


Figure 7.4: Input and output of the flight controller during the mission profile, starting at $H_0 = 2000$ m and $V_0 = 80$ m/s, with an aileron failure at 67 s, indicated by the dashed red line. After this failure, the aileron is 25% less effective.

Figure 7.5 shows the longitudinal and lateral IDHP controller parameters during the mission profile, including the segments selected by CWSD. The progression of $\mathbf{w}_a$ for the longitudinal controller is similar to the progression presented in Part I. The $\mathbf{w}_a$ weights for the longitudinal controller show more changes, to account for the reduced aileron effectiveness. The SHAP results for every segment presented next will illustrate this adaption.

The combined dependence plots for δ_a and δ_r are presented in, respectively, Figure 7.6 and Figure 7.7. The combined dependence plot for δ_e is not included as there are no failures regarding longitudinal control, and the combined dependence plots for δ_e are very similar to those presented in Part I.

Both Figure 7.6 and Figure 7.7 show significant changes in slope for the most important features: p, φ, and err_p. With these steeper slopes, the lateral IDHP controller will output more extensive values for δ_a and δ_r for the same feature values. After the transformation with the 0.25% aileron control effectiveness multiplier, the control output for δ_a is in the same domain as presented in Part I. In theory, when p_ref is accurately



(a) Longitudinal controller

(b) Lateral controller

Figure 7.5: Constant weight segment detection for the lateral controller, during the mission profile with a 25% aileron effectiveness failure after t=67 s, indicated by the vertical red dashed line. The centre segment points, used to copy the $\mathbf{w}_a$ weights are removed to clearly present the time of failure.

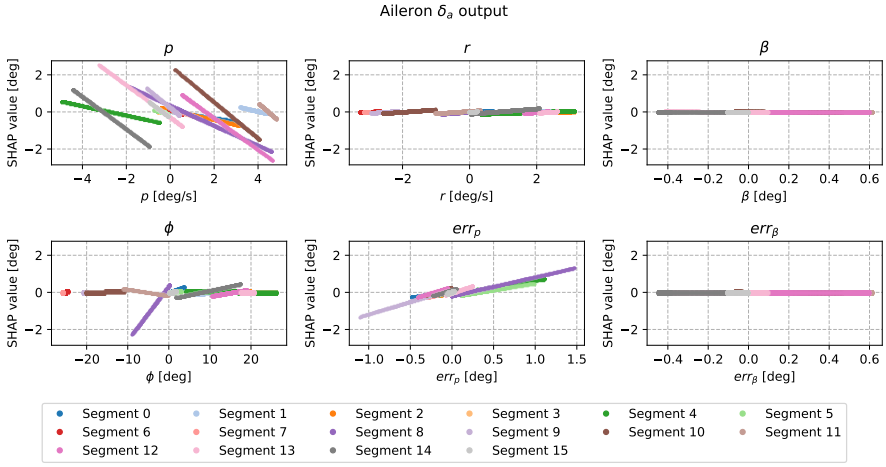


Figure 7.6: Combined dependence plots for δ_a during the mission profile with aileron failure.

followed as in Part I, the rudder output δ_r can remain unchanged. However, Figure 7.7 shows that also rudder is now more sensitive to its inputs. This is likely due to the shared neurons and weights between the input and output layer of the actor NN.

DISCUSSION

The characteristics from the extracted linear control laws can be exploited to determine whether the segments selected using the CWSD algorithm exhibit similar or different learned control strategies. This could help further generalization of the learned control strategy by combining segments exhibiting similar control behavior. Furthermore, the progression of slopes can potentially give clues about the cause of adaption. The correlation between a_α and V_{tas} could for example indicate that V_{tas} is an important state for longitudinal control. Furthermore, the changes in slope shown in the combined dependence plots for δ_a and δ_r illustrate that these plots can be used to illustrate fault-tolerance for adaptive RL frameworks. However. the changes in slope in the combined dependence plots for δ_r are unexpected, since in theory the rudder control strategy can remain unchanged after the lateral IDHP controller has adapted to its original control behavior by commanding more aggressive aileron deflection. This assumed undesired control behavior results in larger values for β than those observed in Part I. SHAP has exposed this divergent behavior. While the rudder strategy might be improved over time, with more interaction data by extending the mission profile after the simulated 270 s, another possible solution to the increased sensitivity of δ_r resulting from failures in δ_a could be to create two independent NNs for the lateral IDHP controller.

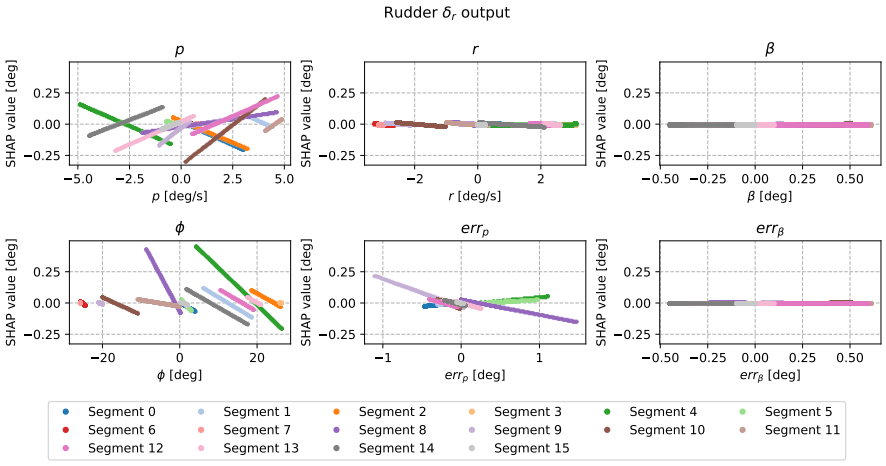


Figure 7.7: Combined dependence plots for δ_r during the mission profile with aileron failure.

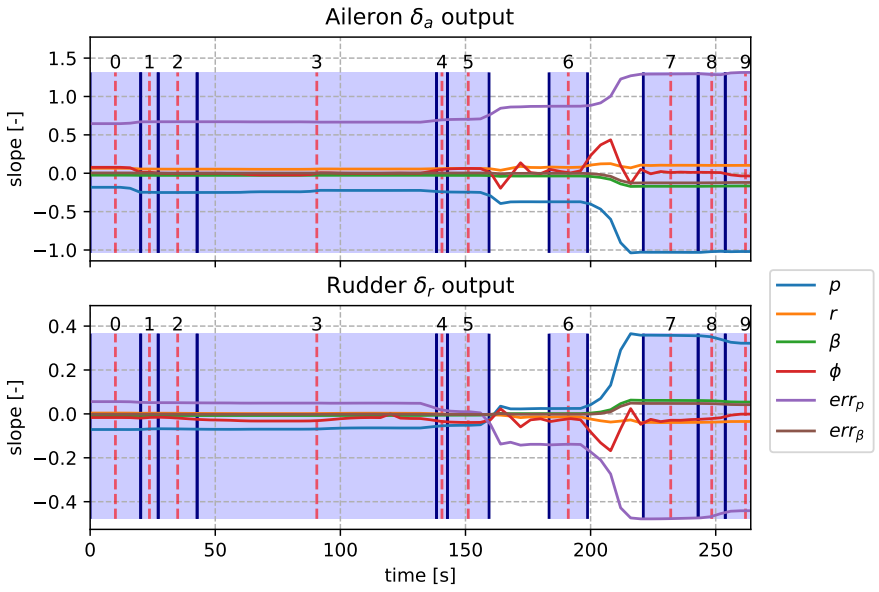


Figure 7.8: Progression of slopes of the linear representation model for δ_a and δ_r during the mission profile, with aileron failure starting at t=67 s.

IV

Wrap-Up

8 | Conclusions

This chapter presents the answers to the research questions, preceded by a synopsis of the research findings and contributions.

8.1. Synopsis

The main contributions of this thesis are twofold. First of all, it is demonstrated how SHAP, a feature-relevance XAI technique, can be used to generate useful and detailed insights into the learned strategy for flight control of an adaptive RL agent. The second main contribution is the CWSD algorithm, enabling the use of XAI techniques, requiring constant weights, for adaptive RL frameworks by identifying time segments with near-constant NN parameters. The increase in interpretability of adaptive RL frameworks for flight control can help control experts improve the robustness of the RL algorithms by highlighting sensitivities of the function approximators, improving trustworthiness of the by understanding the input-output relations of the function approximators, and accelerating development by making debugging of the algorithms easier. Furthermore, the potential of the used explanation methodology involving SHAP and the CWSD algorithm is not limited to flight control, as these techniques can be applied to any adaptive RL problem using a NN as policy function approximator.

The literature review of this thesis shows how RL can be applied to flight control, and how XAI techniques are used to improve the interpretability of ML models. The classification tree in the literature review shows that existing publications into RL for flight control can be labeled into two categories for high-level control, goal-based navigation and autonomous obstacle avoidance, and two categories for low-level control: adaptive control and controller tuning. Adaptive control is chosen as the main focus for this thesis, due to the traction in this research field and the large potential of deep RL for adaptive flight control, due to the adaptability of NNs. From the adaptive control using RL literature, IDHP is selected as the state-of-the-art framework for this thesis, due to its superior performance, fault-tolerance, and adaptability to unknown flight dynamics. To investigate how this framework can be explained, the literature review presents and compares XAI techniques showing potential for explaining deep reinforcement applications for flight control. From these techniques, SHAP is selected as the most promising technique, as it is the most accurate technique of the reviewed post-hoc explainability methods. This post-hoc explainability property prevents any compromises of accuracy for interpretability.

In the preliminary analysis it is shown how SHAP can be used to generate useful explanations for offline deep RL agents. As the agents used in this preliminary analysis do not exhibit online learning, their model parameters are fixed during every episode. As SHAP requires constant weights for the computations of the feature contributions, the offline learning frameworks served as a proof of concept for using SHAP to explain offline RL for flight control, in both discrete and continuous environments. After training discrete A2C and continuous DDPG agents in an high-dimensional environment with continuous states, their decision-making process is globally and locally explained using SHAP. Local explanations, presented using a waterfall plot, show the positive and negative contributions for the prediction of the optimal action at one time step. While this local explanation can give some idea relation between feature value and SHAP value, more useful insights are obtained by concatenation of the local explanations. These global explanations show feature importance, and relation between feature and SHAP value. The global importance of all features can be compared using the SHAP summary plot, while the dependence plot shows a detailed relation between feature value and SHAP value for one feature. Finally, the RL-SHAP diagrams are used in the preliminary analysis to visualize how the SHAP values change over time, and what feature values are causing these contributions to the model output.

The main contributions of this research are presented in the scientific paper, where the proof of concept of SHAP developed during the preliminary analysis is extended to online adaptive RL. The IDHP framework, controlling a simulated Cessna Citation I aircraft, is used to generate online adaptive RL data. The novel proposed CWSD algorithm is used to identify segments with minimal variation in model parameters. It is

demonstrated that SHAP can accurately explain the learned control strategy within these segments. Furthermore, the novel combined dependence plot allows the control strategies of different segments to be compared, in terms of feature importance and relation between feature value and their contribution to the model output. The insights into feature importance can be used to limit RL agent complexity, by only selecting the features relevant during the control task. The use of SHAP and CWSD has led to the discovery that the control strategy of the used IDHP framework can locally be described using linear control laws. It has been verified that the extracted linear control laws match the actual actuator deflections commanded by the IDHP controllers when using fixed weights. Furthermore, these linear control laws have been validated by using these to control the simulated Cessna Citation, showing stable and similar control behavior when following the same mission profile used to explain online adaptive IDHP. This demonstrates that SHAP can be used to extract control laws from flight data. When adaptation to unknown flight circumstances is not required, this control law extraction could be useful to completely or partly replace adaptive RL agents with simpler control laws, for example when the computational budget is limited. Finally, it has been demonstrated how the combined dependence plots can be used to illustrate the fault-tolerance of IDHP. These plots illustrate different input-output mappings for a lateral IDHP controller after a less effective aileron failure is triggered. Furthermore, these combined dependence plots show that also the rudder control strategy is updated due to the adaptation to the aileron failure. This is an example of divergent control behavior identified using SHAP.

8.2. ANSWERS TO THE RESEARCH QUESTIONS

The first research question is answered using the literature review presented in Chapter 3 and the scientific paper presented in Part I.

Research question 1

1. **How can RL methods be used for autonomous flight control?**
   - 1.1. *What is the state-of-the art in RL for flight control?*
     The comparative literature study reviews recent publications in the realm of adaptive critic designs for flight control. From the six classic adaptive critic designs, Dual Heuristic Programming shows the best tracking performance and robustness. Recently DHP has been extended using incremental control techniques, resulting in IDHP. This framework shows better tracking performance, reduced settling-times, and most importantly removes the need of a prior offline training phase. Therefore IDHP is regarded as the state-of-the-art in RL for flight control and is therefore the application for this thesis.
   - 1.2. *Which aspects of RL for flight control are currently the least transparent?*
     The drawback of IDHP, like other approximate RL methods, is the lack of transparency due to the use of complex function approximators. Furthermore, the complex architecture of IDHP, including many feedforward signal and backpropagation routes, makes the decision-making process of the agent harder to understand.
   - 1.3. *Which working principles of RL for flight control are most helpful to explain for flight control engineers or other experts working with the algorithms?*
     In actor-critic RL frameworks, like IDHP, the actor module is ultimately responsible for the actual control output based on the measured system and reference states. By explaining this element of the RL framework, the decision-making process of the agent becomes more interpretable. The results presented in the scientific paper show that illustrating the input-output relations of the actor NN gives meaningful insights into the control strategy employed by the IDHP framework. This explanation methodology, however, only gives insights in what the agent has learned, and not why the agent has learned this behavior. To interpret the reasoning for learning this control strategy, expert knowledge is required. As the other elements in IDHP, the critic and the incremental model, are used to determine the parameters of this actor NN, extracting knowledge from these modules might lead to more direct insights into the reason why the agent has learned a specific control strategy.

The XAI literature review presented in Chapter 4 is used to answer the following question:

Research question 2

2. **How can eXplainable Artificial Intelligence (XAI) techniques be used to gain more insight into the working principles of RL?**
   2.1. *What is the state-of-the-art of transparent RL algorithm design?*
   Out of the available transparent RL publications, explainable fuzzy RL and reward decomposition are selected as potentially useful techniques for adaptive control using RL. Both these methods provide useful explanations for the reward signal in RL. However, this reward signal is often a straightforward penalty for tracking error in adaptive flight control. Furthermore, interpretability is one of the key design drivers of the control system for transparent RL techniques. However, this often results in a compromise of accuracy for improved interpretability, which is unacceptable for many aerospace applications. This suite of techniques is therefore not selected for this research.
   2.2. *What is the state-of-the-art of post-hoc explainability techniques suitable for RL?*
   The reviewed post-hoc explainability techniques, LIME and SHAP, provide insights into the mechanics of already designed AI methods. From this suite of techniques, SHAP is deemed the state-of-the-art due to its superior accuracy in generating explanations. SHAP is proven to provide useful explanations for RL applied for engineering control. Furthermore, SHAP allows ML models to be explained both locally and globally. A local explanation for RL is the prediction of the optimal action at one time step, while global explanations describe the input-output dynamics of the described model on a larger scale.

Finally research question 3 is answered using the findings of the preliminary analysis presented in Chapter 5 and those included in the scientific paper, presented in Part I. The answers to research question 3 are printed on the next page.

Research question 3

3. **How can XRL techniques be used to increase the transparency of RL for adaptive flight control?**
   - 3.1. *Which XAI techniques can be integrated with RL for adaptive flight control?*
     As described in RQ 1.3, the actor module of the IDHP framework is opaque due to the NN function approximator, while it is the most important module to describe the flight mechanics, as this module is ultimately responsible for choosing the actuator outputs based on the measured states. Making this module more interpretable will enhance the transparency of the whole framework. For explaining this input-output mapping, SHAP is a good candidate as it can describe the relation between the output and every individual input.
   - 3.2. *How can the proposed XAI technique be used to identify the most important flight control inputs and analyze how do these influence the actuator output?*
     It has been demonstrated that SHAP can accurately extract the relations between the adaptive RL framework's input and the commanded actuator deflections for both online and offline RL agents. The waterfall plot can be used to create an explanation for a single time step, showing every feature's contribution to the NN output. The key strength of SHAP lies in the global patterns visible through the analysis of many time steps. The feature importance can be inspected using the SHAP summary plot, which ranks the features based on their average contribution to the model output. The relation between model output and a specific feature can be observed in detail using the SHAP dependence plot, and the RL-SHAP diagram can be used to show these relations over time, illustrating the interaction between features. For online adaptive RL applications, where the input-output relations often change during the control task, the combined dependence plot can be used to assess the feature importance and input-output simultaneously for all analyzed time segments. The accuracy of the SHAP explanation model has been verified by comparing its output with the IDHP controller output with fixed weights. Furthermore, the discovered linear relations between feature value and their contribution to the model output have been validated by using these control laws to control the simulated Cessna Citation. As SHAP values can be used to determine feature importance, this analysis can help in selecting the most relevant features out of a list of possible features. Selecting the minimum number of features, while still acquiring good control performance, minimizes the complexity of the control system and hence improves transparency of the control system. Furthermore, this feature selection using SHAP can be used to select sensors required for control of an aerospace application. As an example, for miniature unmanned aerial vehicles where the power and weight budget for sensors is limited, SHAP feature importance analysis can help in selecting the most relevant sensors for control of the vehicle.
   - 3.3. *What is the influence of changing model weights to the accuracy of explanations produced by the XAI technique?*
     The CWSD algorithm is introduced to select segments with minimal variance of actor model parameters. Within these segments, it has been verified that the SHAP model output matches the IDHP controller when its model parameters are fixed. It has been shown that inaccuracies between the SHAP model output and the actual IDHP controller output are caused by the constant-weight assumption, and not by the SHAP calculations. When the model parameters are changing during online adaptive flight control, there is always a trade-off between accuracy of the explanations and generalization when using the proposed CWSD technique. When the limit on NN parameter variation is lenient, long and relatively inaccurate segments are created, improving the generalization and vice versa. However, it is shown that for segments that are too short, the explanations show good global accuracy, but poor local approximation due to the constant weight assumption.

9 | Recommendations

The work presented in this thesis can be considered as an initial effort to increase the transparency of RL for adaptive flight control. To investigate the applicability of the used explanation methodology to different RL frameworks and other control problems, further research is required. Furthermore. based on the mentioned limitations of SHAP and CWSD for explaining RL for adaptive flight control, and other insights gained during the research, the following recommendations for future research are presented:

- As the goal of an explanation is to make something understandable to a human, the quality of an explanation is subjective. Assessing the quality of the generated explanations therefore requires human-in-the-loop experiments. These experiments were out of scope for this research, but inviting control experts to assess the quality and helpfulness of the explanations will likely benefit development of XRL for flight control, as the explanation methodology can be tuned to provide more useful insights. Possible approaches for these human-in-the-loop experiment are presented in [82] and [73], where subjects are asked to assess trust in the ML model's prediction before and after receiving an explanation generated through XAI.
- While the results show that meaningful insights can be extracted from the control strategy learned by IDHP, the RL framework chosen for this thesis, further research is required to investigate whether this also holds true for other adaptive RL frameworks.
- The relations between feature value and SHAP value extracted through SHAP in combination with CWSD are all linear in this research. The adaptive RL framework is used for inner-loop rate control, a relatively simple tracking task. Extending the IDHP hierarchy to perform more complex tasks, such as altitude and pitch angle tracking, and using a more complex NN architecture, could result in more complex and potentially nonlinear input-output relations. To investigate its flexibility, the proposed explanation methodology should be applied to more complex control tasks. The presented combined dependence plots may for example be less informative when the relation between feature value and SHAP value is highly nonlinear, cluttering the plots.
- The proposed explanation methodology allows insights into the learned control strategy, but expert knowledge is still required to interpret why the agent has learned this behavior. Future research could focus on extracting information from other modules present in the RL framework, such as the critic or the incremental model in the case of IDHP, to also explain why the control strategy is learned.
- Feature selection out of a large number of measured states is one of the proposed applications of SHAP for adaptive RL. An idea for future research is therefore to use many potential states for input of the RL agent, and select only the states identified as important through SHAP.
- The progression of the slopes of the extracted linear control laws for δ_e show possible relations with the true air speed V_{tas}. Future research could further investigate this relation to identify if and how the actor NN parameters are adapting due to changes in airspeed. This might result in insights into the adaptation process of the IDHP controller for the selected flight application.
- Finally, one of the intended goals of increased transparency in RL for adaptive flight control is accelerated development of RL algorithms by identifying potential undesired behavior of the agent. During early stages of development, the actor parameters of the RL agent could show more variance than the actor weights in this research. This could result in very short segments, in which the accuracy of the explanations is reduced. Future research should therefore investigate the usefulness of the proposed explanation methodology during early stages of RL algorithm development.

V

APPENDICES

A | Training Convergence - Part of the graded preliminary analysis

Discrete Control Agent Training

The discrete agent is defined to be successfully trained when the mean score of the last 20 episodes surpasses 200, as this score corresponds to a successful landing. The summary of training is illustrated in Figure A.1.

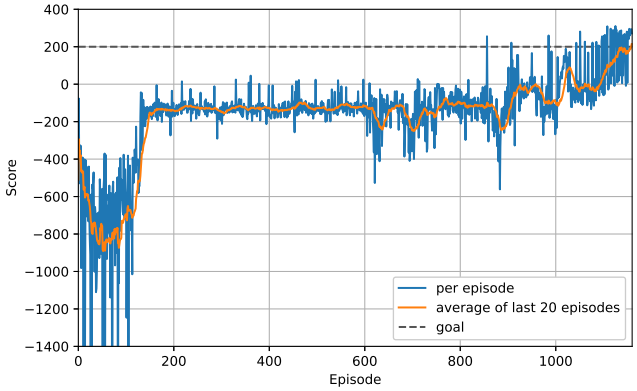


Figure A.1: Agent training for the discrete lunar lander environment, using the A2C algorithm.

Continuous Control Agent Training

The used hyperparameters for training the DDPG agent are summarized in Table 5.3. Training in the continuous lunar lander environment is considered successful when the average score of the last 100 episodes surpasses 200 points. The training curve of the DDPG agent is summarized in Figure A.2.

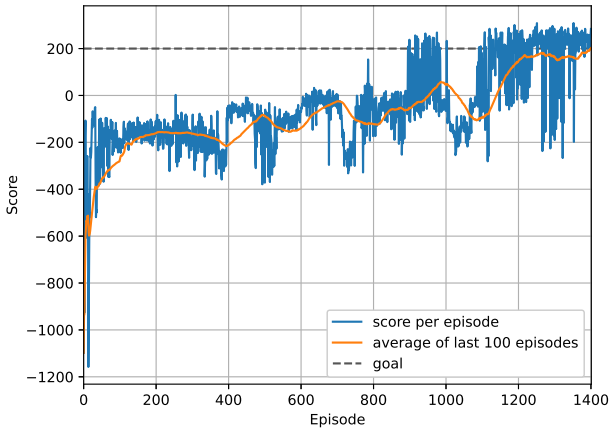


Figure A.2: Agent training for the continuous lunar lander environment, using the DDPG algorithm.

Bibliography

[1] A. Kaplan and M. Haenlein. “Siri, Siri, in my hand: Who’s the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence”. In: *Business Horizons* 62.1 (2019), pp. 15–25. DOI: 10.1016/j.bushor.2018.08.004.

[2] Yogesh K. Dwivedi et al. “Artificial Intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy”. English. In: *International Journal of Information Management* 57 (2019). ISSN: 0268-4012. DOI: 10.1016/j.ijinfomgt.2019.08.002.

[3] K. He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-December (2016), pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[4] L.A. Marina et al. “Deep Grid Net (DGN): A Deep Learning System for Real-Time Driving Context Understanding”. In: *Proceedings - 3rd IEEE International Conference on Robotic Computing, IRC 2019* (2019), pp. 399–402. DOI: 10.1109/IRC.2019.00073.

[5] P. Jamshidi et al. “Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots”. In: *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* 2019-May (2019), pp. 39–50. DOI: 10.1109/SEAMS.2019.00015.

[6] M. Haenlein and A. Kaplan. “A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence”. In: *California Management Review* 61.4 (2019), pp. 5–14. DOI: 10.1177/0008125619864925.

[7] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. “Explainable artificial intelligence: A survey”. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 0210–0215. DOI: 10.23919/MIPRO.2018.8400040.

[8] H.M. El Misilmani, T. Naous, and S.K. Al Khatib. “A review on the design and optimization of antennas using machine learning algorithms and techniques”. In: *International Journal of RF and Microwave Computer-Aided Engineering* 30.10 (2020). DOI: 10.1002/mmce.22356.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[10] Volodymyr Mnih et al. *Human-level control through deep reinforcement learning*. Feb. 2015.

[11] D. Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[12] Kai Arulkumaran et al. “Deep Reinforcement Learning: A Brief Survey”. In: *IEEE Signal Processing Magazine* 34.6 (Nov. 2017), pp. 26–38. ISSN: 1053-5888. DOI: 10.1109/msp.2017.2743240.

[13] F. Santoso, M.A. Garratt, and S.G. Anavatti. “State-of-the-Art Intelligent Flight Control Systems in Unmanned Aerial Vehicles”. In: *IEEE Transactions on Automation Science and Engineering* 15.2 (2018), pp. 613–627. DOI: 10.1109/TASE.2017.2651109.

[14] A.Y. Ng et al. “Autonomous inverted helicopter flight via reinforcement earning”. In: *Springer Tracts in Advanced Robotics* 21 (2006), pp. 363–372. DOI: 10.1007/11552246_35.

[15] W. Koch et al. “Reinforcement learning for UAV attitude control”. In: *ACM Transactions on Cyber-Physical Systems* 3.2 (2019). DOI: 10.1145/3301273.

[16] Danil V. Prokhorov, Roberto A. Santiago, and Donald C. Wunsch. “Adaptive critic designs: A case study for neurocontrol”. In: *Neural Networks* 8.9 (1995), pp. 1367–1372. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(95)00042-9.

[17] Y. Zhou, E.-J. van Kampen, and Q.P. Chu. “Incremental model based online dual heuristic programming for nonlinear adaptive control”. In: *Control Engineering Practice* 73 (2018), pp. 13–25. DOI: 10.1016/j.conengprac.2017.12.011.

[18] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *arXiv* 58.October 2019 (2019), pp. 82–115.

[19] Riccardo Guidotti et al. “A Survey of Methods for Explaining Black Box Models”. In: *ACM Comput. Surv.* 51.5 (Aug. 2018). ISSN: 0360-0300. DOI: 10.1145/3236009.

[20] L. He, N. Aouf, and B. Song. “Explainable Deep Reinforcement Learning for UAV autonomous path planning”. In: *Aerospace Science and Technology* 118 (2021). DOI: 10.1016/j.ast.2021.107052.

[21] Ethem Alpaydın. *Introduction to Machine Learning*. Cambridge, Massachusetts: The MIT Press, 2010.

[22] Kenji Doya. *Reinforcement Learning in Continuous Time and Space*. The MIT Press, 2000.

[23] Dimitri P Bertsekas. *Dynamic Programming and Optimal Control 3rd Edition , Volume II by Chapter 6 Approximate Dynamic Programming Approximate Dynamic Programming*. Vol. II. 2010, pp. 1–200. ISBN: 1886529302.

[24] L. Buşoniu et al. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, Florida: CRC Press, 2010. DOI: 10.1201/9781439821091.

[25] Richard Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Adv. Neural Inf. Process. Syst* 12 (Feb. 2000).

[26] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[27] W.S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259.

[28] J. Schmidhuber. “Deep Learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[29] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-Learning*. Mar. 2016.

[30] George Konidaris, Sarah Osentoski, and Philip Thomas. *Value Function Approximation in Reinforcement Learning Using the Fourier Basis*. 2011.

[31] Richard Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Adv. Neural Inf. Process. Syst* 12 (Feb. 2000).

[32] A. G. Barto, R. S. Sutton, and C. W. Anderson. *Neuronlike adaptive elements that can solve difficult learning control problems*. 1983.

[33] D. V. Prokhorov and D. C. Wunsch. *Adaptive critic designs*. 1997. DOI: 10.1109/72.623201.

[34] H. Van Hoof, J. Peters, and G. Neumann. “Learning of non-parametric control policies with high-dimensional state features”. In: *Journal of Machine Learning Research* 38 (2015), pp. 995–1003.

[35] Richard S Sutton et al. *Policy gradient methods for reinforcement learning with function approximation*. Citeseer, 1999.

[36] Lucian Buşoniu et al. “Approximate reinforcement learning: An overview”. In: *IEEE SSCI 2011: Symposium Series on Computational Intelligence - ADPRL 2011: 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning* May 2014 (2011), pp. 1–8. DOI: 10.1109/ADPRL.2011.5967353.

[37] Ronald J Williams. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. 1992.

[38] B. Kiumarsi and F.L. Lewis. “Actor-critic-based optimal tracking for partially unknown nonlinear discrete-time systems”. In: *IEEE Transactions on Neural Networks and Learning Systems* 26.1 (2015), pp. 140–151. DOI: 10.1109/TNNLS.2014.2358227.

[39] J.D. Boskovic, R. Prasanth, and R.K. Mehra. “A multi-layer autonomous intelligent control architecture for unmanned aerial vehicles”. In: *Journal of Aerospace Computing, Information and Communication* DEC. (2004), pp. 605–628. DOI: 10.2514/1.12823.

[40] D.C. Gandolfo et al. “Energy evaluation of low-level control in UAVs powered by lithium polymer battery”. In: *ISA Transactions* 71 (2017), pp. 563–572. DOI: 10.1016/j.isatra.2017.08.010.

[41] H. Bayerlein, P. De Kerret, and D. Gesbert. "Trajectory Optimization for Autonomous Flying Base Station via Reinforcement Learning". In: *IEEE Workshop on Signal Processing Advances in Wireless Communications, SPAWC* 2018-June (2018). DOI: 10.1109/SPAWC.2018.8445768.

[42] R. Furfaro and R. Linares. "Waypoint-Based generalized ZEM/ZEV feedback guidance for planetary landing via a reinforcement learning approach". In: *Advances in the Astronautical Sciences* 161 (2017), pp. 401–416.

[43] D. Miller and R. Linares. "Low-thrust optimal control via reinforcement learning". In: *Advances in the Astronautical Sciences* 168 (2019), pp. 1817–1834.

[44] Y. Li et al. "Autonomous waypoints planning and trajectory generation for multi-rotor UAVs". In: *DESTION 2019 - Proceedings of the Workshop on Design Automation for CPS and IoT* (2019), pp. 31–40. DOI: 10.1145/3313151.3313163.

[45] Y. Zhao et al. "Q learning algorithm based UAV path learning and obstacle avoidence approach". In: *Chinese Control Conference, CCC* (2017), pp. 3397–3402. DOI: 10.23919/ChiCC.2017.8027884.

[46] Z. Ma et al. "A saliency-based reinforcement learning approach for a UAV to avoid flying obstacles". In: *Robotics and Autonomous Systems* 100 (2018), pp. 108–118. DOI: 10.1016/j.robot.2017.10.009.

[47] F. Fei et al. "Learning extreme hummingbird maneuvers on flapping wing robots". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2019-May (2019), pp. 109–115. DOI: 10.1109/ICRA.2019.8794100.

[48] A. Scorsoglio et al. "Image-based deep reinforcement learning for autonomous lunar landing". In: *AIAA Scitech 2020 Forum* 1 PartF (2020). DOI: 10.2514/6.2020-1910.

[49] S. Heyer, D. Kroezen, and E. van Kampen. "Online adaptive incremental reinforcement learning flight control for a cs-25 class aircraft". In: *AIAA Scitech 2020 Forum* 1 PartF (2020). DOI: 10.2514/6.2020-1844.

[50] S. Lee and H. Bang. "Automatic Gain Tuning Method of a Quad-Rotor Geometric Attitude Controller Using A3C". In: *International Journal of Aeronautical and Space Sciences* 21.2 (2020), pp. 469–478. DOI: 10.1007/s42405-019-00233-x.

[51] M.W. Goedhart et al. "Machine learning for flapping wing flight control". In: *AIAA Information Systems-AIAA Infotech at Aerospace, 2018* 209989 (2018). DOI: 10.2514/6.2018-2135.

[52] D.V. Prokhorov and D.C. Wunsch II. "Adaptive critic designs". In: *IEEE Transactions on Neural Networks* 8.5 (1997), pp. 997–1007. DOI: 10.1109/72.623201.

[53] George G. Lendaris and James C. Neidhoefer. "Guidance in the Use of Adaptive Critics for Control". In: *Handbook of Learning and Approximate Dynamic Programming*. John Wiley Sons, Ltd, 2004. Chap. 4, pp. 97–124. ISBN: 9780470544785. DOI: https://doi.org/10.1002/9780470544785.ch4.

[54] M. Szuster and Z. Hendzel. "Discrete Globalised Dual Heuristic Dynamic Programming in Control of the Two-Wheeled Mobile Robot". In: *Mathematical Problems in Engineering* 2014 (2014). DOI: 10.1155/2014/628798.

[55] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch. "Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator". In: *IEEE Transactions on Neural Networks* 13.3 (2002), pp. 764–773. DOI: 10.1109/TNN.2002.1000146.

[56] J. Si and Y.-T. Wang. "On-line learning control by association and reinforcement". In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 264–276. DOI: 10.1109/72.914523.

[57] S. Ferrari and R.F. Stengel. "Online adaptive critic flight control". In: *Journal of Guidance, Control, and Dynamics* 27.5 (2004), pp. 777–786. DOI: 10.2514/1.12597.

[58] E. Van Kampen, Q.P. Chu, and J.A. Mulder. "Continuous adaptive critic flight control aided with approximated plant dynamics". In: *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2006* 5 (2006), pp. 2989–3016. DOI: 10.2514/6.2006-6429.

[59] Ye Zhou. "Online reinforcement learning control for aerospace systems". PhD thesis. Delft University of Technology, May 2018.

[60] L. Bainbridge. "Ironies of automation". In: *Automatica* 19.6 (1983), pp. 775–779. DOI: 10.1016/0005-1098(83)90046-8.

[61] D. Gunning and D.W. Aha. “DARPA’s explainable artificial intelligence program”. In: *AI Magazine* 40.2 (2019), pp. 44–58. DOI: 10.1609/aimag.v40i2.2850.

[62] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez. “Explainability in deep reinforcement learning”. In: *Knowledge-Based Systems* 214 (2021). DOI: 10.1016/j.knosys.2020.106685.

[63] Erika Puiutta and Eric MSP Veith. *Explainable Reinforcement Learning: A Survey*. 2020. arXiv: 2005.06247 [cs.LG].

[64] Jens Jaekel, Ralf Mikut, and Georg Bretthauer. “Fuzzy Control Systems”. In: Jan. 2004.

[65] Rolando Bautista-Montesano, Rogelio Bustamante-Bello, and Ricardo A. Ramirez-Mendoza. “Explainable navigation system using fuzzy reinforcement learning”. In: *International Journal on Interactive Design and Manufacturing* 14.4 (2020), pp. 1411–1428. ISSN: 19552505.

[66] Sefer Kurnaz, Omer Cetin, and Okyay Kaynak. “Adaptive neuro-fuzzy inference system based autonomous flight control of unmanned air vehicles”. In: *Expert Systems with Applications* 37.2 (2010), pp. 1229–1234. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2009.06.009.

[67] M. Santos, V. López, and F. Morata. “Intelligent fuzzy controller of a quadrotor”. In: *Proceedings of 2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering, ISKE 2010* (2010), pp. 141–146. DOI: 10.1109/ISKE.2010.5680812.

[68] Edgar N. Sanchez, Hector M. Becerra, and Carlos M. Velez. “Combining fuzzy, PID and regulation control for an autonomous mini-helicopter”. In: *Information Sciences* 177.10 (2007). Including Special Issue on Hybrid Intelligent Systems, pp. 1999–2022. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2006.10.001.

[69] Robert L. Wade and Gregory W. Walker. “Flight test results of the fuzzy logic adaptive controller-helicopter (FLAC-H)”. In: *Navigation and Control Technologies for Unmanned Systems*. Ed. by Scott A. Speigle. Vol. 2738. International Society for Optics and Photonics. SPIE, 1996, pp. 200–208.

[70] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. Tech. rep. 2004.

[71] Zoe Juozapaitis et al. “Explainable reinforcement learning via reward decomposition”. In: *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*. 2019.

[72] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[73] M.T. Ribeiro, S. Singh, and C. Guestrin. “"Why should i trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 13-17-August-2016 (2016), pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[74] Yujia Zhang et al. *"Why Should You Trust My Explanation?" Understanding Uncertainty in LIME Explanations*. 2019. arXiv: 1904.12991 [cs.LG].

[75] D. Slack et al. “Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods”. In: *AIES 2020 - Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (2020), pp. 180–186. DOI: 10.1145/3375627.3375830.

[76] R. Liessner, J. Dohmen, and M. Wiering. “Explainable reinforcement learning for longitudinal control”. In: *ICAART 2021 - Proceedings of the 13th International Conference on Agents and Artificial Intelligence* 2 (2021), pp. 874–881.

[77] T. Wood et al. “An interpretable machine learning model of biological age [version 1; peer review: 2 approved with reservations]”. In: *F1000Research* 8 (2019). DOI: 10.12688/F1000RESEARCH.17555.1.

[78] Scott Slundberg. *SHAP Documentation*. URL: https://github.com/slundberg/shap.

[79] S.G. Rizzo, G. Vantini, and S. Chawla. “Reinforcement Learning with Explainability for Traffic Signal Control”. In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019* (2019), pp. 3567–3572. DOI: 10.1109/ITSC.2019.8917519.

[80] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[81] T.P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016).

[82] P. Madumal et al. “Explainable reinforcement learning through a causal lens”. In: *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence* (2020), pp. 2493–2500.