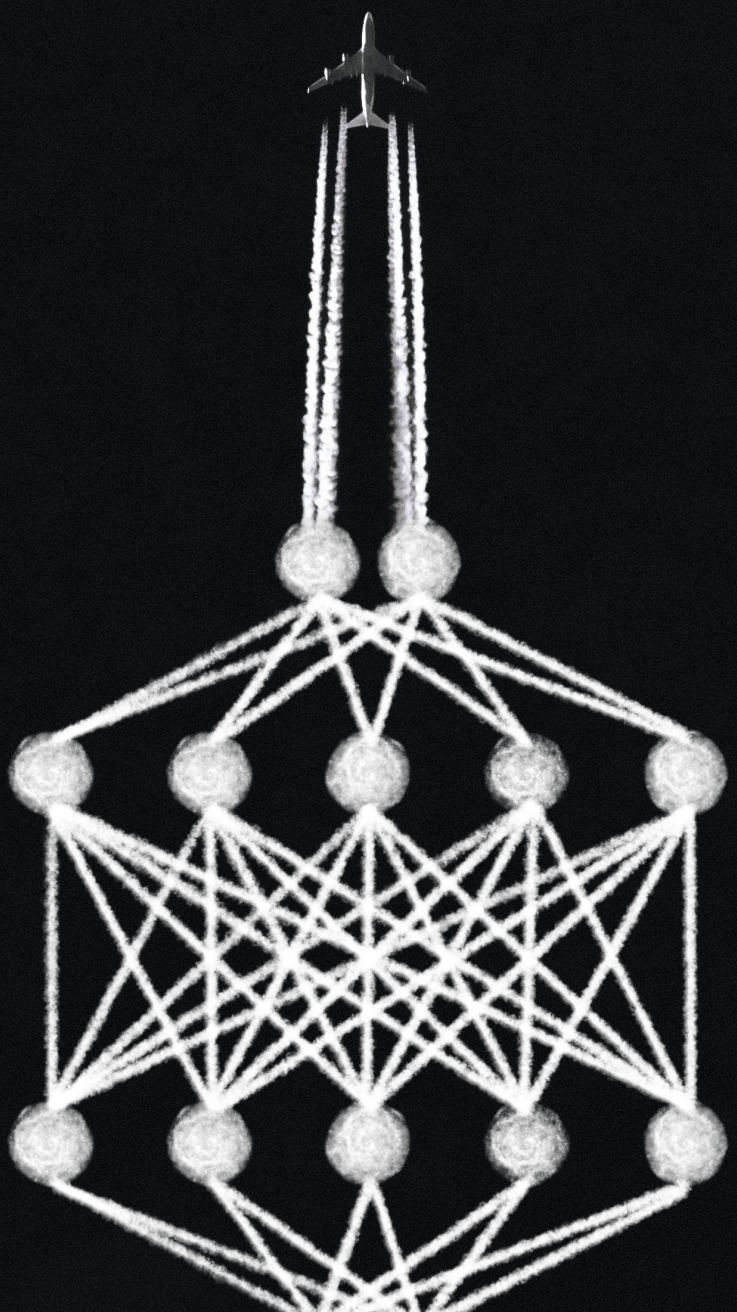
Safe & Intelligent Control

Fault-tolerant Flight Control with Distributional and Hybrid Reinforcement Learning using DSAC and IDHP

Lucas Vieira dos Santos



Delft University of Technology

TU Delft

Safe & Intelligent Control

Fault-tolerant Flight Control with Distributional and Hybrid Reinforcement Learning using DSAC and IDHP

by

Lucas Vieira dos Santos

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on July 13, 2023 at 9:30

Thesis committee:

Chair:	Dr. Coen de Visser
Supervisor:	Dr. Erik-Jan van Kampen
External examiner:	Dr. Erwin Mooij
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	September, 2022 - July, 2023
Student number:	4833570

Cover: Saab Gripen at Farnborough International Airshow by Simon Fitall on Unsplash (Modified)

Preface

We are experiencing a technological revolution where artificial intelligence is aiding us in solving various challenges. In the aviation industry, machine learning is proving to be a promising tool in improving the safety and efficiency of autonomous flight control systems. This thesis delves into implementing Reinforcement Learning to develop robust and reliable flight control systems, ultimately improving aviation safety.

I want to express my gratitude to all those who have supported me on my journey. I am grateful to my supervisor Dr. ir. E. van Kampen for introducing me to the world of autonomous control and guiding me through it. I am also thankful to my family for their unconditional support and love, which has been essential in helping me achieve my dreams. My partner Nick has always been a source of inspiration and a strong believer in my potential. I must also extend my gratitude to my dog Luna, whose simple joy keeps reminding me to appreciate life's little moments. This final work, that marks the end of my learning path in Delft, was a collective effort, and I want you to know that this thesis belongs as much to you as it does to me. Thank you.

Lucas Vieira dos Santos
Delft, June 2023

Contents

Preface ii

Nomenclature v

Acronyms vi

1 Introduction 1
- 1.1 The History of Fault-Tolerant Flight Control Systems . . . 1
- 1.2 The Motivation for Reinforcement Learning in Flight Control . . . 2
  - 1.2.1 The Impact of RL-based Controllers in Current Aviation . . . 2
  - 1.2.2 The Impact of RL-based Controllers on Future Aviation . . . 3
- 1.3 Research Objectives and Questions . . . 4
- 1.4 Report Outline . . . 6

I Research Paper 7

II Literature Study 28

2 Literature Review: A Survey on Reinforcement Learning for Flight Control 29
- 2.1 Reinforcement Learning: The Bridge Between Machine Learning and Decision Making . 29
  - 2.1.1 The Structure of RL Problems and the Markov Property . . . 30
  - 2.1.2 Reward and Returns: Key Concepts in Reinforcement Learning . . . 31
  - 2.1.3 Policy and Value Functions: Enabling Optimal Decision Making . . . 32
  - 2.1.4 Classification of Reinforcement Learning Algorithms . . . 33
- 2.2 Tabular Solutions in Reinforcement Learning . . . 34
  - 2.2.1 Dynamic Programming . . . 34
  - 2.2.2 Monte Carlo Method . . . 36
  - 2.2.3 Temporal Difference . . . 37
- 2.3 Approximate Solutions in Reinforcement Learning . . . 38
  - 2.3.1 Neural Networks as Function Approximators . . . 38
  - 2.3.2 Actor-Critic Designs in Approximate Dynamic Programming . . . 39
- 2.4 Deep Reinforcement Learning: An Overview of Recent Advances . . . 41
  - 2.4.1 Deep Q-Networks . . . 42
  - 2.4.2 Policy Gradient Methods . . . 42
  - 2.4.3 Deep Deterministic Policy Gradient . . . 42
  - 2.4.4 Twin Delayed Deep Deterministic Policy Gradient . . . 43
  - 2.4.5 Soft Actor-Critic . . . 44
  - 2.4.6 Distributional Soft Actor-Critic . . . 44
- 2.5 State-of-the-art of DRL for Flight Control Systems . . . 45
- 2.6 Concluding Remarks . . . 50

3 Preliminary Analysis: Comparative Study of SAC, TD3, and DSAC 51
- 3.1 Motivation for a Preliminary Analysis . . . 51
- 3.2 Experimental Setup of the Plant Dynamics and Environment . . . 51
- 3.3 Experiment I: Comparing Tracking Tasks . . . 53
- 3.4 Experiment II: Comparing Observation Vectors . . . 54
- 3.5 Experiment III: Comparing Reward Functions . . . 56
- 3.6 Concluding Remarks . . . 58

III Additional Results **59**

4 Robustness Analysis of the Hybrid Controller **60**
4.1 The hybrid controller . . . 60
4.2 Sensitivity Analysis of IDHP's Hyperparameters . . . 61
4.3 Robustness to Varying Reference Signals . . . 63
4.4 Robustness to Noise and Bias . . . 63
4.5 Concluding Remarks . . . 64

5 Fault-Tolerance and Safety **68**
5.1 Operation Under Reduced Control Effectiveness . . . 68
5.1.1 Reduced Elevator Effectiveness . . . 68
5.1.2 Reduced Aileron Effectiveness . . . 68
5.2 Concluding Remarks . . . 68

6 Verification and Validation **71**
6.1 Verification . . . 71
6.1.1 Verification of RL algorithms . . . 71
6.1.2 Verification of Citation Environment . . . 72
6.2 Validation . . . 72
6.2.1 Validation of RL algorithms . . . 73
6.2.2 Validation of Citation Environment . . . 73
6.3 Reliability Analysis . . . 73
6.3.1 Reliability Through Stratified Bootstrap Confidence Intervals . . . 74
6.3.2 Reliability Through Performance profile . . . 74
6.3.3 Reliability of Agents Across Aggregated Metrics . . . 74
6.4 Work Reproducibility . . . 76
6.5 Concluding Remarks . . . 76

IV Closure **77**

7 Reflection on Research Questions **78**

8 Conclusion **80**

9 Recommendations **81**

V Appendices **83**

A Short Period Linear Time-Invariant System of Cessna Citation **84**

References **86**

Nomenclature

Greek letters

α	Angle of attack
γ	Discount rate
π	Policy
π'	Greedy policy
π_*	Optimal policy

Latin letters

$\mathcal{S}$	States space
a	Action
A_t	Action at time t
G_t	Return at time t
n	Number of samples
q	Pitch rate
$Q(S_t, A_t)$	Estimated action-value function
q_{ref}	Pitch rate reference signal
$q_\pi(s, a)$	Action-value function under policy π
r	Reward
R_t	Reward at time t
s	State
s'	Immediate next state
S_t	State at time t
$V(S_t)$	Estimated state-value function
v_*	Optimal state-value function
$v_\pi(s)$	State-value function under policy π

Symbols

$\doteq$	Definition
$\leftarrow$	Assignment
$\mathbb{E}$	Expected value

Acronyms

ACD Actor-Critic Design. 39, 40
AD Action-dependent. 40
ADGDHP Action-dependent Global Dual Heuristic Programming. 40
ADHDP Action-dependent Heuristic Dynamic Programming. 40
ADP Approximate Dynamic Programming. 39
AI Artificial Intelligence. 2

DASMAT Delft University Aircraft Simulation Model and Analysis Tool. 45, 46, 48, 49, 72, 73
DDPG Deep Deterministic Policy Gradient. 42–44
DHP Dual Heuristic Programming. 40, 47
DL Deep learning. 41
DNN Deep Neural Network. 41, 42, 48, 50
DOF Degrees Of Freedom. 45
DP Dynamic Programming. 34, 36, 37, 39
DQN Deep Q-Network. 42–44
DRL Deep Reinforcement Learning. 41, 42, 44, 45, 48, 50
DSAC Distributional Soft-Actor critic. 44, 45, 49–58, 60–65, 68, 71–76, 78–81

FCS Flight Control System. 1–3, 41, 45, 47, 51, 60
FTFCS Fault-Tolerant Flight Control System. 2–4, 50

GDHP Globalised Dual Heuristic Programming. 40, 41
GPI Generalised Policy Iteration. 36, 37

HDP Heuristic Dynamic Programming. 39, 40

iADP Incremental Approximate Dynamic Programming. 40, 41
IDHP Incremental Dual Heuristic Programming. 40, 41, 45–51, 60, 61, 63, 64, 71, 72, 78–80
IFC Initial Flight Condition. 49
IGDHP Incremental Globalised Dual Heuristic Programming. 40, 41
IHDP Incremental Heuristic Dynamic Programming. 40
IQM Interquartile Mean. 75

LOC Loss of Control. 3
LTI Linear Time-Invariant. 51, 52, 84

MC Monte Carlo. 36, 37
MDP Markov Decision Process. 30
ML Machine Learning. 2, 29, 30

nMAE normalised Mean Absolute Error. 61–63, 65, 68, 74, 75, 79
NN Neural Network. 38, 42, 43, 48, 61, 78

PI Policy Iteration. 34–36, 39
PID Proportional–Integral–Derivative. 2, 3, 45, 50
PPO Proximal Policy Optimisation. 45, 48

RL Reinforcement Learning. 2–6, 30–34, 36, 38, 39, 41–51, 53, 54, 58, 60, 61, 68, 71–74, 78, 80, 81

SAC Soft-Actor Critic. 44, 45, 47–58, 60–65, 68, 71–75, 78–81

TD Temporal Difference. 37
TD3 Twin Delayed DDPG. 43, 44, 48, 50–58

UAV Unmanned Aerial Vehicle. 45

VI Value Iteration. 36

1

Introduction

1.1. The History of Fault-Tolerant Flight Control Systems

Since the Wright brothers' $59\,\mathrm{s}$ flight in 1903 [1], aviation rapidly progressed towards longer, higher and faster flights. The expansion of the flight envelope brought to the surface the challenges of varying aerodynamics on Flight Control System (FCS) – even if an aircraft is stable in a particular flight condition, it might be unstable in another. The need and development of controllers that provide robust control in different flight regimes have led to remarkable achievements, like autonomous flight control.

Although classical control theory only surged in the late 1930s, the first automatic control demonstration occurred in 1914. Lawrence Sperry, an aviation pioneer, adapted a gyrocompass into an aircraft and demonstrated the first auto-stabilised flight [2]. The demonstration amazed a Parisian crowd as the aircraft remained stable even when Sperry and his copilot, Emil, moved to the vehicle's wings, as shown in figure 1.1. This early invention revolutionised flight control paving the way for future developments. Just 33 years later, the first fully automatic flight, including take-off and landing, was achieved [3].

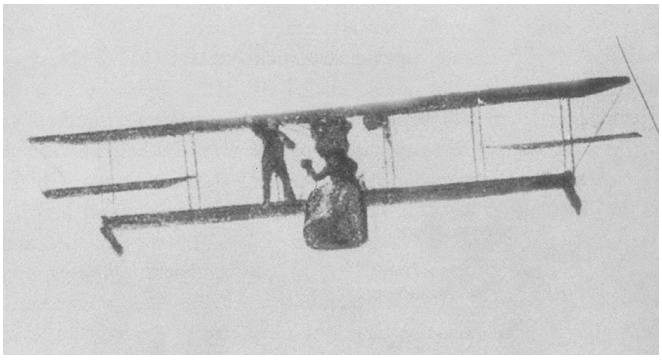


Figure 1.1: Photo of Sperry's groundbreaking flight in 1914. The image show Emil standing on the aircraft's wing while Sperry raises his hands from the control. The photo is retrieved from Historic Wings [4].

During the 1950s, there was a growing interest in FCS that could self-adapt to allow fight in a wide envelope [5]. The field flourished in the 1960s with the development of supersonic aircraft, which required more advanced control. Consequently, gain scheduling was invented to allow a controller to have different parameters for each flight condition, making them optimal throughout the entire envelope [3, 6]. However, gain scheduling controllers lack the means to adapt to changing dynamics, such as the failure of a control surface. In such cases, the controller shows suboptimal performance or even loses control entirely [7].

While adaptive control research aimed to expand the flight envelope, a separate branch was interested in making control systems fault-tolerant [8]. Initially, the focus was on enabling fault tolerance with redundant control devices. However, this approach increases costs and maintenance needs. The

concepts of adaptive control and fault tolerance were later merged, leading to the field of Fault-Tolerant Flight Control System (FTFCS). The combination resulted in controllers that allowed robust control in the entire envelope and could adapt to unexpected conditions, such as a change in dynamics [9].

Research in the 1980s led to further advances on FTFCS with the introduction of Machine Learning (ML) into control system [8]. During this time, Reinforcement Learning (RL), a subfield of ML, was also emerging. RL combined the concepts of learning through trial-and-error from Artificial Intelligence (AI) with the concepts of dynamic programming from optimal control [10].The issue RL addressed was that optimal control by itself could not develop adaptive controllers, and it requires the knowledge of the system dynamics [11]. For flight control, the first flight tests of a ML-based controller happened in the 1990s from a collaboration between NASA and Boeing [12].

Reinforcement Learning (RL) is a fast-growing research field with various applications. As RL algorithms continue to evolve and demonstrate their potential to learn more efficiently and handle more complex tasks, they are becoming increasingly relevant for developing the next generation of FCS. In flight control, RL offers new possibilities for robust and adaptable control. Using RL-based controllers can significantly improve flight safety and reliability, leading to a safer and more efficient industry.

1.2. The Motivation for Reinforcement Learning in Flight Control

In March 2005, Air Transat's Airbus A310-309 experienced a sudden bang followed by a Dutch roll motion, surprising its crew. The pilots disengaged the autopilot and manually controlled the aircraft for the remainder of the flight, successfully landing it despite the difficulty in maintaining control. Subsequent flight control checks revealed no anomalies, and only during a visual inspection after the aircraft had shut down did they discover that most of the rudder was missing [13].

This story is a rare example of a successful response to an aircraft's loss of control. In most cases, neither pilots nor conventional autopilots can adapt fast enough. Future flight control systems that can quickly adapt to adverse conditions and faults can increase the reliability and safety of flight operations. Therefore, investing in research on RL applications for FTFCS is crucial as it can significantly impact the aviation industry. This impact ranges from the design to the operation of aircraft, both in the current aviation industry and the future of aviation, as illustrated in figure 1.2.

	Current aviation	Future aviation
Design	Overcoming gain scheduling limitations with adaptive tuning	Facilitating innovative green aircraft designs with tailored control laws
Operation	Enhancing fault tolerance to prevent loss of control accidents	Enabling safe and efficient autonomous flight operations

Figure 1.2: Matrix illustrating the potential impact of RL-based flight control systems on the design and operations of the aviation industry, both in the present and future.

1.2.1. The Impact of RL-based Controllers in Current Aviation

The aviation industry constantly works towards enhancing safety and efficiency in operations. Despite significant advances in flight control technology, traditional systems have limitations in adaptability and fault tolerance. Moreover, the design process for conventional controllers is complex and time-consuming. As a result, there is a need for novel technologies that can improve design efficiency and enhance flight safety. This section discusses how RL-based FCS can address those issues.

Design motivation: Overcoming limitations of gain scheduling

Proportional–Integral–Derivative (PID) is the most commonly used type of flight controller. These controllers adjust the control action using feedback from the system to reduce deviations from a reference state. However, the gains applied to the feedback are a design parameter for which no unique solution exists [14]. Besides, these parameters do not guarantee matching performance in flight conditions

other than those for which they were tuned. As a result, with any disturbance, the controllers may lose performance or even fail [6].

Gain scheduling is the traditional method for making PID operational in different flight conditions. This technique uses multiple linearisation points, each requiring a different tuning. It works as if a separate controller was designed for each point. While this allows adequate control within the design envelope, the controller may not function properly outside this range or in situations with a change in the system dynamics [3].

Fault-Tolerant Flight Control Systems are a way of overcoming the drawbacks of gain scheduling. Implementing those controllers with Reinforcement Learning is promising due to the field's capability of developing robust, adaptive and model independent controllers. With research in RL-based controller advancing, three major impacts in flight control design are expected:

1. Eliminating the need for manual tuning: Gain scheduling is a time-consuming and expensive manual process. Engineers must design a controller for each flight condition within the aircraft's operational range.
2. Expanding the operational range: Gain-scheduled controllers are only optimal at the design envelope and cannot adapt to adverse conditions.
3. Removing model dependency: Gain scheduling requires identification of a system's model, which increases the complexity and cost of the design process [15].

Operations motivation: Preventing loss of control accidents
The Air Transat incident is an exception to the usual unfortunate outcomes of an aircraft LOC. According to IATA [16] report on LOC accidents from 2009 to 2018, there were 64 incidents in commercial aviation, with only four of them not causing fatalities. LOC accounts for 8% of all types of accidents in aviation, but it is the most deadly accident type, responsible for 60% of all fatalities in commercial aviation.

One promising solution to mitigate the frequency of LOC accidents are FTFCSs, which can preserve control even in adverse situations [17]. Although current control systems can be pre-configured to operate in different failure scenarios [18], intelligent systems that learn to restore control even in unanticipated situations offer a more promising approach. By doing so, these systems can adapt to real-time scenarios, allowing safer operations.

1.2.2. The Impact of RL-based Controllers on Future Aviation

The future of aviation is marked by significant challenges, including the continuous growth of civil aviation, the shift towards autonomous operations, and the need for industry decarbonisation [19]. In this context, flight control systems will play a critical role. Moving from traditional rule-based systems to intelligent and adaptive ones will enable FCS to provide safe and robust control while also contributing to developing other technologies in the field. This section aims to highlight the motivation for investing in research in those systems by emphasising their potential impact on the aerospace industry.

Design Motivation: Advancing Green Aircraft Designs
Since the pre-industrial era, human activity has raised global warming by $1\,^{\circ}\mathrm{C}$. Limiting the increase to $1.5\,^{\circ}\mathrm{C}$ is the most optimistic scenario, as irreversible changes, including ecosystem extinctions, are expected to occur at $2\,^{\circ}\mathrm{C}$ increase [20]. While aviation accounts for only 2% of total CO_2 emissions, its impact on global warming is much more significant due to non-CO_2 emissions such as NOx, water vapour, and soot, as well as emissions at high altitudes [21].

As the aviation sector continues to grow in the coming decades [19], there is an urgent need for greener designs that reduce its environmental impact. This challenge requires a combined effort from all design groups, including flight control. Electrification of aircraft and the use of alternative fuels such as biofuels can play a significant role in reducing aviation's environmental impact. In addition, innovative energy-efficient aircraft designs offer a promising path towards greener aviation [22].

An example of an innovative aircraft design that aims to improve fuel efficiency is the Flying V [23], shown in figure 1.3. This concept is a tailless V-shaped flying wing aircraft that merges the fuselage with the wing. The aircraft contains four control surfaces on each wing, including three elevons at the

trailing edge and a rudder built into the winglet. The elevons provide longitudinal and lateral control by controlling pitch and roll, while the rudder provides yaw control [24]. However, the unconventional design of the Flying V poses a challenge to traditional flight control methods, making them difficult to apply. Therefore, this concept requires dedicated flight control laws [25]. Adaptive controllers based on RL offer a promising solution for allowing these types of designs to be implemented effectively.

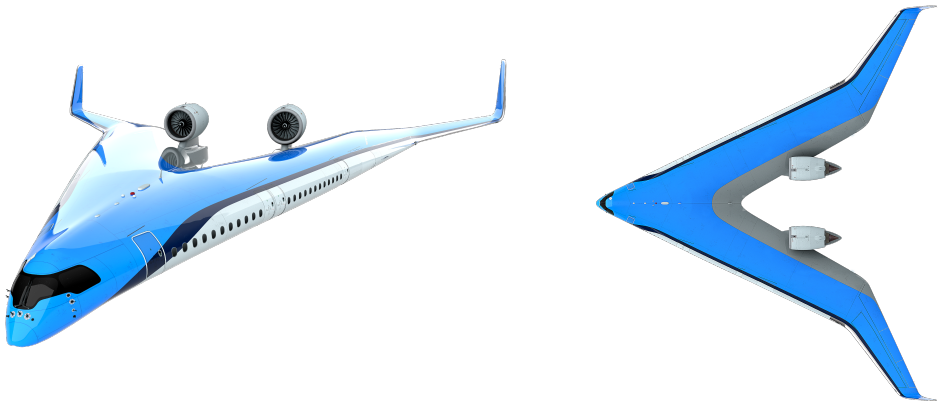


Figure 1.3: Isometric and Top Views of the Flying-V Aircraft Design. Image retrieved from TU Delft [26]

Operations motivation: Enabling autonomous flight
In today's commercial aviation, the cockpit typically consists of two pilots, one serving as a backup in case the other becomes incapacitated. However, the risk of accidents due to pilot incapacitation could be significantly reduced if the onboard control systems could autonomously control the aircraft. Removing this threat will be a first step in allowing single-pilot operations, which could later lead towards fully autonomous aircraft traffic [27].

Moving towards a future of autonomous operations offers many benefits beyond increasing safety levels. For example, it would allow for better airspace utilisation, reducing aircraft separation and operational costs. Additionally, autonomous aircraft systems could contribute to the decarbonisation of the aviation industry, as optimised flight plans could reduce fuel burn by up to 6% [27].

The transition to fully autonomous systems will substantially impact the economy. The shift to single-pilot operations alone could save up to $\$60\,\mathrm{billion}$ annually, and a subsequent transition to fully autonomous systems could save up to $\$110\,\mathrm{billion}$ [27].

1.3. Research Objectives and Questions

Research on flight control is progressing towards intelligent systems, such as FTFCS. Developing and implementing those systems will require extensive research to ensure their reliability and effectiveness in real-world applications. Achieving this goal is crucial to enable aviation to benefit from the potential improvements previously discussed. The objective of this thesis is to contribute to the academic development of this field, with a defined research objective as described below:

Research Objective

This research aims to develop and evaluate a fault-tolerant flight control system with a probability-based and hybrid offline-online Reinforcement Learning algorithm, seeking to advance state-of-the-art methods by enhancing the controller's robustness, adaptability, and safety.

The research objective of this thesis divides into four research questions, the first (Question 1) aims to provide an understanding of the field of Reinforcement Learning (RL), with a focus on its potential applications in flight control systems. The primary goal of this question is to identify the gaps in knowledge that will enable further development of the field through the subsequent research questions.

Research Question 1

Question 1. What is the state-of-the-art in reinforcement learning for fault-tolerant flight control systems?

Question 1.1. What are the key concepts and principles of reinforcement learning, and how are they relevant to control systems design?

Question 1.2. What are the current state-of-the-art algorithms in reinforcement learning, and to what extent do they offer possibilities to fault-tolerant flight control?

Question 1.3. What are the gaps and opportunities for future research in reinforcement learning for flight control systems?

One of the research interests of this thesis is exploring the capabilities of hybrid reinforcement learning to bridge the gap between simulation and real-world applications. With hybrid reinforcement learning, a flight controller can be initially learned and then fine-tuned during operation in changing conditions. The goal of Question 2 is to determine the necessary steps to construct a hybrid controller.

Research Question 2

Question 2. How can hybrid offline-online reinforcement learning techniques be integrated to develop a fault-tolerant flight control system?

Question 2.1. What are the advantages and limitations of a hybrid offline-online reinforcement learning approach?

Question 2.2. How can offline and online reinforcement learning techniques combine to leverage their strengths?

Probabilistic methods in RL are algorithms that deal with uncertainties in the environment and the controller's actions by learning statistical distributions related to their performance. These algorithms are promising for improving the safety of RL-based flight controllers because they make decisions based on various possible outcomes. This research will focus on integrating algorithms that use this distributional information to create safer controllers. This leads to Question 3, which seeks to understand how to apply these methods to flight control.

Research Question 3

Question 3. How can distributional reinforcement learning methods improve the performance of fault-tolerant flight control systems?

Question 3.1. What are the principles and advantages of distributional reinforcement learning, and how does it differ from traditional reinforcement learning methods?

Question 3.2. How can distributional reinforcement learning be used to improve the performance of fault-tolerant flight control systems?

The research's ultimate goal is to demonstrate the effectiveness of the developed controller in real-world flight scenarios, contributing to the development of more reliable autonomous systems. To achieve this goal, Question 4 will compare the performance of the built system with previous methods.

Research Question 4

Question 4. How effective is the resulting fault-tolerant flight control system at achieving robustness, adaptability, and safety in real-world flight scenarios?

Question 4.1. What are the most relevant performance metrics for evaluating the system's performance in different flight conditions?

Question 4.2. How does the fault-tolerant control system compare to traditional controllers regarding stability, performance, and safety?

Question 4.3. How well does the system handle different types of faults and risk situations, and what are the limitations?

1.4. Report Outline

This report describes all the work conducted in this thesis, providing a comprehensive overview of the research. The information is organised into four distinct parts, each with a specific purpose.

- Part I contains the research paper from this study. It encompasses the motivation, the background, the methodology, and the main results of the research.
- Part II contains the literature study that supports this research. Chapter 2 is the literature review, which details the state-of-the-art of Reinforcement Learning and its application to flight control. Chapter 3 is a preliminary analysis comparing the feasibility of various research approaches identified through the literature review process.
- Part III compiles all the additional results not included in the research paper. While chapter 4 showcases extra findings from the investigation into the robustness of the hybrid model, chapter 5 shows the additional results from the fault-tolerance experiments. Lastly, chapter 6 presents the results of the verification and validation procedures.
- Part IV concludes the thesis by reflecting upon the research questions in chapter 7, providing a comprehensive conclusion in chapter 8, and offering recommendations for future research in chapter 9.

Part I

Research Paper

Safe & Intelligent Control: Hybrid and Distributional Reinforcement Learning for Automatic Flight Control

Lucas Vieira dos Santos*
* *Aerospace Engineering, Delft University of Technology (*

The critical challenge for employing autonomous control systems in aircraft is ensuring robustness and safety. This study introduces an intelligent and fault-tolerant controller that merges two Reinforcement Learning (RL) algorithms in a hybrid approach: the Distributional Soft Actor-Critic (DSAC) and the Incremental Dual Heuristic Programming (IDHP). The integration combines the strengths of DSAC in learning a robust control strategy and IDHP in allowing real-time control adaption. Compared to earlier controllers, such as a hybrid using the Soft Actor-Critic (SAC) algorithm and strictly offline DSAC and SAC, our hybrid demonstrates enhanced robustness against changing flight conditions and in the face of sensor noise and bias. During fault tolerance tests, it maintains superior control even when the effectiveness of the aircraft's ailerons and elevators is compromised. By demonstrating the potential of RL-based controllers to provide robustness and fault tolerance, this research advances the feasibility of safe and autonomous flight control operations.

I. Introduction

The advancements in Reinforcement Learning (RL) have led to remarkable achievements in various fields [1]. Notably, RL applications have outperformed humans in video games [2] and enabled robots to handle objects [3] and walk [4]. In the aerospace sector, RL can improve the robustness and fault-tolerance of autonomous systems. Those benefits are reflected in the growing research on RL-based control for aircraft [5], helicopters [6], and unmanned aerial vehicles [7] showing their potential in overcoming the limitations of traditional systems.

Though well-established, traditional Flight Control System (FCS) requires gain scheduling for different points of an aircraft's operational range, introducing complexity to the design and limiting adaptability to adverse conditions, such as system faults and malfunctions [8]. Moreover, Loss of Control (LOC) continues to be a major factor contributing to aviation accidents. Alone, LOC is responsible for 60 % of all fatalities in commercial aviation [9]. This underlines an urgent need for intelligent and adaptive control systems. RL-based FCS emerge as a promising solution. By learning from experience, these systems can offer better control autonomy, simplify the design process, and potentially reduce accidents caused by loss of control.

In the context of autonomous flight control, various RL-based controllers employing different algorithms and strategies have been explored. Offline algorithms, such as Soft Actor-Critic (SAC) and Distributional Soft Actor-Critic (DSAC), have demonstrated their capability to produce robust and efficient aircraft control strategies. However, these offline algorithms have their limitations, as they are not well-suited for adapting the strategies in-flight [10]. In contrast, online algorithms, such as IDHP, can adapt in real-time but raise questions about their robustness and the risks involved in learning while flying. Interestingly, a policy that combines SAC and IDHP has been shown by Teirlinck [11] to leverage the advantages of both algorithms. Most recently, DSAC has caught attention for its ability to create safer policies compared to SAC, as Seres [12] demonstrated the improvement of aircraft control in challenging conditions such as near stalls.

In light of these findings, this study develops an RL-based attitude control system that integrates the offline DSAC algorithm with the online IDHP algorithm. This hybrid model combines the robustness derived from pre-learned policies with the adaptability of online real-time learning. Differently than SAC, DSAC can learn complete distributions of possible outcomes providing deeper insights into the risks associated with different actions. During online operation, IDHP utilises the policy trained by DSAC and dynamically adjusts it according to the flight conditions.

The contributions of this paper are in developing safe and autonomous control systems by constructing fault-tolerant RL-based controllers. We assess the pros and cons of a hybrid offline-online controller, comparing DSAC and SAC as the offline component. Furthermore, we investigate how the hybrid controller contrasts with offline-only policies. At the core of this research is the innovative approach to flight control systems, which combines the hybrid and distributional reinforcement learning approaches to enhance safety without compromising performance.

The remainder of this article is organised as follows. Section II introduces the fundamental concepts of reinforcement learning and the algorithms employed to develop the hybrid controller. Section III delineates the formulation of the flight control task as a reinforcement learning problem and defines the hybrid controller. Section IV presents a discussion on the results of the experiments comparing the different controllers.

II. Background

This section establishes the Reinforcement Learning fundaments and notation, including the algorithms used for developing the hybrid attitude controller.

A. Fundamentals of Reinforcement Learning

Reinforcement Learning is a subfield of machine learning that primarily uses a trial-and-error approach to learn optimal strategies. In RL, an agent learns to make optimal decisions in a task by interacting with an environment. It chooses an action $\boldsymbol{a}_t \in \mathbb{R}^N$ at time t. This action changes the environment state from $\boldsymbol{s}_t$ to $s_{t+1} \in \mathbb{R}^M$ with a probability ρ. Then, the environment informs how satisfactory the state transition was by returning a scalar reward $r_{t+1} \in \mathbb{R}$ [13].

The objective for the agent is to identify the action it should select at each given time to maximise the total rewards, also known as return $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(\boldsymbol{s}_i, \boldsymbol{a}_i)$. The return includes a discount factor $\gamma \in [0, 1]$ that balances the objective of striving for near or long-term rewards.

Actor-critic is a specific type of RL algorithm that divides the tasks of policy learning and return estimation into two separate structures [14]. The Actor, the first structure, is a Neural Network (NN) that learns the policy π. It determines which action $\boldsymbol{a}_t$ should be taken given the state $\boldsymbol{s}_t$. For a stochastic policy, the action is sampled from $\boldsymbol{a}_t \sim \pi(\cdot \mid \boldsymbol{s}_t)$. In contrast, for a deterministic policy, the action is directly derived from $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t)$.

The Critic, the second structure of actor-critic algorithms, approximates the expected return. There are two types of return functions the Critic can approximate: (1) the state-value function in Eq. (1), which informs the expected return for a given state, and (2) the action-value function in Eq. (2), which informs the expected return for a specific state-action pair:

$$V_\pi(\boldsymbol{s}_t) \doteq \mathbb{E}_\pi\left[R_t \mid \boldsymbol{s}_t\right] \tag{1}$$

$$Q_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \doteq \mathbb{E}_\pi\left[R_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t\right] \tag{2}$$

Often, the action-value function is used in its recursive form, better known as the Bellman equation:

$$Q_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\rho, \boldsymbol{a}_{t+1}\sim\pi}\left[Q_\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})\right] \tag{3}$$

B. Incremental Dual Heuristic Programming (IDHP) Algorithm

Incremental Dual Heuristic Programming [15] is an online actor-critic RL algorithm characterised by three parametric structures: the Actor, the Critic and the Incremental Model. Differently from conventional actor-critic algorithms, IDHP incorporates an incremental model to learn an approximation of the system dynamics. The Incremental Model makes IDHP model-free, enabling the online learning process to use an approximation of the system dynamics. The Actor and Critic retain their traditional role in learning the policy and value function.

1. IDHP's Incremental Model

The incremental model learns a linear approximation of the system's non-linear state transition function, denoted as f. The first-order Taylor series expansion is used to approximate the state transition:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx \boldsymbol{x}_t + F_{t-1}\Delta\boldsymbol{x}_t + G_{t-1}\Delta\boldsymbol{u}_t \tag{4}$$

where, $\boldsymbol{x} \in \mathbb{R}^n$ represents the aircraft state vector, and $\boldsymbol{u} \in \mathbb{R}^m$ represents the action vector. F_{t-1} is the system transition matrix, and G_{t-1} is the control effectiveness matrix.

As IDHP is a model-free algorithm, the matrices F_{t-1} and G_{t-1} are unknown. The algorithm learns an approximation of those matrices with Recursive Least Squares (RLS). During each interaction with the environment, the incremental model uses the observed states to enhance the prediction of these matrices, thereby refining the estimation of the system's state transition.

To express the incremental model's update in matrix notation, we define the parameter matrix $\boldsymbol{\Theta} \in \mathbb{R}^{(n+m)\times n}$ as in Eq. (5), and the input information matrix $\boldsymbol{X}_t$ as in Eq. (6). With these matrices, the incremental model can predict the state transition with Eq. (7). The prediction error, denoted as $\boldsymbol{\epsilon}_t \in \mathbb{R}^{(1,n)}$, is computed with Eq. (8), which takes into account the observed change in state $\Delta\boldsymbol{x}_{t+1}$ from the environment.

$$\boldsymbol{\Theta}_{t-1} = \begin{bmatrix} F_{t-1}^T \\ G_{t-1}^T \end{bmatrix} \tag{5}$$

$$\boldsymbol{X}_t = \begin{bmatrix} \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix} \tag{6}$$

$$\Delta\hat{\boldsymbol{x}}_{t+1}^T = \boldsymbol{X}_t^T \cdot \hat{\boldsymbol{\Theta}}_{t-1} \tag{7}$$

$$\boldsymbol{\epsilon}_t = \Delta\boldsymbol{x}_{t+1}{}^T - \Delta\hat{\boldsymbol{x}}_{t+1}{}^T \tag{8}$$

The update to improve the Incremental Model's matrices requires computing the estimation covariance matrix Λ_t with Eq. (9). In this equation γ_{RLS} is the discount factor. It is important to note that this equation is recursive and relies on the covariance matrix from the previous timestep. Consequently, the covariance matrix is unknown at the start of the learning process and must be randomly initialised.

With the prediction error from Eq. (8) and the covariance error from Eq. (9), the Incremental Model's parameter matrix is updated according to Eq. (10). Subsequently, the model makes new predictions on the changes in the system's states and iteratively refines the matrices F and G until their convergence.

$$\Lambda_t = \frac{1}{\gamma_{\text{RLS}}}\left(\Lambda_{t-1} - \frac{\Lambda_{t-1}\boldsymbol{X}_t\boldsymbol{X}_t^T\Lambda_{t-1}}{\gamma_{\text{RLS}} + \boldsymbol{X}_t^T\Lambda_{t-1}\boldsymbol{X}_t}\right) \tag{9}$$

$$\hat{\boldsymbol{\Theta}}_t = \hat{\boldsymbol{\Theta}}_{t-1} + \frac{\Lambda_{t-1}\boldsymbol{X}_t}{\gamma_{\text{RLS}} + \boldsymbol{X}_t^T\Lambda_{t-1}\boldsymbol{X}_t}\boldsymbol{\epsilon}_t \tag{10}$$

2. IDHP's Actor-Network

The Actor-network of the IDHP agent is parameterised with weights denoted as θ. The architecture of this network includes a single hidden layer without a bias term. For attitude tracking, the reward is defined as the negative value of the tracking error, thereby penalising the agent for not following the reference signal. The Actor's loss, denoted as $\mathcal{L}_\pi$, is the negation value of the of the return:

$$\mathcal{L}_\pi(t) = -V(\boldsymbol{s}_t) = -[r_{t+1} + \gamma V(\boldsymbol{s}_{t+1})] \tag{11}$$

Updating the weights of the Actor-network, θ, requires computing the gradient of the loss function with respect to each layer in the network:

$$\nabla_\theta \mathcal{L}_\pi(t) = \frac{\partial \mathcal{L}_\pi(t)}{\partial \theta} = \frac{\partial \mathcal{L}_\pi(t)}{\partial \boldsymbol{a}_t}\frac{\partial \boldsymbol{a}_t}{\partial \theta} = -\Big[\frac{\partial r_{t+1}}{\partial \boldsymbol{s}_{t+1}} + \gamma\hat{\lambda}(\boldsymbol{s}_{t+1})\Big]\hat{G}_{t-1}\frac{\partial \boldsymbol{a}_t}{\partial \theta} \tag{12}$$

Here, $\hat{\lambda}(\boldsymbol{s}_{t+1})$ represents the Critic's prediction and $\partial \boldsymbol{a}_t/\partial\theta$ is computed using backpropagation through the Actor network. Subsequently, the weights of the Actor-Network are updated through gradient descent with a learning rate η_π.

3. IDHP's Critic-Network

The IDHP's Critic network λ is parametrised with weights denoted as ϕ. This network distinguishes itself from traditional critic networks by estimating the derivative of the value function with respect to each state of the aircraft, i.e., $\lambda = \partial V/\partial \boldsymbol{s}$, as opposed to estimating the value function directly. The loss of the Critic is calculated as the mean squared error of the Temporal Difference (TD):

$$\mathcal{L}_\lambda(t) = \frac{1}{2}\Big[\lambda(\boldsymbol{s}_t) - \Big(\gamma\hat{\lambda}(\boldsymbol{s}_{t+1}) + \frac{\partial r_{t+1}}{\partial \boldsymbol{s}_{t+1}}\Big)\frac{\partial \boldsymbol{s}_{t+1}}{\partial \boldsymbol{s}_t}\Big]^2 \tag{13}$$

The term $\partial \boldsymbol{s}_{t+1}/\partial \boldsymbol{s}_t$ is derived from the Incremental Model's approximation of the state transition dynamics.

The Critic's weights, denoted as ϕ, are updated by computing the gradient of the loss function with respect to the network layers:

$$\nabla_\phi \mathcal{L}_\lambda(t) = \frac{\partial \mathcal{L}_\lambda(t)}{\partial \phi} = \frac{\partial \mathcal{L}_\lambda(t)}{\lambda(\boldsymbol{s}_t)}\frac{\partial \lambda(\boldsymbol{s}_t)}{\partial \phi} = \Big[\lambda(\boldsymbol{s}_t) - \Big(\gamma\hat{\lambda}(\boldsymbol{s}_{t+1}) + \frac{\partial r_{t+1}}{\partial \boldsymbol{s}_{t+1}}\Big)\Big(F_{t-1} + G_{t-1}\frac{\partial \boldsymbol{a}_t}{\partial \boldsymbol{s}_t}\Big)\Big]\frac{\partial \lambda(\boldsymbol{s}_t)}{\partial \phi} \tag{14}$$

In this equation, $\partial \boldsymbol{a}_t/\partial \boldsymbol{s}_t$ is determined through backpropagation within the Actor-network. In contrast, $\partial\lambda(\boldsymbol{s}_t)/\partial\phi$ is computed by backpropagation through the Critic network. With the loss gradient, it is possible to update the weight values with gradient descent and a learning rate η_λ.

C. Soft Actor-Critic (SAC) Algorithm

The SAC [16, 17] algorithm is distinguished by three attributes. Firstly, it is an Actor-Critic RL algorithm, which implies that it learns an approximation for both the policy and the value function. Secondly, it is based on an off-policy formulation, which allows for the reuse of previously collected data, thereby enhancing sample efficiency. Lastly, it incorporates concepts from entropy maximisation into the RL objective, which promotes exploration and enhances stability.

1. SAC's Actor-Network

SAC introduces an entropy regularisation term to the conventional RL objective, resulting in the objective in Eq. (15). The additional entropy term $\mathcal{H}$ transforms the learning objective into a dual maximisation problem, emphasising both the expected return and the entropy of actions. As a result, this encourages the Actor to explore the environment more extensively.

$$\begin{aligned} \pi^* &= \underset{\pi}{\operatorname{argmax}} \sum_t \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi} [r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha \mathcal{H}(\pi(\cdot \mid \boldsymbol{s}_t))] \\ \text{with,} \quad & \mathcal{H}(\pi(\cdot \mid \boldsymbol{s}_t))] = \mathbb{E}_{\boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s})} [-\log(\pi(\boldsymbol{a} \mid \boldsymbol{s}))] \end{aligned} \tag{15}$$

Here, the temperature parameter, denoted by α, balances the trade-off between entropy maximisation and expected return. Specifically, setting the temperature to zero reduces the objective to its conventional form, focusing exclusively on maximising expected return.

The actor in SAC aims to find the optimal policy defined in Eq. (15). The loss in achieving this objective is the negative of the value function. Therefore, it includes the Critic value estimation and the entropy term. This loss is shown in Eq. (16).

$$\mathcal{L}_\pi = -V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi} [\alpha \log \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t) - Q(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{16}$$

Nevertheless, the loss in Eq. (16) can lead to policies that make actions oscillate and change rapidly, especially in complex environments. To overcome this issue, we implement the Conditioning for Action Policy Smoothness (CAPS) [18] regularisation method, which adds two additional terms to the loss function, analogous to the approach adopted by Teirlinck [11].

The first term of the CAPS method is the temporal regularisation loss, which encourages each action to be near the immediate previous action. This temporal loss is defined in Eq. (17). The second term, called spatial regularisation, encourages the actions to be close to a randomly chosen action from the distribution $\bar{\boldsymbol{s}} \sim N(\boldsymbol{s}, 0.05)$. The spatial loss is defined in Eq. (18). It is important to note that temporal and spatial regularization distances are calculated using the L2 normalisation. By combining these CAPS terms with the actor loss, we get the full actor loss function shown in Eq. (19), where λ_T and λ_S are the scaling factors for the temporal and spatial terms, respectively.

$$\mathcal{L}_T = D(\pi(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})) = \|\pi(\boldsymbol{s}_t) - \pi(\boldsymbol{s}_{t+1})\|_2 \tag{17}$$

$$\mathcal{L}_S = D(\pi(\boldsymbol{s}_t, \bar{\boldsymbol{s}})) = \|\pi(\boldsymbol{s}_t) - \pi(\bar{\boldsymbol{s}})\|_2 \tag{18}$$

$$\mathcal{L}_\pi^{\text{CAPS}} = \mathcal{L}_\pi + \lambda_T \mathcal{L}_T + \lambda_S \mathcal{L}_S \tag{19}$$

2. SAC's Critic-Network

The SAC's Critic estimates the action-value function Q with a NN parametrised with weights denoted by ϕ. The network aims to approximate the target value in Eq. (20). This target value is a function of the following state's reward and expected value, scaled by a discount factor γ. Even though the target value relies on the state-value function $V(\boldsymbol{s})$, there is no need to create a parameterisation for this function. This is because the state-value function and the action-value function are related, as demonstrated in Eq. (22). Therefore, the target value can be expressed in terms of the action-value function, as shown in Eq. (21).

$$y(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim p}[V(\boldsymbol{s}_{t+1})] \tag{20}$$

$$= r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \min_{i=1,2} [Q_{\phi_{\text{targ},i}}(\boldsymbol{s}_{t+1}, \tilde{\boldsymbol{a}}_{t+1}) - \alpha \log \pi_\theta(\tilde{\boldsymbol{a}}_{t+1} \mid \boldsymbol{s}_{t+1})] \tag{21}$$

$$V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi}[Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)] \tag{22}$$

Note that Eq. (21) uses a Clipped Double Q-learning strategy to reduce overestimation bias. Therefore, it learns two separate Critic networks and takes the smaller value between them. Besides that, the equation uses Target Networks, which are updated less frequently than the actual Critic-Network, providing more stable learning. The $\tilde{\boldsymbol{a}}_{t+1}$ term indicates that the action is sampled from the actor instead of using an action from the replay buffer; therefore, $\tilde{\boldsymbol{a}}_{t+1} = \pi_\theta(\boldsymbol{s}_{t+1})$.

The Critic Network's learning process involves minimising its prediction's TD error. As such, the loss of the Critic Network, given in Eq. (23), is the squared difference between the estimated action-value function $Q(\boldsymbol{s}, \boldsymbol{a})$ and the target value y.

$$\mathcal{L}_Q = \sum_{i=1,2} [Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - y(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})]^2 \tag{23}$$

D. Distributional Soft Actor-Critic (DSAC) Algorithm

The DSAC [19, 20] algorithm is an extension of the standard SAC.The key differentiation between DSAC and SAC lies in the learning approach adopted by the Critic Network: DSAC learns the distribution of returns rather than their expected values. Therefore, the Critic in DSAC approximates a function that describes the distribution of returns.

1. DSAC's Actor-Network

In DSAC, the Actor-Network denoted as π_θ, retains the same parameterised structure used in SAC. Therefore, the SAC's Actor loss function (as shown in Eq. (16)) remains applicable to DSAC. However, since the Critic in DSAC estimates the distribution of returns $Z(\boldsymbol{s}, \boldsymbol{a})$, a transformation function is necessary to convert this distribution into a real-valued action-value function, Q. This transformation is facilitated through a risk measure function Ψ, represented as follows:

$$Q(\boldsymbol{s}, \boldsymbol{a}) = \Psi[Z(\boldsymbol{s}, \boldsymbol{a})] \tag{24}$$

One possible risk measure function could be an expectation operator, $\mathbb{E}[\cdot]$, which effectively converts the Actor loss into a conventional SAC format. Within the context of DSAC, the expectation risk measure is known as risk-neutral. This research uses the Wang risk measure [21]. Furthermore, the CAPS regularisation technique is added to the loss function the same way as done for the SAC

2. DSAC's Distributional Critic-Network

In the DSAC algorithm, the Critic Network aims to approximate the distribution of returns instead of the expectation of returns in the traditional SAC.Specifically, it approximates the quantile function $Z_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t; \tau_i)$, where τ_i denotes the i-th quantile, and ϕ represents the parameters of the NN. The quantile function, which is

the inverse of the Cumulative Distribution Function (CDF), indicates the value corresponding to a specific distribution quantile. For example, if $Z(10\%) = 5$, this indicates a 10% or lower probability of sampling a value of 5 or less from the given distribution.

To approximate the return distribution, the DSAC's Distributional Critic Network has multiple output neurons, each representing the quantile function at different quantiles. When combined, those quantile functions describe the distribution of return. The network input is the environment states, the Actor action, and the quantiles' values.

The error of the Critic Network is theTD error between two quantile fractions, as defined in Eq. (25). A quantile fraction, denoted as $\hat{\tau}$, is the mean value between two subsequent quantiles. Therefore, $\hat{\tau}_i = (\tau_i + \tau_{i+1})/2$. The TD error quantifies how the shape of the action-value distribution has changed after the action.

$$\delta_{ij}^t = r_t + \gamma \left[Z_{\bar{\phi}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}; \hat{\tau}_i) - \log \pi_{\bar{\theta}}(\boldsymbol{a}_{t+1}, \boldsymbol{s}_{t+1}) \right] - Z_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t; \hat{\tau}_j) \tag{25}$$

The loss of the Critic Network in DSAC, denoted as $\mathcal{L}_Z$, is computed as the Huber loss $\mathcal{L}_k^H$ [22] of the TD error across all quantile pairs, as defined in Eq. (26). The Huber loss behaves as a mean squared error when the error value is smaller than the threshold k and behaves linearly beyond this threshold. This loss is chosen because it is less sensitive to outliers than pure mean squared error. The aggregate loss is calculated as a weighted sum of the Huber loss for each quantile pair, where the weight is the difference in the value of the quantile pair.

$$\mathcal{L}_Z = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (\tau_{i+1} - \tau_i) \mathcal{L}_k^H(\delta_{ij}^t) \tag{26}$$

III. Methodology

This section describes the methodology for developing and evaluating the RL-based attitude controllers.

A. Reinforcement Learning for Attitude Tracking Task

This study's Reinforcement Learning environment uses the DASMAT high-fidelity simulation of the Cessna 550 Citation II aircraft. The aircraft simulation has as input the states in Eq. (27) and as actions the vector in Eq. (28). Note that the aircraft states vector $\boldsymbol{x}$ is not the same as the environment observation vector $\boldsymbol{s}$, as the latter is tailored for each RL algorithm.

$$\boldsymbol{x} = \begin{bmatrix} p & q & r & V & \alpha & \beta & \theta & \phi & \psi & h \end{bmatrix}^T \tag{27}$$

$$\boldsymbol{u} = \begin{bmatrix} \delta_e & \delta_a & \delta_r \end{bmatrix}^T \tag{28}$$

In the state vector in Eq. (27), p is the roll rate, q the pitch rate, r the yaw rate, V the airspeed, α the angle of attack. β the sideslip angle, θ the pitch angle, ϕ the roll angle, ψ the heading angle, and h the altitude. In the action vector in Eq. (28), δ_e is the elvator deflection, δ_a the aileron deflection, and δ_r the rudder deflection.

The aircraft model and controller are discretised with a sampling frequency of 100 Hz. The model represents the aircraft in a clean configuration and is trimmed at 2000 m altitude and 90 m s^{-1} airspeed. Furthermore, the DASMAT model includes a built-in yaw damper and auto-throttle. Consequently, throttle control is managed by the model, while the RL agents manage the control surfaces in Eq. (28).

The main objective for the agents within the Citation environment is to control the aircraft such that its attitude match a reference signal. The reference attitude vector $\boldsymbol{x}^{\text{ref}}$, which represents the desired aircraft state, is

defined in Eq. (29). The actual aircraft attitude, $\boldsymbol{x}^{\text{att}}$ is extracted by filtering the state vector $\boldsymbol{x}$ as shown in Eq. (30).

$$\boldsymbol{x}^{\text{ref}} = \left[\theta^r, \phi^r, \beta^r\right]^T \tag{29}$$

$$\boldsymbol{x}^{\text{att}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} \tag{30}$$

The tracking error is computed as the difference between the actual aircraft attitude $\boldsymbol{x}^{\text{att}}$ and the reference signal $\boldsymbol{x}^{\text{ref}}$. A weighted sum of the errors is employed to aggregate the tracking error across all attitude angles into a single metric, utilising the weights specified in Eq. (31). These weights serve the purpose of scaling each state, thereby ensuring they have comparable magnitudes. The values of the scaling factors employed in this study are retrieved from the work by Teirlinck [11]. Consequently, the attitude tracking error is defined in Eq. (32).

$$\boldsymbol{c}^{\text{att}} = \frac{180}{\pi}\left[\frac{1}{30} \quad \frac{1}{30} \quad \frac{1}{7.5}\right]^T \tag{31}$$

$$\boldsymbol{e}^{\text{att}} = (\boldsymbol{x}^{\text{att}} - \boldsymbol{x}^{\text{ref}}) \times \boldsymbol{c}^{\text{att}} \tag{32}$$

The offline (IDHP) and online (DSAC and SAC) agents are built with different reward functions and observation vectors. For the IDHP, the observation vector, defined in Equation Eq. (33), includes some of the aircraft states and the squared tracking error. The reward is calculated as the negative of the squared tracking error, as in Eq. (35). For the DSAC and SAC algorithms, use the observation vector in Eq. (34), and the reward function in Eq. (36).

$$\boldsymbol{s}_{t+1}^{\text{IDHP}} = \left[p \quad q \quad r \quad \alpha \quad \theta \quad \phi \quad \beta \quad \boldsymbol{e}_t^{\text{att}}\right]^T \tag{33}$$

$$\boldsymbol{s}_{t+1}^{\text{DSAC}} = \left[p \quad q \quad r \quad \boldsymbol{e}_t^{\text{att}}\right]^T \tag{34}$$

$$r_{t+1}^{\text{IDHP}} = -\boldsymbol{e}_t^{\text{att}} \times \boldsymbol{e}_t^{\text{att}} \tag{35}$$

$$r_{t+1}^{\text{DSAC}} = -\|\text{clip}[\boldsymbol{e}_t^{\text{att}}, -\vec{\mathbf{1}}, \vec{\mathbf{1}}]\| \tag{36}$$

B. Design of the Hybrid RL-Based Flight Controller

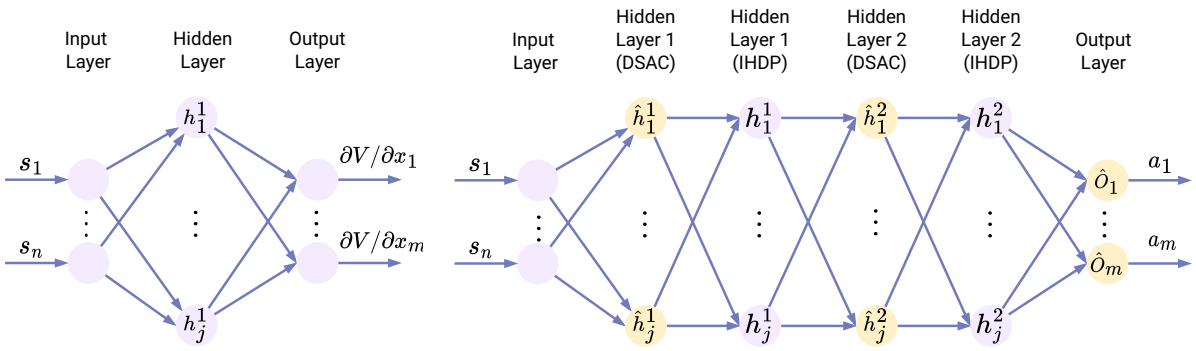
The controller developed in this study comprises a hybrid RL agent that combines the offline DSAC and online IDHP algorithms. This section describes the working principles underlying this controller.

1. Motivation For a Hybrid Controller

An autonomous flight controller must be robust and adaptable to handle unexpected conditions, such as system failures, during flight. Offline RL-based controllers are robust but inefficient in adapting to changing conditions. Online RL controllers are good at adapting in real-time but can be unreliable, and there are safety concerns as they learn during the flight. This study introduces a Hybrid controller that uses IDHP and DSAC to bring together their strengths.

The research by Teirlinck [11] demonstrated the advantages of a hybrid controller that combines IDHP with SAC. Recent studies [12] suggest that DSAC is a safer option than SAC without sacrificing performance. DSAC, being risk-sensitive, tends to avoid states with high uncertainties, as Seres [12] demonstrated a DSAC controller that avoided near-stall conditions.

DSAC also exhibits more stable learning performance. It is more consistent and reliable in learning an optimal policy with a high return and low variance. This research aims to evaluate the performance of a hybrid controller that uses DSAC as an offline algorithm, comparing it with standalone DSAC and SAC, as well as the hybrid that uses SAC.



(a) Hybrid's Critic-Network. **(b) Hybrid's Actor-Network.**

Figure 1. Topology of DSAC-hybrid's Critic and Actor-Network. $\hat{h}$ are the hidden layers derived from the DSAC agent, which are frozen during online learning. h are the layers from the IDHP agent, which can adapt during online learning.

2. Naming Conventions for Hybrid Controllers

Throughout this paper, the SAC algorithm combined with IDHP is referred to as SAC-hybrid, while its standalone version is called SAC-only. Likewise, when DSAC is combined with IDHP, it is called IDHP-hybrid, and DSAC-only when it operates alone.

3. Hybrid Controller architecture

The hybrid controller's architecture is based on the one developed by Teirlinck [11]. In this setup, the Actor-Network of the IDHP algorithm is modified to include neurons from the offline algorithm. That means layers from both offline agent and IDHP are intertwined in a single network. The layers derived from the offline algorithm retain the weight values learned during offline training, while IDHP layers are set to the identity matrix at the beginning. This ensures that at time zero, the Actor-network functions as the network of the offline agent.

The Hybrid Actor Network, which combines layers from IDHP and DSAC, is shown in Fig. 1b. During the online learning phase, the network only updates the IDHP layers, while the layers associated with the offline algorithm remain unchanged. Because only the layers derived from IDHP change, the algorithm update logic discussed in Section II.B does not alter.

On the other hand, the Critic Network of the Hybrid controller does not combine layers from the offline and online algorithms. This network is the same as the one from IDHP, as shown in Fig. 1a. This design choice is due to the IDHP's Critic output, which is significantly different from the one in SAC and DSAC. Consequently, reusing the offline Critic would change the logic of the IDHP algorithm.

C. Training Process for the Hybrid Controller

To create the Hybrid agents, the SAC or DSAC algorithms are first trained offline. For this study, both algorithms were trained over three random seeds with f 1 M steps each. The training episode consisted of 2 K steps(or 20 s) where the agents should track a smoothed step function with variable amplitude. Figure 2 shows the averaged learning curves of the two algorithms across the three random seeds. All agents converged to similar performance levels, with SAC achieving convergence slightly sooner than DSAC.

During offline training, the agents update their networks to improve tracking performance. After each episode,

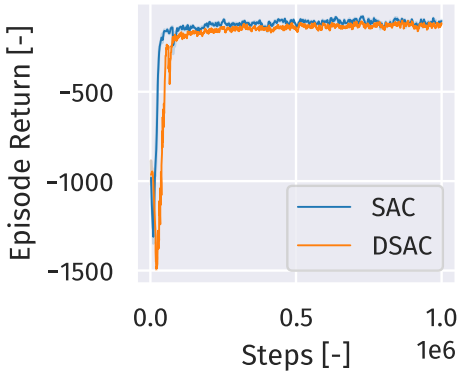


Figure 2. Learning performance curves for the SAC and DSAC algorithms. Each curve represents the average performance of three independent runs. The training was conducted over a total of 10 M **learning steps, partitioned into episodes of** 2 K **steps each.**

the policy is evaluated. The optimal policy at the end of training is the one that showed the best performance during evaluation. The metric used for the evaluation is the normalised Mean Absolute Error (nMAE). This metric normalises the tracking error by dividing it by the range of each tracked state, which enables the comparison of scores across different tracking tasks. The policies learned during this process are shown in Table 1 alongside their best nMAE. It is important to note that, the nMAE quantifies the normalised error; therefore, a lower value indicates superior performance.

Table 1. Evaluation performance of SAC and DSAC agents during training. This table provides the best nMAE score the agents achieved during evaluation after each episode in the training process. The training was conducted over a total of 10 M **learning steps, partitioned into episodes of** 2 K **steps each.**

Id	Agent	nMAE
SAC-1	SAC	8.36 %
SAC-2	SAC	10.02 %
SAC-3	SAC	8.03 %

Id	Agent	nMAE
DSAC-1	DSAC	7.57 %
DSAC-2	DSAC	11.80 %
DSAC-3	DSAC	7.03 %

IV. Results & Discussion

In this section, we present the results of evaluating the performance of the proposed hybrid controller. These results clarify the controller's robustness to changes in reference signals and under conditions of sensor noise and bias, as well as its fault tolerance when subjected to reductions in control surface effectiveness.

A. Analysis of Robustness

This subsection is dedicated to examining the robustness of the hybrid controller under different reference signals and in the presence of sensor noise and bias.

1. Robustness to Different Reference Signals

An important goal for the RL-based controller is to ensure that the policy it learns is general enough to track reference signals different from those used during training. To better understand how the hybrid and

non-hybrid controllers behave under different conditions, we compare their performance in response to three distinct reference signals, shown in Fig. 3.

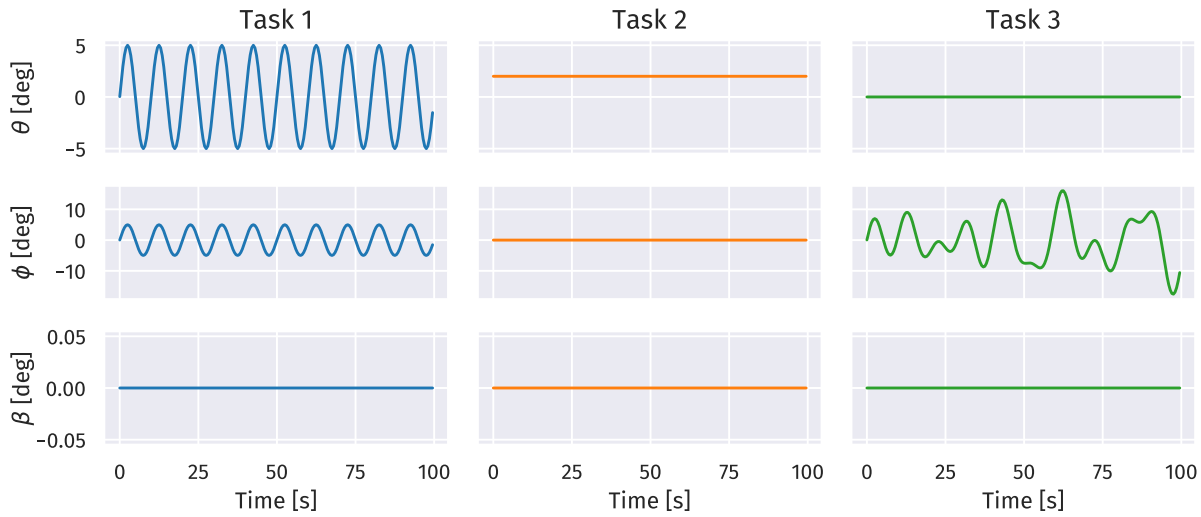


Figure 3. The three tracking tasks used in the robustness experiment, each providing a different attitude reference.

The first reference signal requires tracking a sinusoidal signal with a fixed amplitude and period for the pitch (θ) and roll (ϕ) angles while maintaining the sideslip angle (β) at zero. The second task involves keeping the three attitude angles constant throughout the flight. The third task involves holding both the pitch and sideslip angle at zero while the roll angle follows a pseudo-random sinusoidal trajectory.

The robustness of the agents across different tracking tasks is summarised in Table 2. Each agent was evaluated in a task for five different random seeds using each of the three offline-learned policies. Therefore, a total of 15 runs for each agent. The results indicate that, on average, both SAC-hybrid and DSAC-hybrid improved the performance of their respective offline-only versions.

Furthermore, the table also shows how the nature of the tracking task can influence the algorithms' performance. When the evaluation task is significantly different from the trained task, the agent may not be able to achieve low tracking errors. This is particularly evident in Task 1, where the hybrid agents improve performance compared to the offline agents but are less substantial than in the other tasks.

Table 2. Evaluation of hybrid and non-hybrid agents subjected to varying tracking tasks. The table presents the mean and variance of their tracking nMAE.

	SAC-only	**SAC-hybrid**	**DSAC-only**	**DSAC-hybrid**
Task 1	20.1 ± 1.8%	14.9 ± 1.5%	22.5 ± 7.0%	11.3 ± 0.6%
Task 2	11.1 ± 3.6%	4.5 ± 1.8%	13.4 ± 8.8%	2.9 ± 0.9%
Task 3	16.6 ± 2.2%	4.1 ± 0.8%	18.7 ± 9.8%	3.0 ± 0.7%

It is noteworthy from the results that, between the offline agents, the SAC-only outperforms the DSAC-only in all tasks, with lower variance. However, in hybrid form, the DSAC-hybrid manages to surpass the SAC-hybrid performance. The hybrid configuration enhances the offline policies, making them more robust. The key

takeaway is that the hybrid setup yields more consistent tracking performance results, irrespective of the initial performance of the offline model.

Across all the agents, the DSAC-hybrid consistently achieved the lowest average nMAE in all tasks while also maintaining low variance. Figure 4 shows the extent of improvement in nMAE performance that the hybrid agent brings compared to its offline-only version. The figure reveals that, on average, the DSAC-hybrid enhances performance more effectively than the SAC-hybrid. However, this improvement comes with higher variance. Additionally, during Task 2, the DSAC-hybrid had one run using the DSAC-2 offline policy (as referenced in Table 1), which resulted in reduced performance. This occurred because the DSAC-2 offline policy had the lowest nMAE, highlighting the importance of carefully selecting the initial policy for the hybrid, favouring those with low nMAE.

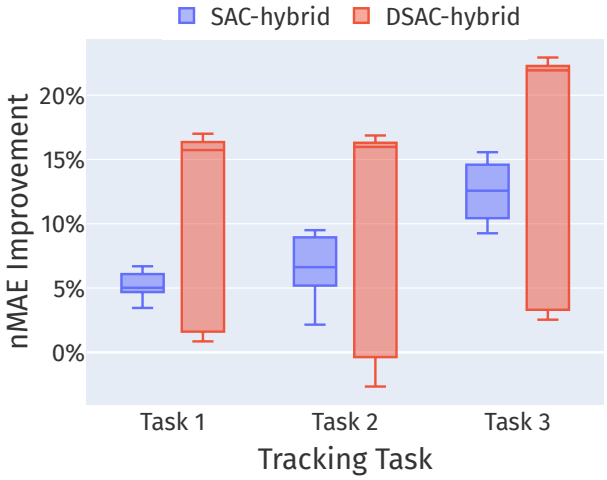


Figure 4. Box plot showing the extent of improvement innMAE that each hybrid algorithm (SAC-hybrid and DSAC-hybrid) brings compared to their respective offline algorithms across different tasks during the evaluation phase.

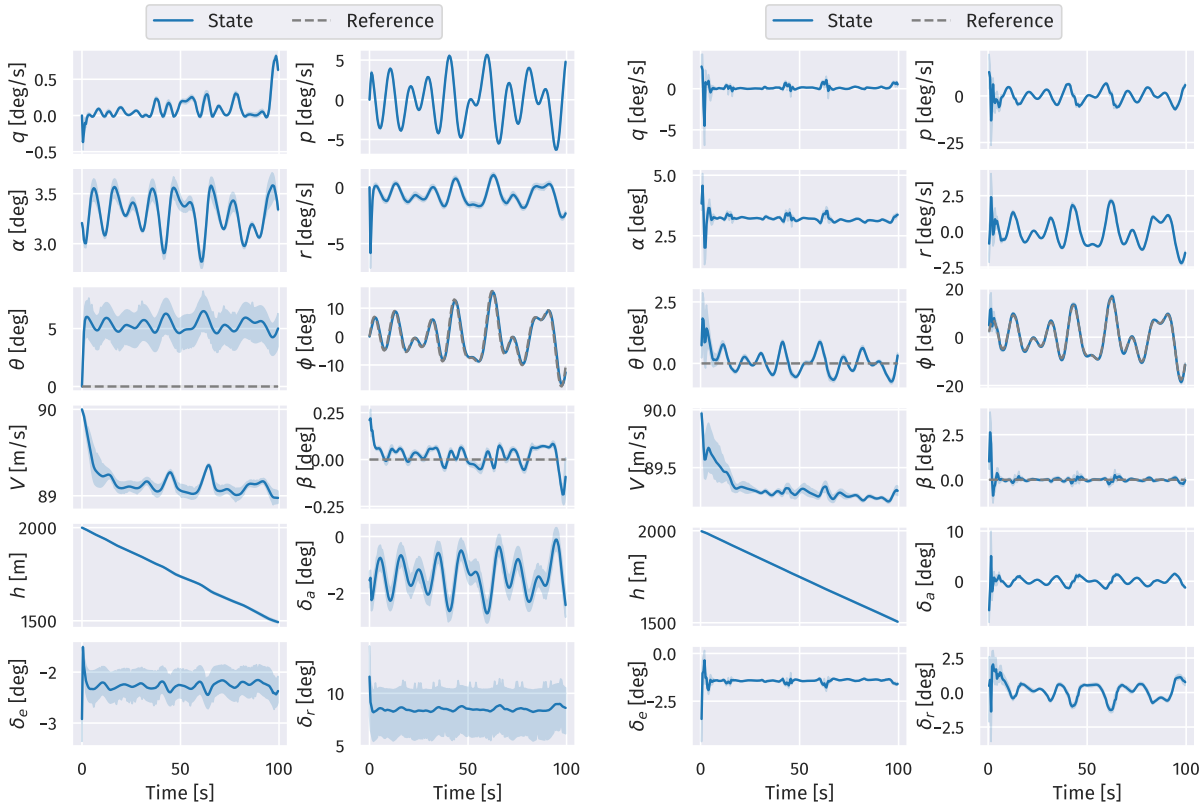
2. Robustness under Sensor Noise and Bias

The final robustness experiment involves testing the agents in a more realistic setting by introducing noise and bias into the aircraft's sensors. In this test, the hybrid agents aim to track a reference signal, while noise and bias are added to the states p, q, and r. The noise and bias are sampled from the normal distribution $\mathcal{N}(\mu = 3 \cdot 10^{-5}, \sigma = 4 \cdot 10^{-7})$, which are based on expected values for the Citation aircraft as found in Grondman *et al.* [23]. Only these three states are modified because they are the only ones shared between the observation functions of the offline algorithms and IDHP, as shown in Eq. (34) and Eq. (33), respectively.

Each offline learned policy was evaluated in a hybrid configuration under noise and bias conditions using five different random seeds for this experiment. This amounts to 15 runs for SAC and 15 runs for DSAC. The results from this experiment are summarised in Table 3. Despite the presence of noise and bias, both hybrid controllers managed to enhance the performance compared to the offline-only versions. The DSAC algorithm again showed a more substantial improvement. The graphs showing the average tracking performance of the agents during the task with noise and bias can be seen in Fig. 5a for DSAC-only, and in Fig. 5b for DSAC-hybrid.

Table 3. Evaluation of hybrid and non-hybrid agents subjected to sensor noise and bias. The table presents the mean and variance of their tracking nMAE.

	Non-hybrid	Hybrid	Improvement
SAC	16.6 ± 2.2%	4.0 ± 0.9%	12.6 ± 2.7%
DSAC	18.7 ± 9.8%	2.8 ± 0.5%	15.9 ± 9.5%



(a) DSAC-only evaluation performance. **(b) DSAC-hybrid evaluation performance.**

Figure 5. Aircraft states during the evaluation of DSAC agents in Task 3, including sensor noise and bias.

B. Assessment of Fault Tolerance

The fault-tolerance experiment examines the performance of the hybrid controllers under reduction in the effectiveness of its control surfaces. First we consider a the elevator effectiveness by 70 %. Then, a reduction in the aileron effectiveness by 90 %. In each condition, the agents are evaluated over 5 random seeds, each tracking Task 3 from Fig. 3 while the fault stats at t =10 s.

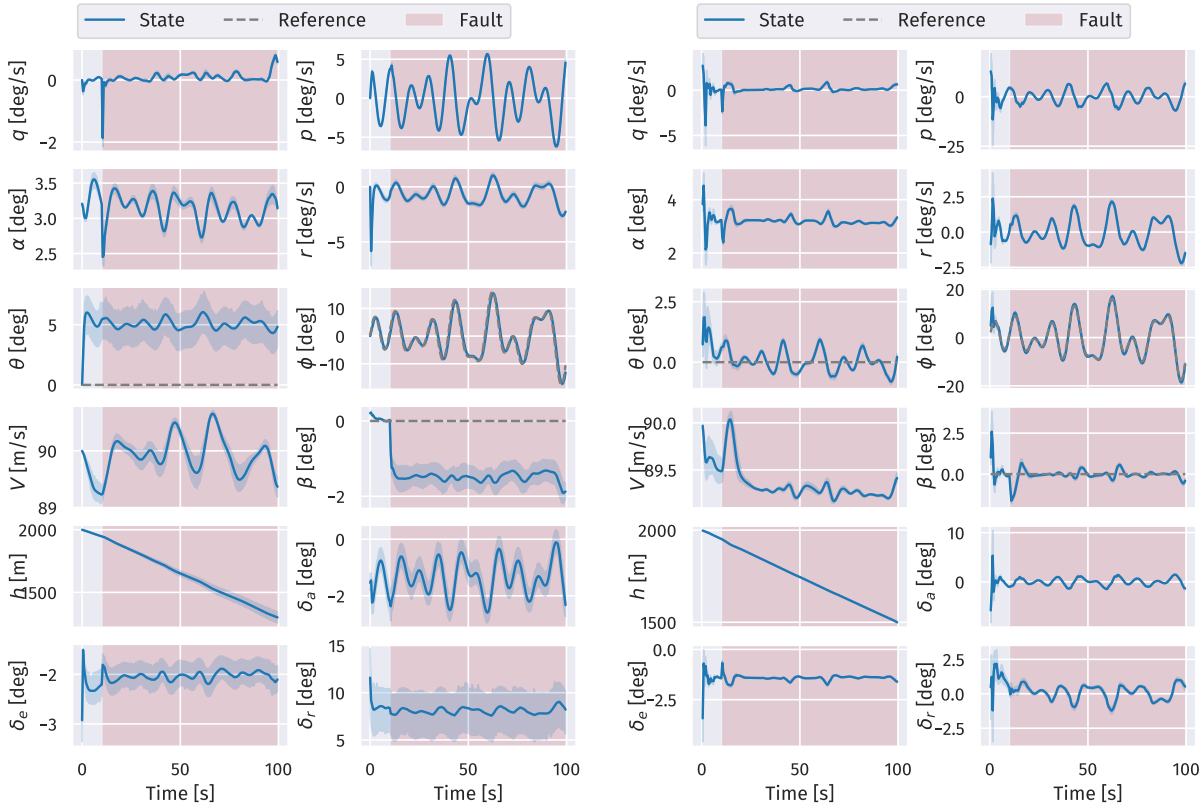
1. Performance under Reduced Elevator Effectiveness

The evaluation of the hybrid and non-hybrid agents with reduction in the elevator effectiveness by 70 % at time 10 s is summarised in Table 4. Those results shows the hybrid compensate to the failure and improve the performance of the offline-only agents. The DSAC-hybrid achieves the lowest nMAE.

Table 4. Evaluation of hybrid and non-hybrid agents subjected to a reduction of elevator effectivess by 70 %. The table presents the mean and variance of their tracking nMAE.

	Non-hybrid	Hybrid	Improvement
SAC	18.4 ± 3.1%	4.1 ± 0.9%	14.3 ± 3.7%
DSAC	22.4 ± 11.8%	3.2 ± 0.6%	19.2 ± 11.4%

The average tracking performance of the DSAC-hybrid and DSAC-only agents is shown in Fig. 6. While DSAC-only does adequate tracking of the roll angle, it struggles to hold the pitch angle and sideslip angle at zero. For the sideslip angle it is visible that the deterioration start at the start of the fault. The DSAC-hybrid, on the other hand, has a worst start on the tracking of the roll angle, but it picks up the tracking of the signal within the first seconds. Additionaly, it is also visible the impact of the fault on pitch and sideslip angle at start of fault. However, the hybrid is able to account for the fault and do a better regulate those angles.



(a) DSAC-only evaluation performance.

(b) DSAC-hybrid evaluation performance.

Figure 6. Aircraft states during the evaluation of DSAC agents in Task 3, including a reduction in elevator effectiveness by 70 % at the 10 s mark.

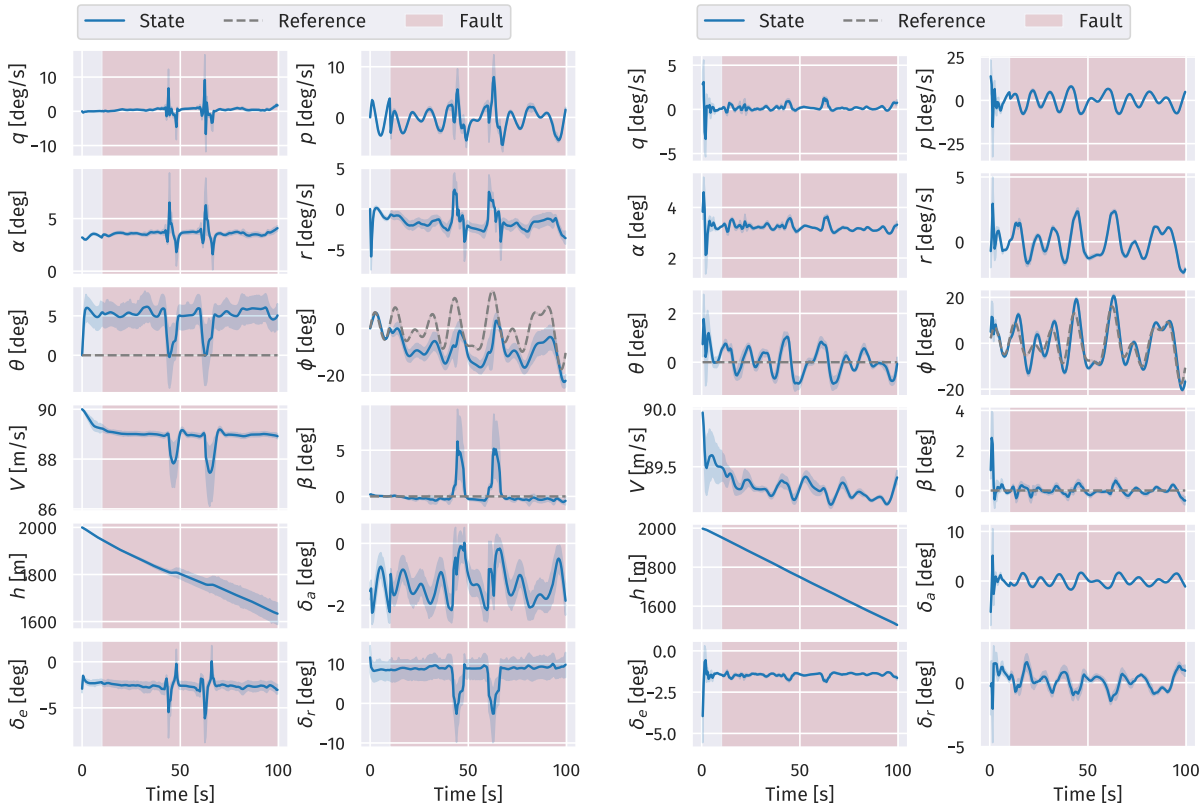
2. Performance under Reduced Aileron Effectiveness

The evaluation of the hybrid and non-hybrid agents with a reduction in the aileron effectiveness by 90 % at time 10 s is summarised in Table 5. The results show that DSAC-hybrid is the controller that achieves the lowest nMAE, followed by SAC-hybrid. This is the experiment where the hybrid architecture showed the greatest nMAE improvement concerning the non-hybrid agents.

Table 5. Evaluation of hybrid and non-hybrid agents subjected to a reduction of aileron effectiveness by 90 %. The table presents the mean and variance of their tracking nMAE.

	Non-hybrid	Hybrid	Improvement
SAC	28.5 ± 8.5%	9.1 ± 2.5%	19.4 ± 10.7%
DSAC	28.7 ± 17.0%	6.1 ± 0.7%	22.6 ± 16.8%

The results of the average tracking performance of the DSAC-hybrid in the evaluation task with the aileron fault is shown in Fig. 7. With the deteriorated aileron, the DSAC-only strugles to track the roll angle, as shown in Fig. 7a. Note how different the tracking becomes when compared to Fig. 7b. The DSAC-hybrid, however, is able to track the roll angle considerably well while still managing to hold the other attitude angles.



(a) DSAC-only evaluation performance. **(b) DSAC-hybrid evaluation performance.**

Figure 7. Aircraft states during the evaluation of DSAC agents in Task 3, including a reduction in aileron effectiveness by 90 % at the 10 s mark.

C. Reliability Analysis of the Controllers

One of the challenges in RL is evaluating the reliability of results. Because the training time is computationally intensive, the number of experiments that can be practically conducted is limited. Consequently, it becomes difficult to assert that the results are reliable and not only an outlier due to randomness in the problem. To address this issue, the study performed a reliability analysis to evaluate the likelihood of obtaining similar results if the same experiments were performed with different random seeds.

The methodology employed for this reliability analysis uses the framework set by Agarwal *et al.* [24]. A central component in the analysis is the stratified bootstrap confidence intervals, which is an aggregated score built by random sampling, with replacement, a set of nMAE values from each experiment. The sample size is proportional to the number of evaluations in that experiment.

1. Performance Profile

The performance profile graph shows the distribution of nMAE across all runs. It uses the data derived from the stratified bootstrap confidence intervals to build a cumulative distribution of the model's scores. The results for the agents are shown in Fig. 8a. The y values in the curve represent the fraction of runs that lead to an nMAE lower than the threshold in the x axis. Therefore, the higher the curve, the better.

In the performance profile graph, the hybrid agents exhibit comparable performance relative to one another, as do the non-hybrid agents. The curve for the DSAC-hybrid is above the others, signifying that this agent demonstrates statistical dominance. This implies that, for this particular agent, the runs are more likely to yield lower nMAE. Among the non-hybrid algorithms, SAC-only exhibits slightly better performance compared to DSAC-only.

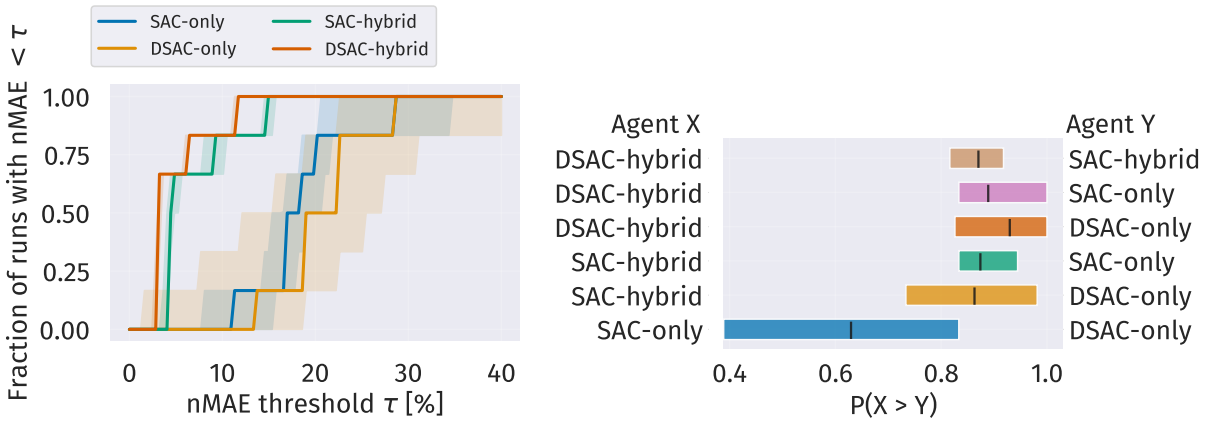
2. Probability of Improvement

A subsequent analysis is the probability of improvement, which quantifies the likelihood that a given algorithm will yield performance improvement compared to others. In other words, it calculates the probability that the performance of algorithm X exceeds that of algorithm Y. The calculation involves a pairwise comparison of the nMAE between the two algorithms, isolating instances where the performance of X surpasses that of Y. The probability of improvement is then computed as the ratio of instances where X performed better to the total number of pairwise comparisons made.

The outcomes of the probability of improvement analysis for the algorithms are shown in Fig. 8b. The data indicate that both hybrid algorithms have greater than 80 % probability of increasing the performance of the offline-only agents. Furthermore, the DSAC-hybrid also has a high probability of improving the SAC-hybrid. Among the offline-only agents, SAC has a slightly above 60 % chance of improving DSAC. However, this is together with high variance, confirming that the performance of those offline algorithms is similar.

V. Conclusion

Robustness and fault tolerance are critical for autonomous flight control systems. Reinforcement Learning (RL) flight controllers emerge as a promising method to achieve these characteristics. While offline RL algorithms are good in providing robustness, they are constrained by a limited capacity to adapt during flight. In contrast, online RL algorithms are highly adaptive but may compromise reliability due to uncertain convergence properties and the inherent risks associated with learning to control during flight. In light of these observations, this research investigated the potential of hybrid controllers, combining the strengths of offline Distributional Soft Actor-Critic (DSAC) and online Incremental Dual Heuristic Programming (IDHP) algorithms, referred to as DSAC-hybrid.



(a) Performance profile of RL agents.

(b) Probability of improvement of RL agents.

Figure 8. Reliability analysis of the hybrid and non-hybrid controllers. Results from studying the stratified bootstrap confidence intervals of the aggregated nMAE scores across the experiments.

The DSAC-hybrid controller demonstrated enhanced robustness compared to a hybrid using Soft Actor-Critic (SAC) as the offline algorithm (denoted to as SAC-hybrid). Besides that, the DSAC-hybrid outperformed controllers exclusively utilising DSAC and SAC. The robustness comparison included evaluating the controllers' ability to track reference signals that deviate from those encountered during training and in scenarios with noisy and biased observation vectors.

Furthermore, the DSAC-hybrid controller showed superior fault-tolerance capabilities. Specifically, in an assessment involving a scenario where elevator effectiveness was reduced by 70 %, the DSAC-hybrid maintained its capacity to track a reference signal. At the same time, the strictly offline agents encountered difficulties. The same was observed with a 90 % reduction in aileron effectiveness. The SAC-hybrid agent also coped with the fault effectively but with lower performance than the DSAC-hybrid.

Overall, the hybrid approach proved to effectively combine offline algorithms' robustness with an online algorithm's real-time adaptability. The DSAC-hybrid emerged as the highest performance model, with SAC-hybrid also demonstrating superiority over the strictly offline algorithms. A reliability analysis provided additional validation to these conclusions, indicating that both hybrid algorithms have a greater than 80 % likelihood of outperforming offline algorithms regarding tracking performance.

This research is a step towards developing safe and autonomous RL-based controllers. By combining offline and online learning algorithms, the hybrid approach opens up new possibilities for flight control. In particular, the DSAC-hybrid controller sets an example of robustness and adaptiveness in attitude tracking. Therefore, these findings have significant implications for future research in autonomous flight control.

References

[1] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields," *IEEE Access*, vol. 8, pp. 209 320–209 344, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3038605.

[2] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 26, 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236.

[3] OpenAI *et al.* "Learning Dexterous In-Hand Manipulation." arXiv: 1808.00177 [cs, stat]. (Jan. 18, 2019), [Online]. Available: http://arxiv.org/abs/1808.00177 (visited on 04/12/2023), preprint.

[4] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. "Learning to Walk via Deep Reinforcement Learning." arXiv: 1812.11103 [cs, stat]. (Jun. 19, 2019), [Online]. Available: http://arxiv.org/abs/1812.11103 (visited on 04/12/2023), preprint.

[5] J. H. Lee and E.-J. Van Kampen, "Online reinforcement learning for fixed-wing aircraft longitudinal control," in *AIAA Scitech 2021 Forum*, VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Jan. 11, 2021, ISBN: 978-1-62410-609-5. DOI: 10.2514/6.2021-0392.

[6] A. Y. Ng *et al.*, "Autonomous Inverted Helicopter Flight via Reinforcement Learning," in *Experimental Robotics IX*, M. H. Ang and O. Khatib, Eds., red. by B. Siciliano, O. Khatib, and F. Groen, vol. 21, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 363–372, ISBN: 978-3-540-28816-9 978-3-540-33014-1. DOI: 10.1007/11552246_35.

[7] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement Learning for UAV Attitude Control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, Apr. 30, 2019, ISSN: 2378-962X, 2378-9638. DOI: 10.1145/3301273.

[8] P. C. Gregory, "Proceedings of the self adaptive flight control systems symposium," Aeronautical Systems Div Wright-Patterson AFB OH Flight Control Lab, United States, Technical Rerport AD0209389, 1959.

[9] IATA, *Loss of Control In-Flight Accident Analysis Report*. Montreal: International Air Transport Association, 2019, ISBN: 978-92-9264-002-6.

[10] K. Dally and E.-J. Van Kampen, "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control," in *AIAA SCITECH 2022 Forum*, San Diego, CA & Virtual: American Institute of Aeronautics and Astronautics, Jan. 3, 2022, ISBN: 978-1-62410-631-6. DOI: 10.2514/6.2022-2078.

[11] C. Teirlinck, "Reinforcement Learning for Flight Control Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance," TU Delft, Delft, 2022.

[12] P. Seres, "Distributional Reinforcement Learning for Flight Control," Delft University of Technology, Delft, 2022.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, ISBN: 0-262-03924-9.

[14] P. J. Werbos, "A menu of designs for reinforcement learning over time," in *Neural Networks for Control*, ser. Neural Network Modeling and Connectionism, W. T. Miller, R. S. Sutton, P. J. Werbos, and N. S. F. (U.S.), Eds., Cambridge, Mass: MIT Press, 1990, pp. 67–97, ISBN: 978-0-262-13261-9.

[15] Y. Zhou, E.-J. van Kampen, and Q. P. Chu, "Incremental model based online dual heuristic programming for nonlinear adaptive control," *Control Engineering Practice*, vol. 73, pp. 13–25, Apr. 2018, ISSN: 09670661. DOI: 10.1016/j.conengprac.2017.12.011.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," version 2, 2018. DOI: 10.48550/ARXIV.1801.01290.

[17] T. Haarnoja *et al.*, "Soft Actor-Critic Algorithms and Applications," version 2, 2018. DOI: 10.48550/ARXIV.1812.05905.

[18] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko, "Regularizing Action Policies for Smooth Control with Reinforcement Learning," version 2, 2020. DOI: 10.48550/ARXIV.2012.06644.

[19] X. Ma, L. Xia, Z. Zhou, J. Yang, and Q. Zhao. "DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning." arXiv: 2004.14547 [cs]. (Jun. 10, 2020), (visited on 10/21/2022), preprint.

[20] J. Duan, Y. Guan, S. E. Li, Y. Ren, and B. Cheng, "Distributional Soft Actor-Critic: Off-Policy Reinforcement Learning for Addressing Value Estimation Errors," version 3, 2020. DOI: 10.48550/ARXIV.2001.02811.

[21] S. S. Wang, "A Class of Distortion Operators for Pricing Financial and Insurance Risks," *The Journal of Risk and Insurance*, vol. 67, no. 1, p. 15, Mar. 2000, ISSN: 00224367. DOI: 10.2307/253675. JSTOR: 253675.

[22] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, Mar. 1964, ISSN: 0003-4851. DOI: 10.1214/aoms/1177703732.

[23] F. Grondman, G. Looye, R. O. Kuchar, Q. P. Chu, and E.-J. Van Kampen, "Design and Flight Testing of Incremental Nonlinear Dynamic Inversion-based Control Laws for a Passenger Aircraft," in *2018 AIAA Guidance, Navigation, and Control Conference*, Kissimmee, Florida: American Institute of Aeronautics and Astronautics, Jan. 8, 2018, ISBN: 978-1-62410-526-5. DOI: 10.2514/6.2018-0385.

[24] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. "Deep Reinforcement Learning at the Edge of the Statistical Precipice." arXiv: 2108.13264. (Jan. 5, 2022), [Online]. Available: http://arxiv.org/abs/2108.13264 (visited on 04/28/2023), preprint.

Appendix

A. DSAC and IDHP Hyperparameters

This section provides the hyperparameters employed during the training process of the Reinforcement Learning algorithms used in this research. The DSAC hyperparameters are shown in Table 6a and IDHP ones in Table 6b.

(a) Hyperparameters for DSAC algorithm, adapted from Seres [12] and Duan *et al.* [20]

Hyperparameter	Symbol	Value
Learning rate	η	$4.4 \cdot 10^{-4}$
Discount Factor	γ	0.99
Hidden Neurons		64×64
Replay buffer size	$\lvert\mathcal{D}\rvert$	$1 \cdot 10^{6}$
Batch Size	$\lvert\mathcal{B}\rvert$	256
Polyak Step		0.995
Optimiser		Adam
Network Activation		ReLu
CAPS Scaling	λ_T, λ_S	400
Nr. of Quantiles		8
Huber Threshold	k	1
Risk Measure	Ψ	Wang

(b) Hyperparameters for IHDP algorithm, adapted from Teirlinck [11] and Zhou *et al.* [15]

Hyperparameter	Symbol	Value
Actor Learning Rate	η_A	0.1
Critic Learning Rate	η_C	0.001
Discount Factor	γ	0.7
RLS Discount Factor	γ_{RLS}	0.7
Hidden Neurons		10
Optimiser		SGD
Network Activation		Tanh

Table 6. Value of hyperparameter for DSAC and IDHP algorithms.

B. Code Reference for RL Algorithms

The code developed to obtain the results presented in this research is publicly available[1]. Additionally, this appendix provides the pseudo-code highlighting the logic behind the algorithms used to build the Reinforcement Learning hybrid controller. Algorithm 1 shows the the Incremental Dual Heuristic Programming pseudo-code and Algorithm 2 the Distributional Soft Actor-Critic pseudo-code.

Algorithm 1 IDHP algorithm

Initialize:
- $\theta, \phi \leftarrow \theta_0, \phi_0$ — *// Initialise actor and critic weights*
- $\Theta, \Lambda \leftarrow \Theta_0, \Lambda_0$ — *// Initialise parameter and covariance matrices*
- $x, s \leftarrow$ env — *// Initialise environment*

for $t = 1, \ldots,$ Episode End **do**
- $a_t \leftarrow \pi_{\theta_t}(s_t)$. — *// Sample action from actor π*
- $s_{t+1}, x_{t+1}, r_{t+1} \leftarrow \text{env}(a_t)$ — *// Gets environment response to action*
- $\lambda, \hat{\lambda}_{t+1} \leftarrow v_{\phi_t}(s_t), v_{\phi_t}(s_{t+1})$ — *// Gets critic for current time and prediction for next time*
- $\nabla_\theta \mathcal{L}_\pi(t), \nabla_\phi \mathcal{L}_\lambda(t) \leftarrow$ Eq. (12), Eq. (14) — *// Compute Actor-Critic gradient loss*
- $\theta \leftarrow \theta - \eta_A \nabla_\theta \mathcal{L}_\pi(t)$ — *// Update Actor weights*
- $\phi \leftarrow \phi - \eta_C \nabla_\phi \mathcal{L}_\lambda(t)$ — *// Update Critic weights*
- $\Lambda \leftarrow$ Eq. (9) — *// Update covariance matrix*
- $\Theta \leftarrow$ Eq. (10) — *// Udate parameter matrix*

Algorithm 2 DSAC algorithm. Adapted from Seres [12] and Duan *et al.* [20]

Initialize:
- $\theta, \phi \leftarrow \theta_0, \phi_0$ — *// Initialise Actor and Critic networks*
- $\theta', \phi' \leftarrow \theta, \phi$ — *// Initialise actor and critic target networks*
- $\mathcal{D} \leftarrow \{\}$ — *// Initialise replay buffer*
- $x, s \leftarrow$ env — *// Initialise environment*

for episode $n = 1, \ldots, N$ **do**
- **while** episode not End **do**
  - $a_t \leftarrow \pi_\theta(s_t)$. — *// Sample action from actor π*
  - $s', r' \leftarrow \text{env}(a_t)$ — *// Gets environment response to action*
  - $\mathcal{D} \leftarrow \mathcal{D} \cup < s, a, r', s' >$ — *// Store state transition*
  - Sample $|\mathcal{B}|$ samples from $\mathcal{D}$
  - $\mathcal{L}_z \leftarrow$ Eq. (26) — *// Compute Critic loss*
  - $Q(s,t) \leftarrow$ Eq. (24) — *// Compute action-value function*
  - $\mathcal{L}_\pi \leftarrow$ Eq. (16) — *// Compute Actor loss*
  - $\mathcal{L}_\pi^{\text{CAPS}} \leftarrow$ Eq. (19) — *// Include smoothness into Actor loss*
  - $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_\pi^{\text{CAPS}}$, $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_Z$ — *// Update Actor and Critic networks*
  - $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$ — *// Update target networks*

[1] **Code available at**: `https://github.com/iamlucasvieira/HybridRL-FlightControl`

Part II

Literature Study

This Part has previously been graded for the course AE4020.

2

Literature Review: A Survey on Reinforcement Learning for Flight Control

Reinforcement learning is a popular field for developing intelligent and autonomous systems that can learn to adapt by interacting with an environment. Flight control can benefit from reinforcement learning by allowing the development of robust and adaptive controllers to provide fault tolerance. This chapter is a literature review on reinforcement learning covering the main concepts of the field, the state-of-the-art techniques and the current status of applications that focus on flight control systems.

2.1. Reinforcement Learning: The Bridge Between Machine Learning and Decision Making

Machine Learning (ML) is a subfield of computer science that makes computers able to learn from patterns and relationships in data to solve problems [28]. There are three primary categories of machine learning, each distinguished by its type of learning: supervised, unsupervised, and reinforcement learning. Figure 2.1 shows the distinction between those methods concerning their input and output.

Figure 2.1: Classification of Machine Learning approaches into Supervised, Unsupervised, and Reinforcement Learning based on their input and output.

Supervised learning algorithms have the task of finding a map between values and labels. This type of learning is similar to function approximation. For example, given a dataset with x and y coordinates, the algorithm must find the most accurate approximation that maps an x into and y value. In this case, learning is a matter of ensuring that predictions match the known labels [29].

In contrast, unsupervised learning algorithms solve problems for datasets with unlabelled values. These algorithms rely on the characteristics of the input values to discover relationships and patterns in the data [28]. For example, an algorithm receiving the same x and y coordinates dataset could be tasked to

find clusters of data points using the distances and correlation between them. Note that the prediction, in this case, is a classification that is not part of the dataset (predicting a cluster label based on a x and y coordinate). While in the supervised learning case, the classification is part of the input dataset (predicting an y label based on a x coordinate).

Reinforcement Learning (RL) algorithms lie between supervised and unsupervised learning. Unlike supervised learning, RL does not use reference data to match a behaviour based on example data, and unlike unsupervised learning, it does not attempt to find a structure in the input data. Instead, RL algorithms learn to perform a task by trial-and-error [10]. The algorithm has as input the state of a system and learns based on the feedback it receives from interacting with this system, which can be positive or negative. The algorithm's goal is to learn the strategy that maximises the positive feedback it receives.

For a machine learning method to learn to control an aircraft, it must be optimal in decision-making tasks. The algorithm needs to learn a strategy to choose what action to take in a circumstance to result in the desirable aircraft response. Of the three ML methods discussed, RL is the most applicable for those types of tasks. Reinforcement learning's fit for control tasks is one of the reasons literature on this field is rich in applications for robotics [30] and flight control [12]. Therefore, this literature review covers the entire spectrum of RL, including tabular, approximate, and Deep Reinforcement learning methods. Figure 2.2 shows a map for the chapters discussing each method.

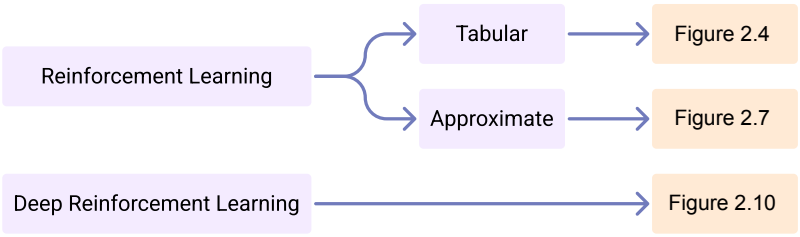


Figure 2.2: Schematic diagram showing the different Reinforcement Learning methods discussed in this literature review and their respective sections.

2.1.1. The Structure of RL Problems and the Markov Property

Reinforcement Learning problems are structured as an interaction between an agent and an environment in discrete time steps. The agent, which is capable of learning, takes actions (A_t) that change the conditions of the environment, also known as the state (S_t). The environment responds to the action's outcome with a new state (S_{t+1}) and a reward value (R_{t+1}) that reflects how satisfactory the new states are [10]. The subscript in these symbols represents the time step at which they occur. Figure 2.3 summarises the interaction between those two elements.

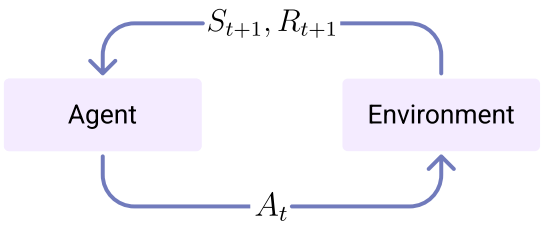


Figure 2.3: Diagram of the interaction between agent and environment in reinforcement learning.

In addition to the interaction framework, RL problems are commonly structured as Markov Decision Process (MDP). That means they are formulated as sequential decision-making problems where the following environment state S_{t+1} only depends on the current state and action and not on anything that occurred before that time [31]. The mathematical representation of the Markov property is defined by equation (2.1); the probability the environment will transition to a new state s' only depends on the action a taken at the current state s.

$$p(s' \mid s, a) \doteq \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} \tag{2.1}$$

With the Markov Property, the consequences of the agent's actions on the environment can directly translate into learning. The agent aims to maximise the rewards it receives, indicating whether its actions result in a desirable behaviour. The agent updates its strategy to avoid actions that yield low rewards and seeks actions with high rewards. Through this process, the agent learns a map that indicates the optimal action in a situation that can lead to the highest sum of rewards.

2.1.2. Reward and Returns: Key Concepts in Reinforcement Learning

The reward signal is the primary information that allows the agent to learn from its actions' consequences in the environment. The agent's objective is to maximise the sum of rewards it receives in the long run, known as the return. This section formalises the definitions of reward and return, which are fundamental concepts for understanding how RL algorithms learn.

Reward Signal

The reward (R_t) is a signal that assigns a numerical score to the outcome of the agent's action. The higher the reward, the more satisfactory the new environment state is. To illustrate, in the classic snake video game, a possible reward signal would reward the agent for each action that keeps it alive and receives a negative reward if it crashes.

If the agent's objective were solely to choose the action that yields the highest immediate reward, it might not find the global maximum solution. This is because an action can influence the future rewards the agent can attain. Therefore, in certain circumstances, it is advantageous to select an immediate action that does not offer an immediate high reward but instead leads to a path of higher rewards in the future [10]. Consequently, relying solely on immediate rewards to learn optimal behaviour is insufficient.

The concept of reward is crucial to RL, as it determines the quality of the agent's actions and shapes its learning process. The design of the reward function is the only way to specify what the agent should accomplish. Therefore, It plays a critical role in shaping the agent's behaviour in learning the optimal strategy.

Returns

In RL, the return (G_t) is a metric that measures the total of rewards the agent can attain from a sequence of actions. It considers the rewards from all interactions in a task, not just the immediate reward from an action. A typical example of a return function is the sum of all received rewards.

When defining a return metric, it is crucial to consider whether the studied problem classifies as episodic or continuous. Episodic tasks have a finite sequence of actions known as an episode. For those types of tasks, the agent learns the optimal behaviour by attempting to perform a task in multiple episodes. In contrast, continuous tasks have actions that do not have a definitive end [10].

The simple return that adds all rewards can be problematic for continuous tasks since its value may reach infinity, making the metric meaningless. One solution is the discounted return, which uses weights to decrease the importance of future rewards exponentially. The discounted return is mathematically expressed in equation (2.2). Here, the discount rate γ reduces the value of rewards that are further in time. As γ is between 0 and 1, the exponential decrease in weight ensures that the sum never reaches infinity. Note that if γ would be 1, this return is the simple sum of rewards [10].

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

2.1.3. Policy and Value Functions: Enabling Optimal Decision Making

The agent's objective is to learn how to act in different situations so that it receives the highest return. However, the return is only known at the end of the episode. During the episode, the agent must rely on expectations based on past experiences to learn the optimal behaviour. This section formalises the strategy that the agent is learning and how it keeps track of the returns from previous experiences to aid during decision-making in an episode.

The Policy

The prior sections have described that the agent's objective is to learn an optimal strategy to solve a problem. In RL, the term 'policy' is commonly used to refer to this strategy. A policy π is a function that maps states to actions, and the process of improving this function involves finding the optimal action for each state that results in the highest return. The ultimate goal is to identify the optimal state-action relations that maximise returns, resulting in the optimal policy π_* [10].

Value Functions

A value function is a crucial component that enables the agent to use information from previous experiences to estimate the expected return for future actions. The expected return for a given state is called the 'value'. For instance, saying that action A has a higher value than action B means that A is expected to yield a higher return than B. The two primary types of value functions used in RL algorithms are the state-value function, and the action-value function [10].

The state-value function informs the expected return for a state; in RL terminology, the state's value [10]. The state-value function is defined in equation (2.3). This conditional expectation equation evaluates the expected return given that the environment is at state s.

$$v_\pi(s) \doteq \mathbb{E}_\pi\big[G_t \mid S_t = s\big] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\Big], \quad \forall s \in \mathcal{S} \tag{2.3}$$

The action-value function gives the expected return for the environment being in a specific state and the agent taking a particular action [10]. Defined in equation (2.4), this value function determines the value for a particular state-action pair.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\big[G_t \mid S_t = s, A_t = a\big] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\Big] \tag{2.4}$$

Bellman Equation

Writing the value function as a relation of the value of the following state results in the Bellman equation [10]. This equation defines the value of a state as the sum of the immediate reward and the value of the next state. This formulation reinforces the Markov property by making explicit that the reward in the next state only depends on the action performed at the current state.

The policy defines the probability that the agent may choose each of the possible actions. Each action has a probability of leading to a particular reward and new state s'. If the probabilities of actions, new states, and rewards are known for all states, the state value function becomes a recursive average calculation, as shown in Equation 2.5 [10]. Here, π denotes the agent's probability of selecting a particular action, while p denotes the probability that an action yields a particular reward r and leads to a new state s'. This new state has its value function $v_\pi(s')$, which is also part of the average.

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \forall s \in \mathcal{S} \tag{2.5}$$

Thus, the Bellman equation computes the state value by estimating the weighted average of the reward of all possible actions from a state and all rewards from all possible new states after each action. This calculation proceeds in a chain, where the value of a state depends on the rewards from an action plus the value of the new state after the action. This type of calculation is known as bootstrapping [10].

2.1.4. Classification of Reinforcement Learning Algorithms

The evolution of RL algorithms has led to solving each time more complex problems. New algorithms usually use some of the concepts from previous ones together with their novelty. The evolution of one algorithm into multiple variants makes the field rich in algorithms. Hence, it is trivial to classify them based on how they learn and what type of problems they solve.

Model-based or Model-free

Model-based RL algorithms are the ones that need a model of the environment dynamics for learning. The model could be either provided to the agent or learned by it. Because those algorithms use the model predictions of rewards and future states, they are highly sample-efficient, needing fewer interactions with the environment [12].

On the other hand, Model-free algorithms do not need a model of the environment. While that makes them not benefit from increased sample efficiency, it makes them more simple and flexible. For instance, a model-free algorithm can learn in different environments, while a model-based algorithm would need a model for each. Besides that, Model-free algorithms are attractive for fault-tolerance applications, where it is interesting to explore the algorithm's behaviour beyond the operational range that models are designed and validated to predict [6].

On- or Off-policy

The policy has two functions during learning; it is the means the agent uses to interact with the environment and what the agent aims to learn. Depending on whether those two functions are explicitly distinguished or not is what classifies algorithms into on- or off-policy.

On-policy algorithms use a single policy for both exploring and learning. Therefore, the policy used to choose an action during interactions with the environment is updated based on the rewards. On the other hand, off-policy methods use two policies; one responsible for acting, the behaviour policy, and the other being updated based on the interactions, the target policy. Splitting the tasks of exploring and learning makes off-policy algorithms more data efficient because the experience from exploration can improve multiple target policies [11].

Value or Policy-based

Value-based methods, also known as Q-learning, learn an approximation of the value function to estimate the expected return of a given state or state-action pair. The estimation is iteratively updated based on the observed rewards and transitions in the environment. On the other hand, policy-based methods directly optimise the policy without estimating a value function [10].

Different applications of RL to flight control have been successful using value-based [32, 33] and policy-based [34, 35] methods. Value-based methods tend to converge to the optimal policy more quickly than policy-based methods. However, they are better suited for problems with discrete state and action spaces, and they can be sensitive to the choice of the value function approximation, leading to overestimation [36]. On the other hand, policy-based methods are well-suited for problems with continuous or high-dimensional state and action spaces with the drawback of being less sample efficient [37].

Offline or Online Learning

Offline learning refers to training an RL agent using a dataset of past experiences without interacting with the actual environment. This approach suits systems that do not change frequently and can be used to train the agent offline before it is deployed in the real-world environment. The advantage of offline learning is that it allows for large amounts of data, leading to a more accurate and efficient agent. However, the agent may not perform well in new or changing environments as it has been unable to learn from these situations. For instance, a flight controller only trained offline can show poor online performance, as during actual flight, different conditions than trained may be found.

On the other hand, online learning involves training the agent in real-time through interaction with the environment. This approach suits systems that change frequently or for which it is difficult to collect a large dataset of experiences. The advantage of online learning is that the agent can learn from new situations as they occur and adapt to changes in the environment. However, the agent may perform worst than an agent trained using offline learning, as it has less data to learn.

An agent that learns to control an aircraft with offline data may not be able to extrapolate the learned policy to the actual flight environment, therefore, not allowing adaptiveness. Online learning alone could provide a tailored policy for the actual flight conditions; however, the required number of sampling to achieve satisfactory performance is unrealistic. An alternative is combining those two methods [38, 39, 40]; offline learning provides a robust policy and online learning updates this policy during flight, allowing adaption to changes in the environment, such as wind conditions or equipment failure.

2.2. Tabular Solutions in Reinforcement Learning

The data structure of the policy and value functions in early RL algorithms are tables - the reason why they classify as tabular solution methods. For instance, the policy is a table that maps a state-action pair to the probability of the agent selecting that action. Similarly, the value function is a table that maps the pair to the expected return of selecting that action. Storing the information into tables makes those algorithms incompatible with continuous problems, where states and actions are not finite. While this makes them unsuitable for flight control applications, they developed the main concepts and strategies used by the state-of-the-art RL algorithms. This section defines three tabular solutions algorithms, which are classified in figure 2.4: Dynamic Programming, Monte Carlo, and Temporal Difference.

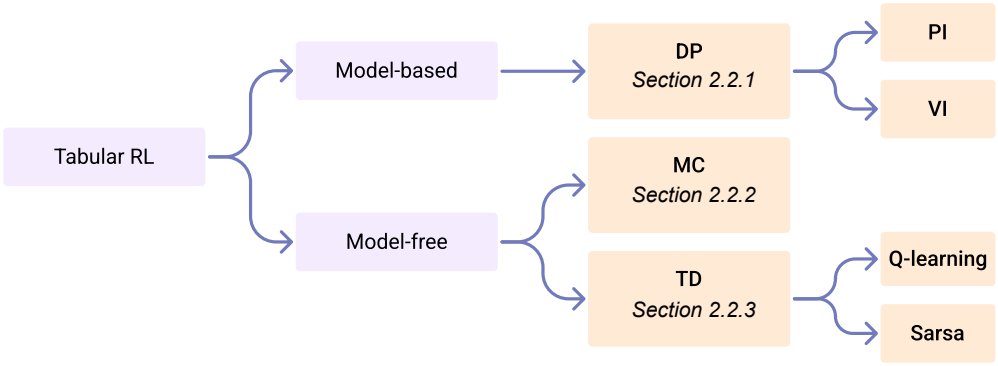


Figure 2.4: Diagram with the classification of algorithms within tabular solutions reinforcement learning.

2.2.1. Dynamic Programming

Dynamic Programming (DP) solves multi-stage decision-making problems to minimise a cost function [41]. The complexity in DP problems is that actions influence the possible outcomes from future actions; therefore, a solution is only possible when decisions are considered in series [42].

To update the policy, DP algorithms evaluate all possible outcomes from a starting state. That includes the possible future states and rewards the agent may encounter from taking different actions. To build this knowledge of possible outcomes, DP uses a model of the environment. Therefore, this algorithm classifies as model-based.

Solving Dynamic Programming Problems: Policy Iteration

Policy Iteration (PI) is the method DP algorithms use to determine the optimal policy. The process consists of iterating between a policy evaluation and policy improvement step, as shown in figure 2.5. The policy evaluation improves the state-value function prediction. Then, the policy improvement uses the new value-function to update the policy such that it selects actions with higher value. Each cycle of PI finds an equal or better policy than the starting one. Therefore, multiple iterations make the policy converge to the optimal policy [10].

Policy Evaluation

The policy evaluation in PI is the stage that calculates the state-value function for the current policy. The environment model provides the reward for choosing an action at a specific state. Determining the state-value functions is a question of iteratively computing the rewards received in all combinations of possible actions and states [10].

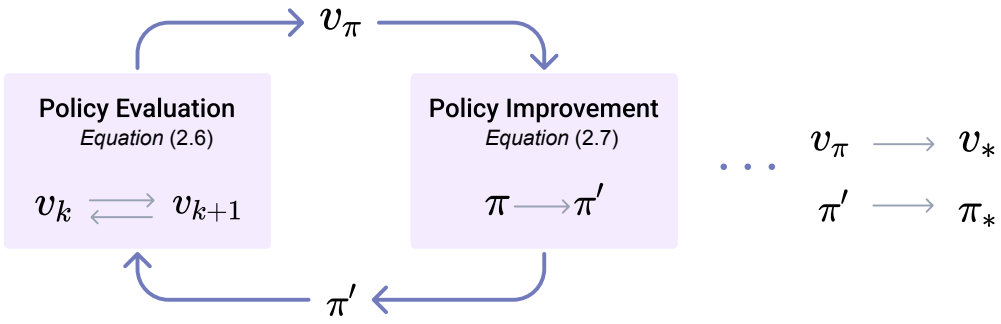


Figure 2.5: Diagram of the policy iteration process with evaluation and improvement steps.

To illustrate this process, consider a policy that chooses a specific action from an initial state. Given this action, the model informs the new state and reward. The policy and model can recursively predict the following states and rewards from this new state until the episode finishes. With all the rewards calculated through the process, it is possible to update the value of the initial state. The complete update of the state-value function requires finding the updated value for all possible initial states.

A single iteration of policy evaluation occurs as described in equation (2.6). After a single update, the state-value function v_{k+1} equals the rewards from all possible actions plus the value of their following state s' according to the previous value function. Updating the value-function infinite times make it converge to the actual state-value function for the current policy [10].

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_k(s')\big] \end{aligned} \tag{2.6}$$

Policy Improvement

The policy improvement updates the actions the policy would choose such that they have the highest value according to the updated value function [10]. The algorithm compares all the values of all possible actions from a state and chooses the one with the highest value. Therefore, the policy improvement is a greedy update where all actions in the updated policy are the ones that give the highest return according to the current value function.

The purpose of the policy improvement, defined by equation (2.7), is determining the greedy policy π' by choosing the greedy actions from the action-value function. It is possible to rewrite the action-value function as a function of the state-value function. Equation (2.8) rewrites the action-value function as the sum of the reward from the action with the state-value of the following state. Then, it is possible to adjust this equation to use the information from the policy evaluation, resulting in equation (2.9) [10].

$$\pi'(s) \doteq \arg\max_a q_\pi(s, a) \tag{2.7}$$

$$= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \tag{2.8}$$

$$= \arg\max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big] \tag{2.9}$$

The policy improvement is the second and final stage of the PI. The policy evaluation determines the value function of the current policy, and the policy improvement uses the gathered knowledge about values to update all actions to build a greedy policy. Iterations of PI result in the policy converging to the optimal policy π_* and the state-value function converging to the optimal state-value v_* function. The equations on the right side of figure 2.5 show the convergence of those two functions.

Solving Dynamic Programming Problems: Value Iteration
Value Iteration (VI) is an alternative method for finding the optimal policy of DP problems. It differs from PI by removing the iteration within the policy evaluation. For VI, the policy evaluation only updates the state-value function once. VI is a specific case of stopping the policy improvement loop. Other methods can truncate the iteration at any other number of steps. In fact, the PI also includes a truncation that stops the policy evaluation iteration after no significant improvement in the value function [10].

Truncating the policy evaluation makes the state-value function never converge to its actual values for the current policy. Using a single step makes this function only receive a single improvement. The consequence of this change is that PI becomes a single-step calculation, described by equation (2.10) [10]. This single-step equation shows that the single-step improvement of the value function includes selecting the action that maximises the returns. Therefore, the value function and the policy receive an update in a single step.

$$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big] \tag{2.10}$$

Looking back into figure 2.5, which describes the cycles in PI, it is possible to examine the differences after truncating the policy evaluation. For VI, the iteration inside the ”Policy Evaluation” block does not exist. In this case, the value update is a single step moving v_k into v_{k+1}. Besides that, the output of this block would be v_{k+1} and not v_π, because VI does not attempt to find the state-value function for the current policy. For the calculation of policy evaluation and improvement, the only equation needed for one iteration cycle is equation (2.10).

Generalised policy iteration
Generalised Policy Iteration (GPI) is the generalisation of the PI idea of learning the optimal policy through evaluation and improvement. Different RL algorithms borrow and adapt the GPI concept of improving value function and policy by building different structures for those processes [10]. For example, both PI and VI use the GPI concept, but they differ in the structure of evaluation and improvement. Ultimately, all GPI methods aim to build interaction between policy and value functions until both reach optimality.

2.2.2. Monte Carlo Method

The Monte Carlo (MC) method extends DP to address model-free problems [10]. Unlike DP, MC does not rely on a model. Therefore, MC algorithms can only discover future states and rewards through interactions with the environment.

Policy Iteration for Monte Carlo
The policy evaluation in the MC method estimates value functions by averaging the returns from different episodes. The agent uses the return from an episode to update the value of the state-action pairs it visited. In an incremental form, the update is defined in equation (2.11). After the update, the new value of a state-action pair $Q(S_t, A_t)$ equals an average of the returns from all episodes that this pair has been visited [10]. In the update equation, $\frac{1}{n}$ is the timestep, with n being the total number of state-action pairs the agent visited from all episodes.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{n}[G_t - Q(S_t, A_t)] \tag{2.11}$$

The policy improvement stage in the Policy Iteration for MC is the same as in DP [10]. The action-value function updates the current policy by making it greedy. In other words, the value function replaces each action in the policy with the one that maximises the return. Therefore, the only difference between MC and acrshortdp in the PI process is the way those methods determine the action-value function.

Pros and Cons in the Monte Carlo Method
The estimation of the action-value function in MC only converges to the actual function if the set of episodes during training explores the environment entirely. Therefore, if the agent never visited a

specific state-action pair, the algorithm would never learn the true value function. That behaviour can be a pro or a con depending on the problem studied.

If only a state-action area is of interest for a task in an environment with ample state-action space, the non-convergence of the value function is an advantage. The algorithm must only visit the state-action in the interest area frequently to learn their values. That creates accurate value functions for that region without the need for learning values for the entire state-action space [10].

2.2.3. Temporal Difference

Temporal Difference (TD) is a tabular solution method that solves model-free problems with higher sample efficiency than the MC. method It does that by merging Monte Carlo with the concept of bootstrapping from Dynamic Programming [10]. The method aims to find a balance between learning completely from experience, like in MC, or learning completely from knowledge, like in DP.

The MC method is not data-efficient because it only updates the estimated value function at the end of an episode (recall that equation (2.11) uses the return G_t of an episode). If a task had a long episode, this algorithm would take a long time to learn. An alternative to making updates only at the end of an episode is to use information from previous episodes. For example, suppose a certain state has been visited in a previous episode. In that case, it is possible to use the value found during that episode, allowing a value-function update before the current episode ends.

Policy Evaluation in Temporal Difference

The TD policy evaluation updates the value function with the estimation of the return, which is the sum of the reward R_{t+1} with the estimated value for the next state $V(S_{t+1})$. This update is shown in equation (2.12). Because this equation only uses an estimation of the return, it allows updating the value after each episode step instead of after each episode termination.

$$V(S_t) \leftarrow V(S_t) + \frac{1}{n}\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \tag{2.12}$$

The working principle of TD algorithms follows the same idea of GPI; First, policy evaluation(i.e. determining the action value function) and then improvement. The evaluation and improvement can be either on- or off-policy. The following parts of this section describe the two main TD algorithms – Sarsa, an on-policy method, and Q-learning, an off-policy method.

Sarsa: An On-policy Temporal Difference Algorithm

Sarsa is an on-policy TD algorithm [10]. The action-value function update in Sarsa is as in equation (2.13). The update only uses the information of the immediate reward from the environment and the value estimation for the following state.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{n}\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \tag{2.13}$$

Iterations of policy evaluation in Sarsa will make the estimated value function approach the true value function of the current policy q_π. When the algorithm improves the policy by selecting the actions that give the highest value, it needs to perform evaluation again, but now using the new policy. Repeating the GPI cycle makes the algorithm learn the optimal policy π_* and its optimal value function q_*.

Q-learning: An off-policy Temporal Difference Algorithm

Q-learning is an off-policy TD algorithm. Because it is off-policy, it does not interact with the environment using the learnt policy as a guide. The policy evaluation in Q-learning updates the action-value function with equation (2.14), which considers immediate reward and estimated values. However, the algorithm does not use the action A_{t+1} suggested by the current policy to estimate the future return. Instead of that, it chooses the action that maximises the expected return.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{n}\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \tag{2.14}$$

Repeating the policy evaluation makes the estimated action-value function approach the actual value function for the current policy q_π. In Q-learning, repeating the policy evaluation does not make the function learn the current policy's action-value function. Instead, the algorithm learns the optimal value function q_* directly. The reason is that the algorithm chooses the action that maximises expected returns when updating the value function (equation (2.14)). That makes the method go off-policy and perform greedy updates. In other words, the algorithm "disrespects" the action the current policy suggests by choosing the action expected to give a higher return.

Although Q-learning algorithms can learn the optimal policy, they may show poor online learning performance. That is because this algorithm always updates the action-value function with the action that provided the highest value, ignoring that, in some states, an action with a considerably low reward has the probability of being taken [10]. For example, consider a state with two possible actions, one with a high value and one with a low value. Q-learning would desire to be in this state, as it only considers and focuses on the action with maximum value. Therefore, this algorithm may enter a dangerous situation (state) as it learns values focusing on profit and neglecting the involved risks.

2.3. Approximate Solutions in Reinforcement Learning

Approximate solutions are the RL methods that generalise the agent's experience, allowing it to learn optimal policies in environments with large or continuous state space. Those methods replace the tabular data structure from tabular solutions with function approximators. This new data structure allows generalisation, such as predicting the value of a state that the agent has never visited.

2.3.1. Neural Networks as Function Approximators

A Neural Network (NN) is the most common function approximator in RL. By mimicking biological neural systems, a NN reduce the complexity of a problem by dividing it into small processing units, known as neurons [29]. Figure 2.6 shows a typical structure of a NN. The input data flows through the network while multiplied by the weights w that connect neurons. The network output y is its prediction. Making the network learn to make accurate predictions requires adjusting the connection weights between the neurons.

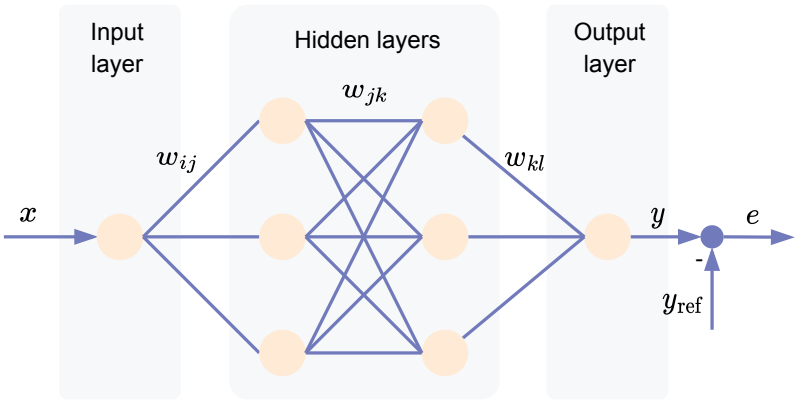


Figure 2.6: Simple Neural Network structure with single input x and single output y. The difference between the output and the reference output y_{ref} gives the network's error e. The subscripts in the weights w indicate the identification of the source and destination neurons in this order.

The traditional method for improving the weights of a NN is backpropagation. It feeds the input training data through the network to obtain the prediction y. The prediction error (or a loss function) can be fed backwards through the network with respect to its partial derivatives. The objective is to determine the current weights' influence on the error and incrementally update the weights to improve the network performance. In other words, backpropagation determines the error gradient concerning the network weights $\partial e/\partial w$ and updates the weights to increase an objective [43].

2.3.2. Actor-Critic Designs in Approximate Dynamic Programming

Approximate Dynamic Programming (ADP) is the group of RL methods that implement function approximators into DP to overcome the *curse of dimensionality* [44], which is the issue of computational cost growing exponentially with the number of variables in the problem [45]. Within ADP, Actor-Critic Design (ACD) [45] are popular methods for control applications. Those methods split the agent into two parametric structures: the critic, approximating the value function, and the actor, approximating the policy. This division makes explicit the agent's function of learning the optimal policy and the optimal value function.

The interaction between actor and critic is a form of Policy Iteration. For the policy evaluation, the actor chooses an action based on the current state, and the critic estimates the state value. Then, for policy improvement, the algorithm uses the environment information about the following state and reward to update the weights of the critic and actor networks [46]. While this learning principle is common to all ACD methods, they differ in how the actor and critic are structured. The different versions of those algorithms result in the families depicted in figure 2.7.

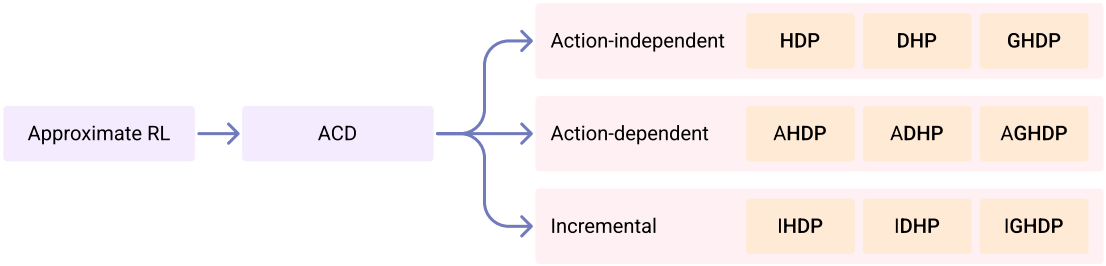


Figure 2.7: ACD algorithms and their classification.

The advantage of ACD for control systems is that learning the critic allows bootstrapping, reducing variance and speeding learning. Bootstrapping uses the knowledge of all states to update values, which reduces the need for additional sampling [37]. Besides that, ACDs allow online implementation as the update in the actor and critic parameters only depends on the information of the current states [47]. However, ACD methods are model-based. That is the case because the parameter update requires estimating the future state, which is determined from a plant mode [48].

Heuristic Dynamic Programming

Of all ACD methods, HDP is the most studied in literature [44]. What defines the HDP method is its critic, which is a neural network that outputs an approximation $V(s_t, w)$ of the value function $v(s_t)$ [49]. The approximation is a function of states s_t and network weights w. The objective is to find the weights for which the critic network approximates the state-value function most accurately. It is possible to use any supervised learning method to update the network weights [45], such as the error backpropagation discussed in section 2.3.1.

The update of the critic- and actor-network weights uses a backpropagation of the critic's estimation error, i.e. the error in value estimation. This error depends on the results from two steps in time. The reason is that it is only possible to know how wrong the critic's predictions are by advancing one time step in the episode and observing the actual reward and following state. The estimation of the critic's error is illustrated in figure 2.8, with the dashed arrows showing the error backpropagation.

The error estimation in figure 2.8 starts with the critic network estimating the value of the current state $V(S_t)$ and the actor moving the environment to a new state by applying action A_t. Then, the critic estimates the value of the state after one-time step advance $V(S_{t+1})$ and observes the received reward $R(S_t)$. In an optimistic scenario, the value prediction of an initial state S_t equals the prediction of the subsequent state S_{t+1} plus the received reward [49]. Thus, the value estimation error, described in equation (2.15), is the difference between the state's value prediction and an updated value that includes a sampling from the environment.

$$e = V(S_t) - [V(S_{t+1}) \cdot \gamma + R_{t+1}] \tag{2.15}$$

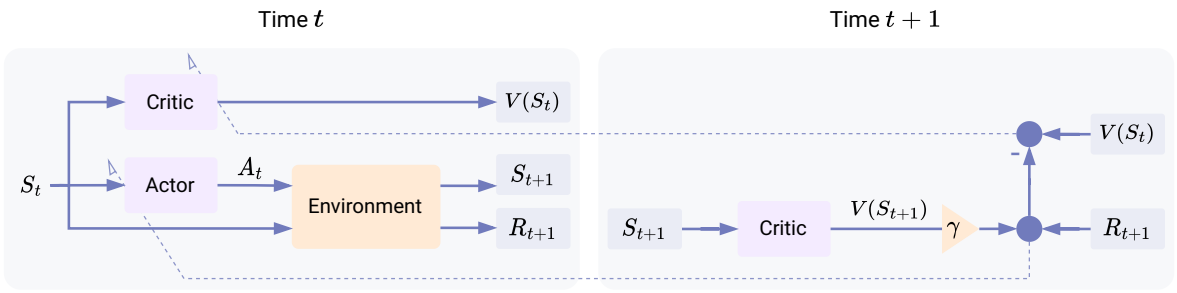


Figure 2.8: The structure of HDP algorithms and the error backpropagation (Adapted from Wang, Zhang, and Liu [50]).

Dual Heuristic Programming

The DHP implementation is similar to the HDP, only differing in the definition of the critic. In DHP, the critic network outputs the gradient of the value function with respect to states $\partial V(S_t)/\partial S_t$ instead of the value function itself [49]. The reason for learning the gradient is that the learned value function in HDP is only needed for backpropagation, which requires the calculation of its gradient. Therefore, directly learning the gradient immediately prepares the network output for backpropagation through the actor and critic network.

Compared with HDP performance, DHP applications have been shown to solve problems faster [48]. However, with the drawback of more complexity due to a network that learns derivatives. The diagram in figure 2.8 is valid for HDP under a few modifications. The critic output should be the value function gradient $\partial V(S_t)/\partial S_t$, and the reward R_{t+1} added to the error value estimation of S_{t+1} should be the reward gradient $\partial R_{t+1}/\partial S_t$.

Globalised Dual Heuristic Programming

Globalised Dual Heuristic Programming (GDHP) is a method that combines HDP and DHP to improve the accuracy of decision-making in complex systems. The critic in GDHP learns an approximation of both the value function and its derivatives, providing twice as much information as traditional HDP and DHP. While this critic allows GDHP to achieve more accurate results, it also makes the computations more intensive [44]. Furthermore, Sun and van Kampen [51] observed that because the critic network learns two different pieces of information with the same layers, it raises coupling errors that lead to instability.

Action-dependent Implementations

The previously discussed ACD are categorized as action-independent, as they assume that the actor and critic are independent. Contrary to that, Action-dependent methods assume that the value function depends on the current action the actor selected. All the three action-independent methods discussed (HDP, DHP, GDHP) have their AD form (ADHDP, ADHDP, ADGDHP). The difference is that in their AD form, the actor's output is also an input for the critic network [44]. A summary of those designs and their action-dependent forms is provided in figure 2.9.

The advantage of the AD forms is that it allows for model-free learning. The backpropagation through the critic allows determining the gradient of the values with respect to the action, which can update the actor's weights without the need for using a model of the systems [52]. However, Van Kampen, Chu, and Mulder [53] shows that keeping the actor and critic independent and instead adding a neural network that learns an approximate plant dynamics can lead to nearly twice the success rate in converging to the correct control law.

Incremental Implementations

Incremental Approximate Dynamic Programming (iADP) are ACDs that, in addition to the actor and critic, learn an approximation of the system dynamics.The incremental form of HDP, DHP, and GDHP are IHDP, IDHP, IGDHP, respectively [54]. The advantage of the incremental forms is that they combine the benefits of the action independent ACD and Incremental approaches. From ACD, it brings the power of general function approximation of the actor and critic, and from incremental models, the ability

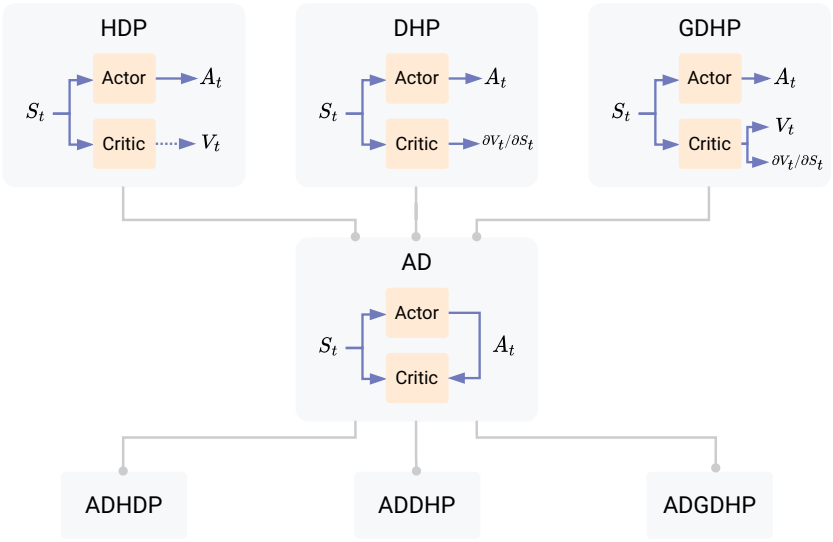


Figure 2.9: Overview of ACD algorithms and their AD forms (Adapted from Ferrari and Stengel [48]).

to speed the learning without needing offline learning. For those reasons, incremental models make iADP suitable for adaptive, model-free and online controllers of nonlinear systems [55].

In iADP, the agent learns an incremental model that approximates the system dynamics and allows predicting the change in states Δx_{t+1} due to an action u_t at the current state x_t. The incremental model assumes that the system dynamics can be represented by a first-order Taylor expansion, as in equation (2.16). In this equation, F and G are the matrices the incremental model learns with system identification techniques such as least squares.

$$\Delta x_{t+1} \approx F[x_{t_0}, u_{t_0}]\Delta x_t + G[x_{t_0}, u_{t_0}]\Delta u_t \tag{2.16}$$

The benefit of iADP methods is that they allow online, model-free, and adaptive learning. For IDHP, different literature [54, 55, 56] have demonstrated its suitability for control tasks. The results show that it is faster and more precise when compared to its action-independent form [54]. Literature using IDHP [57, 58, 59, 35, 60, 38] shows improved efficiency, accuracy, adaptiveness and robustness, making it an attractive method for online and adaptive FCS. While IGDHP shows superiority in performance, it still suffers from high complexity, as GDHP, which makes them less attractive for real applications [61, 51].

2.4. Deep Reinforcement Learning: An Overview of Recent Advances

The Deep learning (DL) field introduced neural networks with multiple processing layers, known as Deep Neural Network (DNN). Each layer learns a different abstraction, known as features, which contributes to the solution of the problem [62]. An example is the layers in an algorithm that identify objects in an image. While one layer could learn edges in the images, another could learn shapes. With the combined information from those layers, the network can classify objects.

Deep Reinforcement Learning (DRL) is the merge of DL into the RL, which makes it able to solve high-dimension problems [63]. This improvement enables solutions for problems that classical RL can not solve and allows scaling previous works in RL to work in higher dimensions. Because of that, in the literature, it is common to find works previously done in classical RL renovated with DRL with a demonstration of performance improvement, e.g. Q-learning vs Deep Q-learning.

This section aims to survey different DRL algorithms while focusing on their potential for flight control systems. Because DRL can solve high-dimension problems, they show great potential to control applications. This section only covers model-free algorithms, which are the types desired to accomplish fault-tolerance tasks. The algorithms discussed are classified in figure 2.10.

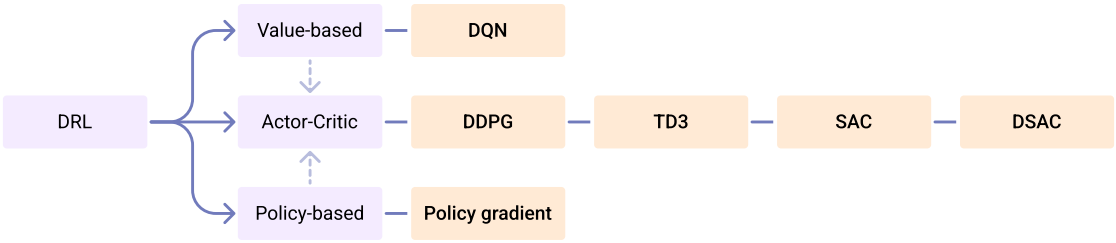


Figure 2.10: Diagram with DRL algorithms and their classification.

2.4.1. Deep Q-Networks

The Deep Q-Network (DQN) algorithm surges from the merge of Q-learning with DNN, making it the first implementation of a RL algorithm that can handle high-dimensional input data [64]. The algorithm is a variant of Q-learning, keeping the characteristics of being a value-based and off-policy algorithm. However, it differs by introducing a DNN approximator for the state-value function. Besides, the algorithm uses a stochastic gradient descent strategy and an experience replay that allows re-using information from previous agent-environment interactions.

The invention of DQN in Mnih et al. [64] showed the algorithm's potential in learning how to play different Atari games. The results showed that the agent outperformed humans [65] and state-of-the-art algorithms [37] in different games. Subsequent researchers have successfully adapted DQN with different strategies to increase its performance and data efficiency; Rainbow [66] is a noteworthy example. Despite the success of applications using DQN, the algorithm can only handle discrete and low-dimensional action spaces [65]. Therefore, it is not ideal for a flight control task, which, in high-fidelity simulations, requires continuous actions

2.4.2. Policy Gradient Methods

Policy gradient methods are the ones that learn an approximation of the policy with a NN. Their policy is represented as π_θ, where θ indicates the parameters of the approximator. The algorithm's ultimate objective is to maximise the expected return, as defined in equation (2.17). To achieve this objective, the agent uses the gradient ascent rule in equation (2.18) to update the policy's parameters [10].

$$J(\pi_\theta) = \mathbb{E}[R(\pi_\theta)] \tag{2.17}$$

$$\theta_{k+1} = \theta_k + \gamma \, \nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{2.18}$$

Policy gradient algorithms have two main advantages over value-based ones. First, they are easier to apply to continuous problems [37]. Second, they avoid overestimation bias, common to algorithms that learn value functions (like Q-learning and DQN). The overestimation occurs because the value function update is greedy (recall the update rule in equation (2.14)). Therefore, any initial overestimation increases through the updated cycle in a vicious iteration which obstructs the learning of the optimal policy [36].

However, policy gradient algorithms also have their drawbacks. Those algorithms are typically on-policy, which makes them significantly less data-efficient than value-based methods. Furthermore, they also show significant variance during learning, making convergence difficult. The variance issue derives from the intrinsic behaviour of the instability in updating a NN weights [37].

The subsequent algorithms discussed in this section combine the ideas from value-based and policy-based algorithms. They aim to acquire the qualities from those two methods and handle their deficiencies. By implementing different strategies, they address the issues of overestimation in value-based methods and the issues of sampling inefficiency and high variance in policy-based methods.

2.4.3. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a model-free and off-policy algorithm that improves DQN performance by introducing an actor-critic structure with a deterministic policy. For a deterministic policy, there is no need for integrating over the action space, which makes the algorithm more efficient for continuous action problems [67]. On top of that, DDPG uses the replay buffer and target network

strategies from the actor-critic structure to address the robustness and stability issues in DQN[65]. Combining those modifications make DDPG a stable algorithm for continuous and high-dimension control actions.

Replay Buffer
Experience replay is a method that reduces the time an agent takes to learn by allowing it to use previous experiences. In this method, each agent-environment interaction, in the form $< s_t, a_t, s_{t+1}, r_{t+1} >$, is stored. Then, when training, the agent can sample one of those experiences and use it as if it was sampled from a real interaction. That makes the algorithm needs fewer environment sampling to converge [68].

Target Network
Implementing a target network is a way to stabilise the training of RL problems. The target network is a copy of the policy network; however, its weights stay fixed for a determined number of iterations. After completing these iterations, the algorithm updates the target network's weight to match the policy network. The training process updates the policy network's weight using the target network's error. As the target network does not change for some time, the policy weight changes are less abrupt [63]. This solves the instability issues of updating NN weights.

The DDPG results showed stable learning in multiple training environments. Compared to DQN, DDPG needed 20 times fewer steps in the same tasks [65]. However, the method still shows the same difficulties typical to model-free algorithms, which is the large number of training steps required before showing satisfactory performance.

2.4.4. Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed DDPG (TD3) is an actor-critic algorithm that builds on DDPG. It addresses the issue where DDPG often fails to learn due to the critic's tendency to exploit overestimation error. To reduce the overestimation, TD3 uses three strategies: Clipped Double Q-learning, Delayed Policy Updates, and Target Policy Smoothing Regularization [36].

Clipped Double Q-learning
Because the value function estimation tends to be overestimated, TD3 uses two critics, each learning an independent value function. When learning, the algorithm opts for the networks that provide the lowest value. This could lead to underestimation, which is preferred over overestimation, which grows because of the exploit of overestimation.

With the clipped double Q-learning, the learning objective becomes the one in equation (2.19). The update uses a target network, $Q_{\theta'}$, which is an approximation with parameter θ'. While the prime symbol in the θ' indicates the network is a target, in the state s' and a', it indicates that those are the immediate next state and action, respectively.

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', a') \qquad a' \sim \pi_{\phi_1}(s') \tag{2.19}$$

Delayed Policy Updates
Updating the value network with a target can reduce the overestimation error growth. The magnitude of this error influences the divergence behaviour of the policy in its update. Therefore, a lower error from the value network can lead to a lower divergence in the policy update. For this reason, TD3 does less frequent policy updates than value function updates.

Target Policy Smoothing Regularization
The issue with deterministic policies is that they have a chance of overfitting by exploiting the overestimated Q-function. Regularization is a way of reducing the variance in the target policy network to smooth the value estimation. The smoothing introduces the idea that similar actions should have similar values. This translates into adding noise to the region around the action, making the algorithm bootstrap nearby actions and behave similarly to a stochastic policy in that region. The regularization

makes the algorithm resemble SARSA, described in section 2.2.3, which also benefits from bootstrapping nearby values.

With the addition of regularization, the policy from the objective with double Q-learning equation (2.19) becomes the one in equation (2.20). The equation shows that the policy is modified to include the random noise ϵ that promotes bootstrapping. The random noise is clipped to the boundaries $[-c, c]$ to stay near the origin action and is shaped by the parameter σ.

$$a' \sim \pi_{\phi_1}(s' + \epsilon) \qquad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c) \tag{2.20}$$

2.4.5. Soft Actor-Critic

Soft-Actor Critic (SAC) is a DRL actor-critic algorithm introduced by Haarnoja et al. [69] to address the challenges of model-free and continuous space RL. Sample efficiency is one of those challenges. Reducing the number of samples needed for learning is vital to allow their applications to real systems. A second challenge is a brittleness in learning. The hyperparameters of the learning algorithm dictate the convergence condition. Removing the hyperparameter's influence in convergence is crucial for safe learning.

Three concepts define the SAC algorithm. (1) It has an actor-critic structure. (2) It is an off-policy algorithm, which allows it to be more sample efficiency by using past experiences during learning. Recall that on-policy algorithms suffer from sample efficiency as they usually require a new sample for every policy update. (3) the agent's objective is to maximise an entropy term in addition to the cumulative rewards. This additional term allows a balance between learning stability and exploration [70].

The SAC's return function contains an entropy term $\mathcal{H}$ in addition to the reward r. Therefore, the algorithm's objective is to maximise the cumulative rewards and entropy. This objective is described in equation (2.21). The idea of adding the entropy term is to allow exploration. In case of low estimation errors, the algorithm is encouraged to explore. Otherwise, it focuses on robust behaviour — the temperature factor α balances the importance of the entropy against the reward.

$$\begin{aligned} \pi^* &= \underset{\pi}{\text{argmax}} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot \mid S_t))] \\ \text{with,} \quad & \mathcal{H}(\pi(\cdot \mid S_t))] = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log(\pi(a \mid s))] \end{aligned} \tag{2.21}$$

2.4.6. Distributional Soft Actor-Critic

Distributional Soft-Actor critic (DSAC) is a RL algorithm that incorporates a distributional perspective into the traditional SAC algorithm. The maximum entropy theory increases the diversity of actions, improving exploration, while distributional RL provides additional information to the value function, allowing it to learn safer policies [71]. Therefore, DSAC merges the benefits of maximum entropy and distributional RL. The main difference in DSAC structure is that its critic outputs a probability distribution of the return. In contrast, in SAC, the critic outputs the expected return.

On-policy algorithms have a stochastic policy that allows vast exploration. However, on-policy learning is data inefficient, and stochastic policies can lead to unstable behaviour. On the other hand, off-policy algorithms, such as DQN, DDPG, and TD3, borrow from Q-learning the idea of adding noise to a deterministic policy, allowing for efficient data use and exploration [71]. In a different approach, SAC uses a stochastic policy with off-policy learning. The algorithm adds an entropy factor to balance how stochastic the policy can be. This combination makes it possible for it to achieve stability and sample efficiency [69]. DSAC extends SAC by considering the probabilities in the rewards to allow for a more robust estimate of the policy gradient, which can lead to improved learning performance.

With the combined benefits of SAC and distributional RL, DSAC has outperformed the state-of-the-art TD3, and SAC in learning performance across various environments [71]. This highlights the benefits of incorporating the distributional information of rewards and actions into the reinforcement learning

process. DSAC has also been shown to reduce the failure rate in risk-sensitive environments. In an experiment in Ma et al. [71] where the robot could fall, DSAC led to a decrease in the failure rate. However, the algorithm also caused a reduction in the maximum return, reflecting a trade-off between performance and safety.

2.5. State-of-the-art of DRL for Flight Control Systems

Deep Reinforcement Learning algorithms have demonstrated their capability of solving problems in various fields, including aviation. This section reviews research that applies DRL to FCS. The objective is to examine the methodology and results of each study to understand the prospects of using DRL for safe, adaptive, and fault-tolerant flight control systems.

Offline PPO Controller | Bøhn et al. [72]

The study in Bøhn et al. [72] is a proof-of-concept for using RL-based controllers on fixed-wing Unmanned Aerial Vehicle (UAV). The objective is to develop a controller that learns to control the UAV and track a reference attitude signal. The proposed controller uses the Proximal Policy Optimisation (PPO) [73] algorithm.

The experiments demonstrate that the PPO controller can stabilise the vehicle from different initial attitudes and airspeed values, indicating its robustness. The controller's generalisation power allows it to control the UAV even under turbulence, a condition not included during the training phase. Compared to a traditional PID controller, the RL controller showed a higher success rate, particularly during aggressive manoeuvres. However, both controllers exhibited similar tracking performance, as shown in figure 2.11.

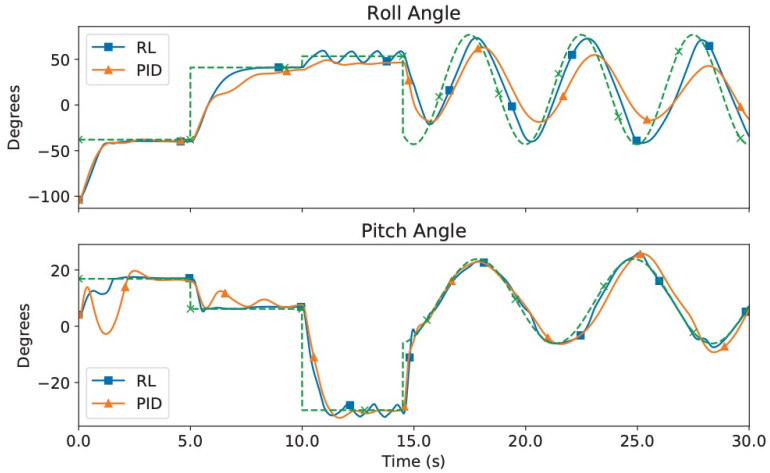


Figure 2.11: Results from Bøhn et al. [72] comparing the PID and RL attitude tracking performance with a UAV simualtion. The dashed line represents the reference signal, and the solid lines are the actual output of the controllers.

The study concludes that DRL shows promise for nonlinear flight control applications. The research also highlights the need for future investigations into the reality gap, the discrepancy between simulated and real-world environments. For the implementation of the PPO, the study recommends further simulations of extreme situations, while for possible alternative algorithms, it suggests exploring controllers built with the SAC algorithm.

Online IDHP + PID Controller | Heyer, Kroezen, and Van Kampen [58]

The research from Heyer, Kroezen, and Van Kampen [58] developed an RL adaptive FCS for the Delft University Aircraft Simulation Model and Analysis Tool (DASMAT)[74], which is a 6 Degrees Of Freedom (DOF) simulation of the Cessna 550 Citation II. The proposed controller consists of two loops; the inner one is an IDHP agent that controls pitch and roll rates, and the outer one is a conventional PID controller for tracking altitude and roll angle.

The results showed that the controller learned to track the reference attitude rates with online learning only, shown in figure 2.12. Additionally, the controller showed it could learn to control the longitudinal

angles even in non-trimmed flight conditions. As part of the result, the research confirms that using a target critic network increases learning stability, despite its disadvantage of slowing convergency. Concerning adaption, the agent can fly a constant-altitude rate-one turn while compensating for two faulty scenarios: low aileron effectiveness and noisy control input.

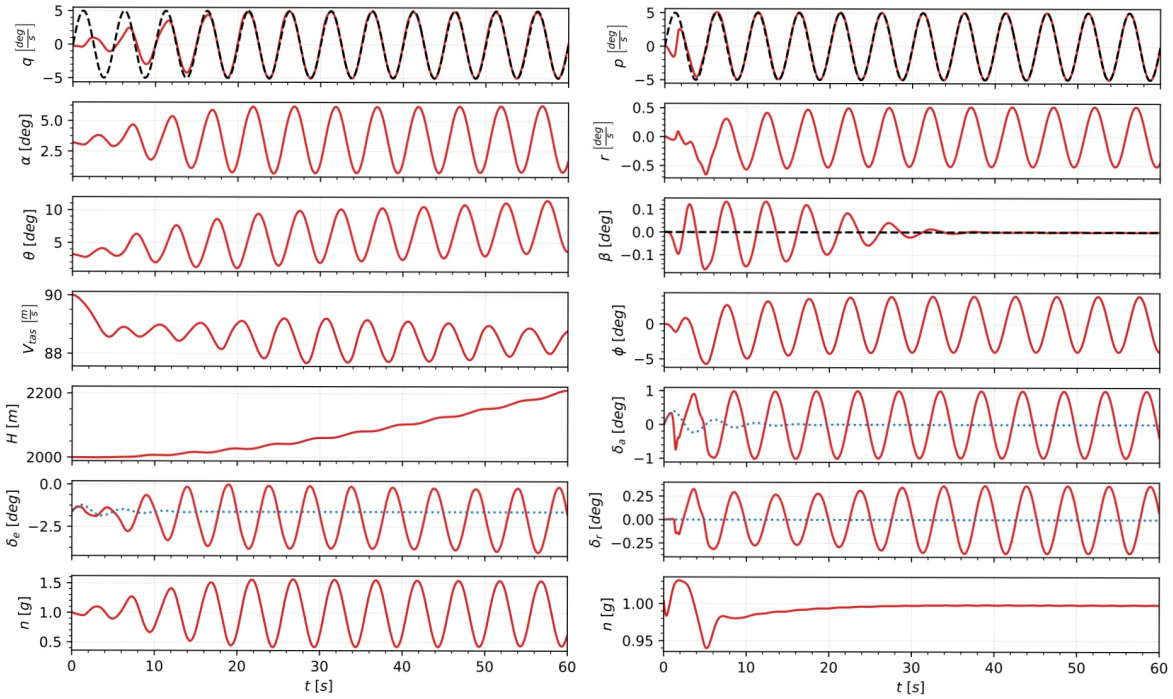


Figure 2.12: Online tracking results from Heyer, Kroezen, and Van Kampen [58]. Graphs on the left show longitudinal, and on the right, lateral tracking, with dashed lines as the reference signal.

Despite the positive outcomes of the IDHP agent in allowing online learning and adaptation, the research leaves some points for improvement in future research. One is the dependence on conventional PID controllers, which, if replaced, could make the overall system more model-independent. A further recommendation is the study of the algorithm on higher control levels or the implementation of a cascade network controller, as done in Enns and Si [75] and Zhou, van Kampen, and Chu [76].

Online Cascaded IDHP Controller | Lee and Van Kampen [35]

The research from Lee and Van Kampen [35] follows up the work in Heyer, Kroezen, and Van Kampen [58]. It develops a cascaded IDHP controller and removes the need for the pre-tuned PID controller, making the algorithm more model-independent. The controller is tested on the Cessna 550 Citation II aircraft through the DASMAT software package. The proposed structure has an outer loop agent that tracks the altitude and an inner loop agent that tracks the pitch angle.

The first experiment compared the performance of the cascaded and a baseline controller. While both structures could learn an approximated optimal policy, the cascaded showed a significantly higher success ratio. However, the increase in the success ratio came with the disadvantage of slower convergence.

One of the research's objectives was to reduce the simulation-reality gap, a deficiency common to general RL applications. For this reason, it analysed the tracking performance of both controllers when dropping the assumption of perfect sensors, which was assumed in Heyer, Kroezen, and Van Kampen [58]. The results show that both controllers can achieve near-optimal control even with noise in the sensor measurements, with the cascaded outperforming the baseline. However, the introduction of noise reduced their learning speed.

A final experiment demonstrated that the baseline controller could perform the tracking task even under gust conditions by adding noise to the input instead of the measurements. While this research moved

towards more independence from system models and addressed the reduction of reality and simulation gaps, it did not show that IDHP alone can perform better than the structure that contains a pre-tuned PID controller.

Offline Cascaded SAC Controller | Dally and Van Kampen [77]

To progress on the development of fault-tolerant FCS, the research in Dally and Van Kampen [77] builds an offline-trained cascaded controller using the SAC algorithm. The cascade structure splits the controller into an inner loop tracking attitude and an outer loop tracking altitude. The outer loop outputs a reference pitch angle value, which is one of the inputs in the inner loop agent. The final structure of the controller is shown in figure 2.13.

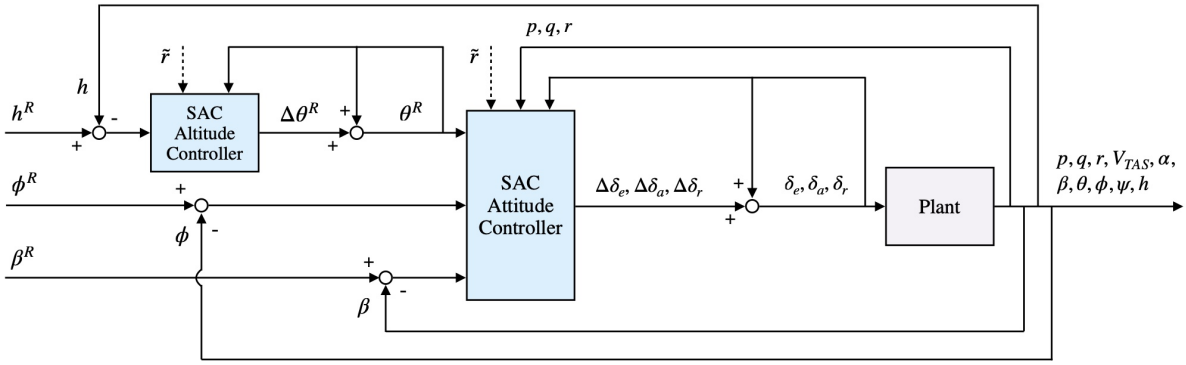


Figure 2.13: Structure of the cascaded controller from Dally and Van Kampen [77]. The inner loop and outer loops track the attitude and altitude, respectively.

In the experiments, the agent learns the controller offline by training the attitude controller and, subsequently, the altitude controller with the already trained inner loop. The results show that learning is unstable even near the final training steps. Even though the agent only controls the input increment, which should be more stable than directly controlling the input, the learning instability is still significant. Those observations show that SAC's low reliability and low sample efficiency make it difficult to use in online applications.

Concerning the evaluation performance, the SAC agent shows good coupled attitude and altitude tracking tasks for nominal flight conditions experiments. It succeeds in a $40°$ bank climbing turn and a $70°$ bank flat turn. As the controller's ultimate objective is to provide fault tolerance, experiments showing operation under six failure cases were studied. Those include a jammed rudder, reduced aileron effectiveness, reduced elevator range, partial loss of horizontal tail, icing, centre-of-gravity shift, and biased sensor noise.

The offline learned controller can not change during online simulations, as SAC is not sample efficient. To allow a demonstration of adaptiveness, the author switches the fixed controller with one previously trained for a specific failure condition. Therefore, when a failure occurs, the research can show the fixed controller's robustness and the adapted controller's behaviour while skipping the adaption process itself.

The switching strategy is a way of overcoming the difficulty of making a RL algorithm sample efficient to the point that it can allow online learning. However, using a pre-trained controller for failure modes is not ideal. (1) Because it is impractical for real applications, as all possible failure modes should be known a priori. (2) An aircraft model should be used to develop a pre-trained controller for each possible failure mode. Despite that, the individual results on the robustness and adaptiveness of the algorithm are still of extreme relevance to demonstrate the extent to which it can provide fault-tolerant control.

In the jammed rudder experiment, the overall robust response shows a more stable and accurate reference tracking than the DHP adaptive implementation in Ferrari and Stengel [78] while performing the same task. For the reduced aileron effectiveness, the SAC robust control performed comparably with the adaptive IDHP controller from Heyer, Kroezen, and Van Kampen [58]. The achievement of the

SAC robust controller performing comparably with adaptive controllers from literature is attainable due to the combined benefits of DNNs generalisation and stochastic policies robustness. However, this behaviour is not consistent in all failure modes. For example, with a shift in the centre-of-gravity shift experiment, the robust controller struggles to pitch down, resulting in a significant altitude error.

The research advances the state-of-the-art by demonstrating a cascaded DRL flight controller operating in nominal and failure conditions. The proposed recommendations for future research are to explore the performance of deterministic policy-based algorithms like TD3 or on-policy like PPO in place of SAC.

Hybrid Cascaded SAC-IDHP Controller | Teirlinck [38]

The work in Teirlinck [38] develops an adaptive controller for a Cessna 550 Citation II simulation through the DASMAT software using a cascaded RL structure. The research novelty is using a hybrid SAC-IDHP agent in the attitude-tracking inner loop of the controller. A hybrid approach could merge the IDHP's capability of online adaption and the SAC's generalising potency. In addition to the hybrid agent, the controller has a SAC-only agent as part of its outer loop, which tracks the altitude. The final controller's structure is shown in figure 2.14.

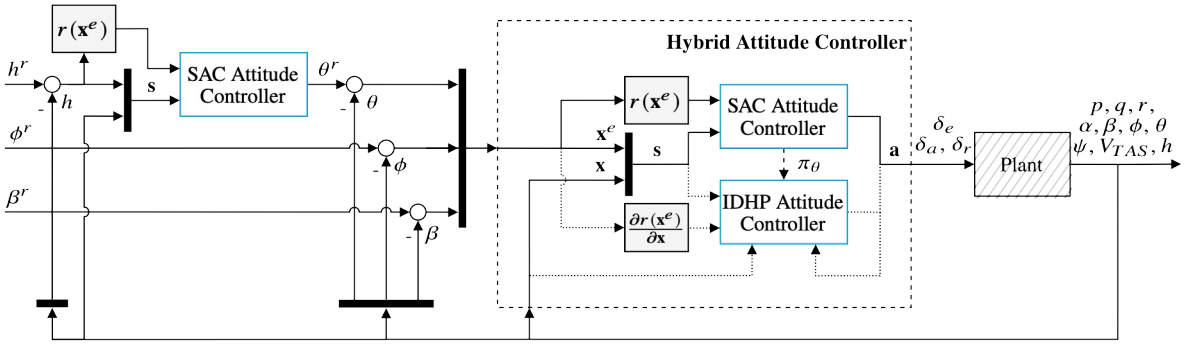


Figure 2.14: Structure of the Hybrid cascade controller from Teirlinck [38]. The controller comprises two feedback loops: an outer loop that uses the SAC algorithm for altitude control and an inner loop that combines SAC and IDHP for attitude control.

The training strategy of the proposed controller requires both offline and online training. The SAC agent from the outer loop and hybrid layers learn to track a reference signal offline. After training those controllers, the IDHP learns online, where only the weights of the IDHP algorithm change. Conventional IDHP-only controllers require initial excitation to allow the identification of the incremental model. However, with the hybrid framework, the output of the pre-trained SAC policy under reference signals can work as the exciting forces on the IDHP agent.

The topology of the IDHP-SAC policy NN combines the neurons from the offline-trained SAC and the online-trained IDHP. That means, during online operation, the SAC neurons do not update, leaving only the IDHP agent neurons able to change. Therefore, the hybrid structure maintains information from the robust SAC agent while allowing some adaptiveness through the IDHP online learning process.

The research results compare the performance of the proposed hybrid controller against a SAC-only controller, as shown in figure 2.15. In nominal operations, the hybrid controller slightly improves altitude tracking performance. That concludes that both controllers can perform altitude tracking in nominal circumstances.

In addition to operating in standard conditions, the research compares the controller's ability to provide fault tolerance. In adverse conditions, the hybrid controller improves the tracking performance of a SAC-only controller with the drawback of increased oscillations in the aircraft response. With a drop in elevator effectiveness, the hybrid approach shows a 5.46% improvement in tracking performance. The improvement in the case of an aileron effectiveness reduction is minor, with a figure of 0.82%. Nevertheless, the hybrid shows improvement in the rise time of the bank angle and keeps the sideslip angle closer to zero.

Additional results show the increased performance of the hybrid controller concerning robustness and

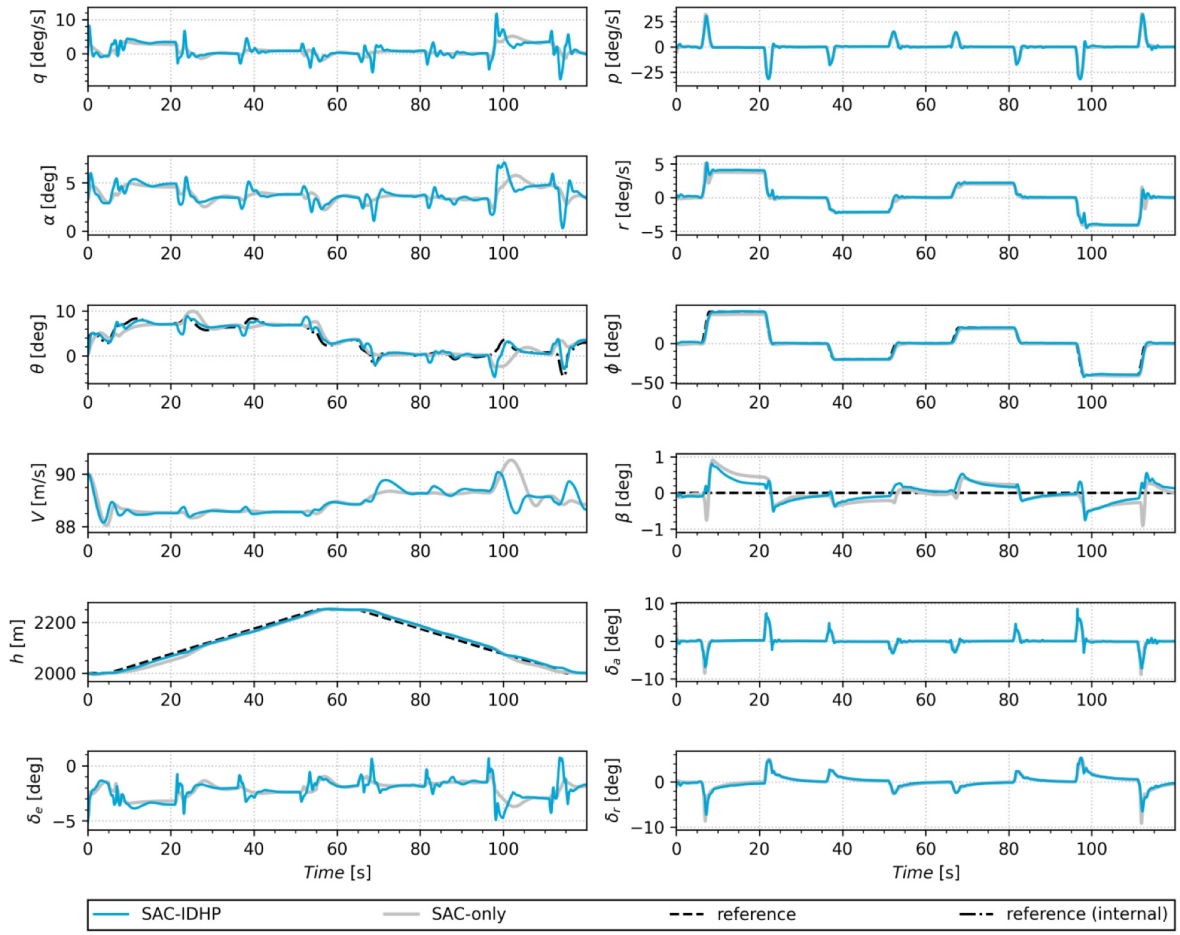


Figure 2.15: Results from Teirlinck [38] comparing the tracking performance of the hybrid SAC-IDHP against the SAC-only controller in nominal flight conditions.

sensitivity. Compared to the SAC-only controller, the hybrid shows more robustness to different IFCs. When considering biased sensor noise, the hybrid controller performs slightly better than the SAC one. However, again with the disadvantage of more oscillatory behaviour.

Inner-loop DSAC Flight Controller | Seres [79]

While literature is rich in RL algorithms that can control simulated environments, a reality gap prevents those algorithms from displaying the same results in real-world environments [6]. Improving those algorithms to be more accurate, robust, adaptive, and efficient is essential to narrowing this gap. Seres [79], proposes an Distributional Soft-Actor critic (DSAC) flight controller, which can consider the risks involved in an environment to increase the consistency in learning.

The Distributional Soft-Actor critic (DSAC) algorithm classifies as a distributional RL. While in traditional RL, the return function maps action-states to an expected return, in RL, the function maps to a return distribution. The information in the distribution allows distributional RL algorithms to provide risk-sensitive learning.

The considered control task is tracking the attitude of the DASMAT high-fidelity model of the Cessna 550 Citation II. The controller should track pitch θ_r and roll ϕ_r angles reference while regulating the sideslip β_r angle to be constantly zero. While Dally and Van Kampen [77] and Teirlinck [38] use a cascaded controller architecture, Seres [79] is only concerned with the inner-loop. The motivation is that the inner-loop is the safety-critical part of the controller, which is sufficient to answer if distributional algorithms can improve safety. Besides that, disregarding the outer-loop reduces the complexity of the system.

The results show that DSAC agents converged earlier, providing a 56.3% increase in sample efficiency. Furthermore, the returns at the end of training reached a higher magnitude and a lower variance than a SAC controller. The lower variance confirms that DSAC improves the controller's stability. The result was also consistent in an environment with more extensive neural networks.

Concerning the evaluation results,DSAC shows a slight increase in tracking performance compared to SAC. The relevance of this outcome is the demonstration that DSAC can cope with SAC's tracking performance while providing safer learning.

In conclusion, Seres [79] provides three significant results. First, It shows that DSAC is more consistent and stable than SAC. Second, It demonstrates that DSAC has a similar tracking performance as SAC. Third, it indicates that Distributional RL learns the uncertainties of the environment and allows the learning of safer policies.

2.6. Concluding Remarks

Reinforcement Learning is a method for learning through repeated experience an optimal strategy to solve a task. Typically, a RL algorithm uses an agent to attempt to accomplish a task multiple times and improve its strategy based on the outcomes. In a process analogous to biological learning, the algorithm learns a strategy that avoids repeating previous ”mistakes” and favours reaching previous ”successes”.

The RL algorithms classify based on the data structure they use to store the information from experiences. Tabular solutions store information in tables, approximate solutions in function approximators, and deep reinforcement learning in Deep Neural Networks. The type of data structure not only classifies the algorithms but also determines their applicability. For instance, tabular solutions are limited to discrete problems, as storing the entire experience of continuous problems in tables is not feasible. In contrast, approximate solutions and DRL can extrapolate experiences to a discrete space. Because DRL uses DNN, it allows learning high-dimensional problems.

The high number of publications on RL reflects its success and popularity in solving problems in different disciplines. A significant part of its popularity is that it allows for determining an optimal strategy for a decision-making problem even when there is no prior knowledge about the problem's rules. The ability to learn strategies for any task defined as a decision-making problem makes RL appealing for different disciplines, such as robotics, game playing, finance, and healthcare. For instance, different RL research shows algorithms learning to play video games from scratch and outperforming humans.

A promising application of RL in aviation is the development of Fault-Tolerant Flight Control System (FTFCS). Unlike automatic systems, adaptive controllers can learn new behaviour during flight to adjust to unpredictable conditions, increasing flight safety. Besides improving safety, adaptive controllers could also make the development of autopilots less costly, as they do not require PID tuning. Moreover, controllers that can learn to adapt will be a bridge towards autonomous flight. Those findings provide an answer to Research Question 1.1.

Research on RL for FTFCS has studied different combinations of algorithms and controller architectures. A cascaded controller architecture shows to result in the highest success rate. Besides that, the strategy of learning offline and online shows to achieve better tracking performance. The IDHP algorithm is the most satisfactory option for the online tracking task as it is far more sample-efficient than other algorithms. For the offline algorithm, SAC exhibits adequate performance, and DSAC shows the potential of increasing learning performance. Furthermore, TD3 is also an option for an offline algorithm that should be studied.

The literature research has demonstrated the promising potential of RL algorithms in enhancing aviation safety and provided insights into the field's current state-of-the-art, addressing Research Question 1.2. Cascaded controllers with a hybrid learning approach have demonstrated robust and adaptive attitude tracking. However, a remaining challenge is ensuring they consistently learn an optimal and safe policy. Hence, in addition to the literature study, a preliminary analysis will study the effectiveness of SAC, DSAC, and TD3 in achieving these goals. The analysis aims to identify the most suitable algorithm for improving consistency and discover methods that optimise learning to generate safer policies, contributing to an answer to Research Question 1.3.

3

Preliminary Analysis: Comparative Study of SAC, TD3, and DSAC

This chapter contributes to the literature study with a comparison of SAC, TD3, and DSAC in a flight control tracking task with offline learning. Section 3.1 gives the motivation for the analysis and section 3.2 informs on the details of the experiments' environment. The experiments compare the algorithms' performance under different reference signals (section 3.3), observation vectors (section 3.4), and reward functions (section 3.5). The chapter concludes in section 3.6.

3.1. Motivation for a Preliminary Analysis

Using RL to learn to control a complex system is time intensive. For instance, while humans can learn to play an Atari game in minutes, Rainbow [66] needed $83\,\mathrm{h}$ to achieve comparable performance [37]. Because of that, experimenting directly with a non-linear and high-fidelity aircraft simulation is unpractical. Therefore, a preliminary analysis with a simplified aircraft environment can allow for more experimentation in a short time. Besides that, dealing with a simpler problem allows a familiarisation with the studied algorithms and concepts, facilitating the subsequent work in the complex environment.

The literature study showed the evolution of RL in FCS applications. Teirlinck [38] demonstrated that a cascaded controller with an offline-trained SAC and online-trained IDHP agents can learn a robust and adaptive policy. Dally and Van Kampen [77] noted the need for study on alternative algorithms to the offline SAC agent, such as TD3, which could potentially lead to improvement in tracking performance. Furthermore, Seres [79] shows that DSAC improved the learning performance when compared to SAC by making the learning more consistent. Those ideas show the potential for improvement in the hybrid offline-online controller with the adjustment of the offline algorithm.

With this motivation, this chapter performs three experiments that compare SAC, TD3, and DSAC learning to track a reference pitch rate angle q of an LTI approximation of the short period dynamics of a Cessna 550 Citation II. The three experiments compare the performance of those algorithms under modifications in the environment formulation. Experiment I explores a comparison under different reference signals, Experiment II under different observation vectors, and Experiment III under different reward functions. The sources from the implementation of those algorithms used in the experiments are listed in table 3.1.

3.2. Experimental Setup of the Plant Dynamics and Environment

The environment in RL informs the consequences of an action on a system state and how satisfactory the new state is. If the system is complex, learning to predict what action can lead to a desirable state is expensive. That is why the plant dynamics used in this analysis is a Linear Time-Invariant system that approximates the short-period motion of the Cessna Citation 500 II aircraft. This plant is a part of the environment of the experiments, where the agent should learn to control the aircraft elevator to

Table 3.1: Details of the sources of the SAC, TD3, and DSAC algorithms implemented in this study, including their original research publications.

Setup	Algorithm	Description
ALGO-1	SAC	Using the SAC implementation from Raffin et al. [80] with default initialisation
ALGO-2	TD3	Using the TD3 implementation from Raffin et al. [80] with default initialisation
ALGO-3	DSAC	Using the DSAC implementation from Seres [79] with default initialisation

track a reference pitch rate.

The short period LTI approximation in state-space form is described in equation (3.1). The aircraft states are the angle of attack α and the pitch rate p; the input is the elevator deflection δ_e. The detailed equations for each term in the state and input matrixes and the values of the aircraft stability and control derivatives are available in appendix A. The equations of motion and the aircraft data are retrieved from Mulder et al. [81].

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} z_\alpha & z_q \\ m_\alpha & m_q \end{bmatrix} \begin{bmatrix} \alpha \\ q \end{bmatrix} + \begin{bmatrix} z_{\delta_e} \\ m_{\delta_e} \end{bmatrix} \delta_e \tag{3.1}$$

The environment in all three experiments has the task of tracking a pitch rate signal. However, each experiment modifies the environment to explore the performance of the studied algorithms in different learning conditions. Experiment I studies modifications in the shape of the reference signal, Experiment II changes the information the environment returns to the agent, and Experiment III changes the function determining the reward an agent receives after an action. Each experiment's modifications are listed in figure 3.1.

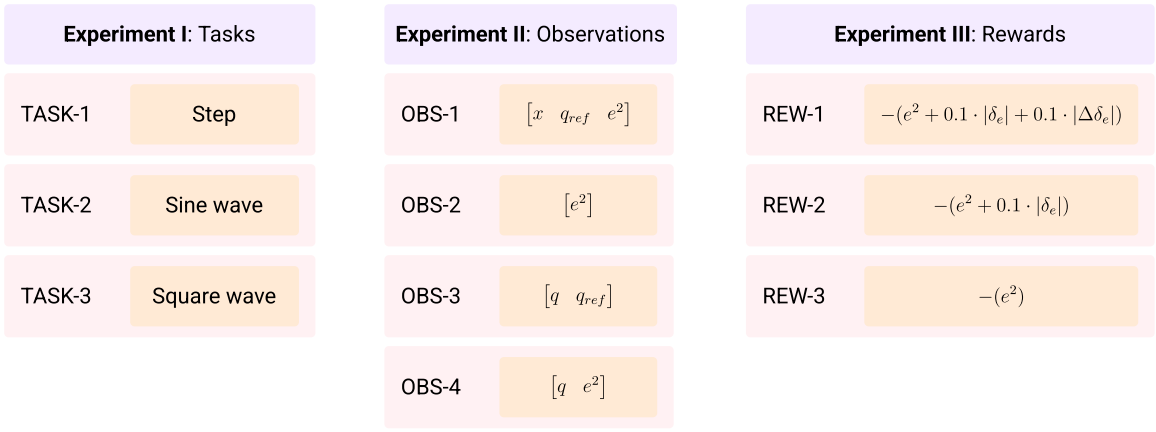


Figure 3.1: Summary of the different configurations in each experiment.

The discussion of the experiments' results focuses on each algorithm's learning and tracking performance. Learning performance concerns the algorithm's extent of updating the model's parameters to improve the return over time. The characteristics of how each algorithm learns to perform a task allow for judging their maximum return, converge time and reliability. The tracking performance, also called evaluation, seeks to understand the quality of the trained agent acting in an episode. The tracking information, which includes the accuracy and success rate through episodes, allows for determining how satisfactory is the policy the agent learned.

3.3. Experiment I: Comparing Tracking Tasks

The shape of the tracking reference signal can entirely change the difficulty or even feasibility of the RL task. For instance, the reference signal could be a smooth curve, a step curve, a random fast-changing signal, or any other function shape. In different tasks, an algorithm might perform differently. Because of that, this experiment compares the performance of the three algorithms in three different tasks, detailed in table 3.2.

Table 3.2: Overview of the experimental setups studied in Experiment I, each comprising a different task.

Setup	Task	Description
TASK-1	Step	The agent must make the aircraft's pitch rate follow a step input applied at the start of the episode with a magnitude of $0.1\,\mathrm{rad\,s^{-1}}$
TASK-2	Sine wave	The agent must make the aircraft's pitch rate follow a sinusoidal wave with amplitude $0.1\,\mathrm{rad\,s^{-1}}$ and a period equal to half of the episode length
TASK-3	Square wave	The agent must make the aircraft's pitch rate follow a squared wave with amplitude $0.1\,\mathrm{rad\,s^{-1}}$ and a period equal to half of the episode length

The graphs in figure 3.2 show the learning performance of the algorithms averaged over the three different environments, each with one of the different tracking functions. DSAC and TD3 find a slightly higher return at the end of training than SAC. While DSAC finds the highest return, it is slower than the other two algorithms. On average, the three agents learn to control the aircraft during the entire episode, as the right plot shows that the mean episode length converges to the maximum episode length of 100 steps. However, TD3 has a high variance in episode length. In some episodes, the agent fails to control the aircraft during the entire length by making the aircraft pitch rate exceed the limits.

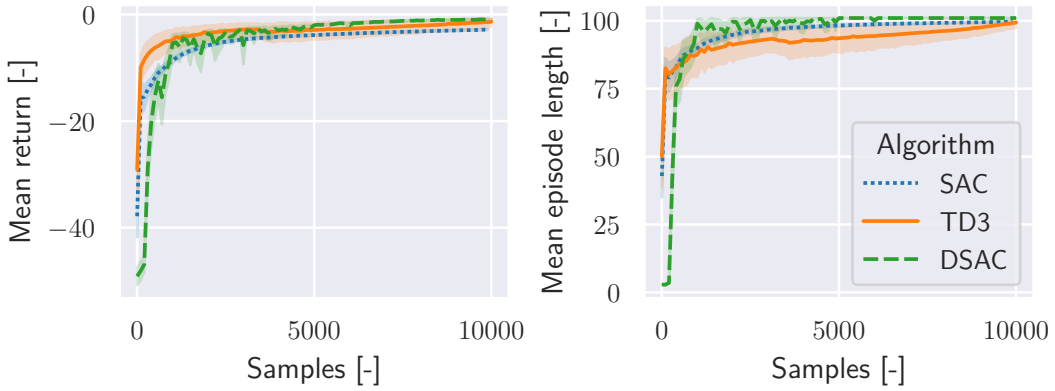


Figure 3.2: Learning performance of SAC, TD3, and DSAC trained on 10,000 samples in Experiment I. The left graph displays the mean return, and the right panel shows the mean episode length. Both graphs average each algorithm's performance over 30 random seeds (ten for each task type) .

While figure 3.2 averaged the algorithm's performance over the three types of tasks, figure 3.3 shows them separately. SAC and DSAC consistently converge to a maximum return in the three tasks with low variance. On the other hand, TD3 shows difficulty learning TASK-1 and a high variance in the maximum return for the three tasks at the end of learning.

In figure 3.4, it is possible to see the average tracking of DSAC in the three tasks together with the reference signal. This image shows that the DSAC has higher difficulty tracking the signal in TASK-2. The same is observed in figure 3.5, which shows the evaluation performance of the three algorithms after running ten episodes on each task. The three agents had more ease in tracking the signal in TASK-1. TD3 shows that despite inferior learning performance, it can still learn a policy that performs well when acting in the environment. TD3's mean tracking error is lower than that of the other two algorithms. However, it has the drawback of showing high variance, which could lead to episodes where it performs poorly or even fails.

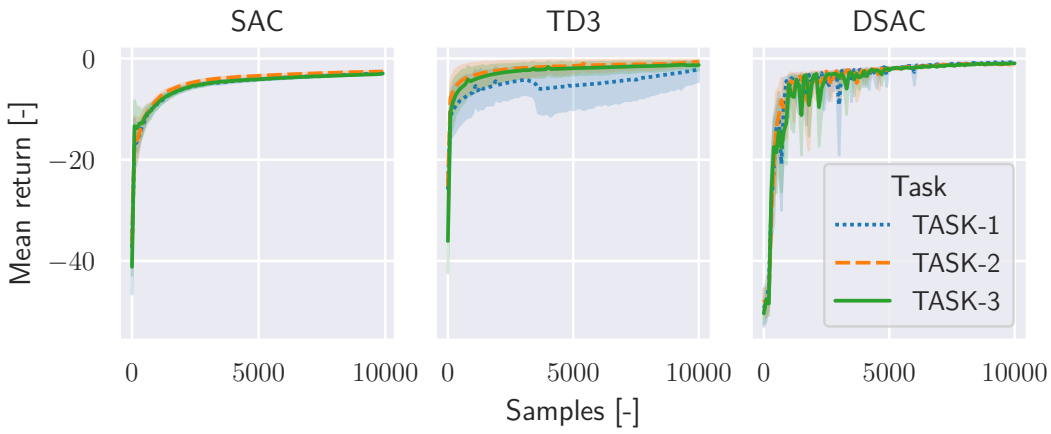


Figure 3.3: Learning performance of SAC, TD3, and DSAC trained on 10,000 samples in Experiment I. The figure displays the mean return of each algorithm for the three different tasks, each averaged over ten random seeds.

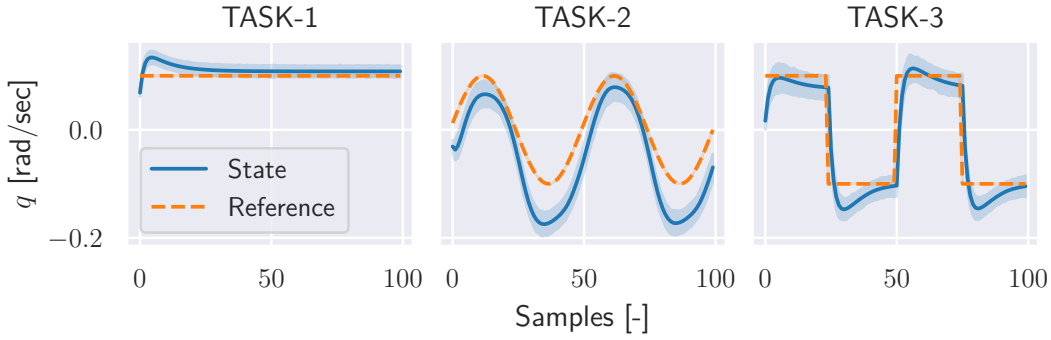


Figure 3.4: Evaluation performance of a trained SAC agent on the three tasks in Experiment I. The figure displays the average state value across 30 different seeds, with an episode length of 100 samples and a sampling frequency of $0.1\,\mathrm{Hz}$, resulting in a $10\,\mathrm{s}$ episode.

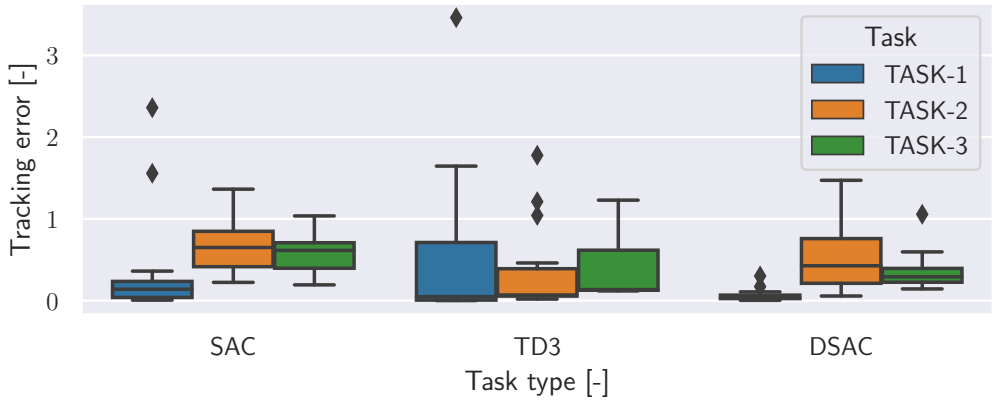


Figure 3.5: Comparison of tracking error for SAC, TD3, and DSAC in Experiment I.

3.4. Experiment II: Comparing Observation Vectors

The observation vector carries the information the RL algorithm uses to learn and decide what action to take next. A poor observation vector can degrade the algorithm's performance to the point that it can obstruct learning. For example, an agent might not learn how to exit a 2D maze if it only receives information about its horizontal position. In this case, the algorithm would not learn the prospect of rewards for being at different heights of the maze.

The second experiment of this analysis compares SAC, TD3, and DSAC learning and tracking performances in four different environments, each containing a different observation vector. The environments have in common the task of tracking a sinusoidal pitch rate reference with an $0.1\,\mathrm{rad\,s^{-1}}$ amplitude and the negative squared tracking error as a reward function. The differences between the environments are detailed in table 3.3. Each algorithm is trained in ten different random seeds for each environment.

Table 3.3: Overview of the experimental setups studied in Experiment II, each comprising a different observation function.

Setup	Observation	Description
OBS-1	$\begin{bmatrix} x & q_{\mathrm{ref}} & e^2 \end{bmatrix}$	This observation vector contains all the aircraft states x (In this case, α and q), the values of the reference signal q_{ref}, and the tracking squared error e^2, which is the squared difference between the actual and the reference pitch rates
OBS-2	$\begin{bmatrix} e^2 \end{bmatrix}$	This observation vector only contains the squared error of the tracked state q and the reference signal q_{ref}
OBS-3	$\begin{bmatrix} q & q_{\mathrm{ref}} \end{bmatrix}$	This observation vectors contains the state tracked q and the reference signal q_{ref}. Note that this differs from the previous observation vector (OBS-2), where the state and reference are used to build the tracking error
OBS-4	$\begin{bmatrix} q & e^2 \end{bmatrix}$	This observations vector provides the agent with the values of the state it is tracking q and the tracking error e^2

The graphs in figure 3.6 show the averaged learning performance of the algorithms over the four environments. It can be observed that DSAC has the highest average return. However, its learning behaviour is slower than SAC and TD3, which quickly attain high returns early in training. At the end of the training, the mean episode length for SAC and DSAC is satisfactory, indicating that both algorithms effectively learn to control the aircraft during the entire episode. Similar to the results in Experiment I, TD3 show high variance in the final mean episode length.

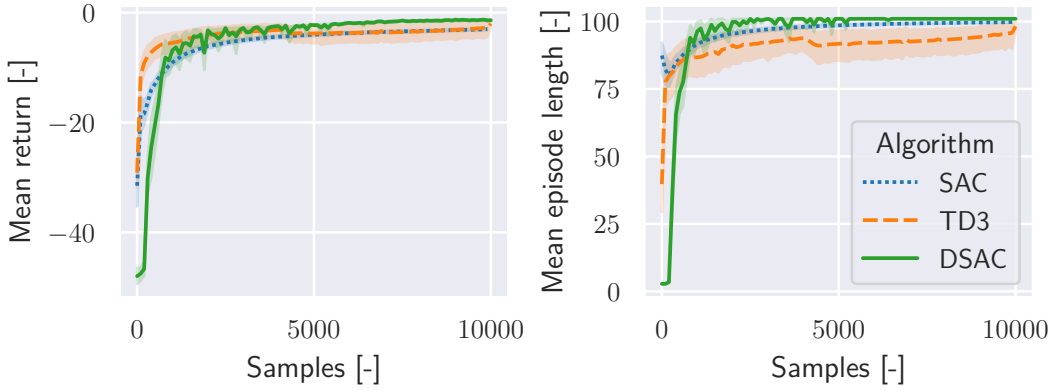


Figure 3.6: Learning performance of SAC, TD3, and DSAC trained on 10,000 samples in Experiment II. The left graph displays the mean return, and the right panel shows the mean episode length. Both graphs average each algorithm's performance over 40 random seeds (ten for each observation vector type).

The learning performance results in figure 3.7 show the mean return of each algorithm for each observation function averaged over the ten random seeds. The return graph shows that TD3 has a high variance in the final return for all four observation vectors. Conversely, DSAC consistently achieves similar maximum returns regardless of the type of observation function. SAC also shows consistency in learning with a low variance but diverges in its returned value for OBS-2.

The tracking performance in figure 3.8 shows the results of each algorithm averaged over ten episodes. For the three algorithms, OBS-3 led to the lowest mean tracking error. While TD3 achieved the lowest

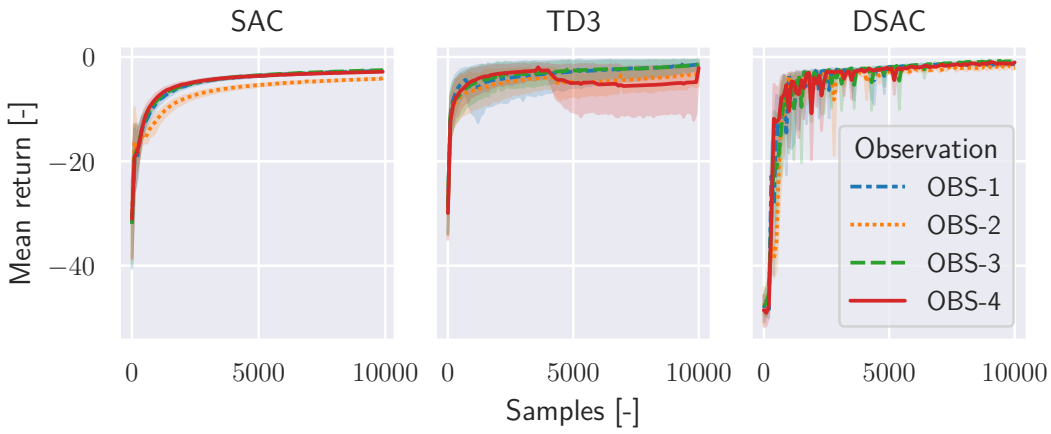


Figure 3.7: Learning performance of SAC, TD3, and DSAC trained on 10,000 samples in Experiment II. The figure displays the mean return of each algorithm for the four different observation vectors, each averaged over ten random seeds.

mean tracking error for all environments, it also showed the highest variance. This suggests that even though TD3 can perform better on average, it is also the most inconsistent regarding its results, potentially leading to undesirable behaviour.

An interesting finding from the results in figure 3.8 is the counterintuitive performance of the algorithms in OBS-1. While the observation vector provides the agent complete information about the environment, it did not lead to the lowest mean tracking error. This may be due to the larger size of the observation vector, which requires a more extended learning period for the agent to learn.

Additionally, the results in figure 3.8 show that providing the agent with the state and reference values results in better tracking performance than providing it with the state and tracking error value. This is evident from OBS-3 outperforming OBS-4 for all algorithms. The reason is that the squared error removes meaning about the tracking performance as the agent can not identify if the error is due to an over- or underestimation. In contrast, OBS-3 provides the agent with the state and reference values, allowing the algorithm to perceive the tracking error better.

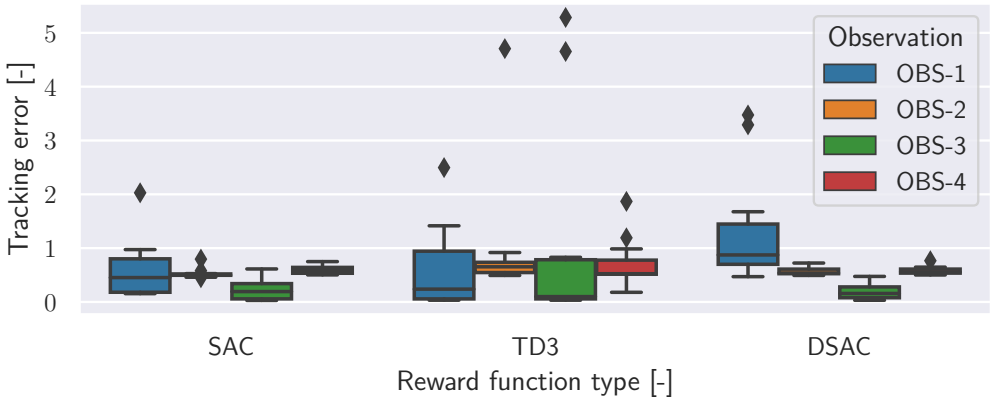


Figure 3.8: Comparison of tracking error for SAC, TD3, and DSAC in Experiment II.

3.5. Experiment III: Comparing Reward Functions

The reward function is what shapes the behaviour the agent should learn. However, no rule can define the best reward function for a specific task. For instance, a poor reward function may not lead the agent to learn the desired behaviour. In addition to that, an algorithm may perform differently depending on the reward function. Because of that, this final experiment compares the three algorithms with three different reward functions, detailed in table 3.4.

Table 3.4: Overview of the experimental setups studied in Experiment III, each comprising a different reward function.

Setup	Reward Function	Description
REW-1	$-(e^2 + \gamma \cdot [\lvert\delta_e\rvert + \lvert\Delta\delta_e\rvert])$	This reward function penalises the agent for tracking error, absolute values of the control action and change in control action. The action terms are scaled with $\gamma = 0.1$, giving more importance to minimising the tracking error.
REW-2	$-(e^2 + \gamma \cdot \lvert\delta_e\rvert)$	This reward function penalises for tracking error and the absolute value of the action with $\gamma = 0.1$.
REW-3	$-(e^2)$	This reward penalises for tracking error.

Comparing the algorithm's learning performance under three different reward functions in figure 3.9 reinforces the results from experiments I and II. SAC and DSAC learn to converge to a maximum return despite the type of reward functions. TD3 learns, on average, an equally high maximum return; however, it has significantly higher variance than the other two algorithms.

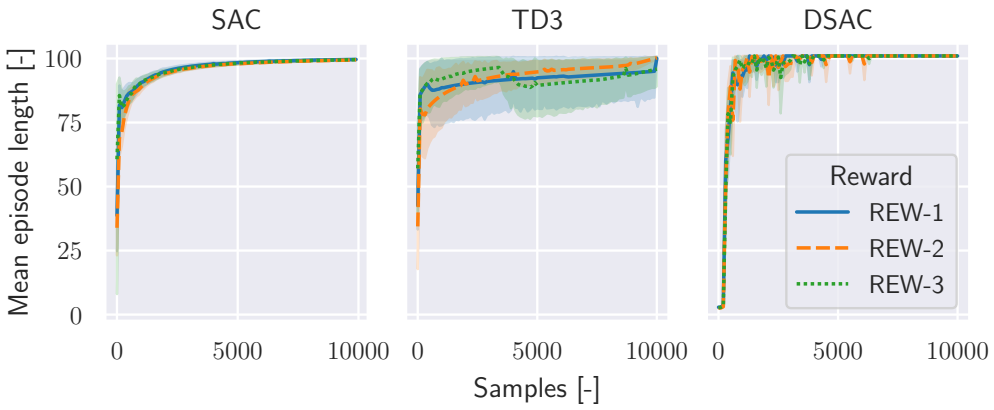


Figure 3.9: Learning performance of SAC, TD3, and DSAC trained on 10,000 samples in Experiment III. The figure displays the mean episode length of each algorithm for the three reward functions, each averaged over ten random seeds.

The tracking performance of the three algorithms is provided in figure 3.10. The reward function REW-2 led to the lowest mean tracking error averaged over ten episodes for the three algorithms. Similarly to the previous experiments, TD3 showed low tracking error. But sufferers from high variance. SAC and DSAC achieved similar results, with SAC showing slightly better tracking performance.

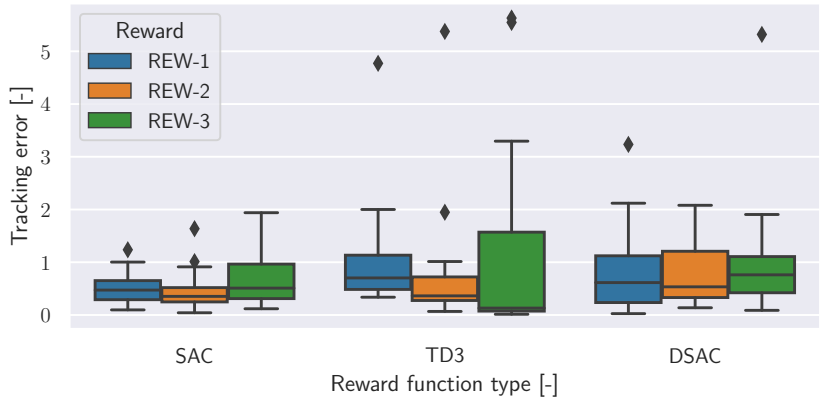


Figure 3.10: Comparison of tracking error for SAC, TD3, and DSAC in Experiment III.

3.6. Concluding Remarks

The preliminary analysis compared the performances of three RL algorithms - SAC, TD3, and DSAC - to determine their possibility of acting as an offline learned agent in a cascaded controller, as demonstrated in Teirlinck [38]. The analysis focused on determining if they can achieve consistent and stable learning while providing adequate tracking performance. The experiments compared the algorithms in three conditions: (I) examining their performance in different tracking tasks, (II) in different observation vectors and (III) in different reward functions.

The results indicate that TD3 can track well on average. However, it has high variance in learning and tracking, with cases where it failed to control the aircraft during the entire episode. On the other hand, SAC and DSAC showed comparable tracking performance, with DSAC exhibiting the most consistent learning with the lowest variance in all three experiments.

Those results conclude that TD3 is ideal for tracking applications where safety is not a priority. The algorithm is superior in average tracking performance but inferior in ensuring consistent results. On the other hand, DSAC showed consistent and robust performance, making it the strongest candidate for building a safe, robust and adaptive offline learned controller. These findings answer Research Question 1.3 and establish DSAC as the algorithm with the highest potential for improving the offline performance on RL applications to flight control systems.

Part III

Additional Results

4

Robustness Analysis of the Hybrid Controller

This chapter contains additional findings concerning the robustness experiments with the hybrid agents. Section 4.1 summarises the hybrid controller definition, including the results from training the offline policy. Section 4.2 shows the process of tuning the IDHP's hyperparameters. Moreover, section 4.3 includes additional results regarding the controller robustness to different reference signals, and section 4.4 the robustness to noise and bias.

4.1. The hybrid controller

Research in FCS using RL has shown the ability of SAC to learn policies offline that are robust and accurate [77]. Additionally, online IDHP has demonstrated its potential for model-free learning and real-time adaptation [57, 76]. Building on this foundation, Teirlinck [38] combined SAC with IDHP to join their strength into a single agent. More recently, Seres [79] showed that DSAC surpasses SAC in learning performance and yields a more conservative policy, which is advantageous for enhancing safety in risky situations. Motivated by these advancements, this study investigates the potential of a hybrid controller that combines DSAC with IDHP.

The main results in this study include a comparison of the developed DSAC-IDHP against the SAC-IDHP, and the offline only SAC and DSAC algorithms. For easier reference to each type of controller, a nomenclature in figure 4.1 is adopted.

	With IDHP	Without IDHP
SAC	SAC-hybrid	SAC-only
DSAC	DSAC-hybrid	DSAC-only

Figure 4.1: The nomenclature adopted to name each RL algorithm used in this research.

For each offline agent (SAC and DSAC), three training processes were performed with three different random seeds. The training consisted of a total of $1\,\mathrm{M}$ steps on episodes with $2\,\mathrm{K}$ steps, tasked with tracking a smooth cosine step signal. The agents are evaluated at the end of each episode. Table 4.1 shows the best performance for each offline agent on its three random seeds.

Table 4.1: Best evaluation performance of SAC and DSAC agents during training. The training was conducted over a total of $10\,\mathrm{M}$ learning steps, partitioned into episodes of $2\,\mathrm{K}$ steps.

Id	Agent	nMAE	Id	Agent	nMAE
SAC-1	SAC	$8.36\,\%$	DSAC-1	DSAC	$7.57\,\%$
SAC-2	SAC	$10.02\,\%$	DSAC-2	DSAC	$11.80\,\%$
SAC-3	SAC	$8.03\,\%$	DSAC-3	DSAC	$7.03\,\%$

4.2. Sensitivity Analysis of IDHP's Hyperparameters

The performance of the IDHP agent dictates the hybrid agent's learning capability. Specific hyperparameters, such as the learning rate, play a crucial role in the learning process; they can significantly affect convergence and, in some instances, even define whether convergence occurs. Therefore, it is critical to examine the effects of different learning rates and discount factor values in the IDHP algorithm, as it provides valuable insights for hyperparameter selection.

The reason for not using the same hyperparameters from prior research is due to the different challenges presented by the hybrid structure. The hybrid modifies the Actor layers by freezing the neurons derived from the offline agent. Furthermore, the hybrid formulation diminishes the influence of stochastic initialisation of weights. Unlike conventional IDHP, which randomly initialises the Actor, the hybrid structure relies on a pre-initialised network. These algorithmic adaptations underline the necessity for parameter tuning in this scenario.

Each test run is assigned one of three labels: ”failed”, ”unacceptable”, or ”acceptable”. A run is labelled ”failed” when the agent cannot maintain control of the aircraft. A run is given an ”unacceptable” label when the agent manages to control the aircraft, but the normalised Mean Absolute Error (nMAE) improvement is less than -30%. Finally, a run is assigned ”acceptable” when the agent successfully controls the aircraft but also keeps the nMAE improvement above -30%.

Understanding the Impact of Learning Rates

The initial hyperparameter study involved examining how the learning rates of the IDHP agent in the SAC and DSAC affected the online learning performance. This process was carried out over 200 individual trials with random seeds. The process involved (1) sampling random values for the Actor and Critic NNs learning rate. (2) Selecting randomly one of the offline trained models from table 4.1 (3) Evaluating the performance of both the hybrid and non-hybrid agents in a tracking task.

The results from the experiments that vary the learning rate of Actor and Critic showed that for different configurations, it is possible to find combinations that are most likely to lead to satisfactory runs. The graph in figure 4.2 and figure 4.3, shows all the runs that were classified as ”acceptable” for the SAC and DSAC, respectively. Those graphs show that the lowest nMAE resulted from runs with Actor and Critic weights that were inversely proportional. i.e. the runs where the value of one parameter was low and the other was high.

In addition to the runs that lead to success, it is essential to investigate the runs that led to failure or unacceptable tracking. Notably, an ”unacceptable” outcome is preferable over a ”failed”. The overview of those runs is shown in figure 4.4 and figure 4.5 for SAC and DSAC, respectively. These figures reveal that an actor learning rate of 0.1 consistently avoided failures for both agents. Consequently, a learning rate of $0.001\,\mathrm{i}$s chosen for the Critic. The reason is that the ”acceptable” runs graph suggests that for an Actor learning rate of 0.1, the Critic learning rate should be lower than 0.1 for greater nMAE improvement.

Understanding the Impact of Discount Factors

In RL algorithms, the discount factor balances the extent to which the agent values immediately versus future rewards. A low discount factor means the agent favours immediate rewards, leading to more exploitation. Two discount factors need tuning for the IDHP algorithm. The first is the policy discount factor, denoted as γ. The second is the incremental model discount factor, designated as γ_{RLS}.

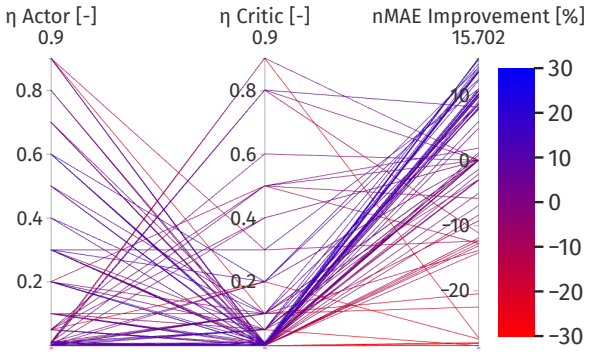


Figure 4.2: Learning rate sensitivity on SAC-hybrid's evaluation performance - Acceptable runs. Impact of the Actor's and Critic's learning rates on online performance improvement.

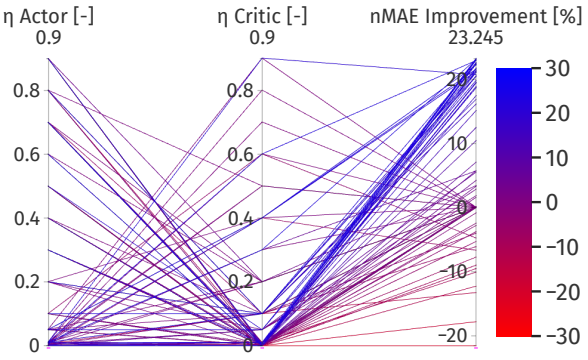


Figure 4.3: Learning rate sensitivity on DSAC-hybrid's evaluation performance - Acceptable runs. Impact of the Actor's and Critic's learning rates on online performance improvement.

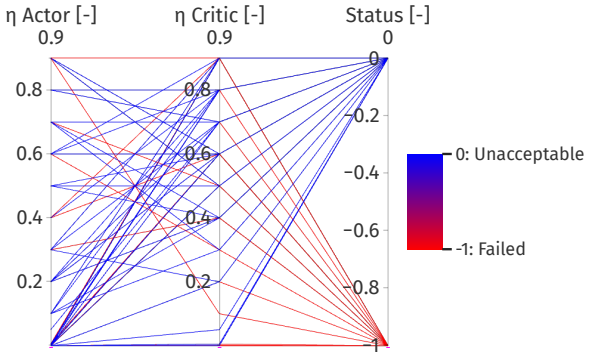


Figure 4.4: Learning rate sensitivity on SAC-hybrid's evaluation performance - Failed and Unacceptable runs.

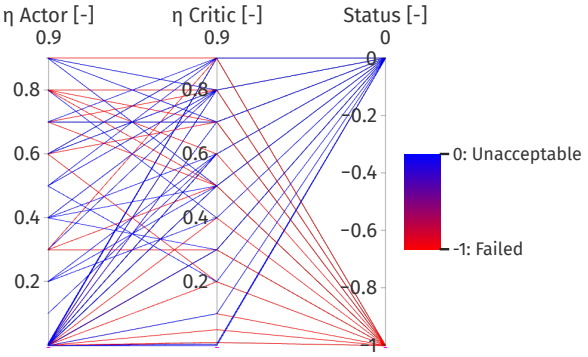


Figure 4.5: Learning rate sensitivity on DSAC-hybrid's evaluation performance - Failed and Unacceptable runs.

Variations in the discount factor values led to the improvements in nMAE shown in figure 4.6 and figure 4.7, for the SAC and DSAC, respectively. Those graphs show only the runs classified as "acceptable". The central insight from this image is that for both models, the discount factor of the incremental model γ_{RLS} is positively correlated with the improvement, indicating higher values are preferred.

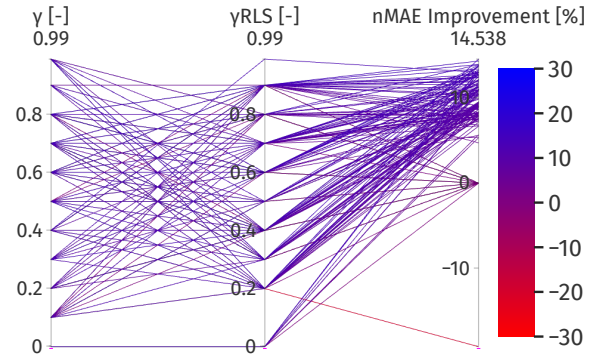


Figure 4.6: Discount factor sensitivity on SAC-hybrid's evaluation performance - Acceptable runs.

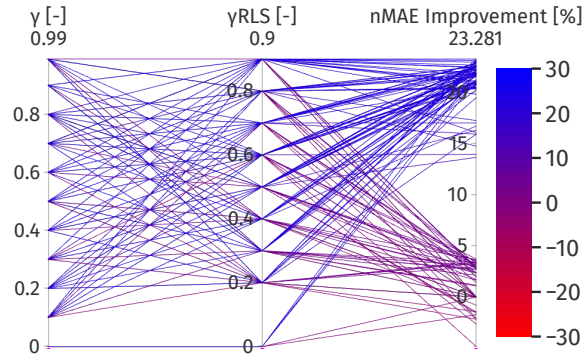


Figure 4.7: Discount factor sensitivity on DSAC-hybrid's evaluation performance - Acceptable runs.

The runs where the hybrid did not lead to adequate improvement in performance are shown in figure 4.8 and figure 4.9 for the SAC and DSAC models, respectively. This image shows that all "unacceptable" runs were the ones where the value of the incremental model discount factor was either higher than 0.9 or lower than 0.1. In this same value range, the SAC algorithm led to two failures, while the DSAC hybrid had no failures. Based on those graphs, the values of both discount factors are set to 0.7, a value where

all runs improved the nMAE while being distant enough from the range that lead to failures.

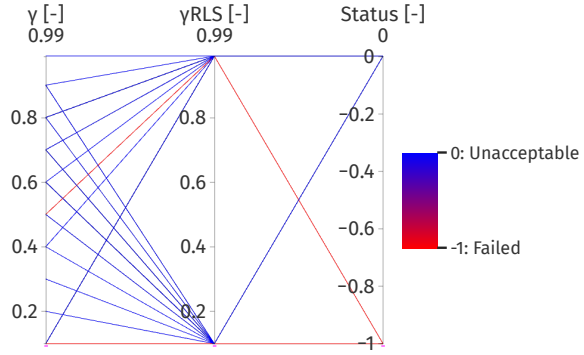


Figure 4.8: Discount factor sensitivity on SAC-hybrid's evaluation performance - Failed and Unacceptable runs.

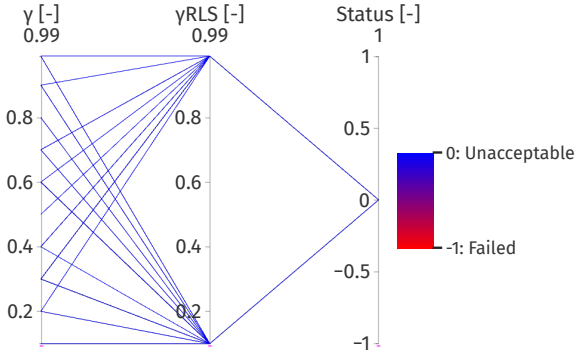


Figure 4.9: Discount factor sensitivity on SAC-hybrid's evaluation performance - Failed and Unacceptable runs.

4.3. Robustness to Varying Reference Signals

The first robustness experiment evaluated the controllers under different reference signals. The results showed that DSAC-hybrid can better adapt to different tasks, followed by the SAC-hybrid. The table 4.2 shows each agent's average nMAE score in the different tasks.

Table 4.2: Evaluation performance of the RL agents on the three different tracking tasks. Performance measured with mean and variance of nMAE tracking error.

	SAC-only	**SAC-hybrid**	**DSAC-only**	**DSAC-hybrid**
Task 1	$20.1 \pm 1.8\%$	$14.9 \pm 1.5\%$	$22.5 \pm 7.0\%$	$11.3 \pm 0.6\%$
Task 2	$11.1 \pm 3.6\%$	$4.5 \pm 1.8\%$	$13.4 \pm 8.8\%$	$2.9 \pm 0.9\%$
Task 3	$16.6 \pm 2.2\%$	$4.1 \pm 0.8\%$	$18.7 \pm 9.8\%$	$3.0 \pm 0.7\%$

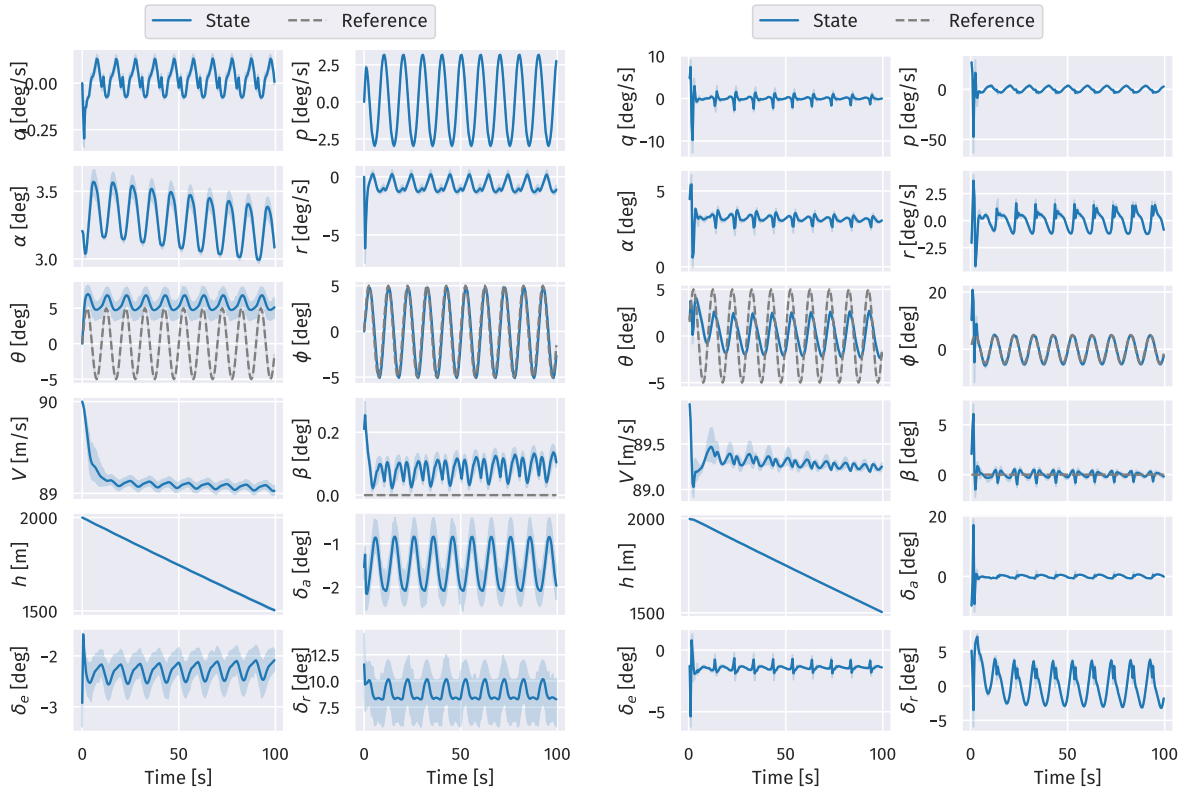
In task 1, the agent tracks sinusoidal pitch and roll angles while maintaining the sideslip angle at zero. Figure 4.10 shows the improvement provided by the DSAC-hybrid over the DSAC-only. A clear improvement is in the sideslip angle, where the hybrid successfully converge to zero. For the pitch angle, it improves the value of the symmetry line of the sinusoidal signal. However, the magnitude could be optimised further.

In Task 2, the agent should maintain constant attitude angles throughout the episode. The tracking performance of the DSAC-only and DSAC-hybrid is shown in figure 4.11. The DSAC-only does not hold the pitch angle adequately. On the other hand, the DSAC-hybrid effectively maintains the pitch angle and enhances the other states' tracking accuracy.

In task 3, the agents must stabilise the pitch and sideslip angles while the roll angle follows a pseudo-random sinusoidal. Figure 4.12 reveals that DSAC-only adequately tacks the roll angle but fails to maintain pitch and sideslip angles at zero. In contrast, the DSAC-hybrid accurately tracks the tree states. Notably, in the initial seconds, the hybrid performs poorly in tracking the roll angle. However, this is an intrinsic behaviour of IDHP agents, which shows some oscillations before convergence.

4.4. Robustness to Noise and Bias

The second experiment evaluated the system's robustness lifts the assumption of ideal sensors by incorporating noise and bias into the observations of angular rates p, q, and r. The values for noise and bias are sampled from the normal distribution $\mathcal{N}(\mu = 3 \cdot 10^{-5}, \sigma = 4 \cdot 10^{-7})$, which were determined by Grondman et al. [82]. Only those three states were affected, as they are the only ones shared by both observation functions of offline algorithms and IDHP. A summary of the experiment under the influence of noise and bias is presented in table 4.3.



(a) DSAC-only evaluation performance.

(b) DSAC-hybrid evaluation performance.

Figure 4.10: Aircraft states during evaluation on tracking Task 1.

Table 4.3: Evaluation performance of the RL agents on Task 3 with sensor noise and bias. Performance measured with mean and variance of nMAE tracking error.

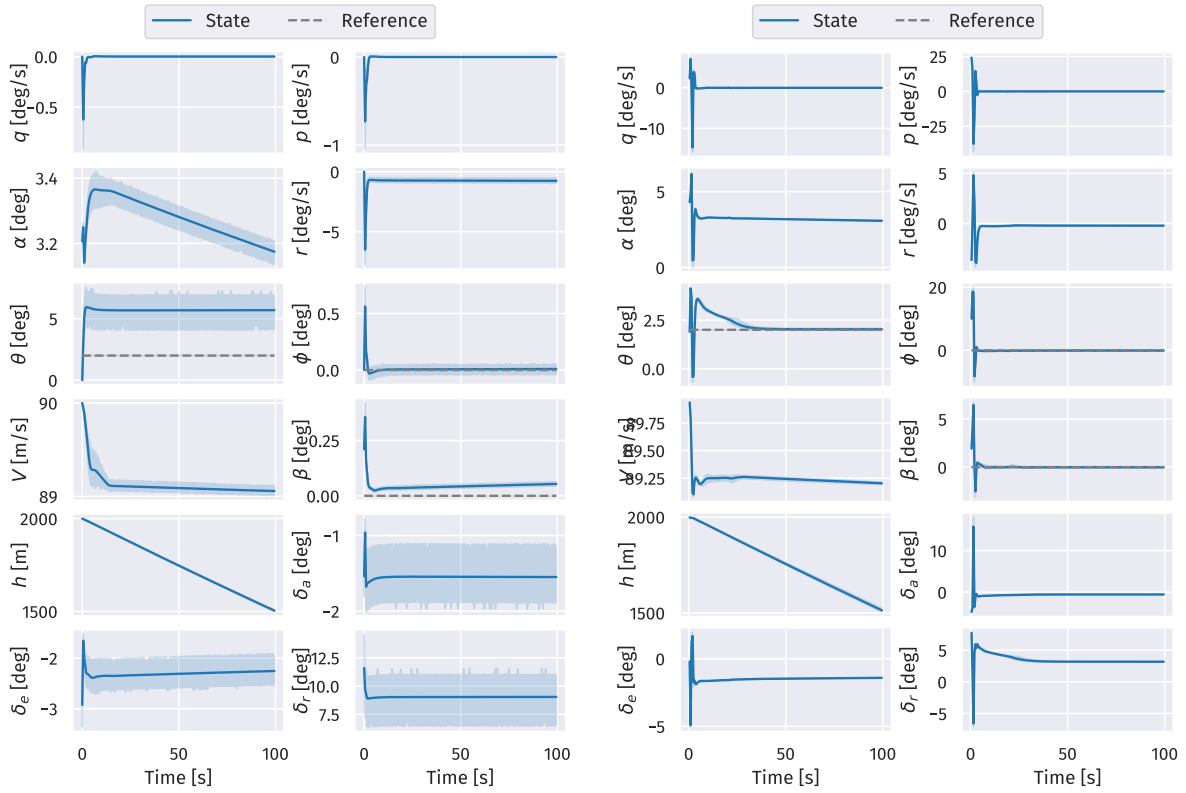
	Non-hybrid	**Hybrid**	**Improvement**
SAC	$16.6 \pm 2.2\%$	$4.0 \pm 0.9\%$	$12.6 \pm 2.7\%$
DSAC	$18.7 \pm 9.8\%$	$2.8 \pm 0.5\%$	$15.9 \pm 9.5\%$

The evaluation of the tracking performance of the two hybrid agents with biased and noisy sensors can be seen in figure 4.13. Both algorithms demonstrate comparable performance when subjected to noise. However, during the initial seconds, the SAC-hybrid shows to be noisier, with more pronounced oscillation. Nonetheless, the hybrid architecture made both agents converge to a better policy than the offline-only.

4.5. Concluding Remarks

This chapter determined the set of hyperparameters for the IDHP agent that is ideal for the hybrid structure. This step was fundamental as the modifications on the architecture of the IDHP's Actor-network require new learning rates and discount factor values. The newly determined values for those parameters were shown to be robust, as they were consistent in learning and did not lead to any failures.

The robustness analysis of the agent's capability of tracking signals that are different from the one it was trained shows that both SAC-hybrid and DSAC-hybrid outperform the SAC-only and DSAC-only, respectively. Between the two hybrids, the DSAC leads to better tracking performance across all tasks.

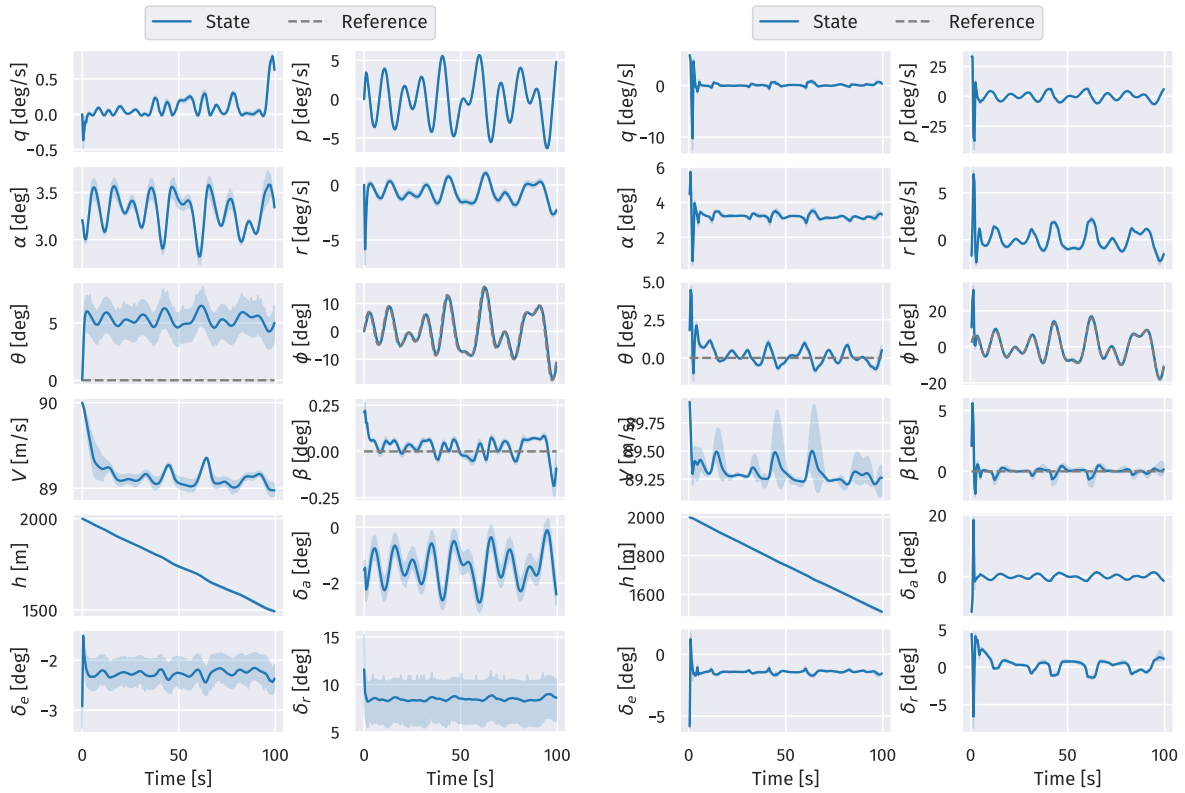


(a) DSAC-only evaluation performance.

(b) DSAC-hybrid evaluation performance.

Figure 4.11: Aircraft states during evaluation on tracking Task 2.

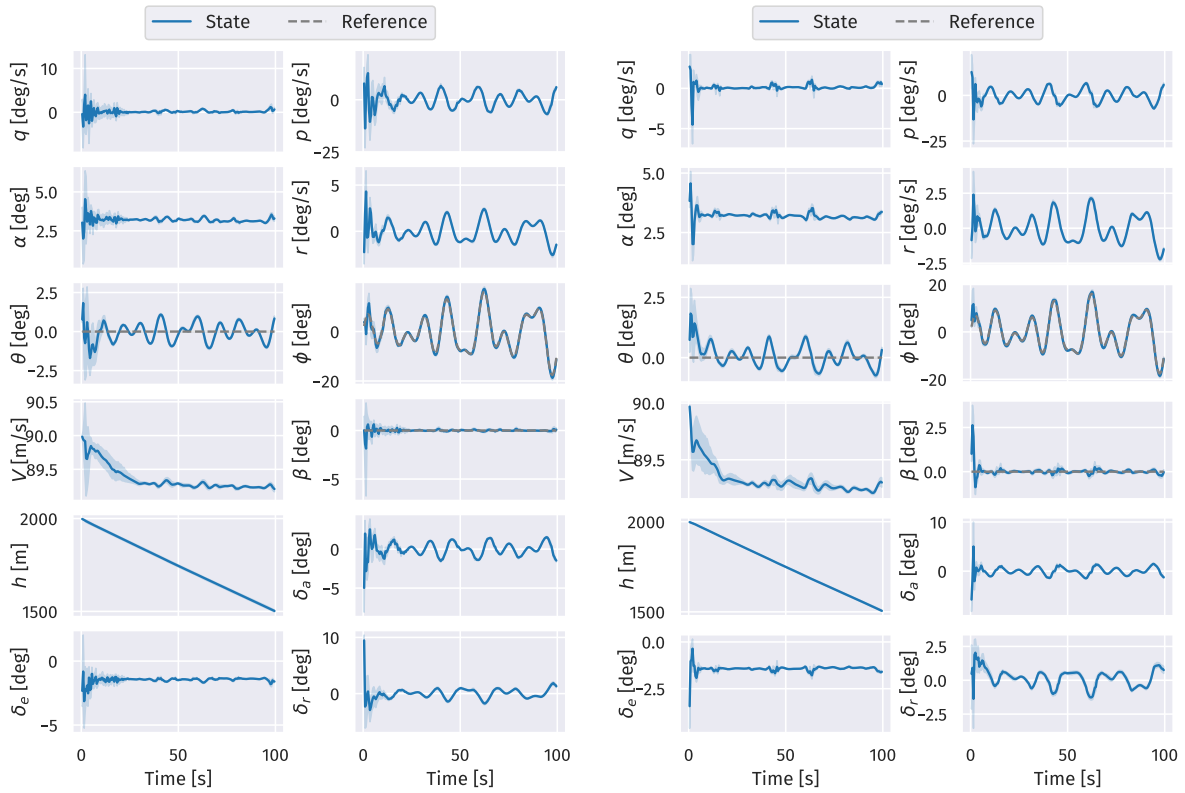
When noise and bias are added to the observation vector of the agents, the hybrid agents also improve the performance of their respective offline counterparts. One more time, the DSAC leads to a hybrid that tracks the reference signal more closely on average and displays a lower variance across random seeds. Besides that, the DSAC-hybrid also leads to a more significant improvement in the nMAE than the SAC-hybrid.



(a) DSAC-only evaluation performance.

(b) DSAC-hybrid evaluation performance with sensor noise and bias.

Figure 4.12: Aircraft states during evaluation.



(a) SAC-hybrid performance with sensor noise and bias.

(b) DSAC-hybrid performance with sensor noise and bias.

Figure 4.13: Citation states with DSAC agents tracking Task 3 including sensor noise and bias.

5

Fault-Tolerance and Safety

RL-based controllers should be able to provide the same robustness as conventional control systems before they can be seen in real systems. Nevertheless, those controllers may bring the novelty of allowing online fault tolerance to conditions not previously conceived. That, combined with the robustness, can allow those controllers to increase the safety of flight control systems; because of that, this section analysis the performance of the hybrid controllers when a fault occurs.

5.1. Operation Under Reduced Control Effectiveness

The first set of experiments to check the performance of hybrid controllers during failure involves reducing the effectiveness of one control surface. For each considered faulty scenario, the agents are tested using five random seeds, and they must follow a pseudo-random reference signal for the roll angle while regulating the pitch and sideslip angle. This section shows additional experiment results by including the graphs with the runs of the SAC-hybrid and SAC-only.

5.1.1. Reduced Elevator Effectiveness

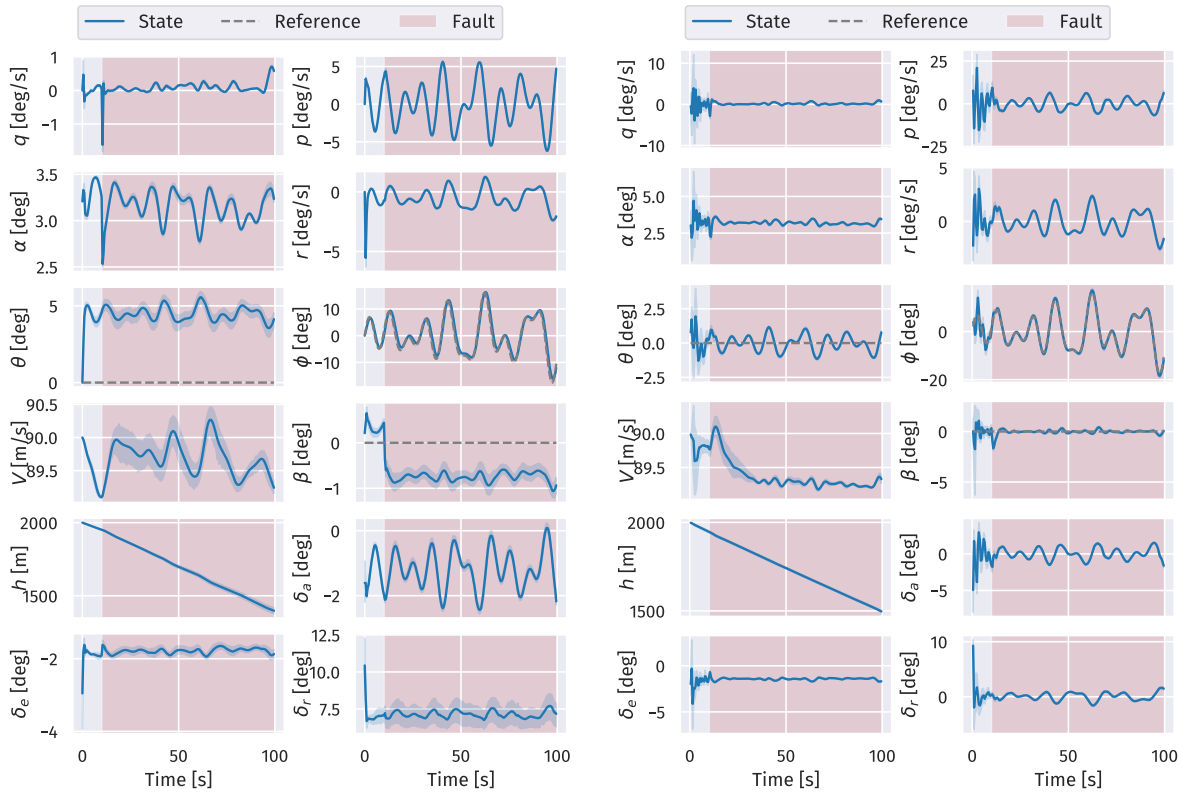
In the first fault scenario, the elevator's effectiveness is reduced by $70\,\%$. This reduction starts at the $10\,\mathrm{s}$ mark. It is desired that the agents can adapt to this change by compensating for the loss and continuing to track the reference signal effectively. Figure 5.1 presents the aircraft states when testing both the SAC-hybrid and SAC-only configurations. The figure reveals that the SAC-hybrid controller can recover from the fault, which is not the case for the SAC-only configuration.

5.1.2. Reduced Aileron Effectiveness

In the second experiment, aileron effectiveness is curtailed by $90\,\%$. This fault is enforced starting at the $10\,\mathrm{s}$ mark. Figure 5.2 illustrates the aircraft states when evaluating both the SAC-hybrid and SAC-only configurations. It is evident from the figure that the SAC-hybrid configuration is capable of facilitating recovery from the fault, in contrast to the SAC-only configuration.

5.2. Concluding Remarks

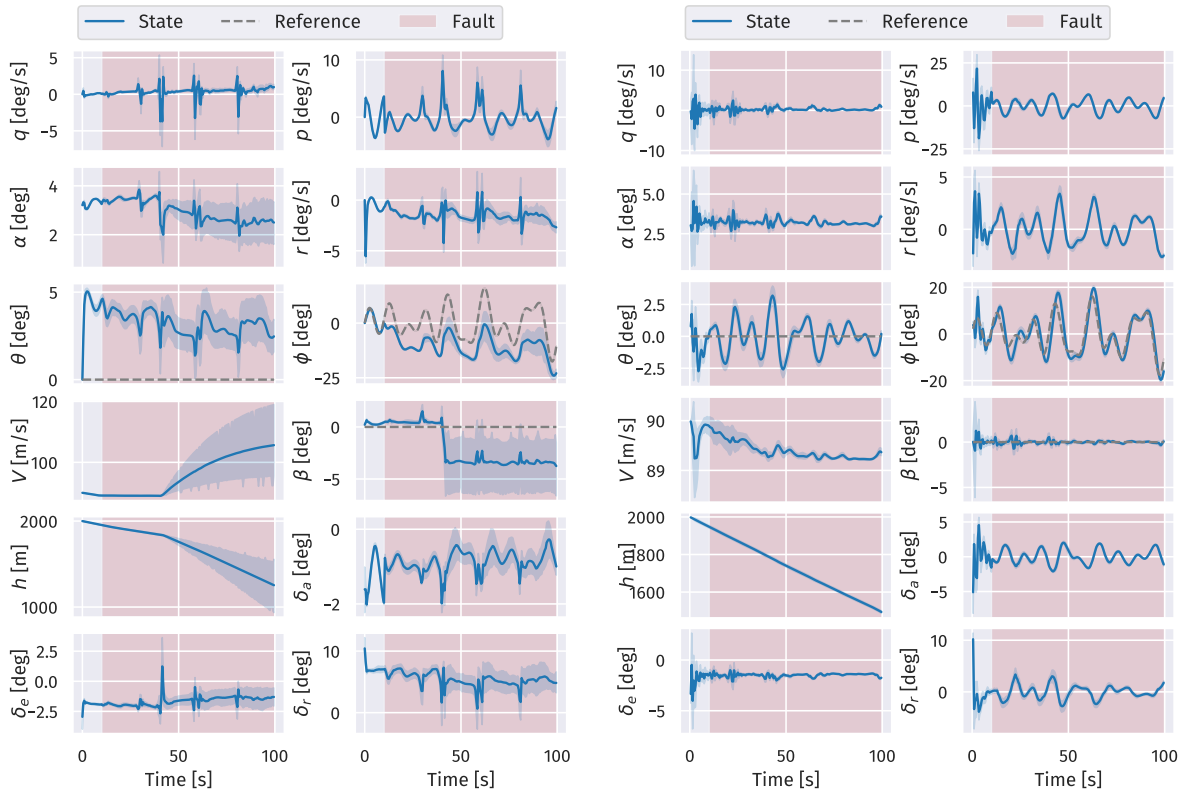
The experiments comparing the performance of different algorithms under reduced elevator effectiveness highlight the benefits of employing a hybrid algorithm instead of solely relying on an offline policy for control. The offline policies did not yield satisfactory performance under scenarios with reduced elevator or aileron effectiveness. In contrast, the hybrid configurations demonstrated an ability to continue tracking the reference signal and recover from the fault. Notably, the DSAC configuration has superior performance in achieving the lowest nMAE. Moreover, the SAC-hybrid agent also leads to better performance under fault than the offline-only policies.



(a) SAC-only's evaluation performance.

(b) SAC-hybrid's evaluation performance.

Figure 5.1: Citation states when evaluating the DSAC agents in a task with reduced elevator effectiveness. Elevator reduced by $70\,\%$ at time $10\,\mathrm{s}$.



(a) SAC-only's evaluation performance.

(b) SAC-hybrid's evaluation performance.

Figure 5.2: Citation states when evaluating the DSAC agents in a task with reduced aileron effectiveness. Aileron reduced by $90\,\%$ at time $10\,\mathrm{s}$.

6

Verification and Validation

The developed aircraft environment, algorithms, and controller architecture must go through a Verification and Validation process. This ensures the reliability and trustworthiness of the research methodology and results. This chapter is organised as follows: Section 6.1 outlines the verification steps, section 6.2 discusses the validation results, and section 6.3 presents the reliability analysis of the research findings.

6.1. Verification

This section provides the results of verifying the RL algorithms, the Citation environment, and the flight controller.

6.1.1. Verification of RL algorithms

An essential part of the verification process is to evaluate the learning performance of the SAC, DSAC, and IDHP. These algorithms should learn to perform the assigned tasks by converging to a maximum return value. The return values for SAC and DSAC during the training steps are shown in figure 6.1. This curve's convergence to a high return value confirm that these agents have learned to perform the tasks. Also, the shape of the learning curve is typical for these RL algorithms, similar to the behaviour seen in the implementations presented by Teirlinck [38] with SAC and Seres [79] with DSAC.

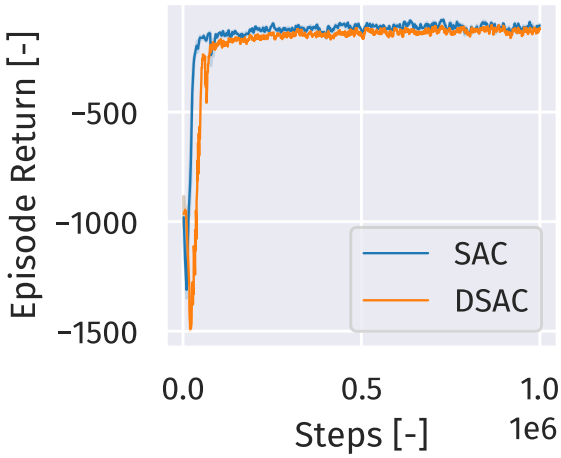


Figure 6.1: Learning curves for the SAC and DSAC algorithms. Each curve averages the performance of the three independent runs of each algorithm. The training was conducted over a total of $10\,\mathrm{M}$ learning steps, partitioned into episodes of $2\,\mathrm{K}$ steps.

In addition to analysing convergence, the code of the developed algorithms is covered by unit and

system tests. These tests ensure that the code is implemented correctly and free of errors. The SAC algorithm has a line coverage of $99\,\%$, DSAC has $97\,\%$, and IDHP has $70\,\%$.

6.1.2. Verification of Citation Environment

The Citation RL environment uses the DASMAT software. To use this aircraft model within Python, the programming language for which the controller was developed, the model had to be converted from MATLAB's Simulink model into a Python module. Verifying this environment involved comparing the responses to a rectangular pulse input in MATLAB and Python. This comparison confirmed that the Citation model's Python behaviour matches its original MATLAB implementation. Figure 6.2 shows this verification as both models result in the same response.

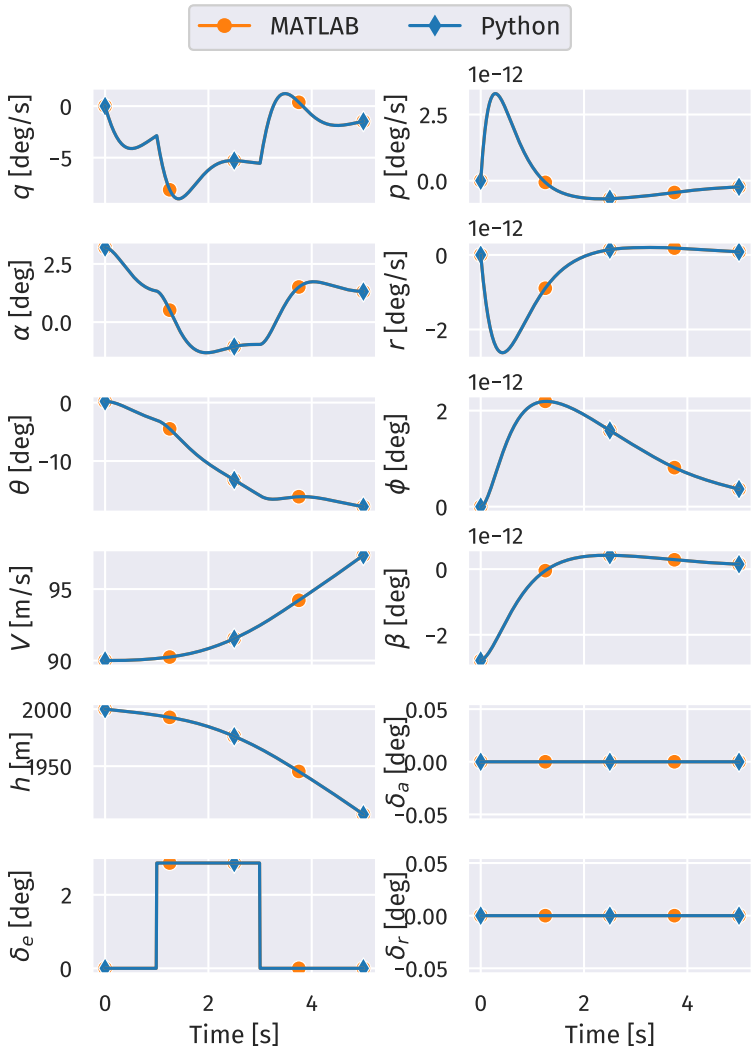


Figure 6.2: verification of the consistency of the Citation model in MATLAB and Python when responding to a rectangular pulse.

Additionally, the Python implementation of the Citation environment is also covered with unit tests to prevent unintended behaviour and code bugs. In total, the Citation environment contains a $80\,\%$ line coverage.

6.2. Validation

This section elaborates on the work to validate the RL algorithms, the Citation environment, and the flight controller, ensuring that these components accurately represent the real-world behaviour they are intended to model.

6.2.1. Validation of RL algorithms

The validation of the RL algorithms was executed by comparing the training and evaluation performance of the developed algorithms with those from Raffin et al. [80] implementations in a benchmark RL environment. Specifically, the benchmark environment used for this purpose was Gym's Brockman et al. [83] Pendulum environment, where the agent is tasked with learning how to maintain the pendulum vertically aligned.

The algorithms' learning performance is illustrated in figure 6.3. This figure shows that the developed SAC and DSAC converge to a similar range of values as the SAC from Raffin et al. [80]. Moreover, it is essential to note that the developed RL algorithms not only converge but also exhibit competitiveness to the compared implementation, evidenced by their earlier convergence, suggesting enhanced efficiency in learning.

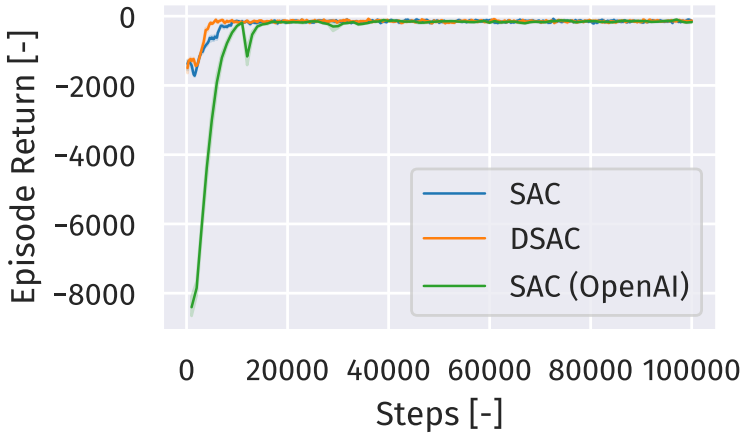


Figure 6.3: Learning curves for the developed SAC and DSAC algorithms, and SAC from Raffin et al. [80]. Each curve averages the performance of the five independent runs of each algorithm. The training was conducted over a total of $100\,\mathrm{k}$ learning steps, partitioned into episodes of $1\,\mathrm{K}$ steps in the Pendulum environment from Brockman et al. [83].

6.2.2. Validation of Citation Environment

The Citation environment is based on the DASMAT software, which was previously validated in the study by van den Hoek, de Visser, and Pool [74]. In that study, the performance of the Citation model was assessed by comparing it with actual flight data from the same type of aircraft. This comparison affirmed the fidelity of the Citation model to real-world flight dynamics.

Given that the Citation model has already been validated, this research did not require revalidation. Instead, the focus was placed on ensuring the model's Python implementation remained faithful to its original form. As outlined in the Verification section, this involved confirming that the behaviour and responses of the Python implementation are consistent with those of the original MATLAB implementation.

6.3. Reliability Analysis

An issue on RL research is that training is computationally expensive, which reduces the number of experiments feasible within a time of research. That is why this research only trains three of each offline agent. Consequently, it becomes difficult to confirm that the results found are trustable and not only an outlier. Therefore, a reliability analysis is essential to confirm to what extent the same results would be obtained in the case of different random seeds.

A challenge regarding the data collected is the difference in the size of the evaluation results of the hybrid and non-hybrid algorithms. The hybrid algorithm has some stochastic behaviour in its critic, which it learns to improve during flight. With the hybrid, running different evaluation runs and gathering various results across random seeds is possible. Contrary to that, the non-hybrid agents are deterministic and perform in evaluation the same independent of the random seed. Therefore, for each evaluation

in this paper, the hybrid was tested with five random seeds, giving more insight into the variation of its results. This difference in data collection is one more reason for reliability analysis.

The reliability analysis in this report uses the nMAE from all the runs throughout all experiments discussed in chapter 4 and chapter 5. The tools used to perform the analysis are determined from the recommendations in [84]. This paper defines three methods to assess the influences of randomness in RL even when only a few runs are available. (1) Stratified bootstrap confidence intervals. (2) Performance profiles. (3) Aggregated metrics.

6.3.1. Reliability Through Stratified Bootstrap Confidence Intervals

The stratified bootstrap confidence intervals is a method for aggregating the nMAE metric from different experiments to predict how likely the same results would be observed if the experiments were repeated with different random seeds. The advantage of this method is that it aggregates normalised scores across experiments, which allows a better understanding of the statistical uncertainty of the results.

The stratified bootstrap method consists of random sampling, with replacement, a set of nMAE values from each experiment. The sample size is proportional to the number of evaluations in that experiment. Then it is possible to determine a distribution of those samples' aggregate scores. Because the draws are proportional to the size, it makes it more fair for the comparison between the hybrid and non-hybrids.

6.3.2. Reliability Through Performance profile

A performance profile graph represents the nMAE distribution across all runs. The graph uses the data from the stratified bootstrap confidence intervals to show a cumulative distribution of the model's score. The results for this research's agents are shown in figure 6.4. The y values in the curve represent the fraction of runs that lead to an nMAE lower than the threshold in the x axis. Therefore, the higher the curve concerning the others, the better.

In the graph, it is possible to note that the hybrids have similar performance when compared to each other and the non-hybrids when compared to each other. Because the DSAC-hybrid curve is strictly above all others, this agent is statistically dominant. In other words, it is an agent for which runs are more likely to have lower nMAE than the others. Between the non-hybrid algorithms, SAC only performs better than DSAC. Besides that, it is possible to see that those agents lead to higher confidence intervals, which is due to the reason that they have fewer runs than the hybrid agents.

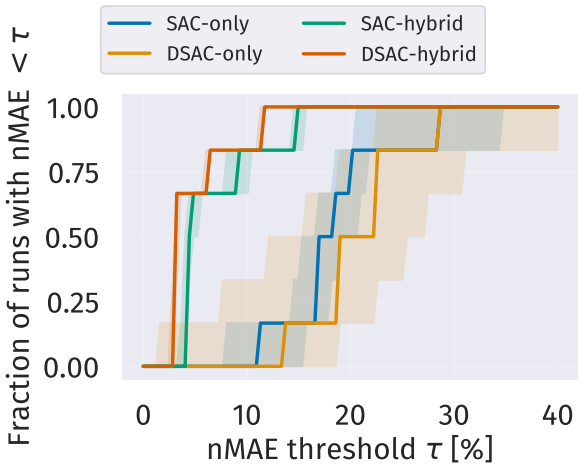


Figure 6.4: Performance profile curve comparing the RL agents across all experiment tasks.

6.3.3. Reliability of Agents Across Aggregated Metrics

While the performance profile allows a comparative performance between the models at a glance, it still does not show the entire picture of the models' statistical Uncertainties. For instance, the performance

profile curves have intersection points, whereas the models have similar performance. For this reason, this section looks into the aggregate metrics of the Stratified Bootstrap Confidence Intervals.

For the first analysis on aggregated metrics, three metrics are considered. The first is the median, which shows each agent's middle nMAE value. The second is the IQM, an alternative to the median. The IQM discards the $25\,\mathrm{percent}$ top and bottom values of the scores. Therefore, it is a version of the median that removes outliers. The final metric is the optimality gap, which is a measure of how far an algorithm is from the perfect performance (which would be $0\,\%$ as the perfect nMAE). Essentially, this metric shows how much more error the algorithms have when compared to the ideal scenario.

The graph in figure 6.5 shows the performance of the algorithms on the aggregated metrics. In those graphs, lower median, IQM, and optimality gap are preferred, as they correspond to a lower nMAE. This graph shows one more time the statistical dominance of DSAC-hybrid over the other algorithms, with SAC-hybrid following it closely. Those two agents are also the ones with the lower optimality gap, showing that they are the ones that get closer to the ideal controller.

This picture also shows a clearer understanding of the comparison between the non-hybrid agents. SAC-only achieves lower nMAE. While DSAC-only has a comparable average performance, it has a significantly higher variance in evaluation performance.

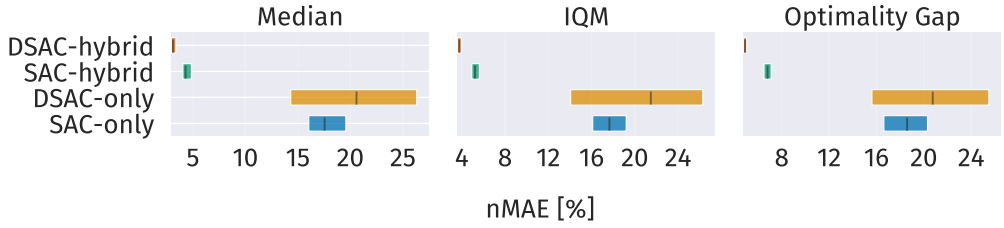


Figure 6.5: Aggregate metrics across RL algorithms: Median, IQM, mean, and optimality gap.

Another metric is the probability of improvement, which shows how likely an algorithm is to improve its performance compared to others. Therefore, how probable is the performance of algorithm X higher than algorithm Y. The way this metric is calculated consists of a pair-wise average of the times an algorithm performed better than others.

The probability of improvement analysis results for the studied algorithms is shown in figure 6.6. This image shows that both hybrid algorithms have a higher than $80\,\%$ probability of increasing the performance of the offline-only agents. Furthermore, the DSAC-hybrid also has a high probability of improving SAC-hybrid. Between the offline-only agents, SAC has more than $60\,\%$ of a chance of improving DSAC. However, this is together with a high variance, confirming that the offline algorithms' performance is comparable.

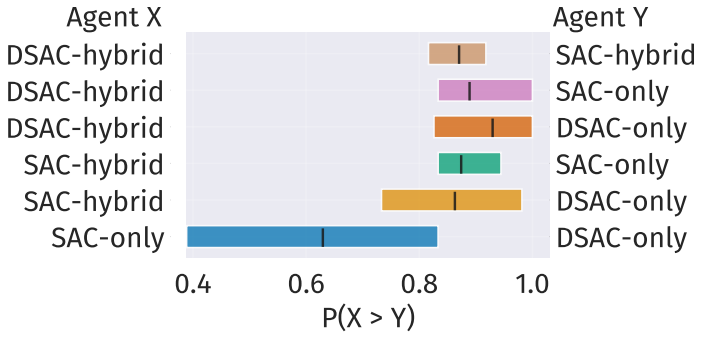


Figure 6.6: Probability improvement of the studied algorithms.

6.4. Work Reproducibility

Ensuring that others can reproduce the work in this study is very important. This helps in checking that the results are reliable. To help with this, the seeds for the experiments are saved in the experiment files. These files are in a repository of this thesis. This means that getting the same results as in this study only requires running those files. This step helps in making the study more open and trustworthy.

6.5. Concluding Remarks

This chapter showed the steps for verifying and validating the built algorithms, environment, and controller. Therefore, assuring they are correctly implemented and worked as they should. For example, the results are consistent with other research. When using the developed algorithms in a benchmark environment, they showed similar or even better performance compared to the state-of-the-art algorithms' publicly available implementations.

Besides that, this chapter looked into how reliable the results are. This was needed to provide a statistical overview of the comparisons between the algorithms, as the low number of trained models requires a further investigation of the effects of randomness. This analysis showed that the hybrid agents outperform the non-hybrid ones. Moreover, it showed that the DSAC-hybrid agent was likely to perform better than the other algorithms more than $80\,\%$ of the time.

Part IV

Closure

7

Reflection on Research Questions

This research introduced a novel hybrid Reinforcement Learning (RL) flight controller for robust and fault-tolerant attitude control. The controller combines Distributional Soft-Actor critic (DSAC) and Incremental Dual Heuristic Programming (IDHP) into a single algorithm. The DSAC is trained offline to learn a robust policy. Then this policy is used to initialise the weights of the IDHP agent, which can provide adaptability online. The joining of both methods results in a robust and safe controller for autonomous control.

The study began by answering Question 1 through a literature review. Here, it laid out the RL concepts and main algorithms by responding to Question 1.1 and Question 1.2. This study found that a hybrid controller using Soft-Actor Critic (SAC) and IDHP showed improved performance over offline learning only. Additionally, DSAC research has shown that it shows better learning properties than SAC and leads to safer policies. This opened the question of whether combining DSAC with IDHP in a hybrid setup could lead to better results, addressing Question 1.3.

Research Question 1

Question 1. What is the state-of-the-art in reinforcement learning for fault-tolerant flight control systems?

- **Question 1.1.** What are the key concepts and principles of reinforcement learning, and how are they relevant to control systems design?
- **Question 1.2.** What are the current state-of-the-art algorithms in reinforcement learning, and to what extent do they offer possibilities to fault-tolerant flight control?
- **Question 1.3.** What are the gaps and opportunities for future research in reinforcement learning for flight control systems?

After looking into literature, an initial analysis studied how changing the offline algorithm might improve performance. It was found that a hybrid can take the strengths of both offline and online algorithms. However, it also became clear that making a hybrid would introduce more complexity to the RL algorithm, answering Question 2.1. The study also found that the best way to create a hybrid is by building an Actor NN that includes layers from offline and online algorithms, answering Question 2.2.

Research Question 2

Question 2. How can hybrid offline-online reinforcement learning techniques be integrated to develop a fault-tolerant flight control system?

Question 2.1. What are the advantages and limitations of a hybrid offline-online reinforcement learning approach?

Question 2.2. How can offline and online reinforcement learning techniques combine to leverage their strengths?

Further analysis confirmed that the DSAC algorithm improved the learning performance when compared to SAC. The results showed that DSAC was more reliable in reaching the best return, leading to safer policies that avoid risky situations. The main downside with DSAC was that it needed more clock time to train. Those results answered Question 3.1. For a more robust, reliable, and fault-tolerant controller, it is envisioned building a hybrid controller that merges the distributional algorithm to IDHP, which answer Question 3.2

Research Question 3

Question 3. How can distributional reinforcement learning methods improve the performance of fault-tolerant flight control systems?

Question 3.1. What are the principles and advantages of distributional reinforcement learning, and how does it differ from traditional reinforcement learning methods?

Question 3.2. How can distributional reinforcement learning be used to improve the performance of fault-tolerant flight control systems?

The developed DSAC-hybrid controller showed superior performance than SAC-hybrid, and the offline-only DSAC and SAC. The performance was measured using the nMAE of the tracking performance error of each algorithm. A reliability analysis was conducted to ensure the results were reliable. Altogether, those findings answered Question 4.1.

The hybrid controllers showed they improved robustness and higher tracking performance compared to SAC-only and DSAC-only. Besides that, they also showed superior fault tolerance, where they managed to track the attitude even under reduced control effectiveness. Additionally, DSAC-hybrid outperforms in those same aspects the SAC-hybrid. Those findings answer Question 4.2 and Question 4.3.

Research Question 4

Question 4. How effective is the resulting fault-tolerant flight control system at achieving robustness, adaptability, and safety in real-world flight scenarios?

Question 4.1. What are the most relevant performance metrics for evaluating the system's performance in different flight conditions?

Question 4.2. How does the fault-tolerant control system compare to traditional controllers regarding stability, performance, and safety?

Question 4.3. How well does the system handle different types of faults and risk situations, and what are the limitations?

8

Conclusion

Reinforcement Learning (RL) holds great promise in developing novel autonomous flight control systems that are both intelligent and safe. Offline algorithms, such as Soft-Actor Critic (SAC) and Distributional Soft-Actor critic (DSAC), have demonstrated the capability to derive control strategies that are both robust and efficient for aircraft control. However, these algorithms exhibit limitations in terms of adaptability during flight. In contrast, online algorithms like IDHP facilitate real-time adaptation but come with concerns regarding robustness and the risks associated with learning during flight. Previous research has indicated that a combination of SAC and IDHP can effectively combine the strengths of both algorithms. Notably, DSAC has shown potential in generating safer policies than those derived from SAC. Thus, this research was focused towards the development and evaluation of a hybrid algorithm that integrates DSAC and IDHP, referred to as DSAC-hybrid.

The robustness experiments of this research evaluated the SAC and DSAC algorithms in a hybrid and non-hybrid controller architecture in two scenarios. First, tracking attitude reference signals that are different from the ones used during training. Second, tracking a reference signal while the aircraft sensors contained noise and bias. The DSAC-hybrid outperformed the SAC-hybrid in both scenarios. Moreover, both hybrid controllers exhibited superior performance compared to their offline-only counterparts.

When considering fault tolerance, the controllers were evaluated in situations where the effectiveness of the control surface was decreased. Both SAC and DSAC struggled to cope with reductions in elevator and aileron effectiveness. Conversely, the DSAC-hybrid proved aptitude at controlling the aircraft and accurately tracking the reference signal even under these adverse conditions. Although the SAC-hybrid also yielded improved performance compared to the offline algorithms, it had a lower performance than DSAC-hybrid.

The main conclusion from those experiments is the significant potential in hybrid controllers (either with DSAC or SAC) for ensuring safe flight control. They maintain the robustness of offline learning and enhance performance through online adaptiveness. Moreover, employing distributional RL to generate an initial policy for a hybrid agent proved advantageous, as DSAC consistently surpassed other algorithms in all experiments. This marks a significant step towards intelligent, robust, and adaptive flight control systems. The DSAC-hybrid demonstrates the capability to deliver safety, accuracy, and reliability, which are critical components for the future of autonomous control systems.

9

Recommendations

This chapter outlines possible paths for future research to deepen our understanding of RL-based flight control systems.

Re-use of critic network
In both DSAC-hybrid and SAC-hybrid controllers discussed in this work, only the Actor layers from the offline agent are transferred to the online agent. This means that all the valuable information gathered during offline learning of the value function is not used when the system is online. This is especially concerning for DSAC, where the critic network is more advanced and understands how to explore the environment to achieve higher rewards safely. Future research should consider finding ways also to use the information from the critic network in the online agent.

Alternative transfer learning
The hybrid approach in this research uses a shared neural network for transferring knowledge from the offline agent to the online agent by freezing the neurons that come from the offline agent. But this transfer could be achieved in many other ways, and some might lead to better results. Future research should investigate different network structures for transfer learning.

Figure 9.1 illustrates several possible architectures for the Actor-Network. The present study adopts the hybrid approach. In the Sequential structure, the output of the offline agent is integrated as one of the inputs for the online agent. The Add topology sums the outputs of both networks; it makes the offline network operates like an excitation function to the online agent. Lastly, the Switch architecture works by alternating between the networks during flight. Under nominal conditions, the control system may employ the offline policy and transition to the online agent in response to escalating errors.

Broadening Safety Analysis
This study looked at safety mainly through the lens of fault tolerance. However, it is also essential to look at safety in situations where high performance is not the primary goal, such as near-stall conditions where it is more important for the agent to recover than to keep tracking a signal. Future research should include a comparison between the hybrid and non-hybrid DSAC approaches to confirm whether those safety characteristics can be transferred to the online agent.

Considering More Complex Controllers
This work focused on the safety-critical control loop and thus only looked at attitude control. To get a complete picture of how these controllers can behave in more advanced conditions, looking at altitude control in a cascade structure is essential. This will provide deeper insights into how RL-based controllers operate in more challenging flight scenarios.

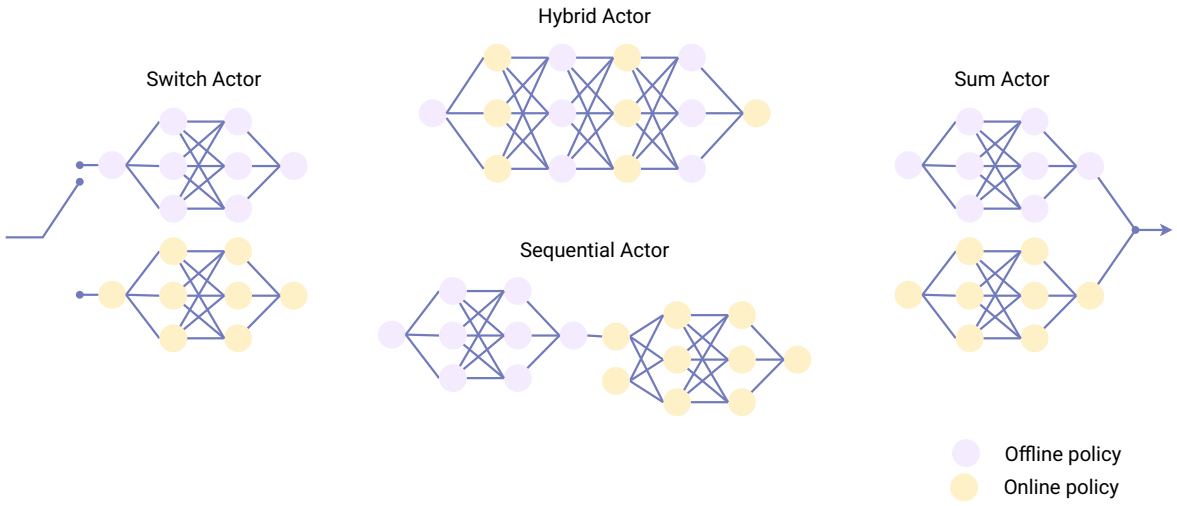


Figure 9.1: Some of the possible transfer learning for the Hybrid's Actor-Network structure.

Part V

Appendices

A

Short Period Linear Time-Invariant System of Cessna Citation

This chapter discusses an Linear Time-Invariant (LTI) short-period approximation of the Cessna Citation aircraft, as provided by Mulder et al. [81]. The equations of motion for the aircraft are represented by equation (A.1). Here, α denotes the angle of attack, q represents the pitch rate, and δ_e signifies the elevator deflection.

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = \underbrace{\begin{bmatrix} z_\alpha & z_q \\ m_\alpha & m_q \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \alpha \\ q \end{bmatrix} + \underbrace{\begin{bmatrix} z_{\delta_e} \\ m_{\delta_e} \end{bmatrix}}_{\mathbf{B}} \delta_e \tag{A.1}$$

The state matrix **A** and the input matrix **B** in equation (A.1) are composed of simplified notations to represent the linearised dynamics concisely. These matrices are further elaborated in equation (A.2) and equation (A.3).

$$\mathbf{A} = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{Z_\alpha}}{2\mu_c - C_{Z_{\dot{\alpha}}}} & \frac{V}{\bar{c}} \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V}{\bar{c}} \frac{C_{m_\alpha} + C_{Z_\alpha} \frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} & \frac{V}{\bar{c}} \frac{C_{m_q} + C_{m_{\dot{\alpha}}} \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix} \tag{A.2}$$

$$\mathbf{B} = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{z_{\delta_e}}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V}{\bar{c}} \frac{C_{m_{\delta_e}} + C_{Z_{\delta_e}} \frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix} \tag{A.3}$$

These control and stability derivatives in those equations are essential in modelling the aircraft's behaviour within the state-space system. The values of these derivatives for the Cessna 550 Citation II aircraft are extracted from Mulder et al. [81] and are shown in table A.1.

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = \underbrace{\begin{bmatrix} -0.746 & 0.974 \\ -1.462 & -1.540 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \alpha \\ q \end{bmatrix} + \underbrace{\begin{bmatrix} -0.090 \\ -6.857 \end{bmatrix}}_{\mathbf{B}} \delta_e \tag{A.4}$$

V	$=$	$59.9\,\mathrm{m\,s^{-1}}$	m	$=$	$4547.8\,\mathrm{kg}$	$\bar{c}$	$=$ $2.022\,\mathrm{m}$
S	$=$	$24.2\,\mathrm{m^2}$	l_h	$=$	$5.5\,\mathrm{m}$	μ_c	$=$ 102.7
K_Y^2	$=$	0.980	x_{cg}	$=$	$0.30\bar{c}$		
C_{X_0}	$=$	0	C_{Z_0}	$=$	-1.1360		
C_{X_u}	$=$	-0.2199	C_{X_u}	$=$	-2.2720	C_{X_u}	$=$ 0
C_{X_α}	$=$	0.4653	C_{X_α}	$=$	-5.1600	C_{X_u}	$=$ -0.4300
$C_{X_{\dot{\alpha}}}$	$=$	0	$C_{X_{\dot{\alpha}}}$	$=$	-1.4300	C_{X_α}	$=$ -3.7000
C_{X_q}	$=$	0	C_{X_q}	$=$	-3.8600	$C_{X_{\dot{\alpha}}}$	$=$ -7.0400
$C_{X_{\delta_e}}$	$=$	0	$C_{X_{\delta_e}}$	$=$	-0.6238	C_{X_q}	$=$ -1.5530
b	$=$	$13.36\,\mathrm{m}$	C_L	$=$	1.1360	μ_b	$=$ 15.5
K_X^2	$=$	0.012	K_Z^2	$=$	0.037	K_{XZ}	$=$ 0.002
C_{Y_β}	$=$	-0.9896	C_{l_β}	$=$	-0.0772	C_{n_β}	$=$ 0.1638
C_{Y_p}	$=$	-0.0870	C_{l_p}	$=$	-0.3444	C_{n_p}	$=$ -0.0108
C_{Y_r}	$=$	0.4300	C_{l_r}	$=$	0.2800	C_{n_r}	$=$ -0.1930
$C_{Y_{\delta_a}}$	$=$	0	$C_{l_{\delta_a}}$	$=$	-0.2349	$C_{n_{\delta_a}}$	$=$ 0.0286
$C_{Y_{\delta_r}}$	$=$	0.3037	$C_{l_{\delta_r}}$	$=$	0.0286	$C_{n_{\delta_r}}$	$=$ -0.1261

Table A.1: Control and Stability Derivatives for the Cessna 550 Citation II, Adapted from Mulder et al. [81]

References

[1] G. D. Padfield and B. Lawrence. “The Birth of Flight Control: An Engineering Analysis of the Wright Brothers’ 1902 Glider”. In: *The Aeronautical Journal* 107.1078 (Dec. 2003), pp. 697–718. ISSN: 0001-9240, 2059-6464. DOI: 10.1017/S0001924000013464.

[2] William W. Davenport. *Gyro! The Life and Times of Lawrence Sperry*. New York: Scribner, 1978. 348 pp. ISBN: 978-0-684-15793-1.

[3] Brian L. Stevens, Frank L. Lewis, and Eric N. Johnson. “Aircraft Dynamics and Classical Control Design”. In: *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. Third edition. Hoboken, N.J: John Wiley & Sons, 2016, pp. 250–376. ISBN: 978-1-118-87098-3.

[4] Historic Wings. *George the Autopilot*. HistoricWings.com: A Magazine for Aviators, Pilots and Adventurers - A Magazine for Aviators, Adventurers and Pilots. Aug. 30, 2012. URL: http://fly.historicwings.com/2012/08/george-the-autopilot/ (visited on 04/03/2023).

[5] P. C. Gregory. *Proceedings of the Self Adaptive Flight Control Systems Symposium*. Technical Rerport AD0209389. United States: Aeronautical Systems Div Wright-Patterson AFB OH Flight Control Lab, 1959.

[6] William Koch et al. “Reinforcement Learning for UAV Attitude Control”. In: *ACM Transactions on Cyber-Physical Systems* 3.2 (Apr. 30, 2019), pp. 1–21. ISSN: 2378-962X, 2378-9638. DOI: 10.1145/3301273.

[7] M. M. Polycarpou and J. A. Farrell. “Neural Control”. In: *Control System Advanced Methods*. Ed. by W. S. Levine. 2nd ed. The Electrical Engineering Handbook Series. Boca Raton: CRC Press, 2011. ISBN: 978-1-4200-7364-5.

[8] M. Steinberg. “Historical Overview of Research in Reconfigurable Flight Control”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 219.4 (2005), pp. 263–275. ISSN: 0954-4100. DOI: 10.1243/095441005X30379.

[9] M. H. Smaili et al. “Intelligent Flight Control Systems Evaluation for Loss-of-Control Recovery and Prevention”. In: *Journal of Guidance, Control, and Dynamics* 40.4 (Apr. 2017), pp. 890–904. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.G001756.

[10] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0-262-03924-9.

[11] B. Kiumarsi et al. “Optimal and Autonomous Control Using Reinforcement Learning: A Survey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.6 (2018), pp. 2042–2062. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2017.2773458.

[12] S.A. Emami, P. Castaldi, and A. Banazadeh. “Neural Network-Based Flight Control Systems: Present and Future”. In: *Annual Reviews in Control* 53 (2022), pp. 97–137. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2022.04.006.

[13] Transportation Safety Board of Canada. *Loss of Rudder in Flight - Air Transat Airbus A310-308 C-GPAT, Miami, Florida, 90 Nm S, 06 March 2005*. Aviation Investigation Report A05F0047. Gatineau, Québec: Transportation Safety Board of Canada, 2007. 74 pp. ISBN: 978-0-662-46783-0.

[14] Oluwasegun Ayokunle Somefun, Kayode Akingbade, and Folasade Dahunsi. “The Dilemma of PID Tuning”. In: *Annual Reviews in Control* 52 (2021), pp. 65–74. ISSN: 13675788. DOI: 10.1016/j.arcontrol.2021.05.002.

[15] D. Vrabie and F. L. Lewis. “Approximate Dynamic Programming”. In: *Control System Advanced Methods*. Ed. by W. S. Levine. 2nd ed. The Electrical Engineering Handbook Series. Boca Raton: CRC Press, 2011, pp. 1414–. ISBN: 978-1-4200-7364-5.

[16] IATA. *Loss of Control In-Flight Accident Analysis Report*. Montreal: International Air Transport Association, 2019. ISBN: 978-92-9264-002-6.

[17] C.M. Belcastro and C.M. Belcastro. “Application of Failure Detection, Identification, and Accomodation Methods for Improved Aircraft Safety”. In: Proceedings of the American Control Conference. Vol. 4. 2001, pp. 2623–2624.

[18] Hossam Faris, Ibrahim Aljarah, and Seyedali Mirjalili. “Evolving Radial Basis Function Networks Using Moth–Flame Optimizer”. In: *Handbook of Neural Computation* (Jan. 1, 2017), pp. 537–550. DOI: 10.1016/B978-0-12-811318-9.00028-4.

[19] ICAO. *Innovation for a a Green Transition: 2022 Environmental Report*. 2022.

[20] IPCC. “Summary for Policymakers”. In: *Global Warming of 1.5°C* (June 9, 2022), pp. 1–24. DOI: 10.1017/9781009157940.001.

[21] D.S. Lee et al. “The Contribution of Global Aviation to Anthropogenic Climate Forcing for 2000 to 2018”. In: *Atmospheric Environment* 244 (Jan. 2021), p. 117834. ISSN: 13522310. DOI: 10.1016/j.atmosenv.2020.117834.

[22] Robert A. McDonald et al. “Future Aircraft Concepts and Design Methods”. In: *The Aeronautical Journal* 126.1295 (Jan. 2022), pp. 92–124. ISSN: 0001-9240, 2059-6464. DOI: 10.1017/aer.2021.110.

[23] Wilco Oosterom. “Flying-V Family Design”. 2021.

[24] Simon van Overeem. “Modelling, Control, and Handling Quality Analysis of the Flying-V”. 2022.

[25] Alberto Ruiz Garcia. “Aerodynamic Model Identification of the Flying V Using Wind Tunnel Data”. 2019.

[26] TU Delft. *Flying-V*. TU Delft. 2023. URL: https://www.tudelft.nl/lr/flying-v (visited on 04/06/2023).

[27] Aerospace Technology Institute. *The Journey Towards Autonomy in Civil Aerospace*. Insight 15. Cranfield, United Kingdom, 2020.

[28] Christian Janiesch, Patrick Zschech, and Kai Heinrich. “Machine Learning and Deep Learning”. In: *Electronic Markets* 31.3 (Sept. 2021), pp. 685–695. ISSN: 1019-6781, 1422-8890. DOI: 10.1007/s12525-021-00475-2.

[29] Anke Meyer-Baese and Volker Schmid. “Foundations of Neural Networks”. In: *Pattern Recognition and Signal Analysis in Medical Imaging*. Elsevier, 2014, pp. 197–243. ISBN: 978-0-12-409545-8. DOI: 10.1016/B978-0-12-409545-8.00007-8.

[30] Hai Nguyen and Hung La. “Review of Deep Reinforcement Learning for Robot Manipulation”. In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. 2019 Third IEEE International Conference on Robotic Computing (IRC). Naples, Italy: IEEE, Feb. 2019, pp. 590–595. ISBN: 978-1-5386-9245-5. DOI: 10.1109/IRC.2019.00120.

[31] Andrew G. Barto and Thomas G. Dietterich. “Reinforcement Learning and Its Relationship to Supervised Learning”. In: Jennie Si et al. *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ: IEEE Press, 2010. ISBN: 978-0-470-54478-5.

[32] Dongjin Lee, Mooncheon Choi, and Hyochoong Bang. “Model-Free Linear Quadratic Tracking Control for Unmanned Helicopters Using Reinforcement Learning”. In: *The 5th International Conference on Automation, Robotics and Applications*. 2011 5th International Conference on Automation, Robotics and Applications (ICARA 2011). Wellington, New Zealand: IEEE, Dec. 2011, pp. 19–22. ISBN: 978-1-4577-0330-0 978-1-4577-0329-4 978-1-4577-0328-7. DOI: 10.1109/ICARA.2011.6144849.

[33] Robert J. Clarke et al. “Closed-Loop Q-Learning Control of a Small Unmanned Aircraft”. In: *AIAA Scitech 2020 Forum*. AIAA Scitech 2020 Forum. Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 6, 2020. ISBN: 978-1-62410-595-1. DOI: 10.2514/6.2020-1234.

[34] Hongxun Liu et al. “Robust Control Strategy for Quadrotor Drone Using Reference Model-Based Deep Deterministic Policy Gradient”. In: *Drones* 6.9 (Sept. 12, 2022), p. 251. ISSN: 2504-446X. DOI: 10.3390/drones6090251.

[35] Jun H. Lee and Erik-Jan Van Kampen. “Online Reinforcement Learning for Fixed-Wing Aircraft Longitudinal Control”. In: *AIAA Scitech 2021 Forum*. AIAA Scitech 2021 Forum. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Jan. 11, 2021. ISBN: 978-1-62410-609-5. DOI: 10.2514/6.2021-0392.

[36] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. Version 3. In: (2018). DOI: 10.48550/ARXIV.1802.09477.

[37] Hao-nan Wang et al. “Deep Reinforcement Learning: A Survey”. In: *Frontiers of Information Technology & Electronic Engineering* 21.12 (Dec. 2020), pp. 1726–1744. ISSN: 2095-9184, 2095-9230. DOI: 10.1631/FITEE.1900533.

[38] Casper Teirlinck. “Reinforcement Learning for Flight Control Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance”. Delft: TU Delft, 2022.

[39] Amrut Sekhar Panda et al. “Combined Online and Offline Inverse Dynamics Learning for a Robot Manipulator”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020 International Joint Conference on Neural Networks (IJCNN). Glasgow, United Kingdom: IEEE, July 2020, pp. 1–7. ISBN: 978-1-72816-926-2. DOI: 10.1109/IJCNN48605.2020.9206604.

[40] Jacob Buckman et al. “Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion”. Version 2. In: (2018). DOI: 10.48550/ARXIV.1807.01675.

[41] Richard Bellman. *Dynamic Programming*. Princeton, NJ: Princeton Univ. Pr, 1984. 339 pp. ISBN: 978-0-691-07951-6.

[42] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. 3e ed. Athena Scientific Optimization and Computation Series. Belmont (Mass.): Athena scientific, 2007. ISBN: 978-1-886529-26-7 978-1-886529-30-4.

[43] Jun Lu. “Gradient Descent, Stochastic Optimization, and Other Tales”. Version 1. In: *arXiv preprint* (2022). DOI: 10.48550/ARXIV.2205.00832.

[44] D. Liu et al. “Adaptive Dynamic Programming for Control: A Survey and Recent Advances”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1 (2021), pp. 142–160. ISSN: 2168-2216. DOI: 10.1109/TSMC.2020.3042876.

[45] Paul J. Werbos. “A Menu of Designs for Reinforcement Learning over Time”. In: *Neural Networks for Control*. Ed. by W. Thomas Miller et al. Neural Network Modeling and Connectionism. Cambridge, Mass: MIT Press, 1990, pp. 67–97. ISBN: 978-0-262-13261-9.

[46] D.V. Prokhorov and D.C. Wunsch II. “Adaptive Critic Designs”. In: *IEEE Transactions on Neural Networks* 8.5 (1997), pp. 997–1007. ISSN: 1045-9227. DOI: 10.1109/72.623201.

[47] G. G. Lendaris and J. C. Neidhoefer. “Guidance in the Use of Adaptive Critics for Control”. In: Jennie Si et al. *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ: IEEE Press, 2010, pp. 97–124. ISBN: 978-0-470-54478-5.

[48] Silvia Ferrari and F. Stengel. “Model-Based Adaptive Critic Designs”. In: Jennie Si et al. *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ: IEEE Press, 2010. ISBN: 978-0-470-54478-5.

[49] Paul J. Werbos. “Approximate Dynamic Programming for Real-Time Control and Neural Modeling”. In: *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Ed. by David A. White and Donald A. Sofge. VNR Computer Library. New York: Van Nostrand Reinhold [u.a.], 1992. ISBN: 978-0-442-30857-5.

[50] Fei-Yue Wang, Huaguang Zhang, and Derong Liu. “Adaptive Dynamic Programming: An Introduction”. In: *IEEE Computational Intelligence Magazine* 4.2 (May 2009), pp. 39–47. ISSN: 1556-6048. DOI: 10.1109/MCI.2009.932261.

[51] Bo Sun and Erik-Jan van Kampen. “Incremental Model-Based Global Dual Heuristic Programming with Explicit Analytical Calculations Applied to Flight Control”. In: *Engineering Applications of Artificial Intelligence* 89 (Mar. 2020), p. 103425. ISSN: 09521976. DOI: 10.1016/j.engappai.2019.103425.

[52] "Direct Neural Dynamic Programming". In: Jennie Si et al. *Handbook of Learning and Approximate Dynamic Programming*. IEEE, 2009. ISBN: 978-0-470-54478-5. DOI: 10.1109/9780470544785.ch5.

[53] 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. Vol. 5. 2006, pp. 2989–3016. ISBN: 978-1-56347-819-2. DOI: 10.2514/6.2006-6429.

[54] Ye Zhou, Erik-Jan Van Kampen, and Q.P. Chu. "Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control". In: *The International Micro Air Vehicle Conference and Competition 2016*. Beijing, China, Oct. 17, 2016.

[55] Ye Zhou, Erik-Jan van Kampen, and QiPing Chu. "Nonlinear Adaptive Flight Control Using Incremental Approximate Dynamic Programming and Output Feedback". In: *Journal of Guidance, Control, and Dynamics* 40.2 (Feb. 2017), pp. 493–496. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.G001762.

[56] Pedro Miguel Dias, Ye Zhou, and Erik-Jan Van Kampen. "Intelligent Nonlinear Adaptive Flight Control Using Incremental Approximate Dynamic Programming". In: *AIAA Scitech 2019 Forum*. AIAA Scitech 2019 Forum. San Diego, California: American Institute of Aeronautics and Astronautics, Jan. 7, 2019. ISBN: 978-1-62410-578-4. DOI: 10.2514/6.2019-2339.

[57] Ye Zhou, Erik-Jan van Kampen, and Qi Ping Chu. "Incremental Model Based Online Dual Heuristic Programming for Nonlinear Adaptive Control". In: *Control Engineering Practice* 73 (Apr. 2018), pp. 13–25. ISSN: 09670661. DOI: 10.1016/j.conengprac.2017.12.011.

[58] Stefan Heyer, Dave Kroezen, and Erik-Jan Van Kampen. "Online Adaptive Incremental Reinforcement Learning Flight Control for a CS-25 Class Aircraft". In: *AIAA Scitech 2020 Forum*. AIAA Scitech 2020 Forum. Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 6, 2020. ISBN: 978-1-62410-595-1. DOI: 10.2514/6.2020-1844.

[59] Rick Feith. "Safe Reinforcement Learning in Flight Control: Introduction to Safe Incremental Dual Heuristic Programming". 2020.

[60] Ramesh Konatala, Erik-Jan Van Kampen, and Gertjan Looye. "Reinforcement Learning Based Online Adaptive Flight Control for the Cessna Citation II(PH-LAB) Aircraft". In: *AIAA Scitech 2021 Forum*. AIAA Scitech 2021 Forum. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Jan. 11, 2021. ISBN: 978-1-62410-609-5. DOI: 10.2514/6.2021-0883.

[61] B. Sun and E. van Kampen. "Incremental Model-Based Global Dual Heuristic Programming for Flight Control". In: *IFAC-PapersOnLine* 52.29 (2019). ISSN: 1474-6670. DOI: 10.1016/j.ifacol.2019.12.613.

[62] Y. Lecun, Y. Bengio, and G. Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539.

[63] 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.

[64] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". Version 1. In: (2013). DOI: 10.48550/ARXIV.1312.5602.

[65] Timothy P. Lillicrap et al. "Continuous Control with Deep Reinforcement Learning". Version 6. In: (2015). DOI: 10.48550/ARXIV.1509.02971.

[66] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". Version 1. In: (2017). DOI: 10.48550/ARXIV.1710.02298.

[67] David Silver et al. "Deterministic Policy Gradient Algorithms". In: *31st International Conference on Machine Learning, ICML 2014* 1 (June 2014).

[68] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8.3-4 (May 1992), pp. 293–321. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00992699.

[69] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". Version 2. In: (2018). DOI: 10.48550/ARXIV.1801.01290.

[70] Tuomas Haarnoja et al. “Soft Actor-Critic Algorithms and Applications”. Version 2. In: (2018). DOI: 10.48550/ARXIV.1812.05905.

[71] Xiaoteng Ma et al. *DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning*. June 10, 2020. DOI: 10.48550/arXiv.2004.14547. arXiv: 2004.14547 [cs]. (Visited on 10/21/2022). preprint.

[72] Eivind Bøhn et al. “Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization”. In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2019 International Conference on Unmanned Aircraft Systems (ICUAS). June 2019, pp. 523–533. DOI: 10.1109/ICUAS.2019.8798254.

[73] John Schulman et al. “Proximal Policy Optimization Algorithms”. Version 2. In: (2017). DOI: 10.48550/ARXIV.1707.06347.

[74] M. A. van den Hoek, C. C. de Visser, and D. M. Pool. “Identification of a Cessna Citation II Model Based on Flight Test Data”. In: *Advances in Aerospace Guidance, Navigation and Control*. Ed. by Bogusław Dołęga et al. Cham: Springer International Publishing, 2018, pp. 259–277. ISBN: 978-3-319-65282-5 978-3-319-65283-2. DOI: 10.1007/978-3-319-65283-2_14.

[75] R. Enns and J. Si. “Helicopter Trimming and Tracking Control Using Direct Neural Dynamic Programming”. In: *IEEE Transactions on Neural Networks* 14.4 (July 2003), pp. 929–939. ISSN: 1045-9227. DOI: 10.1109/TNN.2003.813839.

[76] Ye Zhou, Erik-Jan van Kampen, and QiPing Chu. “Incremental Approximate Dynamic Programming for Nonlinear Adaptive Tracking Control with Partial Observability”. In: *Journal of Guidance, Control, and Dynamics* 41.12 (Dec. 2018), pp. 2554–2567. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.G003472.

[77] Killian Dally and Erik-Jan Van Kampen. “Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control”. In: *AIAA SCITECH 2022 Forum*. AIAA SCITECH 2022 Forum. San Diego, CA & Virtual: American Institute of Aeronautics and Astronautics, Jan. 3, 2022. ISBN: 978-1-62410-631-6. DOI: 10.2514/6.2022-2078.

[78] Silvia Ferrari and Robert F. Stengel. “Online Adaptive Critic Flight Control”. In: *Journal of Guidance, Control, and Dynamics* 27.5 (Sept. 2004), pp. 777–786. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.12597.

[79] Peter Seres. “Distributional Reinforcement Learning for Flight Control”. Delft: Delft University of Technology, 2022.

[80] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. ISSN: 1533-7928.

[81] J. A. Mulder et al. *Flight Dynamics: Lecture Notes AE3202*. Delft: Delft University of Technology, 2013.

[82] Fabian Grondman et al. “Design and Flight Testing of Incremental Nonlinear Dynamic Inversion-based Control Laws for a Passenger Aircraft”. In: *2018 AIAA Guidance, Navigation, and Control Conference*. 2018 AIAA Guidance, Navigation, and Control Conference. Kissimmee, Florida: American Institute of Aeronautics and Astronautics, Jan. 8, 2018. ISBN: 978-1-62410-526-5. DOI: 10.2514/6.2018-0385.

[83] Greg Brockman et al. *OpenAI Gym*. June 5, 2016. arXiv: 1606.01540 [cs]. URL: http://arxiv.org/abs/1606.01540 (visited on 04/14/2023). preprint.

[84] Rishabh Agarwal et al. *Deep Reinforcement Learning at the Edge of the Statistical Precipice*. Jan. 5, 2022. arXiv: 2108.13264. URL: http://arxiv.org/abs/2108.13264 (visited on 04/28/2023). preprint.