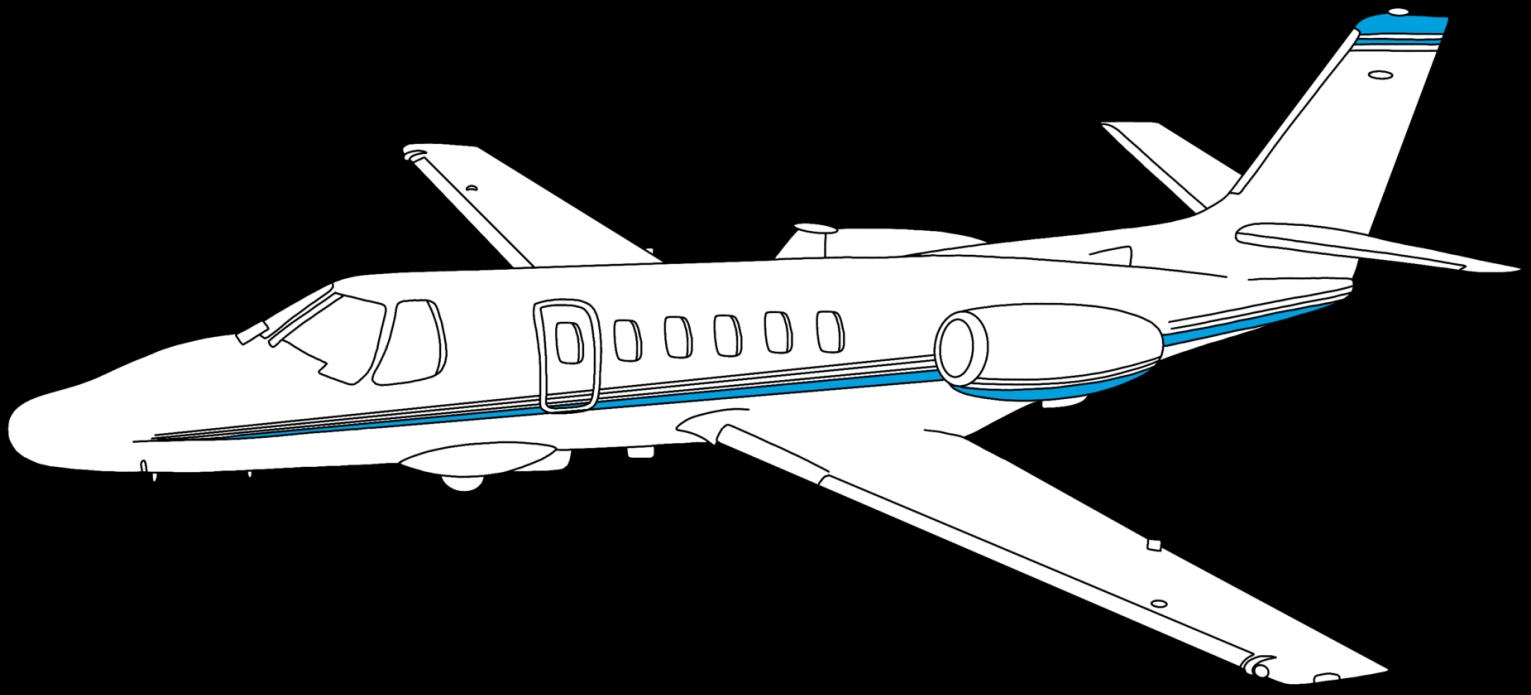
Uncertainty-Aware Learning for Fault-Tolerant Flight Control

Hybrid Reinforcement Learning for the Flight Control System of a Cessna 550 Citation II

Garcia de Vinuesa Garcia, Pablo



TU Delft

Uncertainty-Aware Learning for Fault-Tolerant Flight Control

Hybrid Reinforcement Learning for the Flight Control System of a Cessna 550 Citation II

by

Garcia de Vinuesa Garcia, Pablo

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on March 19th, 2026, at 14:00.

Supervisor:	Dr.ir. Erik-Jan van Kampen
Chair:	Dr. Spilios Theodoulis
External	Examiner: Dr. Xuerui Wang
Additional Member:	Ir. Isabelle El-Hajj
Faculty:	Faculty of Aerospace Engineering, Delft

Aircraft illustration adapted from FlyCraft Aviation [11].

TU Delft

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AFCS	Automatic Flight Control System
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
ADP	Approximate Dynamic Programming
IDHP	Incremental Dual Heuristic Programming
INDI	Incremental Nonlinear Dynamic Inversion
NDI	Nonlinear Dynamic Inversion
SAC	Soft Actor-Critic
DSAC	Distributional Soft Actor-Critic
RUN-DSAC	Returns Uncertainty-Navigated Distributional Soft Actor-Critic
TD	Temporal Difference
DDPG	Deep Deterministic Policy Gradient
TD3	Twin Delayed Deep Deterministic Policy Gradient
CAPS	Conditioning for Action Policy Smoothness
RLS	Recursive Least Squares
UAV	Unmanned Aerial Vehicle

Symbols

Symbol	Definition
$\mathcal{S}$	State space of the Markov Decision Process
$\mathcal{A}$	Action space of the Markov Decision Process
$\mathcal{P}$	State transition dynamics
$\mathbf{s}_t$	State vector at time step t
$\mathbf{a}_t$	Action vector at time step t
r_t	Reward at time step t
G_t	Discounted return
γ	Reward discount factor
π	Policy mapping states to actions
π_θ	Parameterized actor policy
V^π	State-value function
Q^π	Action-value function
θ	Actor network parameters
ϕ	Critic network parameters
τ	Soft target network update coefficient
$\mathbf{x}$	Aircraft state vector
$\mathbf{u}$	Control input vector
$\mathbf{F}$	Incremental state transition matrix
$\mathbf{G}$	Control effectiveness matrix
λ_ϕ	Critic estimate of the value-function gradient (co-state)
η_a	Actor learning rate

Symbol	Definition
μ_c	Critic learning rate
γ_{RLS}	Recursive least-squares forgetting factor
$\mathcal{L}_\pi$	Actor loss function
$\mathcal{L}_\lambda$	Critic loss function
δ_t	Temporal-difference error
$Z(s,a)$	Return distribution in distributional reinforcement learning
σ_Q	Standard deviation of the return distribution
Q_V	Uncertainty-aware action-value function
p, q, r	Body-axis roll, pitch, and yaw rates
α	Angle of attack
β	Sideslip angle
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle
$\delta_e, \delta_a, \delta_r$	Elevator, aileron, and rudder deflections

1 Introduction

1.1. Background

1.1.1. History of Automatic Flight Control

Loss of control in-flight remains one of the leading contributors to aviation accidents [26]. This underscores the need for flight control systems that can adapt to unforeseen failure scenarios and thus exert fault-tolerant control. With the rise of increasingly autonomous aircraft in both civil aviation and urban air mobility, the development of fault-tolerant and adaptive flight controllers has become a critical research priority.

Early automatic flight control systems (AFCS) were developed using classical linear control techniques, requiring gain scheduling to switch between controllers based on the aircraft's operating point [41]. These controllers perform well under nominal conditions, yet, due to their dependence on high-fidelity models of the aircraft dynamics, they are unable to deal with fault-scenarios changing the dynamics of the aircraft [5]. These models, however, are expensive and difficult to develop, and often fail to generalise for many unknown faulty scenarios. For this reason, there is a growing need to develop model-free flight control systems, ensuring adaptability and safe operation in the face of uncertain circumstances and failure modes.

To address some of these limitations, nonlinear adaptive methods such as Nonlinear Dynamic Inversion (NDI) and Backstepping (BS) were introduced [25, 7]. These approaches improved robustness by dealing with nonlinear dynamics and use Lyapunov stability theory to guarantee convergence and performance. However, both of these approaches require an accurate model of the system dynamics and thus remain sensitive to modeling errors and parameter uncertainties [52]. To reduce the reliance on the high-fidelity model, the incremental versions INDI and IBS were developed. INDI uses incremental changes in the measurements of sensors to adjust control commands in real time, enabling adaptable, robust control under varying unforeseen conditions [38, 14, 46]. IBS combines the incremental technique with Lyapunov-based backstepping, improving fault-tolerance and disturbance rejection [1, 13].

Despite these advances, incremental methods rely heavily on accurate sensor data and are sensitive to challenges related to sensor synchronization and filtering [21]. This motivates the research of techniques based on bio-inspired Artificial Intelligence (AI), known as Reinforcement Learning (RL).

1.1.2. Reinforcement Learning for Flight Control

Reinforcement Learning refers to a classification of machine learning algorithms, in which an agent learns control strategies through interaction with the environment, guided by a reward function [48]. Unlike classical control, RL does not always need a model of the system dynamics, making it suitable for non-linear, high-dimensional tasks, such as fault-tolerant flight control.

Traditional RL methods, such as Dynamic Programming (DP), Monte Carlo (MC) methods, and Temporal Difference (TD) learning, are known as tabular methods, as these use discrete action and state spaces to perform tasks [48]. These are unfeasible in the flight control domain, due to the complexity of flight control systems. However, the introduction of function approximators, such as linear methods or Artificial Neural Networks (ANNs), to reinforcement learning algorithms has led to the use of algorithms based on these methods for continuous and high-dimensional control tasks [48].

Approximate Dynamic Programming (ADP) methods combine principles from RL and Dynamic Programming (DP) and use an actor-critic structure [37]. These methods have been used in various flight control applications such as missile control [17], autolanding for aircraft [34], or the six-degree-of-freedom control of a business jet [9]. However, ADP methods require a model of the system dynamics, limiting their application to unforeseen failure types. More recent Adaptive Critic Methods (ACDs) implement an incremental approach similar to INDI and IBS, and can successfully be applied to flight control problems while eliminating the model dependence. Incremental Heuristic Dynamic Programming (IHDP) [57] and Incremental Dual Heuristic Programming (IDHP) [58] have been used for online flight control. These methods do not require an online learning phase, and are able to adapt online to changes in the plant's dynamics, making them suitable for fault-tolerant flight control [59, 56]. IADP methods are sample-efficient and demonstrate adaptive control, but lack safety, as the controllers need to explore in order to learn [8].

Due to the improvements in Deep Neural Networks (DNN), reinforcement learning has branched off into a field called Deep Reinforcement Learning (DRL). Deep Q-Networks (DQN), a notable DRL algorithm using the principle of TD learning, was used to achieve human-like performance on a number of classic Atari games [32]. DQN, however, is limited to discrete-time problems, making it unsuitable for the continuous flight control problem. Deep Deterministic Policy Gradient (DDPG) [30] and Trust Region Policy Optimization (TRPO) [35] are two extensions to DQN that handle continuous reinforcement learning tasks. These methods have been used for attitude control of UAVs [3] and for aircraft guidance in a 3D space [53]. DDPG was improved with the introduction of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [12]. TD3 reduces overestimation in DDPG and has been successfully applied for 3D position tracking of a quadcopter [24], and UAV navigation through an obstacle course [18]. Soft Actor-Critic (SAC) methods have also been developed to address DDPG's problem with overestimation. Unlike TD3, these methods use a stochastic policy, which encourages exploration and increases sample efficiency [5]. SAC can be applied offline, and is one of the more stable and sample-efficient algorithms for continuous control, with various applications for flight control of Cessna 550 Citation II aircraft [5, 36, 21]. However, these algorithms are limited by their inability to adapt to changing flight conditions in flight, and can only generalize from the learned experience [51].

1.1.3. The Need for Hybrid Reinforcement Learning

Despite these advances, the application of reinforcement learning to flight control faces some challenges. Purely offline-trained RL approaches are limited by the quality and diversity of the dataset; thus, policies trained solely on data might fail to generalize to unforeseen failures, decreasing the fault-tolerance capability of the algorithm [28]. Online-trained approaches, like incremental approximate dynamic programming methods, while highly adaptive, are more sample inefficient and can lead to unsafe exploration in the search for higher rewards [40].

To avoid the disadvantages associated with offline and online methods, hybrid methods were developed. These combine the positives of both by first training policies offline, taking advantage of the high sample efficiency of offline methods, and then refining policies online to adapt to new scenarios as the agent interacts with the environment. Recent work by Teirlinck combined SAC and iDHP to develop a hybrid controller for the Cessna 550 Citation II [49]. Later work by Vieria combined DSAC and iDHP to develop a fault-tolerant controller for the same aircraft model [51]. This approach has made promising developments in the field of fault-tolerant flight control, combining the safety and efficiency of offline learning with the adaptability of offline learning.

1.2. Research Objective and Questions

Reinforcement learning for flight control applications has progressed toward hybrid methods, leveraging the sample efficiency of offline methods and the adaptability of online RL algorithms. However, these methods must be extended to fault-tolerant flight control applications to ensure safety in flight control. Motivating the research objective for this thesis:

Research Objective

This research aims to develop and evaluate a hybrid offline-online Reinforcement Learning framework for the fault-tolerant flight control of the Cessna Citation II

This research objective can be further broken down into three separate research questions. The first aims to explore the principles and state-of-the-art algorithms in reinforcement learning applied to flight control. This includes understanding the fundamentals of RL, evaluating state-of-the-art algorithms, and defining what ”fault tolerance” entails in the context of flight control. The first research question, **RQ1**, can be seen in the following box:

Research Question 1

RQ1. What is the most suitable reinforcement learning framework for fault-tolerant flight control?

- **RQ1.1.** What are the key principles of reinforcement learning, and how are they relevant to flight control systems?
- **RQ1.2.** What are the current state-of-the-art Reinforcement Learning algorithms and architectures used for general aircraft flight control?
- **RQ1.3.** What defines a fault-tolerant flight control system, and to what extent can the state-of-the-art RL approaches be applied to achieve this capability?

The next research question investigates how to design and implement a hybrid reinforcement learning controller. The goal is to identify how to combine the sample efficiency of offline learning with the adaptability of online methods for flight control applications.

Research Question 2

RQ2. How can a hybrid offline-online reinforcement learning controller be designed and implemented?

- **RQ2.1.** What are the defining characteristics and theoretical advantages of a hybrid offline-online architecture for control?
- **RQ2.2.** Which specific offline and online RL algorithms are most suitable to serve as the components of the hybrid controller?
- **RQ2.3.** What is the methodology for designing and implementing the selected hybrid framework?

Once the hybrid controller has been developed, it must be compared against non-hybrid baselines to determine the advantages and limitations of the developed controller. This evaluation will be conducted in both nominal and fault-tolerant scenarios, using previously defined performance metrics. The results from the evaluation will help identify the strengths and weaknesses of the proposed framework, as well as its practical limitations.

Research Question 3

RQ3. How does the performance of the hybrid controller compare to non-hybrid baselines in fault-tolerant scenarios?

- **RQ3.1.** What metrics are used to evaluate the performance and flight-tolerance of an aircraft flight controller?
- **RQ3.2.** How does the hybrid controller's performance and learning stability compare against a purely offline-trained controller when subjected to nominal and failure modes?
- **RQ3.3.** How does the hybrid controller's performance and stability compare to a purely online-trained controller when adapting to failure modes?
- **RQ3.4.** Based on the experimental results, what are the key advantages and disadvantages of the proposed hybrid approach for fault-tolerant flight control?

1.3. Project Plan

This section includes a plan and timeline of the activities to be done from the end of the literature review phase until the completion of the research.

Phase 1: Framework Development (25/08 - 27/09)
The immediate focus of this phase is building a functional hybrid controller.

- Familiarise myself in-depth with the frameworks for RUN-DSAC and iDHP algorithms and identify how to combine these in a hybrid framework
- Develop the RUN-DSAC offline component for policy pre-training.
- Develop the iDHP online module, and integrate both into a hybrid RUN-DSAC-IDHP architecture
- Verify integration through simple nominal scenarios and test viability of hybrid framework with DASMAT simulation.

Phase 2: Baseline and Comparative Experiments (29/09 - 31/10)
The next phase will evaluate the hybrid controller and baseline controllers under nominal flight conditions (if time allows it, Phase 3 will also be done during this time period). The steps to be taken include:

- Develop working offline RUN-DSAC, and DSAC controller.
- Use work by Vieira [51] to develop hybrid DSAC-iDHP controller.
- Develop a working online iDHP controller.
- Test hybrid RUN-DSAC-IDHP against all 3 baseline controllers developed for tracking tasks under nominal flight conditions.
- Analyse the learning performance of the proposed controller against the baseline controllers.
- Analyse the tracking performance of the proposed controller against the baseline controllers.

Phase 3: Expanded Fault-Tolerant Experiments (03/11 - 21/11)
The hybrid controller and the developed baseline controllers will then be tested against a set of fault-tolerant scenarios to evaluate

- Investigate commonly used fault-tolerant scenarios and identify the faults to be tested.
- Conduct experiments testing the hybrid controller against baseline controllers for each fault-tolerant scenario.
- Analyse the tracking performance and the results of the experiments.
- Refine the controller based on findings.

Phase 4: Verification and Validation (24/11 - 19/12)
Once the experimental results have been achieved, the focus will shift towards verifying and validating the results, to ensure robustness and reproducibility of the findings.

- Repeat the experiments with different data and/or nominal flight conditions to confirm consistency.

- Verify the algorithms are correctly implemented.

Phase 5: Analysis and Drafting (22/12 - 02/02)
Ahead of the greenlight meeting, this next phase involves the analysis and drafting of the thesis. This involves extracting key insights from the results, interpreting them, and developing a cohesive narrative for the thesis.

- Generate figures and tables for research.
- Draft and write methodology, results, validation, and verification sections, and the scientific article.
- Apply all the recommended changes to the literature review.

Phase 6: Finalization (08/02 - 11/03)
The final phase of the thesis will be to adopt any changes needed since the green light meeting, finish obtaining any results that are still needed, and finish writing the thesis.

- Integrate feedback to refine arguments, results, and address any gaps.
- Ensure consistency in figures, writing style, references, and formatting for the report.
- Obtain any results that are still missing or any minor tweaks that must be made.

This is a general outline of the steps needed to be taken in order to successfully complete this project. It must be noted that these changes are subject to change as the project progresses, but should serve as a general timeline to guide the progress of the report. Below is a table with a summary of the phases described above and timeline to be followed:

Table 1.1: Summary of Project Phases and Timeline

Phase	Dates	Description
Phase 1: Framework Development	25/08 – 27/09	Build the hybrid framework by implementing the RUN-DSAC offline module and the iDHP online module, and integrate both. Test initial performance under nominal conditions.
Phase 2: Baseline and Comparative Experiments	29/09 – 31/10	Develop and train baseline controllers (DSAC, RUN-DSAC, DSAC-iDHP, iDHP) and compare them with the hybrid controller under nominal tracking tasks.
Phase 3: Fault-Tolerant Experiments	03/11 – 21/11	Evaluate hybrid and baseline controllers in predefined fault-tolerant scenarios, refining the hybrid framework where needed.
Phase 4: Verification and Validation	24/11 – 19/12	Verify reproducibility and correctness of experiments and algorithms; validate performance metrics and dataset integrity.
Phase 5: Analysis and Drafting	22/12 – 02/02	Analyze results, generate figures, and write the methodology, results, and discussion sections. Prepare a full draft for the green light meeting.
Phase 6: Finalization	08/02 – 11/03	Incorporate feedback, finalize experiments if needed, and polish the thesis for submission in mid-March.

Part I

Research Paper

Hybrid Risk-Aware Reinforcement Learning for Fault-Tolerant Flight Control

P. Garcia de Vinuesa Garcia*
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Recent advances in reinforcement learning (RL) have led to the creation of high-performance RL-based flight controllers, but ensuring fault-tolerance and adaptability remains a major challenge in safety-critical aerospace systems. This study introduces a hybrid RL flight control framework combining an offline uncertainty-aware distributional policy trained using Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC) and an online Incremental Dual Heuristic Programming (IDHP) controller, selected to provide a robust control baseline while enabling real-time adaptation. RUN-DSAC is chosen over SAC and DSAC due to its exploitation of return uncertainty, enabling more reliable learning, improved convergence consistency, and increased generalization across fault scenarios. The proposed hybrid controller demonstrates improved attitude-tracking performance and enhanced fault-mitigation capabilities when compared to stand-alone offline and online controllers across a range of fault scenarios in a high-fidelity Cessna 550 Citation II simulation. These results demonstrate that combining uncertainty-aware offline learning with online adaptive control provides a pathway toward adaptive fault-tolerant flight control systems.

Nomenclature

$\mathcal{S}, \mathcal{A}$	=	state and action spaces of the Markov decision process
$\mathbf{s}, \mathbf{a}$	=	state and action vectors
$\mathbf{x}, \mathbf{u}$	=	aircraft state vector and control input vector
$\mathcal{P}$	=	state-transition dynamics
r, G_t	=	reward function and discounted return
γ	=	reward discount factor
$\pi, \pi_\theta, \pi_{\bar{\theta}}$	=	policy, actor policy, and target policy
V^π, Q^π, V, Q	=	state-value and action-value functions
$\mathcal{H}$	=	policy entropy
$\theta, \phi, \bar{\theta}, \bar{\phi}$	=	actor and critic parameters and target parameters
$\mathcal{L}$	=	generic loss function
$\lambda_\phi, \lambda_{\phi'}$	=	critic estimate of the value-function gradient (co-state) and its target
$\mathbf{F}, \mathbf{G}$	=	incremental state-transition and control-effectiveness matrices
$\boldsymbol{\epsilon}$	=	recursive least-squares prediction error
γ_{RLS}	=	recursive least-squares forgetting factor
τ	=	soft update coefficient for target networks
p, q, r	=	body-axis roll, pitch, and yaw rates
α, β	=	angle of attack and sideslip angle
θ, ϕ, ψ	=	pitch, roll, and yaw angles
$\delta_e, \delta_a, \delta_r$	=	elevator, aileron, and rudder deflections
Δt	=	simulation time step
$Z(\mathbf{s}, \mathbf{a})$	=	return distribution
σ_Q	=	standard deviation of the return distribution
μ	=	uncertainty sensitivity coefficient
Q_V	=	uncertainty-aware action-value function

*M.Sc. Student, Faculty of Aerospace Engineering, Department of Control & Operations, Section Control & Simulation, Delft University of Technology

I. Introduction

Loss of control in flight remains one of the leading causes of fatal aviation accidents, according to the International Air Transport Association, accounting for 47% of all fatal aviation accidents from 2014-2023 [1]. This safety challenge has driven continuous advances in automatic flight control systems (AFCS), with increased emphasis on adaptability, robustness, and fault tolerance. As aircraft systems become increasingly autonomous, flight control systems must not only operate nominally but also adapt to and mitigate fault scenarios when necessary.

Conventional AFCS architectures are predominantly designed using linear control techniques combined with gain scheduling, in which controllers are tuned around predefined operating points across the flight envelope [2]. While these approaches perform well under nominal flight conditions, they can only mitigate faults using robust control techniques at the expense of performance. Non-linear control techniques, like Non-linear Dynamic Inversion (NDI), are more suited for fault-tolerant flight control since they invert the dynamics of the controlled system to formulate control laws [3]. Yet these methods also require an accurate model of the environment dynamics, and thus are limited to modelling errors, uncertainties, and the correctness of the state estimates [2]. To reduce model dependency, incremental variants were developed, including incremental NDI (INDI) and incremental backstepping (IBS), which use sensor measurements to estimate an incremental model of the dynamics [4, 5]. Although these methods have demonstrated improved fault tolerance, they remain limited by sensor accuracy and errors.

Reinforcement Learning (RL) offers a fundamentally different paradigm for flight control, in which control policies aiming to meet an objective are learned through interaction with the environment rather than relying on plant models. Using optimal control and dynamic programming, RL improves control behaviour based on the rewards obtained. Early RL applications were limited to low-dimensional problems due to discrete state-action spaces and the *curse of dimensionality* [6]. The introduction of function approximation methods enabled RL to scale to high-dimensional, continuous control tasks such as aircraft flight control [6].

Within the RL framework, Approximate Dynamic Programming (ADP) has demonstrated potential for adaptive and fault-tolerant flight control. Online adaptive critic designs showed early success in real-time flight control applications [7, 8], and the subsequent development of incremental ADP allowed learning without the need for a global model [9–12]. In particular, Incremental Dual Heuristic Programming (IDHP) has demonstrated sample-efficient online learning and the ability to adapt to failures in fixed-wing aircraft applications [13, 14]. However, purely online learning approaches face safety concerns during exploration and suffer from reduced scalability when applied to high-dimensional, strongly coupled control problems.

In parallel, advances in deep reinforcement learning (DRL) have shown the ability to produce robust and efficient aircraft control strategies. Value-based and policy-based methods have achieved success in complex control tasks [15, 16], and actor-critic algorithms such as DDPG, PPO, and SAC have shown strong performance in continuous-action environments [17–19]. These methods have been successfully applied to flight control problems such as fixed-wing attitude control and fault-tolerant manoeuvring [20–22]. Despite their ability to learn highly non-linear and coupled dynamics, DRL methods require extensive offline training and lack adaptability once deployed, limiting their fault-mitigation ability.

Recent research addressed these limitations through the use of distributional reinforcement learning, where the full return distribution is learned instead of just the expected value. The distributional approach improves stability during training and robustness, which is particularly useful for safety-critical flight control tasks [23]. Building on this, uncertainty-aware algorithms such as the Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC) have shown improved learning efficiency, tracking performance, and robustness compared to SAC and DSAC in flight control applications [24]. These offline methods, especially the new RUN-DSAC, can generalize across faults despite being trained under nominal conditions; however, they lack the real-time adaptability needed to fully mitigate unforeseen faults.

Hybrid reinforcement learning frameworks have thus emerged as a promising solution, combining the higher sample efficiency and safety of offline algorithms during training with the adaptability of online algorithms [25]. Prior work has demonstrated the effectiveness of these approaches in flight control applications, with SAC-IDHP and DSAC-IDHP controllers outperforming their stand-alone counterparts under nominal and fault conditions [26, 27].

Motivated by these findings, the contribution of this paper is a hybrid fault-tolerant flight control framework that integrates an offline-trained RUN-DSAC agent with an online IDHP agent. The offline component provides a baseline that can handle non-linear dynamics, while the online IDHP controller incrementally adapts the policy in response to unforeseen failures. By combining the complementary advantages of offline distributional DRL and online adaptive control, the proposed approach aims to achieve both high nominal performance and reliable fault tolerance in controlling the Cessna Citation II simulation.

The theoretical background of IDHP and the progression from SAC to RUN-DSAC are explained in Section II. Section III then describes the simulation environment and the proposed hybrid controller architecture. Section IV presents the experimental results and discusses performance under nominal and faulty conditions. Finally, Section V concludes the paper and outlines directions for future research.

II. Background

This section introduces the algorithms used to form this hybrid approach. First, the RL problem is formally formulated, then the online IDHP algorithm is introduced, and its main working principles are explained. Finally, the progression from SAC to DSAC and then to RUN-DSAC is shown, and the working principles of these algorithms are detailed.

A. Reinforcement Learning Problem Formulation

Reinforcement learning addresses sequential decision-making problems by modelling the interaction between the agent and its environment as a Markov Decision Process (MDP) [6]. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space, $\mathcal{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ the state transition dynamics, $r(\mathbf{s}_t, \mathbf{a}_t)$ the reward function, and $\gamma \in [0, 1]$ the discount factor. The RL agent at time t selects an action $\mathbf{a}_t \in \mathbb{R}^m$ which acts on the environment with state $\mathbf{s}_t \in \mathbb{R}^n$. The state transition function in Equation 2, then, determines the next state $\mathbf{s}_{t+1}$, and a scalar reward r_{t+1} is obtained from this state and used as feedback for the agent. The objective of the agent is to learn a policy $\pi(\mathbf{a}|\mathbf{s})$ that maximises the expected discounted return G_t defined in Equation 1 [6].

$$G_t = \sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_{t+k}, \mathbf{a}_{t+k}) \tag{1}$$

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

The objective of the agent can be expressed through the value function $V^\pi(\mathbf{s})$ or the action value function $Q^\pi(\mathbf{s}, \mathbf{a})$ expressed in Equation 3 and Equation 4.

$$V^\pi(\mathbf{s}) = \mathbb{E}_\pi \left[G_t \mid \mathbf{s}_t = \mathbf{s} \right] \tag{3}$$

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}'} \mathcal{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) V^\pi(\mathbf{s}') \tag{4}$$

Deep RL methods use function approximation to estimate the policy and value functions for a problem. IDHP uses a critic to estimate the derivative of the state value function with respect to the state. SAC, on the other hand, estimates an action value function.

B. Online Agent: Incremental Dual Heuristic Programming

This work selects IDHP as the online component of the hybrid agent. This agent was implemented with reference to the work of [26] and [10, 28]. IDHP builds on the Dual Heuristic Programming (DHP) principle by incorporating an incremental model that identifies local system dynamics online, thereby eliminating model dependency and making the algorithm suitable for unknown flight conditions. The IDHP agent follows an actor-critic architecture, comprising an actor that uses Neural Networks to approximate the control policy and a critic that approximates the derivative of the state value function with respect to the states [29].

1. Incremental Model

The incremental model of IDHP provides a local linear approximation of the environment states at the current time based on the conditions from the previous instant. This model can be derived by means of a first-order Taylor series expansion shown in Equation 5, where $\mathbf{F}_{t-1}$ and $\mathbf{G}_{t-1}$ are the system transition matrix and the control effectiveness matrices, respectively.

$$\Delta\mathbf{x}_{t+1} \approx \mathbf{F}_{t-1}\Delta\mathbf{x} + \mathbf{G}_{t-1}\Delta\mathbf{u}_t \tag{5}$$

Since IDHP is model-free, the $\mathbf{F}_{t-1}$ and $\mathbf{G}_{t-1}$ matrices are unknown. The algorithm uses a Recursive Least Squares (RLS) estimator to identify these matrices at every time step. After each interaction with the environment, the incremental model uses the observed states to improve its predictions of the matrices, thereby refining the estimate of the system dynamics.

The incremental model is expressed in matrix form as follows:

$$\mathbf{\Theta}_{t-1} = \begin{bmatrix} \mathbf{F}_{t-1}^T \\ \mathbf{G}_{t-1}^T \end{bmatrix} \tag{6}$$

$$\mathbf{X}_t = \begin{bmatrix} \Delta\mathbf{x}_t \\ \Delta\mathbf{u}_t \end{bmatrix} \tag{7}$$

$$\Delta\hat{\mathbf{x}}_{t+1}^T = \mathbf{X}_t^T \hat{\mathbf{\Theta}}_{t-1} \tag{8}$$

$$\boldsymbol{\epsilon}_t = \Delta\mathbf{x}_{t+1}^T - \Delta\hat{\mathbf{x}}_{t+1}^T \tag{9}$$

The parameter matrix $\mathbf{\Theta}_{t-1}$ in Equation 6 and the input information matrix $\mathbf{X}_t$ in Equation 7 are then used to calculate the state transition with Equation 8. The prediction error $\boldsymbol{\epsilon}_t$ is then calculated in Equation 9 by subtracting the state prediction obtained in Equation 8 from the observed change in state $\Delta\mathbf{x}_{t+1}$.

$$\mathbf{\Lambda}_t = \frac{1}{\gamma_{\text{RLS}}} \left(\mathbf{\Lambda}_{t-1} - \frac{\mathbf{\Lambda}_{t-1}\mathbf{X}_t\mathbf{X}_t^T\mathbf{\Lambda}_{t-1}}{\gamma_{\text{RLS}} + \mathbf{X}_t^T\mathbf{\Lambda}_{t-1}\mathbf{X}_t} \right) \tag{10}$$

$$\hat{\mathbf{\Theta}}_t = \hat{\mathbf{\Theta}}_{t-1} + \frac{\mathbf{\Lambda}_{t-1}\mathbf{X}_t}{\gamma_{\text{RLS}} + \mathbf{X}_t^T\mathbf{\Lambda}_{t-1}\mathbf{X}_t}\boldsymbol{\epsilon}_t \tag{11}$$

Equation 11 shows the update of the parameter matrix, yet this update requires computing the covariance matrix $\mathbf{\Lambda}_t$ and the model's forgetting factor γ_{RLS}. The forgetting factor controls the influence of past data on current estimates and is a number between 0 and 1. The covariance matrix quantifies the covariances among the parameter estimates, indicating the confidence in these estimates, and is updated according to Equation 10. The parameter matrix and the covariance matrix are calculated using a previous estimate of itself. For this reason, the first is initialized to zeros and the second to a high-magnitude identity matrix, indicating very low confidence in the current estimates.

Additionally, when the magnitude of the prediction error $\boldsymbol{\epsilon}_t$ exceeds a predefined threshold, the covariance matrix is reset to its initial high value $\mathbf{\Lambda}_0$ to restore adaptability. This mechanism, adapted from the work of Teirlinck [26], prevents divergence under rapidly changing dynamics, thereby improving the algorithm's online adaptability.

2. IDHP Actor

The actor in the IDHP framework is responsible for generating the parametrized control policy π_θ, which guides the controller's actions. The policy takes the current observation states from the environment $\mathbf{s}_{t+1}$, the previous control action $\mathbf{a}_t$, and the tracking error as inputs and outputs the current control action. With this action, the new system states and the tracking error are calculated. The negative of the tracking error makes up the reward signal, r_t, which is used for calculating the Actor's loss $\mathcal{L}_\pi(t)$ in Equation 12.

$$\mathcal{L}_\pi(t) = -V(\mathbf{s}_t) = -\left[r_{t+1} + \gamma V(\mathbf{s}_{t+1})\right] \tag{12}$$

The loss function of the control policy, Equation 12, is merely the sum of the tracking error and the next Bellman value estimate $V(s_{t+1})$ multiplied by a discount factor γ. The gradient of this loss function with respect to the parameters of the control policy θ is then used to update the parameters, and thus, the Actor's control policy [27]. This can be seen in Equation 13 and Equation 14. These policy-gradient-based updates allow the controller to incrementally improve performance online, making IDHP suitable for adaptive and fault-tolerant scenarios.

$$\begin{aligned} \nabla_\theta\mathcal{L}_\pi = \frac{\partial\mathcal{L}_\pi}{\partial\theta} &= -\left[\frac{\partial r_{t+1}}{\partial\mathbf{x}_{t+1}} + \gamma\lambda_{\phi'}(\mathbf{s}_{t+1})\right]\frac{\partial\mathbf{x}_{t+1}}{\partial\mathbf{a}_t}\frac{\partial\mathbf{a}_t}{\partial\theta} \\ &= -\left[\frac{\partial r_{t+1}}{\partial\mathbf{x}_{t+1}} + \gamma\lambda_{\phi'}(\mathbf{s}_{t+1})\right]\mathbf{G}_{t-1}\frac{\partial\mathbf{a}_t}{\partial\theta} \end{aligned} \tag{13}$$

$$\theta_{t+1} = \theta_t - \eta_a\nabla_\theta\mathcal{L}_\pi \tag{14}$$

The gradient of $\mathcal{L}_\pi$ is calculated by simply taking the derivative of the loss function with respect to the observation $\mathbf{x}_t$, and multiplying it by $\frac{\partial\mathbf{x}_{t+1}}{\partial\mathbf{a}_t}$ and $\frac{\partial\mathbf{a}_t}{\partial\theta}$. Since the critic of the IDHP framework outputs the derivative of the value function $\frac{\partial V(\mathbf{s}_t)}{\partial\mathbf{x}_t} = \lambda_{\phi'}(\mathbf{s}_t)$, instead of the value function, we do not need to backpropagate through the critic network, and can use its output directly in the gradient calculation as shown in Equation 13. This co-state representation simplifies the gradient computation. Additionally, the derivative $\frac{\partial\mathbf{x}_{t+1}}{\partial\mathbf{a}_t}$ can be replaced by the control effectiveness matrix $\mathbf{G_{t-1}}$ as per

its definition. Using this incremental control effectiveness matrix enables efficient policy updates without requiring a full system model. Finally, the term $\frac{\partial \mathbf{a}_t}{\partial \theta}$ is obtained through backpropagation through the actor network. The policy is then updated by performing gradient ascent on the parameters, as seen in Equation 14, where η_a represents the learning rate of the Actor network, which controls how aggressive updates will be.

3. IDHP Critic

The critic in the IDHP framework approximates the derivative of the value function with respect to the state, defined as $\lambda_\phi(\mathbf{s}_t) = \frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{x}_t}$, where ϕ are the parameters of the critic network. The critic neural network takes as input the observation states, the tracking error, and the previous action, and, at each timestep, calculates the temporal-difference (TD) error δ_t. The TD error measures the discrepancy between the predicted and observed value estimates. This error is calculated by subtracting the current value estimate $V(\mathbf{s}_t)$ from the next value estimate, the TD target $r_{t+1} + \gamma V(\mathbf{s}_{t+1})$, as shown in Equation 15

$$\delta_t = r_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t) \tag{15}$$

Where r_{t+1} is the reward function. The loss of the critic is defined as the mean squared error of the state derivative of the TD error $\frac{\partial \delta_t}{\partial \mathbf{x}_t}$, shown in Equation 16 and Equation 17.

$$\mathcal{L}_\lambda = \frac{1}{2}\left(-\frac{\partial \delta_t}{\partial \mathbf{x}_t}\right)\left(-\frac{\partial \delta_t}{\partial \mathbf{x}_t}\right)^T \tag{16}$$

$$\frac{\partial \delta_t}{\partial \mathbf{x}_t} = \left[\frac{\partial r_{t+1}}{\partial \mathbf{x}_{t+1}} + \gamma \lambda_{\phi'}\left(\mathbf{s}_{t+1}\right)\right]\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}} - \lambda_\phi\left(\mathbf{s}_t\right) \tag{17}$$

Taking the state derivative of the TD error in Equation 15 results in Equation 17, in this equation, $\lambda_{\phi'}(\mathbf{s}_{t+1})$ represents the target critic value. Additionally, the state derivative of the reward, $\frac{\partial r_{t+1}}{\partial \mathbf{x}_{t+1}}$, is computed analytically from the reward function, and the term $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$ is computed using the incremental model, shown in Equation 18.

$$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} = \mathbf{F}_{t-1} + \mathbf{G}_{t-1}\frac{\partial \mathbf{a}_t}{\partial \mathbf{x}_t} \tag{18}$$

Where $\frac{\partial \mathbf{a}_t}{\partial \mathbf{x}_t}$ can be obtained by back-propagation through the Actor network. Like the Actor, the Critic is also updated using gradient descent, the gradient of the loss function with respect to the critic parameters ϕ is given in Equation 19. This gradient propagates the TD error through the critic network.

$$\nabla_\phi \mathcal{L}_\lambda = \frac{\partial \mathcal{L}_\lambda}{\partial \phi} = \frac{\partial \mathcal{L}_\lambda}{\partial \lambda_\phi(\mathbf{s}_t)}\frac{\partial \lambda_\phi(\mathbf{s}_t)}{\partial \phi} = \frac{\partial \delta_t}{\partial \mathbf{x}_t}\frac{\partial \lambda_\phi(\mathbf{s}_t)}{\partial \phi} \tag{19}$$

Finally, the update is performed using:

$$\phi_{t+1} = \phi_t - \mu_c \nabla_\phi \mathcal{L}_\lambda \tag{20}$$

$$\phi'_{t+1} = \tau\phi_t + (1-\tau)\phi'_t \tag{21}$$

Where μ_c represents the learning rate of the critic network. Previous research indicates that improving learning stability is also crucial, particularly for flight control applications. Thus, the critic weights are not updated using gradient descent; instead, the soft update rule in Equation 21 is used. Where τ is a set constant ensuring smoothness by combining the previous weights ϕ_t with the newly calculated ones ϕ'_t, reducing oscillations caused by rapidly changing critic targets.

C. Soft Actor-Critic

The Soft Actor-Critic (SAC) algorithm is an off-policy Deep RL algorithm designed for continuous control problems. It learns a stochastic policy, enabling consistent exploration of the action space and improving generalization. SAC is formulated within the framework of maximum entropy reinforcement learning [30].

Unlike conventional RL approaches that maximize the expected return, SAC augments the objective with an entropy term. Entropy measures the randomness of the policy's probability distribution; a higher entropy increases exploration. Entropy-regularized objectives have shown to display increased learning stability and reduce hypersensitivity to hyperparameter tuning [31]. In addition, the off-policy nature of the SAC algorithm improves sample efficiency, as the agent can learn from a buffer of past interactions [30]. The actor in the SAC framework estimates a stochastic control policy, while the critic learns an entropy-regularized action-value function that guides policy improvement.

1. SAC Actor

The SAC actor learns a stochastic policy by solving an entropy-regularized dual maximization problem. This encourages continuous exploration throughout training, preventing early convergence to deterministic policies and improving tracking performance and training stability [21]. Because the policy is stochastic and optimized in a continuous action space, SAC is well-suited to control tasks such as attitude tracking, where smooth exploration during training improves the policy [32].

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \left(r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t))\right)\right] \tag{22}$$

The optimal policy π^* is defined in Equation 22, where $\mathcal{H}(\pi(\cdot|\mathbf{s})) = -\mathbb{E}_{\mathbf{a}\sim\pi}[\log \pi(\mathbf{a}|\mathbf{s})]$ denotes the policy entropy, and α is a temperature parameter controlling the exploration–exploitation trade-off. To optimize this objective, the actor network minimizes the negative soft value function, which leads to the following actor loss:

$$\mathcal{L}_{\pi} = -V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim\pi}\left[\alpha \log \pi(\mathbf{a}_t \mid \mathbf{s}_t) - Q(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{23}$$

In Equation 23, the policy is encouraged to select actions that minimize the critic's action value while maintaining sufficient entropy. The entropy coefficient α may be adapted over time to track a target level, thereby improving robustness in hyperparameter tuning and learning performance in flight control applications [21].

The former loss equation can lead to highly oscillatory control signals; thus, the Conditioning for Action Policy Smoothness (CAPS) regularization method has been implemented by Teirlinck [26] and Vieira [27] to reduce abrupt changes in actuator inputs, enhancing stability in flight control applications. A temporal regularization loss and a spatial regularization loss, defined in Equation 24 and Equation 25 respectively, are combined with Equation 23 in order to obtain the complete actor loss expression shown in Equation 26. In this equation, λ_T and λ_S are the weights of their corresponding regularization terms.

$$\mathcal{L}_T(\pi(\mathbf{s}_t, \mathbf{s}_{t+1})) = || \pi(\mathbf{s}_t) - \pi(\mathbf{s}_{t+1}) ||_2 \tag{24}$$

$$\mathcal{L}_S = D(\pi(\mathbf{s}_t, \bar{\mathbf{s}})) = ||\pi(\mathbf{s}_t) - \pi(\bar{\mathbf{s}})||_2 \tag{25}$$

$$\mathcal{L}_{\pi}^{\text{CAPS}} = \mathcal{L}_{\pi} + \lambda_T \mathcal{L}_T + \lambda_S \mathcal{L}_S \tag{26}$$

2. SAC Critic

The SAC critic approximates the soft action-value function $Q^{\pi}(\mathbf{s}, \mathbf{a})$ using two independent Q-networks to mitigate positive bias during value estimation [33]. The critic is implemented as a Deep Neural Network (DNN) parametrized by weights ϕ. For a given transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$, the critic target is constructed using a one-step Bellman backup combining the immediate reward and the discounted value of the next state:

$$\begin{aligned} y(\boldsymbol{s}_t, \boldsymbol{a}_t) &= r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1}\sim p}\left[V(\boldsymbol{s}_{t+1})\right] \\ &= r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \min_{i=1,2}\left[Q_{\phi_{\text{targ},i}}(\boldsymbol{s}_{t+1}, \tilde{\boldsymbol{a}}_{t+1}) - \alpha \log \pi_{\theta}(\tilde{\boldsymbol{a}}_{t+1} \mid \boldsymbol{s}_{t+1})\right] \end{aligned} \tag{27}$$

The state-value function $V(\mathbf{s}_t)$ appearing in Equation 27 is not learned explicitly as a separate network. Instead, it is defined in Equation 28 as the expectation of the action-value function under the current policy, minus the entropy term. Substituting this into Equation 27 yields a fully entropy-consistent Bellman backup without introducing an additional value network.

$$V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t\sim\pi}\left[Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)\right] \tag{28}$$

Each critic network is trained by minimising the squared temporal-difference error between the predicted Q-value and the target value, as expressed in Equation 29.

$$\mathcal{L}_Q = \sum_{i=1,2}\left[Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - y(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\right]^2 \tag{29}$$

Finally, to increase learning stability, the target critic networks $Q_{\phi_{\text{targ},i}}$ in Equation 27 are updated more slowly than the critic networks using a soft update rule. This stabilises training by preventing rapid changes in the target values. This

combination of entropy-aware targets, double critics, and delayed target updates enables the SAC critic to learn stable and reliable value estimates in complex control problems.

D. Distributional Soft Actor-Critic

The Distributional SAC (DSAC) framework extends SAC by replacing the scalar action-value function with a return distribution, allowing the critic to capture information about the mean and variability of future returns [23]. Instead of approximating $Q(\mathbf{s}, \mathbf{a})$, the critic approximates the random variable $Z(\mathbf{s}, \mathbf{a})$, which is defined as the discounted cumulative reward, represented in Equation 30.

$$Z(s,a) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{30}$$

$$Q(\mathbf{s}, \mathbf{a}) = \Psi[Z(\mathbf{s}, \mathbf{a})] \tag{31}$$

The risk distribution can be represented as a real-valued action-value function through the transformation $\Psi[]$, where the expectation operator $\mathbb{E}[]$ can be used. In this work, Wang's risk measure was employed, similar to the approach in [27].

The critic network in the DSAC framework approximates the distribution of returns shown in Equation 30. Learning the future distribution allows the critic to quantify the expected performance and uncertainty associated with future outcomes. This is beneficial in flight control tasks where disturbances, model mismatch, or faults can introduce uncertainty in the system response.

In practice, the return distribution is approximated using a discrete quantile representation, where the critic approximates $Z_\phi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}; \tau_i)$, where τ_i represents the i-th quantile, and ϕ the parameters of the Neural Network. The quantile function is the inverse of the cumulative distribution function (c.d.f) of the random return for a specific quantile.

The Bellman backup for DSAC is extended to the distributional setting by replacing the action-value function with the return distribution.

$$y = r_t + \gamma \left[Z_{\bar{\phi}_{min}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha \log \pi_{\bar{\theta}}(\mathbf{a}_{t+1}|\mathbf{s}_{t+1}) \right] \tag{32}$$

Where $\mathbf{a}_{t+1}$ is sampled from the target policy $\pi_{\bar{\theta}}$ and $Z_{\phi_{min}}$ denotes the return distribution belonging to the critic that outputs the smaller value estimate, adhering to the clipped double Q-learning principle used before. This bellman backup is used to calculate the TD error between each target quantile y_i and each predicted quantile Z_j.

$$\delta_{ij}^t = r_t + \gamma \left[Z_{\bar{\phi}} \left(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}; \hat{\tau}_i \right) - \log \pi_{\bar{\theta}} \left(\boldsymbol{a}_{t+1}, \boldsymbol{s}_{t+1} \right) \right] - Z_\phi \left(\boldsymbol{s}_t, \boldsymbol{a}_t; \hat{\tau}_j \right) \tag{33}$$

Where $\hat{\tau}$ is the average between two subsequent quantiles $\hat{\tau}_i = (\tau_i + \tau_{i+1})/2$. The pairwise temporal difference between all predicted and target quantiles serves as an intermediate construct required for quantile regression using Huber loss [23]. For a given quantile fraction τ, the Huber loss is defined as:

$$\rho_\tau(\delta) = |\tau - \mathbb{I}[\delta < 0]| \mathcal{L}^\kappa(\delta) \tag{34}$$

Where $\mathbb{I}[]$ is the indicator function, and $\mathcal{L}^\kappa(\delta)$ is the Huber loss with threshold κ defined in Equation 35. The Huber loss behaves quadratically for small errors and linearly for large errors. This provides smooth gradient clipping and improves numerical stability during training. This feature helps prevent unstable updates in the critic when compared to SAC.

$$\mathcal{L}^\kappa(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq \kappa \\ \kappa\left(|\delta| - \frac{1}{2}\kappa\right) & \text{otherwise} \end{cases} \tag{35}$$

$$\mathcal{L}_Z = \mathbb{E}\left[\frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \rho_{\tau_j}^\kappa(\delta_{ij}^t) \right] \tag{36}$$

The overall critic loss is then obtained by averaging the quantile regression Huber loss over all predicted-target quantile pairs as seen in Equation 36. Minimizing this loss ensures that each predicted quantile converges to the corresponding

quantile of the true return distribution. This allows the critic to learn both the expected value and the uncertainty of future events.

E. Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC)

RUN-DSAC builds on the DSAC framework by exploiting the uncertainty encoded in the return distribution. While DSAC improves learning stability by modelling the full return distribution, it still selects actions using a scalar risk-adjusted distribution. RUN-DSAC refines this method by explicitly selecting actions based on the quantified uncertainty, allowing the agent to modulate its behaviour based on both the expected performance and the confidence in that performance. This enhances sample efficiency, learning consistency, and safety when dealing with complex flight control tasks [24].

Given the distributional critic $Z(\mathbf{s}, \mathbf{a})$, the expected value and standard deviation of the return are computed as:

$$Q_\theta(\mathbf{s}, \mathbf{a}) = \mathbb{E}[Z_\theta(\mathbf{s}, \mathbf{a})] \tag{37}$$

$$\sigma_Q(\mathbf{s}, \mathbf{a}) = \sqrt{\mathbb{V}[Z_\theta(\mathbf{s}, \mathbf{a})]} \tag{38}$$

These quantities are combined into an uncertainty-aware action-value function

$$Q_V(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, \mathbf{a}) + \mu\sigma_Q(\mathbf{s}, \mathbf{a}) \tag{39}$$

Where the coefficient μ determines the agent's sensitivity to return uncertainty. Positive values of μ encourage risk-seeking behaviour, while negative values promote risk-averse action selection.

The uncertainty-aware value $Q_V(\mathbf{s}, \mathbf{a})$ replaces the standard expected value in the critic evaluation and policy update, allowing the agent to account for both performance and uncertainty during learning. This mechanism improves robustness and learning efficiency, particularly in environments subject to disturbances, modelling errors, or fault conditions. In flight control applications, RUN-DSAC has demonstrated improved tracking performance and fault tolerance compared to SAC and DSAC, without increasing algorithmic complexity [24].

III. Methodology

A. Simulation Environment and Control Task

The proposed controlled framework is evaluated using a high-fidelity simulation of the Cessna 550 Citation II aircraft. This environment is identical to that used in prior work on RL-based hybrid controllers in Teirlinck [26] and Vieira [27]. The aircraft model was discretized at a sampling frequency of 100 Hz, and trimmed at an altitude of 2000 m and an airspeed of 90 ms^{-1}. From this trim point, the controller is tasked with tracking time-varying attitude reference signals while maintaining stable flights.

The state vector $\mathbf{x}$ and input vector $\mathbf{u}$ are defined below:

$$\mathbf{x} = [p, q, r, V_{TAS}, \alpha, \beta, \theta, \phi, \psi, h_e, x_e, y_e]^T \tag{40}$$

$$\mathbf{u} = [\delta_e, \delta_a, \delta_r, T_{N_1}, T_{N_2}]^T \tag{41}$$

However, the observation vector $\mathbf{s}$ varies depending on the RL algorithm used. Limiting the observation space to dynamically relevant quantities improves sample efficiency during controller training. The observation, action, and tracking error formulations are shown below:

$$\mathbf{s}_{\text{online}} = [p, q, r, \alpha, \theta, \phi, \beta, \mathbf{e}_t^{att}]^T \tag{42}$$

$$\mathbf{s}_{\text{offline}} = [p, q, r, \mathbf{e}_t^{att}]^T \tag{43}$$

$$\mathbf{a} = [\delta_e, \delta_a, \delta_r]^T \tag{44}$$

$$c^{\text{att}} = \frac{180}{\pi}\begin{bmatrix} \frac{1}{30} & \frac{1}{30} & \frac{1}{7.5} \end{bmatrix}^T \tag{45}$$

$$\boldsymbol{e}^{\text{att}} = \left(\boldsymbol{x}^{\text{att}} - \boldsymbol{x}^{\text{ref}}\right) \times \boldsymbol{c}^{\text{att}} \tag{46}$$

Where p, q, and r denote the body angular rates, α the angle of attack, θ and ϕ the pitch and roll angles, β the sideslip angle. The observation vector for the online agents omits the translational and positional states (V_{TAS}, h_e, x_e, y_e, and ψ). For the offline agents, the attitude angles and the angle of attack are also excluded as seen in Equation 42 and Equation 43, respectively. The action vector excludes thrust components, as seen in Equation 44. Finally, the tracking

error is defined as the difference between the commanded and measured attitude states, and is scaled to balance the contributions of each controlled variable in the reward function, as shown in Equation 46 and Equation 45.

$$r_{t+1}^{\text{online}} = -\boldsymbol{e}_t^{\text{att}} \times \boldsymbol{e}_t^{\text{att}} \tag{47}$$

$$r_{t+1}^{\text{offline}} = -\left\| \text{clip}\left[\boldsymbol{e}_t^{\text{att}}, -\overrightarrow{\mathbf{1}}, \overrightarrow{\mathbf{1}}\right] \right\| \tag{48}$$

The rewards for each agent type are given in Equation 47 and Equation 48 for online and offline agents, respectively.

B. The Hybrid RUN-DSAC-IDHP Architecture

Purely offline deep reinforcement learning controllers, such as SAC, DSAC, and RUN-DSAC, learn policies through exhaustive interaction with the simulation environment [21, 23, 24]. This exhaustive interaction allows the agent to explore a wide range of operating conditions and reach policies with strong generalization capabilities across the flight envelope. Yet, once deployed, these controllers are inherently non-adaptive and may perform poorly under changing dynamics, varying parameters, or actuator faults [26].

On the other hand, online adaptive control approaches such as IDHP are capable of real-time adaptation, but are sensitive to policy initialization and require precise hyperparameter tuning. This can lead to large transient errors and even the loss of control.

The hybrid offline-online reinforcement learning architecture aims to combine the strengths of both approaches: the robustness and high generalization capabilities of offline learning with the adaptability of online learning. In this work, the hybrid controller is designed by initializing an IDHP agent using a pre-trained offline policy obtained from the RUN-DSAC algorithm and allowing the IDHP controller to adapt incrementally during online operation.

1. Hybrid Agent Architecture

The hybrid controller is implemented as a single composite policy network composed of a frozen offline-trained subnetwork and a trainable IDHP actor layer, following the approach described by Teirlinck [26] and Vieira [27]. Rather than operating as two sequential controllers, the offline and online components are integrated into a unified policy representation. This structure preserves the stability and prior knowledge of the offline policy while allowing the online layer to perform incremental corrections, avoiding the instability and coordination issues that may arise when combining multiple parallel controllers.

The offline policy obtained after training the RUN-DSAC agent is embedded into the IDHP actor as a fixed nonlinear mapping. Its role is to transform the input state into a baseline control representation. This policy remains frozen during the online training phase. The IDHP actor then uses a single shallow trainable layer to map the offline policy's output to the final control action. Consequently, the hybrid policy can be expressed as:

$$\mathbf{u}_t = \tanh(W_{\text{IDHP}}\, \pi_{\text{off}}(\mathbf{s}_t)) \tag{49}$$

Where π_{off} represents the frozen offline policy and W_{IDHP} represents the trainable IDHP parameters. This structure, illustrated in Figure 1, represents a single hybrid policy network in which the offline policy provides a frozen non-linear feature mapping, while the IDHP actor generates the final control action.

2. Hybrid Agent Training

The training of the hybrid agent is divided into offline and online phases. In the offline phase, the RUN-DSAC policy is learned through extensive interaction with the simulation environment using the procedure described in Algorithm 2 in section V. Once the training converges, the policy parameters are frozen and transferred to the hybrid controller.

During online operation, only the IDHP components are updated. The incremental model estimates the local system dynamics, the critic approximates the gradient of the value function, and the actor parameters are updated accordingly, as described in Algorithm 1 in section V. The offline policy remains fixed throughout this process and does not receive gradient updates.

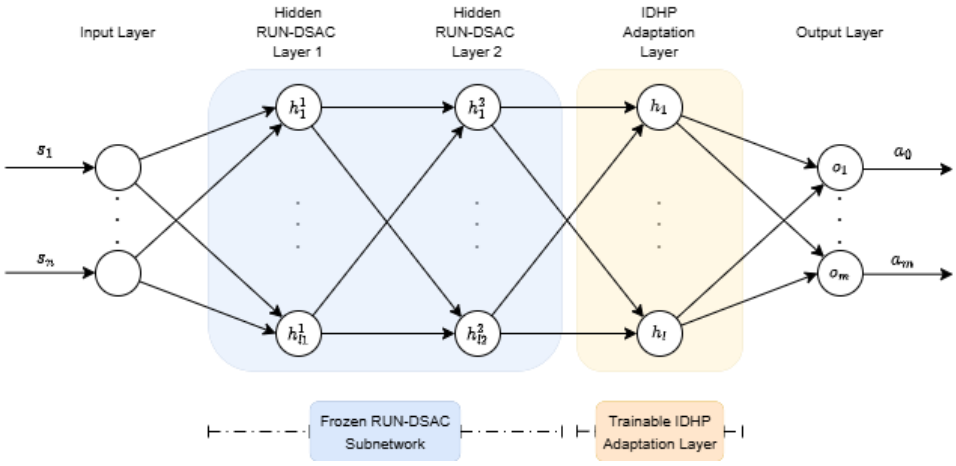


Fig. 1 RUN-DSAC-IDHP Hybrid Policy Architecture

By restricting online learning to the IDHP components, the hybrid controller maintains the improved generalization capabilities of the offline-trained policy, while being able to adapt in real-time to changing dynamics or fault conditions.

C. Experimental Framework and Analysis

To evaluate the performance of the proposed hybrid reinforcement learning framework, a series of simulation-based experiments is conducted using the high-fidelity model of the Cessna 550 Citation II aircraft. These experiments aim to assess the tracking performance and adaptability of the developed agents under both nominal and faulty operating conditions, and to enable a fair comparison between the offline, online, and hybrid approaches.

The task for all controllers is to track predefined attitude signals using the elevator, aileron, and rudder control surfaces. The simulation is executed with a fixed timestep of $\Delta t = 0.01$ s over a 30-second time horizon, and the environment enforces actuator limits and rate constraints.

The evaluated controllers include an online-only adaptive controller based on Incremental Dual Heuristic Programming (IDHP), three offline reinforcement learning controllers trained using Soft Actor-Critic (SAC), Distributional Soft Actor-Critic (DSAC), and Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC), and three hybrid controllers obtained by combining IDHP with pretrained SAC, DSAC, and RUN-DSAC policies. The hybrid controllers are denoted as SAC-IDHP, DSAC-IDHP, and RUN-DSAC-IDHP, respectively.

For the offline agents, the learned policies are executed deterministically during the evaluation and remain fixed during the simulation. In contrast, the online IDHP agent and the hybrid agents continuously update their parameters during execution. Yet, for the hybrid agents, only the IDHP parameters are updated online, while the offline policy remains frozen.

Each controller is evaluated under a set of predefined fault scenarios. In addition to a nominal baseline case without failures, the fault conditions listed in Table 2 are considered.

These faults are standard failure modes, previously used in the work by Teirlinck [26]. To account for stochastic effects, each experiment is repeated using eight random seeds. The performance of the controllers is evaluated by observing the time responses and measuring the normalized mean absolute error (nMAE) between the tracking signals and the actual response for the three tracked variables. The nMAE is averaged over time across all seeds. Failed runs, where the aircraft states diverge, are recorded separately.

The results from this set of experiments are presented and discussed in the following section, where the performance of the proposed hybrid controller RUN-DSAC-IDHP is compared against the other hybrid agents, as well as their offline-only and online-only counterparts.

Table 2 Evaluated Fault Scenarios

Fault Case	Description
Nominal	No failure
dr_stuck	Rudder jammed at $-15°$
da_reduce	Aileron effectiveness reduced to 10%
da_limit	Aileron deflection limited to $\pm 5°$
de_reduce	Elevator effectiveness reduced to 30%
de_reduce_extreme	Elevator effectiveness reduced to 10%
de_limit	Elevator deflection limited to $\pm 2.5°$
de_invert	Inverted elevator response
ht_reduce	Horizontal tail effectiveness reduced to 70%
icing	Aerodynamic icing effects enabled

IV. Result and Discussion

This section presents and discusses the performance of the proposed hybrid RUN-DSAC-IDHP framework compared to baseline offline, online, and hybrid agents. The results first establish baseline behaviour under nominal conditions, followed by significant results from some of the evaluated fault scenarios. Performance is assessed in terms of tracking accuracy, using the normalized mean absolute error (nMAE) as a metric, and qualitatively by analyzing the time responses of the control inputs and the system. This section aims to showcase the capabilities and limitations of the proposed framework.

A. Nominal Case

Table 3 Normalized Mean Absolute Error (nMAE) under nominal conditions

Agent	θ nMAE	ϕ nMAE	β nMAE	total nMAE
SAC	12.28%	4.87%	1.74%	6.30%
DSAC	13.63%	11.02%	3.41%	9.35%
RUN-DSAC	12.15%	10.81%	1.95%	8.30%
SAC-IDHP	3.49%	1.68%	1.37%	2.18%
DSAC-IDHP	4.52%	1.95%	5.66%	4.04%
RUN-DSAC-IDHP	2.96%	2.21%	2.75%	2.64%

1. Baseline Controllers: IDHP and Offline Agents

Figure 2 compares the time response of the standalone IDHP controller with the three offline-trained agents under nominal conditions. The IDHP agent alone is unable to stabilize the aircraft and regulate the body rates. It shows decent tracking of the pitch (θ) and roll (ϕ) reference signals' general shapes, yet exhibits significant oscillations during tracking, and the control signals are highly oscillatory. This behaviour is consistent with prior IDHP-based studies, which showed that this method was well-suited for body-rate tracking but less effective for directly tracking attitude angles [13]. This can be explained by the fact that attitude tracking introduces integration, slower dynamics, and stronger coupling, which are less well-suited to IDHP's capabilities. As a result, further tuning improves tracking of one of the reference signals but does not resolve the tracking limitations.

On the other hand, all three offline agents achieve accurate and stable attitude tracking. SAC displays the tightest tracking but exhibits aggressive, oscillatory control inputs, particularly at the beginning of the response for the elevator (δ_e) and rudder (δ_r) deflections. DSAC shows the largest deviation from the reference signal, which can be linked to the added complexity of learning the full return distribution via quantile regression. This can introduce noisier value

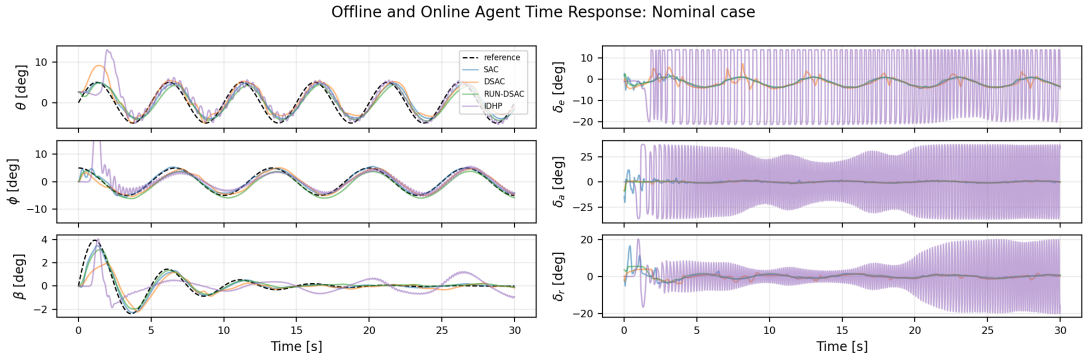


Fig. 2 Tracked States and Control Signal Response with IDHP, SAC, DSAC, and RUN-DSAC under Nominal Conditions

gradients during training, which is reflected in the oscillatory response of the control surface deflection. RUN-DSAC differs from DSAC in that, instead of using a risk distortion, it augments the action-value estimate with an uncertainty parameter μ that makes the agent risk-prone. The resulting policy yields smoother control inputs than SAC and DSAC while maintaining accurate tracking, as shown in Figure 2.

2. Hybrid Controllers: SAC, DSAC, and RUN-DSAC with Online IDHP Adaptation

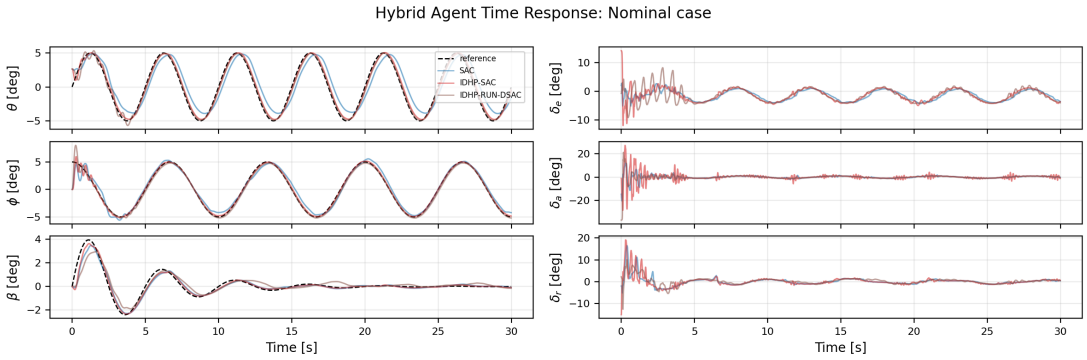


Fig. 3 SAC, SAC-IDHP, and RUN-DSAC-IDHP Time Response Under Nominal Conditions

Figure 3 shows the response of SAC-IDHP, RUN-DSAC-IDHP and a baseline SAC. DSAC-IDHP is not shown for clarity, but displays a response similar to SAC-IDHP. In the case of these two, the hybrid controllers track the reference signals more accurately than their offline counterparts, particularly in the steady state regime. This demonstrates that the online IDHP adaptability effectively enhances tracking performance by adjusting the initial offline policy to the observed system dynamics. Yet, this improvement in tracking accuracy is also accompanied by increased oscillations in the control signals, particularly during the initial seconds of the oscillation. These occur because online updates to the IDHP actor and critic temporarily amplify control corrections before converging on an optimal policy. In the case of SAC, this oscillatory behaviour decays over time, indicating stable learning. However, the oscillations in the DSAC-IDHP agent persist over time due to the greater control variability in its offline form. It is also important to note that although the control signals display somewhat oscillatory behaviour, this behaviour is always bounded and never reaches the actuator limits.

The behaviour of RUN-DSAC-IDHP differs from the previous two hybrids, as show in Figure 3. Compared to offline RUN-DSAC, the hybrid agent displays better tracking accuracy while maintaining a relatively smooth control response. Unlike DSAC-IDHP, the oscillations introduced by the online agent are damped after the initial phase of the simulation. This indicates that the risk-prone RUN-DSAC policy provides a better initialization for the online IDHP controller, allowing the hybrid agent to benefit from more accurate tracking without excessive amplification of control variability.

Under nominal conditions, this agent trades off tracking performance and control smoothness.

B. Fault Scenarios

In addition to the nominal case, the results for four of the analysed fault scenarios are presented in the following section.

1. Elevator Saturation ($\delta_e \in [-2.5°, 2.5°]$)

Table 4 Normalized Mean Absolute Error (nMAE) under elevator saturation

Agent	θ nMAE	ϕ nMAE	β nMAE	total nMAE
SAC	17.07%	6.65%	1.96%	8.56%
DSAC	18.31%	11.24%	3.26%	10.93%
RUN-DSAC	16.50%	10.76%	1.94%	9.73%
SAC-IDHP	11.38%	4.85%	4.20%	6.81%
DSAC-IDHP	12.03%	2.53%	7.18%	7.25%
RUN-DSAC-IDHP	11.10%	2.88%	3.99%	5.99%

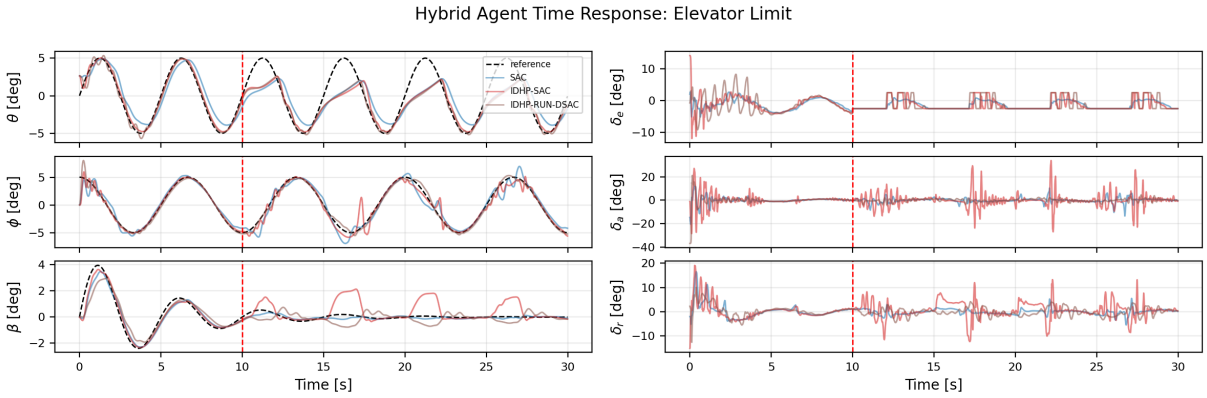


Fig. 4 SAC, SAC-IDHP, and RUN-DSAC-IDHP Time Response Under Elevator Saturation Fault (Vertical Red Line Indicates the Start of the Fault)

Figure 4 shows the response of the SAC-based and RUN-DSAC-based offline and hybrid controllers under an elevation saturation fault. As seen in Table 4, RUN-DSAC-IDHP has the lowest overall tracking error among all the evaluated agents, followed by SAC-IDHP and DSAC-IDHP.

The actuator constraints prevent full tracking of the pitch reference signal across all controllers. Nonetheless, Figure 4 shows the offline agent's ability to generalize and be able to maintain reasonable tracking performance during a fault that these agents were not trained for. In particular, tracking of the sideslip reference was superior for the offline agents than for the hybrids, as the hybrids tend to excite additional coupling effects when attempting to compensate for the limited elevator authority.

Augmenting SAC with online IDHP adaptation improves pitch and roll tracking accuracy, yet this also significantly increases control oscillations. These are more pronounced in SAC-IDHP than in RUN-DSAC-IDHP because the SAC baseline's more aggressive nature amplifies corrective actions when the elevator is saturated. In Figure 4, it is seen that RUN-DSAC-IDHP shows improved tracking performance while maintaining lower magnitude oscillations in the control inputs, benefitting from the better-conditioned offline initialization provided by the RUN-DSAC offline baseline.

Importantly, none of these controllers fully mitigates the fault; the observed tracking limitations are caused by the saturated actuator, which can be seen in the top right plot of Figure 4.

2. Rudder Stuck at $-15°$

Table 5 Normalized Mean Absolute Error (nMAE) under rudder stuck fault

Agent	θ nMAE	ϕ nMAE	β nMAE	total nMAE
SAC	56.78%	93.21%	65.52%	71.84%
DSAC	250.97%	559.60%	88.38%	299.55%
RUN-DSAC	103.92%	33.17%	64.11%	67.07%
SAC-IDHP	23.87%	20.28%	63.08%	35.74%
DSAC-IDHP	5.97%	2.95%	65.11%	24.68%
RUN-DSAC-IDHP	14.91%	7.83%	63.29%	28.68%

One of the more limiting fault scenarios occurred when the rudder got stuck at $-15°$ after 10s of simulation. Figure 5 compares all three offline agents under this fault. All three agents fail to track the reference signal once the failure occurs. SAC achieves the best pitch tracking, while RUN-DSAC performs better in roll, showing different exploitation of control coupling across agents. None of the agents is able to maintain accurate sideslip tracking, as the loss of rudder control limits control over β.

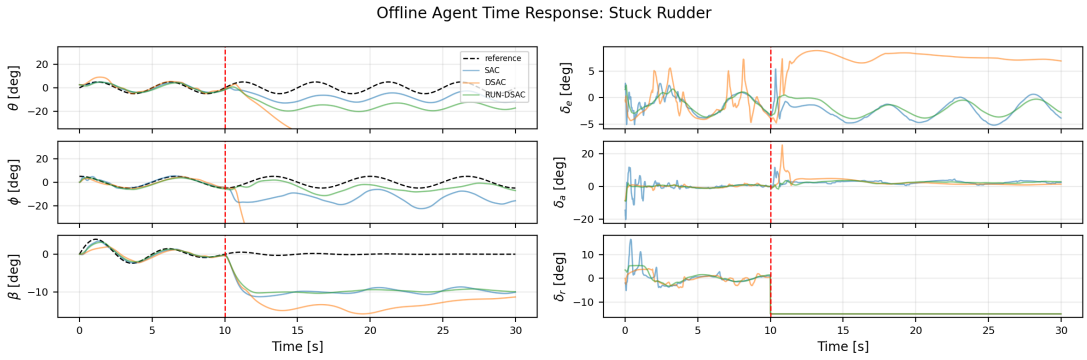


Fig. 5 Tracked States and Control Signal Response for Offline Agents under $\delta_r = -15°$

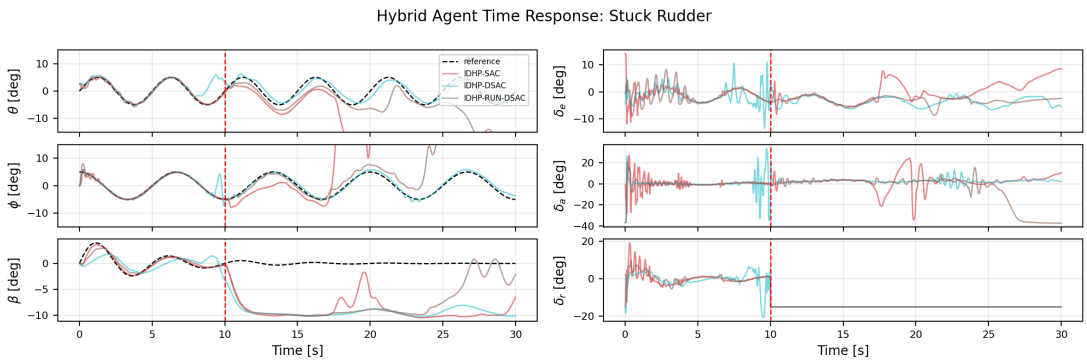


Fig. 6 Tracked States and Control Signal Response for Hybrid Agents under $\delta_r = -15°$

The corresponding hybrid responses are shown in Figure 6. In contrast to the offline controllers, DSAC-IDHP and RUN-DSAC-IDHP significantly improve tracking of the pitch and roll reference signals after the fault, indicating that the

online IDHP adaptation can compensate for the loss of rudder authority. Among the hybrid controllers, DSAC-IDHP achieves the lowest tracking error, followed by RUN-DSAC-IDHP. In this scenario, the DSAC agent's more aggressive and variable behaviour appears to provide greater freedom for the IDHP to adjust the policy online, whereas the smoother RUN-DSAC initialization leads to more constrained but stable adaptation. Like the offline agents, none of the hybrids can recover sideslip tracking due to loss of rudder actuator control, indicating that the agent does not fully mitigate the fault but partially reduces its effects through online adaptation.

3. Horizontal Tail Reduction (30%)

Table 6 Normalized Mean Absolute Error (nMAE) under horizontal tail reduction

Agent	θ nMAE	ϕ nMAE	β nMAE	total nMAE
SAC	16.76%	7.26%	2.00%	8.68%
DSAC	23.58%	12.05%	3.25%	12.96%
RUN-DSAC	23.23%	10.86%	1.80%	11.96%
SAC-IDHP	5.78%	2.90%	1.60%	3.42%
DSAC-IDHP	8.95%	2.48%	5.60%	5.68%
RUN-DSAC-IDHP	6.76%	2.53%	3.09%	4.13%

The next fault case evaluated occurs when the horizontal tail is reduced by 30%. As shown in Table 6, SAC-IDHP achieves the best tracking performance; however, the control response of this agent is characterized by high-frequency, high-magnitude oscillations across all three control surfaces over the 30-second period. The next best tracking accuracy is achieved by the RUN-DSAC-IDHP agent. As seen in Figure 7, the offline controller exhibits degraded pitch tracking following the fault. The hybrid RUN-DSAC-IDHP controller rapidly compensates for this fault by increasing the elevator actuator deflection.

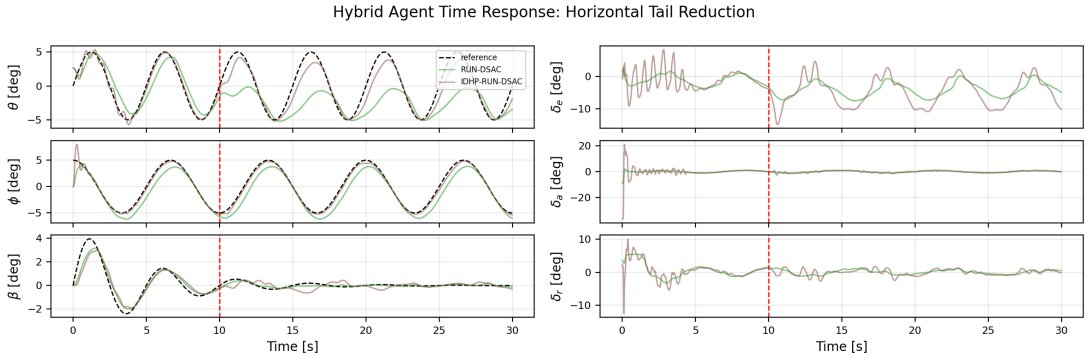


Fig. 7 Tracked States and Control Signal Response for RUN-DSAC and RUN-DSAC-IDHP under 30% Horizontal Tail Reduction

The hybrid response is characterized by transient oscillations during the initial seconds of the simulation and of the fault, which decay rapidly after adaptation converges. Unlike in the actuator saturation case with limited elevator authority, this fault is effectively mitigated by the hybrid agent, and nominal performance is restored despite the loss of control effectiveness. Both the offline and hybrid agents adapt to this fault, yet the elevator deflections exhibit greater-amplitude oscillations when the IDHP component is active. The online component of the hybrid agent learns a new policy that requires larger elevator deflection angles to compensate for the loss of horizontal-tail effectiveness, thereby achieving accurate tracking. This demonstrates the fault-mitigation capabilities of the hybrid architecture.

4. Icing

The final fault analysed is the aerodynamic effects of icing. This fault was simulated in the Cessna 550 Citation II simulation environment [34]. As shown in Table 7, the novel RUN-DSAC-IDHP agent achieves the best tracking

Table 7 Normalized Mean Absolute Error (nMAE) under icing conditions

Agent	θ nMAE	ϕ nMAE	β nMAE	total nMAE
SAC	12.36%	6.13%	2.00%	6.83%
DSAC	13.76%	11.63%	3.39%	9.59%
RUN-DSAC	14.06%	10.81%	1.90%	8.92%
SAC-IDHP	3.98%	1.94%	1.44%	2.45%
DSAC-IDHP	5.29%	2.10%	5.70%	4.36%
RUN-DSAC-IDHP	3.84%	2.27%	2.72%	2.94%

accuracy under this condition, followed closely by SAC-IDHP. It is also evident from the table that all the hybrid agents outperform the offline agents. Figure 8 shows the response of RUN-DSAC and RUN-DSAC-IDHP under icing conditions. The hybrid agent significantly reduces the immediate impact of the fault, improving attitude tracking during the initial phase of the simulation. However, as the simulation is extended in time, tracking performance gradually degrades.

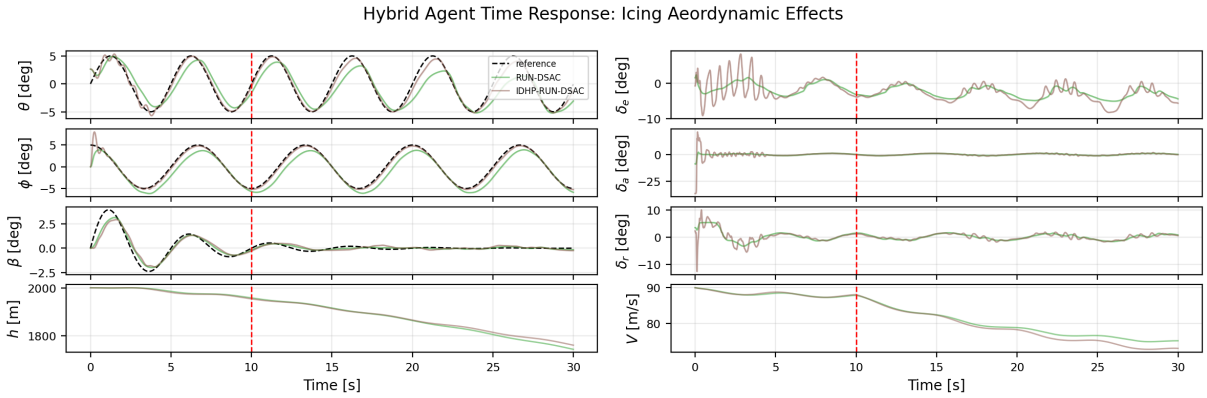


Fig. 8 Tracked States and Control Signal Response for RUN-DSAC and RUN-DSAC-IDHP under Icing Aerodynamic Effects

Inspection of the airspeed and altitude responses reveals a progressive loss of velocity that results in altitude decay. This indicates that icing primarily affects the aerodynamic performance, rather than the control authority. While the hybrid controller partially compensates for these effects through online adaptation, full mitigation would require active thrust control to counteract the reduced lift and increased drag. The hybrid agents only accommodate for this fault and reduce its effects, but don't fully mitigate it since throttle control is outside the scope of this work. This case showcases a fundamental limitation of attitude-only control when faced with aerodynamic degradation.

V. Conclusions

The results presented in this section demonstrate the effectiveness of the proposed hybrid RUN-DSAC-IDHP architecture across nominal and faulty flight conditions. Under nominal conditions, the hybrid agents all improve tracking performance relative to their offline counterparts, confirming that the online IDHP component in the hybrid agents is able to refine pre-trained control policies to better match the observed system dynamics. While SAC-IDHP achieves the lowest normalized mean absolute error, the novel RUN-DSAC-IDHP finds a balance between tracking accuracy and smooth control signals. This showcases the advantages of risk-prone offline learning in achieving a better initial offline policy, which is later fine-tuned via online adaptation.

In fault scenarios, the hybrid controllers exhibit clear advantages over purely offline controllers. For the limited-elevator and stuck-rudder faults, the hybrid agents can partially accommodate the loss of control authority after the fault,

displaying improved pitch and roll tracking despite the physical limitations imposed by the faults. In the case of the horizontal tail reduction, the hybrid RUN-DSAC-IDHP agent fully mitigates the fault, rapidly adapting to the reduced control effectiveness and restoring nominal performance. These results highlight the strength of the hybrid architecture in handling changes in control effectiveness via online learning.

However, the results also reveal significant limitations. Faults that directly constrain the actuators are never fully mitigated by attitude control alone, and the agents remain limited by these constraints. This results in errors in tracking the attitude variable for the constrained actuator. Similarly, under icing conditions, although the hybrid controllers reduce the immediate impact of aerodynamic degradation, long-term performance deteriorates due to losses in airspeed and altitude. This shows that the proposed implementation can reduce faults, particularly compared to the offline counterparts, but does not achieve complete fault recovery.

Overall, the results show that the hybrid RUN-DSAC-IDHP agent provides an adaptable and robust control strategy, suited to faults that reduce control effectiveness. The proposed algorithm's offline component also exhibits better generalization than SAC and DSAC, resulting in smoother control signals at the expense of a slight reduction in tracking performance in certain fault scenarios compared to the SAC-based agent. These findings highlight the capabilities of the proposed algorithm, but also delineate the boundaries of what can be achieved through attitude control alone.

References

[1] International Air Transport Association, "Recommendations for Accident Prevention," Tech. rep., International Air Transport Association (IATA), 2023. URL `https://www.iata.org/contentassets/95e933e1ad794068812f073cf883cb08/recommendations-for-accident-prevention---2023.pdf`, iATA Annual Safety Report.

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

[3] de Visser, C. C., Wang, X., and van Kampen, E. J., "AE4311 Nonlinear and Adaptive Flight Control – Lecture 1: Introduction and NDI," , 2024. Lecture slides.

[4] Farrell, J., Sharma, M., and Polycarpou, M., "Backstepping-based flight control with adaptive function approximation," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 6, 2005, pp. 1089–1102.

[5] Sieberling, S., Chu, Q., and Mulder, J., "Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction," *Journal of guidance, control, and dynamics*, Vol. 33, No. 6, 2010, pp. 1732–1742.

[6] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, 2nd ed., The MIT Press, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[7] Ferrari, S., and Stengel, R. F., "Online adaptive critic flight control," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 777–786.

[8] Ferrari, S., and Stengel, R. F., *Model-Based Adaptive Critic Designs*, John Wiley Sons, Ltd, 2004, Chap. 3, pp. 65–95. https://doi.org/https://doi.org/10.1002/9780470544785.ch3, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470544785.ch3.

[9] Zhou, Y., van Kampen, E., and Chu, Q. P., "Incremental approximate dynamic programming for nonlinear flight control design," *Proceedings of the EuroGNC 2015*, 2015.

[10] Zhou, Y., van Kampen, E.-J., and Chu, Q., "Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control," 2016.

[11] Zhou, Y., van Kampen, E.-J., and Chu, Q., "Nonlinear Adaptive Flight Control Using Incremental Approximate Dynamic Programming and Output Feedback," *Journal of Guidance, Control, and Dynamics*, Vol. 40, 2016, pp. 1–8. https://doi.org/10.2514/1.G001762.

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

[13] Lee, J. H., and van Kampen, E.-J., "Online reinforcement learning for fixed-wing aircraft longitudinal control," *AIAA Scitech 2021 Forum*, 2021, p. 0392.

[14] Sun, B., and van Kampen, E.-J., "Incremental Model-Based Heuristic Dynamic Programming with Output Feedback Applied to Aerospace System Identification and Control," *2020 IEEE Conference on Control Technology and Applications (CCTA)*, 2020, pp. 366–371. https://doi.org/10.1109/CCTA41146.2020.9206261.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., "Human-level control through deep reinforcement learning," *nature*, Vol. 518, No. 7540, 2015, pp. 529–533.

[16] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M., "Deterministic policy gradient algorithms," *International conference on machine learning*, Pmlr, 2014, pp. 387–395.

[17] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] van Kampen, E.-J., Chu, Q., and Mulder, J., *Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics*, ???? https://doi.org/10.2514/6.2006-6429, URL https://arc.aiaa.org/doi/abs/10.2514/6.2006-6429.

[20] Bøhn, E., Coates, E. M., Moe, S., and Johansen, T. A., "Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization," *2019 international conference on unmanned aircraft systems (ICUAS)*, IEEE, 2019, pp. 523–533.

[21] Dally, K., and van Kampen, E.-J., "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control," *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics, 2022. https://doi.org/10.2514/6.2022-2078, URL http://dx.doi.org/10.2514/6.2022-2078.

[22] Li, Y., and van Kampen, E.-J., "Reinforcement Learning-based Intelligent Flight Control for a Fixed-wing Aircraft to Cross an Obstacle Wall," *2024 European Control Conference (ECC)*, IEEE, 2024, pp. 1636–1641.

[23] Seres, P., "Distributional Reinforcement Learning for Flight Control: A risk-sensitive approach to aircraft attitude control using Distributional RL," Master's thesis, Delft University of Technology, Delft, The Netherlands, Nov. 2022. Defended November 9, 2022.

[24] Homola, M., Li, Y., and van Kampen, E.-J., "Uncertainty-Driven Distributional Reinforcement Learning for Flight Control," *AIAA SCITECH 2025 Forum*, 2025, p. 2793.

[25] Song, Y., Zhou, Y., Sekhari, A., Bagnell, J. A., Krishnamurthy, A., and Sun, W., "Hybrid rl: Using both offline and online data can make rl efficient," *arXiv preprint arXiv:2210.06718*, 2022.

[26] Teirlinck, C., "Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance," Master's thesis, Delft University of Technology, Delft, The Netherlands, Sep. 2022. URL https://resolver.tudelft.nl/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85.

[27] Vieira, L., "Evaluating Reinforcement Learning for Quadcopter Control: From Simulation to Real World," Master's thesis, Delft University of Technology, Delft, The Netherlands, Nov. 2023. Defended November 17, 2023.

[28] Heyer, S., Kroezen, D., and van Kampen, E.-J., "Online adaptive incremental reinforcement learning flight control for a CS-25 class aircraft," *AIAA Scitech 2020 Forum*, 2020, p. 1844.

[29] Werbos, P. J., "A Menu of Designs for Reinforcement Learning Over Time," *Neural Networks for Control*, The MIT Press, 1991. https://doi.org/10.7551/mitpress/4939.003.0007, URL https://doi.org/10.7551/mitpress/4939.003.0007.

[30] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *International conference on machine learning*, Pmlr, 2018, pp. 1861–1870.

[31] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S., "Soft Actor-Critic Algorithms and Applications," , 2019. URL https://arxiv.org/abs/1812.05905.

[32] Barros, G. M., and Colombini, E. L., "Using soft actor-critic for low-level UAV control," *arXiv preprint arXiv:2010.02293*, 2020.

[33] Fujimoto, S., Hoof, H., and Meger, D., "Addressing function approximation error in actor-critic methods," *International conference on machine learning*, PMLR, 2018, pp. 1587–1596.

[34] van den Hoek, M. A., de Visser, C. C., and Pool, D. M., "Identification of a Cessna Citation II Model Based on Flight Test Data," *Proceedings of the 4th CEAS Specialist Conference on Guidance, Navigation and Control (EuroGNC)*, Warsaw, Poland, 2017.

Appendix

A. Pseudo-code for Developed Algorithms

The following section includes the pseudo-code explaining the working of the online IDHP algorithm, and the offline SAC, DSAC, and RUN-DSAC algorithms.

Algorithm 1 IDHP Online Training Procedure

Initialize:
Actor parameters θ, critic parameters ϕ
Target critic parameters $\bar{\phi} \leftarrow \phi$
RLS parameters $\hat{\Theta}_0 \leftarrow 0$, covariance $\Lambda_0 \leftarrow \beta I$
Given: forgetting factor γ_{RLS}, critic soft update factor τ
Reset environment and observe initial state s_0 and tracking error e_0
Set previous action $a_{-1} \leftarrow 0$
for each timestep t **do**
    Actor forward pass: $\mathbf{a}_t \leftarrow \pi_\theta(\mathbf{s}_t, \mathbf{a}_{t-1}, e_t)$
    Apply $\mathbf{a}_t$ to environment, observe $\mathbf{s}_{t+1}$, compute tracking error e_{t+1} and reward r_{t+1}
    Compute increments Δx_t, $\Delta \mathbf{u}_t$, $\Delta \mathbf{x}_{t+1}$
    Incremental model (RLS update):
    Form regressor $\mathbf{X}_t \leftarrow \begin{bmatrix} \Delta \mathbf{x}_{t+1} \\ \Delta \mathbf{u}_t \end{bmatrix}$
    Predict $\Delta \hat{x}_{t+1}^T \leftarrow textbf X_t^T \hat{\Theta}_{t-1}$
    Prediction error $\epsilon_t \leftarrow \Delta x_{t+1}^T - \Delta \hat{x}_{t+1}^T$
    Update covariance Λ_t using Eq. (10)
    Update parameters $\hat{\Theta}_t$ using Eq. (11)
    Extract $\mathbf{F}_{t-1}, \mathbf{G}_{t-1}$ from $\hat{\Theta}_t$
    Critic update (co-state learning):
    Compute $V(\mathbf{s}_t)$ and $V(\mathbf{s}_{t+1})$ (consistent with Eq. (15))
    TD error $\delta_t \leftarrow r_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t)$
    Compute TD state derivative $\frac{\partial \delta_t}{\partial \mathbf{x}_t}$ using Eq. (17)
    Critic loss $\mathcal{L}_\lambda \leftarrow \frac{1}{2} \left\| -\frac{\partial \delta_t}{\partial \mathbf{x}_t} \right\|^2$
    Update critic parameters $\phi \leftarrow \phi - \mu_c \nabla_\phi \mathcal{L}_\lambda$
    Soft-update target critic $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$
    Actor update (policy improvement):
    Compute $\nabla_\theta \mathcal{L}_\pi$ using Eq. (13) with G_{t-1}
    Update actor parameters $\theta \leftarrow \theta - \eta_a \nabla_\theta \mathcal{L}_\pi$
    Optional: if $\|\epsilon_t\|$ exceeds threshold, reset covariance $\Lambda_t \leftarrow \Lambda_0$
end for

Algorithm 2 Offline SAC-family Training Procedure (SAC / DSAC / RUN-DSAC)

Initialize:
Actor parameters θ
Critic parameters ϕ_1, ϕ_2
Target critic parameters $\bar{\phi}_1 \leftarrow \phi_1,\ \bar{\phi}_2 \leftarrow \phi_2$
Replay buffer $\mathcal{D}$
Entropy temperature α
for each environment interaction step t **do**
  Sample action $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$
  Execute $\mathbf{a}_t$, observe reward r_t and next state $\mathbf{s}_{t+1}$
  Store transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in $\mathcal{D}$
  Sample mini-batch $\{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$ from $\mathcal{D}$
  Sample next actions $\mathbf{a}'_i \sim \pi_\theta(\cdot \mid \mathbf{s}'_i)$
  Critic target computation:
  if SAC **then**
    $y_i \leftarrow r_i + \gamma \left[\min_j Q_{\bar{\phi}_j}(\mathbf{s}'_i, \mathbf{a}'_i) - \alpha \log \pi_\theta(\mathbf{a}'_i \mid \mathbf{s}'_i)\right]$
  else ▷ DSAC / RUN-DSAC
    $Y_i \leftarrow r_i + \gamma \left[Z_{\bar{\phi}_{\min}}(\mathbf{s}'_i, \mathbf{a}'_i) - \alpha \log \pi_\theta(\mathbf{a}'_i \mid \mathbf{s}'_i)\right]$
  end if
  Critic update:
  if SAC **then**
    Minimize $\mathcal{L}_Q = \|Q_\phi(\mathbf{s}_i, statenext_i) - y_i\|^2$
  else ▷ DSAC / RUN-DSAC
    Minimize quantile regression Huber loss $\mathcal{L}_Z$
  end if
  Actor value computation:
  if SAC **then**
    $Q^{\text{actor}} \leftarrow \min_j Q_{\phi_j}(\mathbf{s}_i, \mathbf{a}_i)$
  else if DSAC **then**
    $Q^{\text{actor}} \leftarrow \Psi(Z_\phi(\mathbf{s}_i, \mathbf{a}_i))$ ▷ Risk-distorted expectation
  else ▷ RUN-DSAC
    $Q^{\text{actor}} \leftarrow Q_\phi(\mathbf{s}_i, \mathbf{a}_i) + \mu\sigma_Q(\mathbf{s}_i, \mathbf{a}_i)$
  end if
  Actor update:
  Minimize $\mathcal{L}_\pi = \mathbb{E}\left[\alpha \log \pi_\theta(\mathbf{a}_i \mid \mathbf{s}_i) - Q^{\text{actor}}\right]$
  Temperature update (optional):
  Update α to track target entropy
  Target critic update:
  $\bar{\phi}_j \leftarrow \tau\phi_j + (1-\tau)\bar{\phi}_j$
end for

Part II

Literature Study

2

Literature Review

This chapter presents a comprehensive literature review establishing the theoretical background for this thesis. It explores Reinforcement Learning (RL), from its fundamental principles to its state-of-the-art applications in flight control. In section 2.1 and section 2.2, the fundamentals of RL are explained, starting from the fundamentals of Markov Decision Processes and progressing to tabular methods. Then, section 2.3 explains function approximation methods necessary to apply RL to continuous flight-control applications. These first three sections are based on the theory presented in Sutton and Barto [48]. Section 2.4 and section 2.5 then explore two families of advanced RL algorithms: Approximate Dynamic Programming (ADP) and Deep Reinforcement Learning (DRL) techniques, both algorithm families applicable for continuous flight control problems. Section 2.6 explains the difference between offline, online, and hybrid RL methods, highlighting the advantages and disadvantages of each and answering research question *RQ2.1*. Finally, section 2.7 showcases the performance, strengths, and weaknesses of state-of-the-art RL algorithms for flight control. This section aims to answer research question *RQ1*,

2.1. Foundations of Reinforcement Learning

Machine learning (ML) describes the process through which computers can learn patterns in data to solve problems. Machine learning can be divided into three main categories: supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning algorithms find the relation between values and labels, given knowledge of the result. More formally, supervised learning is learning from a training set of labelled examples provided by a knowledgeable external supervisor. The goal is to generalise from these examples so that the system can respond correctly to new, unseen situations.

On the other hand, unsupervised learning algorithms typically involve finding correlations and structure in sets of unlabelled data. The goal of unsupervised learning is to discover patterns and correlations within the data.

The third category, Reinforcement Learning (RL), differs from both. RL is goal-directed through interaction with the environment. Unlike supervised learning, the agent is not told which actions to take, instead, it must discover which actions yield the maximum reward by trying them. Unlike unsupervised learning, the goal is not to find structure and correlations but to identify which set of actions will maximize a reward signal. As stated by Barto and Sutton, the most important characteristics of RL are ”trial-and-error search and delayed reward”. In Reinforcement Learning, an action may affect not only the immediate reward, but also all subsequent ones, making this a powerful framework for solving decision-making problems under uncertainty. This section explains the fundamental concepts of RL, and provides a theoretical basis for the methods discussed later.

2.1.1. The Agent-Environment Interaction

The main principle of reinforcement learning stems from an interaction between an *agent* and its *environment*. The agent is the learner and decision-maker, while the environment represents everything external to the agent with which it interacts over a sequence of discrete time steps, $t = 0, 1, 2, \ldots$.

The interaction between the agent and its environment is as follows. At time step t, the agent observes the environment state $\mathbf{s}_t \in \mathcal{S}$ and, based on this state, selects an action $\mathbf{a}_t \in \mathcal{A}$. At the next time step, the agent receives a scalar reward $r_{t+1} \in \mathbb{R}$ and transitions to a new state $\mathbf{s}_{t+1} \in \mathcal{S}$. Here, $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, respectively. The objective of reinforcement learning is to select actions that maximise the accumulated reward over time. This interaction generates a trajectory of states, actions, and rewards of the form $(\mathbf{s}_0, \mathbf{a}_0, r_1, \mathbf{s}_1, \mathbf{a}_1, r_2, \ldots)$.

The objective of the agent is to maximise the total reward accumulated over time. The reward signal is a scalar numerical value r_t resulting from an action $\mathbf{a}_t$ in state $\mathbf{s}_t$, it defines the goal for the agent. The agent learns to map the states to actions by developing a *policy* that maximises the cumulative reward.

2.1.2. Markov Decision Processes

Reinforcement Learning problems are commonly structured as Markov Decision Processes (MDPs). An MDP is a framework for modeling sequential decision-making processes that must satisfy the *Markov Property*: the next state $\mathbf{s}_{t+1}$ depends only on the current state S_t and the selected action $\mathbf{a}_t$. This property can be more formally defined in Equation 2.1

$$p(\mathbf{s'}, r|\mathbf{s}, \mathbf{a}). = Pr\{r_{t+1} = r, \mathbf{s}_{t+1} = \mathbf{s'}|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\} \tag{2.1}$$

The probability of transition to a new state $\mathbf{s'}$ and the rewards of that state r depend only on the action taken $\mathbf{a}$ at the current state s. A Markov Decision Process satisfies the Markov property. If the state and action spaces are finite, then it is called a finite MDP. MDPs translate directly into learning, since the agent can maximise the reward in the next state by choosing the optimal action in the current state, as that is the only factor affecting the reward and the next state.

2.1.3. Reward and Return Signals

The reward signal R_t returns a numerical score to the agent after the agent has selected an action. The higher the reward, the more desirable the new state is. For example, an agent attempting to escape a maze might encounter a reward of -1 for each time step that passes when it has not escaped. This will ensure the agent will aim to escape the maze as quickly as possible. "The reward signal is your way of communicating to the robot *what* you want it to achieve..." [48]

However, if the agent had as an objective to maximise the immediate reward, it might be impossible to find the global optimal solution. Instead, the reinforcement learning agent is given the objective to maximise the reward over time. This is often referred to as the expected return G_t. This measure considers the rewards from all future timesteps.

For episodic tasks (tasks with a limited number of timesteps), the agent learns the optimal behaviour and the return is calculated as the sum of all rewards of all selected actions until the terminal state.

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{2.2}$$

where T is the final time step of the episode. When the interaction is not divided into episodes, the task is said to be *continuous*. In these cases, T would be infinite, and thus the return we aim to maximize could also be infinite. To prevent this a discounting term is introduced to Equation 2.2.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.3}$$

where γ is the discount rate and $0 \leq \gamma \leq 1$. This ensures the return never reaches infinity. It is also worth noting that if $\gamma = 0$. The agent only chooses the immediate best reward, while if $\gamma = 1$ the agent values all future rewards the same, leading to Equation 2.2.

2.1.4. Policies and Value Functions

A policy is the strategy an agent deploys to solve a problem. It is a law that determines which action to select at each state, or, more mathematically, a mapping from the state to the probability of selecting an action, $\pi(\mathbf{a}|\mathbf{s})$. The ultimate goal of an RL task is to identify the optimal policy $\pi*$, the state-action mapping that maximises returns.

The *value function* estimates "how good" it is to be in a given state, or to perform a given action in a given state under a policy π. The state-value function of a policy π determines the expected return starting in a state **s** and following policy π after. This is denoted by Equation 2.4

$$V_\pi(\mathbf{s}) = E_\pi \left[G_t|\mathbf{s}_t = \mathbf{s}\right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|\mathbf{s}_t = \mathbf{s}\right] \tag{2.4}$$

In Equation 2.4 E_π is the expected value of a variable given that policy π is followed. Similarly, the expected return of starting from a state **s** and choosing an action a, and after following policy π is known as the action-value function of a policy, $Q_\pi(\mathbf{s}, \mathbf{a})$. This function estimates the value of a particular state-action pair and is defined by Equation 2.5.

$$Q_\pi(\mathbf{s}, \mathbf{a}) = E_\pi[G_t|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right] \tag{2.5}$$

Both the value function and action-value function can be estimated from experience.

2.1.5. The Goal: Bellman Equations and Optimality

The value functions defined in Equation 2.4 and Equation 2.5 can be written in a recursive form. This decomposition is the core of Bellman equations, which are fundamental to understanding and solving reinforcement learning problems. Bellman equations express a relationship between the value of a state and the value of its following states. They define the value of a state as the sum of the immediate reward and the discounted value of the next state you will land in. This relates to the Markov property, since it shows that the reward in the next state depends solely on the action chosen at the current state.

For a given policy π, the Bellman equation for the state value function v_π breaks down the value $V_\pi(\mathbf{s})$ into the sum of the expected rewards over all possible actions and subsequent state transitions.

$$V_\pi(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \sum_{\mathbf{s}',r} p(\mathbf{s}', r|\mathbf{s}, \mathbf{a})[r + \gamma V_\pi(\mathbf{s}')] \tag{2.6}$$

In Equation 2.6 the value of the state s is an average over all actions **a** (weighted by the policy $\pi(\mathbf{a}|\mathbf{s})$) of the expected return of each action. This expected return is an average over all possible outcomes $(\mathbf{s}', r)$ (weighted by the environment dynamics $p(\mathbf{s}', r|\mathbf{s}, \mathbf{a})$) of the immediate reward r plus the discounted value $\gamma V_\pi(\mathbf{s}')$ of the following state. The Bellman equation computes the state value by taking a weighted average of the rewards from all actions available from a state, and the rewards from all possible new states $\mathbf{s}'$ resulting from each action. Likewise, the Bellman form of the action-value function is the following:

$$Q_\pi(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}',r} p\left(\mathbf{s}', r, |\ \mathbf{s}, \mathbf{a}\right) \left[r + \gamma \max_{\mathbf{a}'} Q_\pi\left(\mathbf{s}', \mathbf{a}'\right)\right] \tag{2.7}$$

To solve an RL task, it is necessary to find the policy that achieves the highest reward in the long run. Value functions are used to evaluate how good a policy is compared to others. A policy π is better than a policy π' if its expected return is greater than the expected return of π' for all states ($\pi \geq \pi'$ if and only if $V_\pi(\mathbf{s}) \geq V_{\pi'}(\mathbf{s})$ for all $\mathbf{s} \in \mathcal{S}$). This implies that there will always be a policy that is greater than or equal

to all other policies; this is called the *optimal policy*. This optimal policy may be equal to others; in this case, these policies are all called optimal policies, $\pi*$, and they all share the same *optimal state-value function*, $V_*(\mathbf{s})$, and *optimal action-value function*, $Q_*(\mathbf{s}, \mathbf{a})$.

$$V^*(\mathbf{s}) = \max_{\mathbf{a}} \sum_{\mathbf{s}',r} p\left(\mathbf{s}', r \mid \mathbf{s}, \mathbf{a}\right)\left[r + \gamma V_*\left(\mathbf{s}'\right)\right] \tag{2.8}$$

$$Q^*(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}',r} p\left(\mathbf{s}', r \mid \mathbf{s}, \mathbf{a}\right)\left[r + \gamma \max_{a'} Q_*\left(\mathbf{s}', \mathbf{a}'\right)\right]. \tag{2.9}$$

Equation 2.8 and Equation 2.9 are the Bellman optimality equations for V_* and Q_*. They state that the value of a state or action-state pair under an optimal policy equals the return from the best action in that state. Once an agent has found V_*, it acts optimally by selecting the action a that maximizes the right-hand side of Equation 2.8, requiring the agent to perform a one-step-lookahead search only. Knowing Q_* provides a way to find the optimal action without the need of a transition model $p(\mathbf{s}', r|\mathbf{s}, \mathbf{a})$, in this case, the agent does not require doing a one-step look-ahead, it can simply select the action that maximizes $Q_*(\mathbf{s}, \mathbf{a})$, $\underset{\mathbf{a}}{\text{argmax}} Q_*(\mathbf{s}, \mathbf{a})$.

The Bellman optimality equations provide a clear definition of Reinforcement Learning: to find a value function that is recursively consistent with the optimal selection of actions. These equations are the theoretical target for all RL agents. The next section explores the foundational algorithms designed to solve these equations or approximate their solutions.

2.2. Tabular Methods

The earliest and most fundamental Reinforcement Learning algorithms operate in a *tabular setting*, in which the state and action spaces are small enough to be stored in arrays or tables. For example, the policy will be represented in a table that maps each possible state-action pair to the probability of that action being chosen from that state. Although flight control problems are continuous and thus cannot be represented in tables, it is important to understand these methods, as they provide the foundation to more complex approximation methods that will be discussed later. Tabular methods can be divided into three different main families, distinguished by how they gather experience and their knowledge of the environment: Dynamic Programming, Monte Carlo Methods, and Temporal Difference Learning.

Model-based vs. Model-free Learning

A further distinction between reinforcement learning algorithms is whether they are model-based or model-free.

Model-based algorithms are those that use a model of the environment to perform planning. The agent uses a model to predict the outcomes of the actions before selecting them, and the policy is computed by using the model to predict the return associated with each action. Model-based algorithms have a high sample efficiency and learn at a quicker rate than model-free methods; however, they require full knowledge of the environment dynamics and thus are limited in their utility. Dynamic Programming algorithms fall under this category.

On the other hand, model-free methods do not require any knowledge of the environment or its transition probabilities. The agent learns by interacting with the environment. This has advantages when dealing with changing environments or fault-tolerant applications, allowing the agent to adapt to changing circumstances. Monte Carlo methods and Temporal Difference learning are both model-free algorithm families.

2.2.1. Dynamic Programming

Dynamic programming (DP) algorithms are those able to solve optimal policies given a perfect model of the environment. They work under the assumption that the environment is a finite MDP. DP methods use value functions to structure the search for good policies, iteratively improving them by applying Bellman equations as update rules. The two most prominent DP methods are Policy Iteration and Value Iteration.

Policy Iteration

Policy Iteration is a method used to determine the optimal policy in DP algorithms. It is a process that alternates between policy evaluation and policy improvement steps. The first evaluates the state-value function of a given policy, while the latter uses this state-value function to update the policy. Multiple iterations of these alternating steps converge to the optimal policy and state-value function.

$$\pi_0 \xrightarrow{\text{E}} V_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} V_*$$

Policy Evaluation

This is the step that calculates the value function by iteratively updating approximations of the value function starting from an arbitrarily chosen $V_0(s)$ for all states. This algorithm iteratively applies the Bellman equation for V_π from Equation 2.8 as an update rule, to estimate the value of each of the states:

$$V_{k+1}(\mathbf{s}) \leftarrow \sum_{\mathbf{a}} (\mathbf{a}|\mathbf{s}) \sum_{\mathbf{s}',r} p(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) \left[r + v_k(\mathbf{s}')\right] \tag{2.10}$$

After each update, the value of each state V_{k+1} is updated to the sum of the rewards from all possible actions plus the value of the following state $V_k\mathbf{s}'$ weighted by the probability of choosing each action according to the policy. As k approaches infinity, the values of the states converge to the actual value of the policy $V_\pi(\mathbf{s})$. This means that a single Policy Evaluation step will move the estimate of the value function closer to its actual value under a given policy.

Policy Improvement

Finding the value of a policy is useful to compare it against other policies. If V_π is the value for a policy π, then to improve upon this policy, at a certain state, an action **a** must be chosen that does not follow policy π, and then continue using policy π for subsequent steps. If the state-action value of choosing **a** at state **s**, $Q_\pi(\mathbf{s}, \mathbf{a})$, is greater than or equal to the value of the state **s** under policy π, $V_\pi(\mathbf{s})$, then it would be better to select action **a** every time we encounter the state **s**. Therefore, the policy π' which chooses **a** at state **s** must be better than the original policy π.

$$Q_\pi(\mathbf{s}, \pi'(\mathbf{s})) \geq V_\pi(\mathbf{s}) \rightarrow V_{\pi'}(\mathbf{s}) \geq V_\pi(\mathbf{s}) \tag{2.11}$$

In other words, selecting a *greedy policy* with respect to the value function of the previous policy, as shown in Equation 2.12, will result in a policy π' that is better or equal to the original policy π.

$$\pi'(\mathbf{s}) = \underset{\mathbf{a}}{\text{argmax}} \sum_{\mathbf{s}',r} p\left(\mathbf{s}', r \mid \mathbf{s}, \mathbf{a}\right) \left[r + \gamma V_\pi\left(\mathbf{s}'\right)\right], \tag{2.12}$$

Once a policy has been improved upon, it can be evaluated again to find $V_{\pi'}$, then this policy can be improved upon again to find π'' and so forth.

Value Iteration

The policy evaluation step is computationally expensive and requires many iterations to converge to exactly V_π. However, this step can be truncated without losing the guarantee of convergence. *Value Iteration* is a special case where the policy evaluation step is truncated after just one step; it can be written as an update operation that combines both policy improvement and truncated policy evaluation:

$$V_{k+1}(\mathbf{s}) = \underset{\mathbf{a}}{\text{argmax}} \sum_{\mathbf{s}',r} p\left(\mathbf{s}', r \mid \mathbf{s}, \mathbf{a}\right) \left[r + \gamma V_k\left(\mathbf{s}'\right)\right], \tag{2.13}$$

Generalized Policy Iteration

Generalized Policy Iteration (GPI) is the idea of allowing the policy evaluation and policy improvement processes to interact. Most RL methods use the concepts of improving value functions and policies to converge to optimal ones. This can be seen in Figure 2.1. The GPI process is characterised by steps of

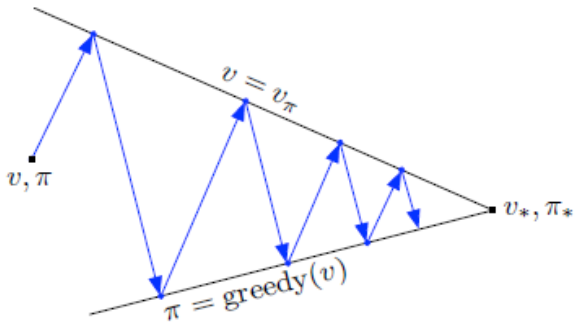


Figure 2.1: GPI interaction between evaluation and improvement from Sutton and Barto [48]

policy improvement towards the optimal policy, and steps of policy evaluation towards the optimal value function. Each process drives the solution towards one of the goals, and as GPI progresses, both the policy and the value function converge to their optimal states.

2.2.2. Monte Carlo (MC) Methods

Monte Carlo (MC) methods do not require a model of the environment to learn, only experience gained through interaction with it. MC methods learn value functions by averaging empirical returns over sampled episodes. In contrast to DP, MC does not *bootstrap*, meaning it does not use subsequent state-value estimates to compute the value of the current state.

MC Prediction: Policy Evaluation

MC prediction evaluates a policy π by generating many episodes that follow such a policy, and averaging the returns observed after each visit to a state in an episode. As more episodes are generated, the average should converge to the expected value of each state under that policy. *First-visit MC method* averages the returns from the first visit of the agent to a state **s**. *Every-visit MC method* averages the returns from every time a state **s** is visited in an episode. The update after each episode for a specific state-action pair $Q(\mathbf{s}, \mathbf{a})$ is given by Equation 2.14 below.

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha \left[G_t - Q(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{2.14}$$

where α is the step size, and G_t is the return.

For MC prediction problems where there is no model available, it is more useful to estimate the action-value functions using sampled probabilities. However, if the policy evaluated is deterministic, both the first-visit and every-visit MC methods will fail to estimate the value of state-action pairs that are not selected. In this case, the principle of *exploring starts* is needed, which states that each state-action pair has a non-zero probability of being selected at the beginning. This ensures the policy is properly evaluated and $Q_\pi(\mathbf{s}, \mathbf{a})$ is determined as the number of episodes increases.

MC Control

In GPI, the policy is improved and evaluated repeatedly, converging to the optimal policy and action values. In MC control, the same principle applies: policy evaluation is done using the strategies described in 2.2.2 where Q_π is approximated, while policy improvement is done by making the policy greedy with respect to the current action-value function.

$$\pi_0 \xrightarrow{\text{E}} Q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} Q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} Q_*$$

Since the value functions in use are action-value functions, policy improvement is done simply by choosing the action with the highest action-value:

$$\pi(\mathbf{s}) \doteq \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \tag{2.15}$$

MC methods are foundational for model-free learning, but can also be inefficient due to their high variance and because learning occurs only at the end of each episode.

2.2.3. Temporal Difference (TD) Learning

While Monte Carlo methods provide a robust, model-free way to learn from complete episodes, the requirement to wait until the end of each episode can slow learning. *Temporal-Difference (TD) learning* is an improvement to reinforcement learning that removes this limitation by combining the sampling from MC with the bootstrapping from DP. TD methods learn after each time step and update their value estimates based on the immediate reward and the current estimate of the next state's value, rather than waiting for the final return.

TD Prediction

Monte Carlo methods require knowledge of the return at the end of the episode G_t to update the value of a state $V(\mathbf{s}_t)$, TD methods on the other hand, require only knowledge of the reward at time $t+1$, r_{t+1} and the previous estimate of the value of the subsequent state $V(\mathbf{s}_{t+1})$. The update-law for the simplest TD methods, $TD(0)$ is the following:

$$V(\mathbf{s}_t) \leftarrow V(\mathbf{s}_t) + \alpha \left[r_{t+1} + \gamma V(\mathbf{s}_{t+1} - V(\mathbf{s}_t)\right] \tag{2.16}$$

where α represents the timestep, and γ the discount rate. TD methods also follow the idea of GPI: first, the policy is evaluated using Equation 2.16 or a similar update law, and then the policy is improved by making it greedy with respect to the value function. The expression between brackets in Equation 2.16 is called the TD error, δ_t, and is changed depending on the algorithm used.

On-Policy vs. Off-Policy Learning

Like MC control, TD control must balance exploration and exploitation. In MC control, the assumption of exploring starts was introduced to ensure exploration. However, a method called *off-policy learning* was developed to enable exploration without relying on this assumption.

On-policy learning methods are those that use the same policy to interact with the environment to generate data as the policy for learning. Off-policy methods, on the other hand, use two separate policies; the *behaviour policy*, which is used to generate data, and the *target policy*, which is the one being learned about. On-policy methods are simpler, while off-policy methods are more powerful, as they can evaluate multiple target policies at once; however, they are slower to converge and have greater variances.

Sarsa: On-Policy TD Control

Sarsa uses the action-value function of the subsequent state and action $Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$ and the reward r_{t+1} to update the estimate of the action-value at state $\mathbf{s}$.

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha[r_{t+1} + \gamma Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q(S_t, A_t)] \tag{2.17}$$

It can be seen in Equation 2.17 that the quintuple $(\mathbf{s}_t, \mathbf{a}_t, r_{t+1}, \mathbf{s}_{t+1}, \mathbf{a}_{t+1})$ is used to update estimates of the action-value function Q_π, hence the name Sarsa. Policy improvement is then done by making the policy π greedy with respect to the action-value function Q_π.

Q-Learning: Off-Policy TD Control

Q-learning is a foundational off-policy TD control algorithm. It learns an estimate of the optimal value function q_* regardless of the policy being followed. Q-learning does not use A_{t+1}, the action suggested by the current policy, to update the action-values, instead, it selects the action a that maximises the action value at the next state, as can be observed in Equation 2.18

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha[r_{t+1} + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a}) - Q(\mathbf{s}_t, \mathbf{a}_t)] \tag{2.18}$$

In on-policy methods, repeatedly updating the estimate of the action-value function would converge in the action-value function for the policy being evaluated, Q_π. Yet, in this method, repeating the update law Equation 2.18 will converge to the optimal policy Q_*. The agent chooses the best action and "ignores" the action that the behaviour policy being evaluated would select, eventually leading to the optimal action-values. This flexibility has made it one of the most widely used RL algorithms.

2.3. Scaling to Complex Control: Approximate Solutions

The tabular methods described in section 2.2, Dynamic Programming, Monte Carlo methods, and Temporal Difference learning, provide the theoretical foundations for reinforcement learning. They guarantee optimal solutions under specific conditions and illustrate the core principles of policy improvement and evaluation. However, they are limited to small problems, with discrete state and action spaces $\mathcal{S}, \mathcal{A}$.

Real-world problems, such as aircraft flight control, require the agent to operate in continuous or high-dimensional environments. Representing this space in a tabular fashion is impossible. This challenge is referred to as the *curse of dimensionality*: the number of possible states grows exponentially with the number of state variables. Additionally, with such a large number of states, it becomes almost impossible to generalize from previous encounters, as most states encountered will most likely have never been encountered before.

To address this limitation, reinforcement learning algorithms are extended through the use of *function approximation* methods. These generalise from a set of limited states to a larger set of states not yet experienced. This section introduces the principles of function approximation and presents the modern reinforcement learning architectures used for high-dimensional control problems.

2.3.1. Function Approximation in Reinforcement Learning

The core idea of function approximation is to represent the value function with a parametrised function $\hat{V}(\mathbf{s}, \mathbf{w}) \approx V_\pi(\mathbf{s})$, where $\mathbf{w}$ is a weight vector. The function "$\hat{V}$ might be the function computed by a multi-layer artificial neural network, with $\mathbf{w}$ the vector of connection weights in all layers", Sutton and Barto [48]. When the agent updates the value of a state it has visited, the weight vector will affect the estimated values of other unvisited states.

Function approximation, from a machine learning perspective, can be considered an instance of supervised learning. Function approximation methods aim to mimic an input-output relation by adjusting the weights $\mathbf{w}$ to minimise the error between the predictions $\hat{V}(\mathbf{s}, \mathbf{w})$ and the target values. In reinforcement learning, function approximation is applied by treating each state and its corresponding return (or TD target) as a training sample.

The objective is typically to minimise the Mean Squared Value Error (VE) over the distribution of states $\mu(\mathbf{s})$ the agent encounters while following its policy (the on-policy distribution):

$$\text{VE}(\mathbf{w}) = \sum_{\mathbf{s} \in \mathcal{S}} \mu(\mathbf{s})[V_\pi(\mathbf{s}) - \hat{V}(\mathbf{s}, \mathbf{w})]^2 \tag{2.19}$$

This objective can be minimised using gradient descent methods. When the target $V_\pi(\mathbf{s})$ is estimated using Monte Carlo methods or bootstrapped updates, the gradient update is the following:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{V}(\mathbf{s}_t, \mathbf{w}_t) \right] \nabla \mathbf{w} \hat{V}(\mathbf{s}_t, \mathbf{w}_t) \tag{2.20}$$

In Equation 2.20 U_t represents the target, which depends on the method used. For Monte Carlo methods, the full return is used $U_t = G_t$, whereas for ***semi-gradient TD(0)***, $U_t = r_{t+1} + \gamma \hat{V}(\mathbf{s}_{t+1}, \mathbf{w}_t)$ is used. Additionally, α is the step-size parameter.

Linear vs. Nonlinear Approximators

Linear function approximators are the most straightforward. They express the estimated value function as a weighted combination of features $\mathbf{x}(\mathbf{s})$. This makes the value function linear in the weights, even if

the features are non-linear functions of the state.

$$\hat{V}(\mathbf{s}, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(\mathbf{s}) \tag{2.21}$$

Monte Carlo updates in the linear setting are guaranteed to converge to the minimum of the mean squared error VE. Similarly, the semi-gradient TD(0) update is also guaranteed to converge, but to the TD fixed point, not to the minimum VE solution.

Linear features can be constructed using a variety of techniques such as:

- Polynomial basis: combining powers of state variables.
- Fourier basis: representing state features as sinusoidal functions of different frequencies.
- Tile coding: uses overlapping grids to discretize continuous states into binary features.
- Radial basis functions (RBF): uses Gaussian-like features centered at selected states.

Selecting and configuring the features has a direct impact on learning performance and generalization. However, all linear approximators are limited in their representational power.

In contrast, non-linear approximators like Artificial Neural Networks (ANNs) can approximate complex, high-dimensional mappings between a feature vector and the estimated value of the states. They are networks composed of interconnected layers, characterized by an input layer that receives the feature vector of the state, an output layer that produces the final result, and one or more hidden layers where non-linear computations occur.

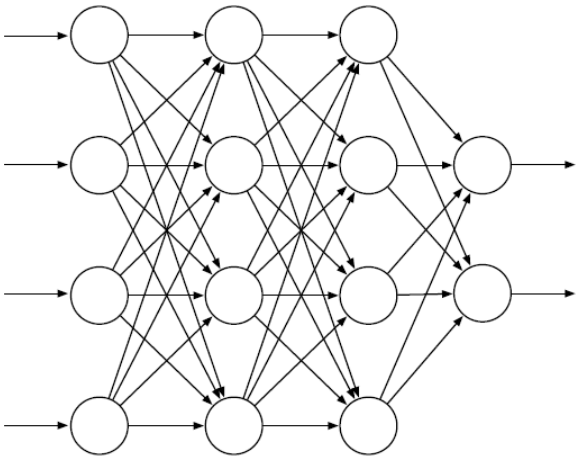


Figure 2.2: Generic feedforward ANN diagram from Sutton and Barto [48]

ANNs use stochastic gradient descent to learn, through the back-propagation algorithm. Unfortunately, unlike their linear counterparts, non-linear function approximation methods do not guarantee convergence and introduce stability challenges.

Gradient and Semi-gradient Methods

Function approximation methods can be differentiated by whether they use true gradient descent or not. Methods that perform gradient descent update their weights towards the gradient of a well-defined objective function. Methods that use bootstrapping, such as TD methods, have a target that depends on the current estimate $\hat{v}$, so the gradient cannot be exactly defined. These are called semi-gradient methods, which still converge with linear approximators, although not to the minimum VE. When the approximator is non-linear, however, these methods may become unstable, especially in the off-policy setting.

The Deadly Triad

Off-policy methods combined with function approximation introduce a problem known as the deadly triad. This is characterized by the combination of:

1. Function approximation
2. Bootstrapping
3. Off-policy learning

This combination can lead to instability in the solution and divergence of the value estimates. An example of this issue is Baird's counterexample, where a simple linear off-policy TD method with function approximation diverges catastrophically. Some solutions used to mitigate this problem are Gradient-TD methods, Emphatic-TD learning, and constrained update rules.

In summary, function approximation allows RL algorithms to generalise from limited experience, making them suitable for continuous tasks. However, the choice of the approximation technique and the learning paradigm (on-policy or off-policy) are critical design considerations that have enabled the development of increasingly sophisticated methods.

2.4. Actor-Critic Designs in Approximate Dynamic Programming

Approximate Dynamic Programming (ADP) refers to a class of methods that merge concepts from reinforcement learning and optimal control to overcome the "curse of dimensionality" [31]. Within ADP, adaptive critic design methods (ACDs) are those that use a forward dynamic programming (FDP) and derive relations for the optimal policy, the cost, and their derivatives to ensure convergence to a solution approximating the optimal one over time [10]. They are composed of two components: an actor (or action network), which approximates the policy, and a critic (or critic network), which approximates the value function [55, 10].

The interaction between actor and critic in ACDs can be considered Policy Iteration. Given a policy, the actor chooses an action based on the current state, and the critic estimates the state value to perform Policy Evaluation. Then, to perform Policy Improvement, the actor and critic network weights are updated based on the next state and reward obtained from the environment [10]. ACD methods can be divided into three main algorithms: Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP), and Globalized Dual Heuristic Programming (GDHP). Each of these has an Action-Dependent (AD) and an incremental variant; this classification can be observed in Figure 2.3 below.

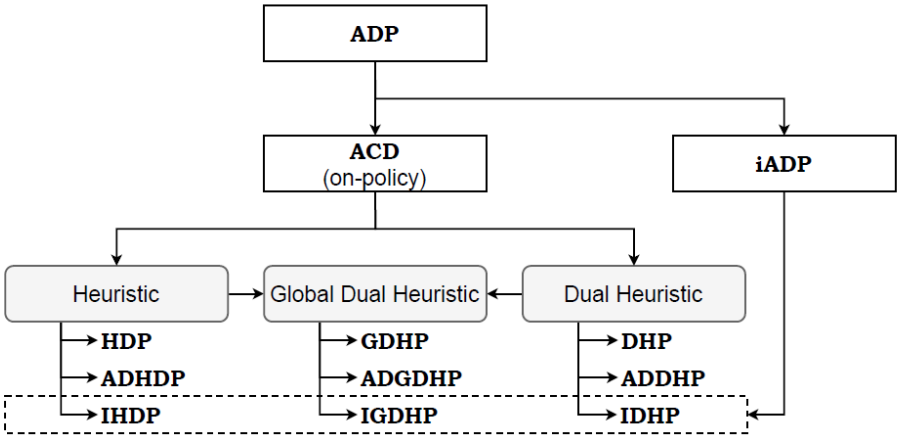


Figure 2.3: Approximate Dynamic Programming Classification from Teirlinck [49]

2.4.1. Heuristic Dynamic Programming

HDP is the simplest form of ACD, It is characterized by a critic network trained to approximate the value function, which represents the total future cost, or "cost-to-go" of a given state [55]. The critic is trained by receiving the state $\mathbf{s}_t$ and outputting an estimate $V(\mathbf{s}_t, \phi)$ of the state value function $V(\mathbf{s}_t)$ at time t. The objective of the critic is to determine the weights ϕ for which the critic network best approximates the state-value function. This is done by minimizing the error in Equation 2.22:

$$E_1(t) = V(\mathbf{s}_t) - \gamma V(\mathbf{s}_{t+1}) - r_{t+1} \tag{2.22}$$

Updating the actor requires knowledge of the derivative of the state value with respect to the action. To obtain this, it is necessary to backpropagate $\delta V / \delta V$ through the network [33]. Additionally, HDP requires knowledge of the state-transition model [33].

On the other hand, the action-dependent version of HDP, ADHDP, is model-free and is more widely known as a generalization of Q-learning [31, 33]. In ADHDP, the critic estimates the action-value function $Q(s_t, a_t)$. The derivative of the value function is then computed by using backpropagation solely, without the need for the state-transition model. Van Kampen, Chu, and Mulder [22] show that HDP methods have numerous advantages when compared to ADHDP in the task of adaptive flight control of an F-16 model. The HDP controller has a higher success rate and faster convergence during initial training, adapts better to unexpected changes, and has a wider operational flight envelope, whereas ADHDP performs better in the presence of measurement noise.

2.4.2. Dual Heuristic Dynamic Programming

The DHP implementation varies from the HDP implementation in that the critic network outputs the estimate of the gradient of the value function with respect to the states $\delta V(\mathbf{s}_t)/\delta \mathbf{s}_t$ [55]. This gradient is then used for backpropagation through the actor and critic networks, without needing to compute the derivative of the value function with respect to the action. Compared to HDP, DHP yields faster and more accurate gradients [10]. Like HDP, DHP also requires knowledge of the model. Action Dependent DHP (ADDHP) is the model-free version of DHP, in which the critic network outputs the partial derivatives of the action value function $\frac{\delta Q(\mathbf{s}_t, \mathbf{a}_t)}{\mathbf{s}_t}$ and $\frac{\delta Q(\mathbf{s}_t, \mathbf{a}_t)}{\mathbf{a}_t}$.

2.4.3. Globalized Dual Heuristic Dynamic Programming

GDHP is a method that combines both principles from DHP and HDP. The critic approximates both the value function and its derivatives [55]. This approach provides the "best of both worlds": an estimate of the value function for high-level decisions and gradient information for fine-grained adjustments to the control policy. The original formulation was computationally complex, requiring the computation of second-order derivatives [31]. However, recent work by Prokhovor and Wunsch proposes frameworks that are more practical to implement by using a single critic network with two separate outputs for the value function and its derivatives [33]. The action-dependent version of this implementation (ADGDHP) is a unified framework for all ACDs and can emulate all the frameworks described above [33].

2.4.4. iADP and State-of-the-art Frameworks

The previously discussed ACDs provide a useful framework for optimal control. However, their application to complex, non-linear systems with unknown dynamics presents many difficulties. ADP methods are suitable when the system states are fully observable, but this may not hold when measurements are noisy. To address this problem, a new class of methods known as incremental Approximate Dynamic Programming (iADP) was developed, combining the principles of ADP and incremental nonlinear control techniques [56, 59].

iADP methods do not need a priori information of the model or online identification of the nonlinear model. Instead, they calculate the required control increment at a given time based on the system conditions from the previous instant. This is done by linearizing the nonlinear system dynamics around the current point, and then, assuming a high sampling frequency, the system can be linearized around

the previous time-step's state and action $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, resulting in the following discrete incremental form for a non-linear system: [56].

$$\Delta \mathbf{x}_{t+1} \approx \mathbf{F}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})\Delta \mathbf{x}_t + \mathbf{G}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})\Delta \mathbf{u}_t \tag{2.23}$$

Where, **F** and **G** are the system matrix and control effectiveness matrix, respectively. This approach is similar to control methods such as Incremental Nonlinear Dynamic Inversion (INDI), as it leverages the system's incremental dynamics to achieve robustness without requiring a perfect model [59].

The incremental framework has proven highly effective in practice. Research has demonstrated that Incremental Heuristic Dynamic Programming (iHDP), using a simplified quadratic value function, can accelerate online learning and improve tracking precision compared to traditional HDP methods [57, 42]. It has also been shown that Incremental Dual Heuristic Programming (IDHP) methods can accelerate online learning, improve tracking precision, and offer higher fault tolerance than DHP, thereby handling a wider range of initial states and sudden changes in system dynamics [58]. Additionally, incremental Global Dual Heuristic Programming shows advantages over its non-incremental counterpart; however, both approaches are less suited for real-world applications due to their high complexity [43, 44]. These methods have been applied to several high-fidelity simulations of real aircraft, under full observability and partial observability conditions, and have all outperformed their non-incremental counterparts in stability, robustness, and adaptability when faced with unforeseen scenarios [27, 45, 19].

2.5. Deep Reinforcement Learning

The previous section, section 2.4 introduced ADP methods and their incremental extensions as a way to address the curse of dimensionality by combining function approximation with dynamic programming. These methods, except for the novel incremental versions and ADHDP, all relied on explicit or approximate models of the system dynamics. This model dependence limits the applicability of these methods to highly uncertain or partially unknown environments.

Deep Reinforcement Learning (DRL), on the other hand, is a family of model-free methods that combine reinforcement learning principles with deep neural networks to scale decision-making to high-dimensional domains. Model-free algorithms do not try to build a model of the environment; instead, the agent improves its performance by direct interaction with the environment. This makes model-free methods more suitable for fault-tolerant flight control scenarios where unforeseen failures and disturbances can occur. Additionally, learning a model of these failure modes would require the agent to put itself in situations that compromise flight safety.

A common way to categorise deep reinforcement learning algorithms is to distinguish between value-based methods and policy-based methods. Value-based methods optimise the action-value functions $Q_\pi(\mathbf{s}, \mathbf{a})$ and derive the policy from these. Policy-based methods, on the other hand, optimise the policy directly. Actor-critic methods combine both approaches by using a value-based method to learn an action-value function (critic) to stabilize and guide the learning of a policy (actor). This section introduces the main DRL algorithms, starting with DQN, then moving on to policy gradient methods, and finally examining advanced actor-critic architectures such as DDPG, TD3, and SAC.

2.5.1. Value-Based Methods

Value-based methods aim to learn an estimate of the action-value function $Q_\pi(\mathbf{s}, \mathbf{a})$, representing the expected return of choosing action **a** in state **s** according to the policy. The policy is then iterated by selecting the action that maximizes the action-value function, eventually converging to the optimal action-value function $Q^*(\mathbf{s}, \mathbf{a})$.

In Section 2.2.3, the off-policy TD control algorithm known as Q-learning was introduced. This algorithm updates the value of a state-action pair according to Equation 2.18. This algorithm was one of the most significant breakthroughs in reinforcement learning and has been proven to converge to the optimal solution in the tabular case; however, it is not applicable to flight control problems, since it is unable to represent high-dimensional state and action spaces.

Deep Q-Networks

To address this issue, Mnih et al. introduced the Deep Q-Network framework, which combines Q-learning with deep neural networks as function approximators [32]. DQN uses a deep convolutional neural network to approximate the Q-function, enabling it to process high-dimensional state inputs. However, since this meets all the conditions of the "deadly triad" introduced in Section 2.3.1, two key mechanisms were developed to address the issue of instability:

- **Replay Buffer:** past interactions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{r}_{t+1})$ are stored in a replay buffer, and training is performed from randomly sampled mini-batches of samples from this replay buffer. This reduces correlations between consecutive samples, thereby reducing variance and increasing data efficiency.
- **Target Network:** A separate network from the Q-network, known as the target network, is used to generate the Q-learning targets. Every C steps, the target network will be synchronised with the Q-network. This reduces oscillations and divergence during learning.

These innovations enabled the DQN framework to achieve performance superior to that of humans in Atari games [32]. DQN is characterized by its high sample efficiency and low variance of the value function estimation; however, it can often result in overestimation [6].

This framework was later enhanced with several different extensions. Double DQN aims to mitigate overestimation in DQN by using two separate networks for action selection and value estimation [50]. Dueling DQN estimates the state value and action advantage separately, increasing the learning efficiency in states where many actions have similar effects, since these do not need to be learned [54]. Prioritized Experience Replay (PER) prioritizes transitions with higher learning potential, i.e., transitions that were frequented more, further improving DQN [6].

However, despite these advances, the DQN algorithm can only handle discrete action spaces, making it poorly suited for continuous control tasks such as aircraft flight control. In such spaces, discretizing the action space is often impractical; this limitation motivated the development of policy-based methods.

2.5.2. Policy-Based Methods

While action-based methods learn the action value and then derive the policy indirectly, policy-based methods learn by optimizing the weights θ of a parametrized policy $\pi_\theta(\mathbf{a}|\mathbf{s})$ with respect to the expected return. This makes policy-based methods more suitable for high-dimensional, continuous action spaces, without requiring discretization [6].

The theoretical basis of these methods stems from the Policy Gradient Theorem, which states that the gradient of the performance objective with respect to the parameters of a given policy can be expressed as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi \left[\nabla_\theta (\log \pi_\theta(\mathbf{a}|\mathbf{s})) Q_{\pi_\theta}(\mathbf{s}, \mathbf{a})\right] \tag{2.24}$$

Where $J(\pi_\theta)$ represents the expected return of a policy π_θ. Equation 2.24 shows that an estimate of the gradient can be obtained from sample trajectories, without needing any knowledge of the system dynamics.

REINFORCE

REINFORCE is one of the first applications of the policy-gradient learning method. It uses samples from the gradient derived using the full return G_t. Although it provides an unbiased estimate of the gradient, this method suffers from high variance because it is a Monte Carlo method. To reduce the variance of REINFORCE, a baseline is introduced, typically the state value function $V(\mathbf{s})$. This modification improves stability and sample efficiency compared to REINFORCE [47].

Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) is an extension of the policy gradient framework that aims to improve learning stability. Methods like REINFORCE can suffer from large parameter updates that destabilize learning if the step size is poorly tuned. TRPO mitigates this shortcoming by introducing a trust region, ensuring that policy updates do not deviate too far from the previous policy [35]. The algorithm is bounded by ensuring the Kullback-Leibler divergence between the policy and the updated policy remains below a certain threshold. However, this method requires second-order optimization, making it more computationally expensive and more complex to implement.

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an algorithm proposed as a simpler and more efficient alternative to TRPO. Like TRPO, it was designed to ensure similarity between a policy π_θ and the updated policy π'_θ [6]. The step size in this algorithm is bounded by clipping the ratio between the two policies and taking the minimum of the unclipped objective. Additionally, PPO supports mini-batch updates and is compatible with first-order optimization methods, making this method easier to implement and more efficient than TRPO [35].

PPO is one of the most commonly used policy-gradient methods, and has been proven to handle aircraft flight control tasks. Bøhn et al. demonstrated that a controller using PPO is capable of controlling a UAV's attitude, and demonstrated improved performance when compared to a baseline PID controller under moderate and severe turbulence conditions [3]. Wang et al. developed pretrained PPO algorithm with reward shaping functions for aircraft guidance in a continuous 3D space. This algorithm was capable of fulfilling complex guidance tasks with high efficiency and performance [53].

2.5.3. Actor-Critic Methods

Actor-critic methods combine elements of policy-based and value-based methods. The actor employs policy-based methods to learn a policy for action selection, while the critic estimates the value function to guide policy updates using TD error as a bootstrapping method [23]. Actor-critic methods combine the advantages of both policy-based and value-based methods, allowing them to handle continuous action spaces like policy-based methods and exhibit reduced variance like value-based methods. However, they also absorb the disadvantages of overestimation and insufficient exploration [6, 47].

Actor-critic methods are suited for flight control tasks due to the ability to handle continuous action spaces. It is important to note that adaptive critic designs (ACDs), mentioned previously in section 2.4 are also actor-critic methods; these are an extension of dynamic programming into the continuous domain.

Synchronous Advantage Actor-Critic

Synchronous Advantage Actor-critic (A2C) estimates the advantage function, how much better an action is compared to the average action at that state. Using the advantage function as a baseline reduces the variance in gradient estimates, leading to more stable learning. This method is synchronous, meaning the algorithm consists of a master node and worker nodes; each worker contains an actor and critic network, and, through interaction with the environment, they estimate the value function and policy. After all workers have finished interacting with the environment, the master node updates the global actor and critic based on the experienced trajectories and synchronises all workers with these updated actor and critic.

Asynchronous Advantage Actor-Critic

The Asynchronous Advantage Actor-critic (A3C) algorithm is similar to A2C, except that the coordinator node is removed. Each agent interacts with the environment, and whenever a worker finishes the gradient computation, the master is updated, increasing computational efficiency compared to A2C.

2.5.4. Combining Deep Q-Networks with Actor-Critic

As explained in subsection 2.5.1, the DQN algorithm is an off-policy method that uses deep neural networks to estimate the value of states. Yet this method only works for discrete, low-dimensional action

spaces. Combining this method with a critic to estimate the policy can yield a method that leverages the advantages of DQN and actor-critic methods. Because of DQN, actor-critic methods can be transformed into off-policy methods, leveraging the replay buffer to improve sample efficiency and reduce sample correlations. Additionally, as an actor-critic method, it can handle problems with high-dimensional, continuous action spaces [6]. The classical algorithms that use this technique are Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), and Soft Actor-Critic (SAC), which are explained in further detail in the next sections.

Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient, combines the Deterministic Policy Gradient and deep neural networks [30]. It is an extension of DQN into continuous action spaces via the use of a critic to estimate the Q-function, and an actor that estimates a deterministic policy [6]. The critic uses TD methods to update the Q-function, as done in DQN, and the actor uses the deterministic policy gradient theorem to update its policy function. Deterministic policy gradient is used, as it can be estimated more efficiently than its stochastic counterpart, since there is no need to compute a problematic integral over the action space [39].

DDPG performs better than DQN and learns 20 times quicker for Atari solutions; however, the method still requires a large number of training episodes when finding solutions for model-free algorithms [30]. Yet this method has proven effective for autonomous UAV navigation, achieving 84% accuracy on a simple obstacle course and 82% on a complex one [4]. DDPG has also been applied to the flight control of a fixed-wing aircraft gliding over an obstacle wall, showing a 57% success rate when crossing the wall [29]. However, the applications of DDPG have been limited to UAVs and small-scale aircraft, and this method has been shown to display overestimation in the critic network [6, 5]

Twin Delayed Deep Deterministic Policy Gradient

The Twin Delayed Deep Deterministic Policy Gradient Algorithm (TD3) improves on DDPG, aiming to reduce its overestimation problem. To reduce overestimation, TD3 uses three techniques:

- **Clipped Double Q-learning:** Since the value function tends to be overestimated with DDPG, TD3 uses two critics, each estimates an independent value function. When learning, TD3 selects the minimum of the two values to compute the Bellman equation. This may introduce underestimation bias, which is preferable to overestimation bias, since underestimated actions will not be propagated through the policy update [6, 12]. A second advantage of this method is that the minimum operator will provide a higher value to the states with the lower variance estimation error, which will lead to more conservative policy updates and more stable learning [12].

- **Delayed Policy Updates:** Deep function approximators need multiple updates to converge to a target network, and target network updates are a good tool to achieve stability. However, policy updates can propagate errors due to high error states. Therefore, the value network (actor) should be updated more frequently than the policy network (critic) in order to achieve smaller variance in the value function, a higher quality policy, and thus stable learning [6].

- **Target Policy Smoothing Regularization:** Noise is added to the target action during critic updates, which smooths the Q-value function and avoids overfitting [6]. This technique mimics the learning from SARSA, making the algorithm bootstrap nearby actions [12].

These three techniques ensure the TD3 algorithm has a better learning speed and performance than the DDPG algorithm, making it more suitable for the continuous control setting. This algorithm has been used to develop a flight controller for the 3D position tracking of a quadcopter, resulting in a success rate of 94% when stabilizing the quadcopter attitude [24]. He et al. developed a TD3fD (from Demonstration) controller for UAV navigation in unknown environments that used expert demonstrations to pre-train the policy and Q-value networks, resulting in faster learning than regular TD3 [18].

Soft Actor-Critic

Soft Actor-Critic (SAC) is a framework that extends soft policy iteration with function approximation. Instead of learning a deterministic policy, like in DDPG, it learns a stochastic policy [6]. SAC was

introduced by Haarnoja et al. [16] to solve the challenges of model-free continuous reinforcement learning. There are three key characteristics of the SAC algorithm:

- It has an actor-critic structure with separate policy and value networks.
- It is an off-policy algorithm, allowing the reuse of past experiences to increase sample efficiency.
- The agent aims to maximise entropy as well as the reward, to ensure stability and exploration [16].

The third characteristic is key, since maximum entropy reinforcement learning has numerous advantages. It guides the policy learning towards high-reward areas, improves exploration and robustness, and can handle model and estimation errors [15, 16]. The objective with the entropy factor is defined in Equation 2.25:

$$\begin{aligned} J(\pi) &= \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r\left(s_t, a_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid s_t\right)\right)\right] \\ \text{with,} \quad & \mathcal{H}\left(\pi\left(\cdot \mid S_t\right)\right)] = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log(\pi(a \mid s))] \end{aligned} \tag{2.25}$$

Where, $\mathcal{H}$ represents the entropy term and α represents the temperature parameter, which defines the importance of entropy against the reward r.

Like TD3, the SAC framework also makes use of a double Q-network, and a double target network, to mitigate the overestimation bias [6]. SAC converges to a stochastic policy, for evaluation, the mean of this policy is taken, making it deterministic to improve performance [16].

The SAC framework has been succesfully applied to flight control tasks. Dally and van Kampen employed a cascaded control structure to demonstrate that a single, offline-trained SAC controller could robustly handle six different unforeseen failures. Their research highlighted that this robustness came at the expense of a challenging training process, with a success rate of only 26% [5]. Similarly, Barros and Colombini applied SAC for the control of a quadrotor with a go-to-target task, achieving a 100% success rate in tests with extreme initial conditions, and even generalizing to track moving targets without being trained to do so [2]. This highlights SAC's ability to produce highly robust and generalizable policies, making it a strong candidate for complex, fault-tolerant flight control applications.

Distributional Soft Actor-Critic

Distributional Soft Actor-Critic (DSAC) extends the SAC framework by modelling the full distribution of future returns rather than only their expected value. Instead of learning a scalar action-value function $Q(\mathbf{s}, \mathbf{a})$, the critic estimates the random return variable $Z(\mathbf{s}, \mathbf{a})$, which represents the discounted cumulative reward. This allows the algorithm to capture both the expected performance and the variability of future outcomes, providing additional information about uncertainty during learning [36] [21]

In practice, the return distribution is approximated using a discrete quantile representation, and the critic is trained through quantile regression with a Huber loss. By learning the full return distribution, DSAC can better represent uncertainty in value estimates, improving stability and robustness in complex control problems such as flight control [36]. However, learning the full distribution increases algorithmic complexity and can introduce noisier gradients during training compared to SAC.

2.6. Classification of Reinforcement Learning Algorithm Training

Reinforcement learning algorithms can be classified according to the type of interaction they rely on during training: online, offline, and hybrid. By design, all of the algorithms developed above operate are classified as online; they interact with the environment continuously, collecting their data and updating the policy as they go [40]. However, online RL algorithms are sample-inefficient, and data collection through interaction with the environment can be dangerous [28]. Due to the sample inefficiency of online methods, research has turned to offline methods, which use large datasets to train the agent, and to hybrid methods, which combine both approaches. These three categories will be presented in further detail in the subsections below.

2.6.1. Online Reinforcement Learning

Online reinforcement learning follows the core principle of RL; the agent interacts with the environment continuously, collecting data, making decisions based on the data and a given policy, and updating the policy iteratively. Online methods have the advantage of being able to adapt to unforeseen conditions, making these methods suitable for fault-tolerant flight control applications.

The most notable drawback to online reinforcement learning is that it is sample inefficient, sometimes ”needing billions of interactions to achieve suitable performance” Song et al. [40]. Additionally, online RL requires active exploration, which can be hazardous in safety-critical aerospace systems, as exploration early on might lead to actions that jeopardize the integrity of the system.

2.6.2. Offline Reinforcement Learning

Offline reinforcement learning, also referred to as batch RL, relies on datasets collected previously in order to train a policy, without any additional interaction with the environment [28]. This alternative is attractive for flight control, as agents can be trained with large datasets without the risks of unsafe exploration.

A well-known disadvantage of offline learning is the problem of distributional shift: if the agent encounters state-action distributions during application different from those encountered during training, performance might degrade sharply. This can pose a problem for fault-tolerant flight control applications, where actuator or sensor failures encountered in flight would not have been encountered during training [28]. However, there are techniques used to mitigate distributional shift.

Recent work by Homola et al. [21] introduced an offline reinforcement learning algorithm using an Uncertainty-Driven Distributional Soft Actor-Critic framework. This controller shows improved learning efficiency and accuracy when compared with SAC and DSAC controllers, and the article recommends a future implementation of this algorithm in an online control setting.

2.6.3. Hybrid Reinforcement Learning

Hybrid reinforcement learning combines the strengths of both offline and online approaches [40]. This approach is characterized by using an offline dataset to pre-train the algorithm, and then the policy is fine-tuned during operation through interaction with the environment and the use of an online RL algorithm. This allows greater sample efficiency and the ability to deal with unexpected faults or disturbances.

Recent work by Teirlinck [49] implemented a cascaded controller using a hybrid SAC-iDHP framework, resulting in improved tracking performance when compared to a SAC controller, and demonstrating robustness to biased sensor noise. This work was later improved by Vieira [51], who used a DSAC framework for the offline algorithm in the hybrid network, and demonstrated that the DSAC-iDHP controller was more robust than the SAC-iDHP controller, and baseline DSAC and SAC offline controllers. By combining safety, sample efficiency, and adaptability, hybrid reinforcement learning frameworks are more suitable for flight control applications. The offline phase reduces the risky exploration, while online adaptation ensures fault-tolerance when faced with adverse conditions.

2.7. State-of-the-Art Reinforcement Learning Algorithms for Flight Control Applications

The theoretical foundations of RL described in the previous section provide a wide array of algorithms, each with varying working principles and characteristics. However, their application to the task of fault-tolerant flight control for a fixed-winged aircraft requires careful consideration. This section analyses some state-of-the-art algorithms from a selection of papers that have applied reinforcement learning techniques to control a fixed-winged aircraft.

To provide some further context, Table 2.1 summarizes ten key papers in the field of reinforcement

learning for flight control, outlining their core contributions to the field and differentiating between the learning algorithm and the type of learning (offline, online, or hybrid). In the following section, five key papers from the ones present in Table 2.1 are reviewed, to have a better understanding of the field and select the best approach for the controller to be designed

Table 2.1: Summary of State-of-the-Art Reinforcement Learning Papers for Flight Control

Paper (Author, Year)	Algorithm(s)	Control Task	Paradigm	Key Contribution & Relevance
Dally & Van Kampen (2022) [5]	SAC	6-DOF Fault-Tolerant Control	Offline	Establishes a benchmark for robust offline fault-tolerant DRL controllers.
Heyer, Kroezen, & Van Kampen (2020) [19]	IDHP + PID	Online Adaptive Rate Control	Online	Demonstrates online, adaptive RL controller (iADP) for a CS-25 class aircraft with high sample efficiency.
Teirlinck (2022) [49]	SAC-IDHP	6-DOF Fault-Tolerant Control	Hybrid	Combines offline SAC with online IDHP, showing the hybrid controller outperforms the standalone SAC.
Seres (2022) [36]	DSAC	Attitude Control	Offline	Improves on SAC by using distributional RL for more stable learning.
Vieira dos Santos (2023) [51]	DSAC-IDHP	Attitude Control (Fault-Tolerant)	Hybrid	Builds directly on Teirlinck [49] and Seres [36] by creating a hybrid with the more advanced DSAC.
Lee & van Kampen (2021) [27]	IDHP	Longitudinal Altitude Control	Online	Extends IDHP to a purely online cascaded structure, removing the need for PIDs in the outer loop.
Bøhn et al. (2019) [3]	PPO	Attitude Control	Offline	A key proof-of-concept for using PPO for attitude control, demonstrating robustness to turbulence.
Wang et al. (2021) [53]	PPO	3D Aircraft Guidance	Offline	Uses pre-training and reward shaping for an aircraft guidance task.
Li & van Kampen (2024) [29]	DDPG	Longitudinal Obstacle Crossing	Offline	Application of DDPG, demonstrates DRL for a complex task.
Homola, Li, & van Kampen (2025) [21]	RUN-DSAC	Attitude Control	Offline	Cutting-edge enhancement to DSAC using uncertainty to guide policy decisions, improving safety and performance.

Offline Cascaded SAC Controller | Dally and Van Kampen (2022)

Dally and Van Kampen [5] developed a cascaded SAC controller for a Cessna 500 Citation aircraft able to withstand different failure types. The controller was trained offline, due to the challenges arising from the low sample efficiency of online training methods. The authors implemented a cascaded control structure, where an inner loop agent was responsible for tracking the attitude, while an outer loop was responsible for tracking the altitude. This controller can be observed Figure 2.4

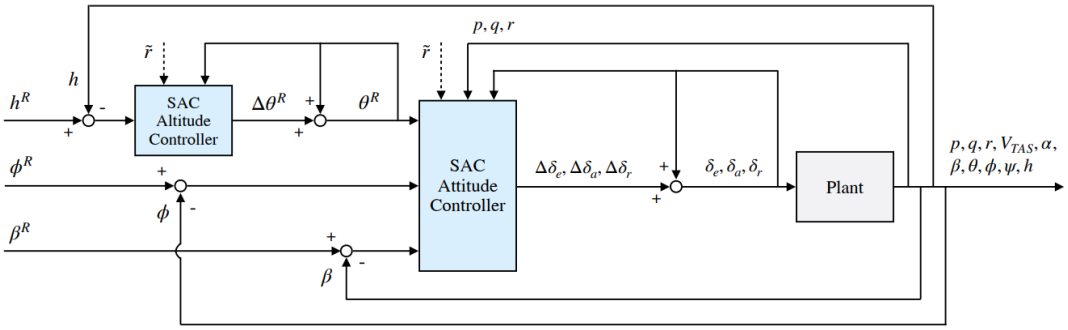


Figure 2.4: Cascaded controller from Dally and Van Kampen [5]

The inner loop was trained first, with initial attitude and speed of 2000 m and 90 m/s respectively. After 10^6 timesteps, training reached a plateau and the outer loop was trained, with the fully trained attitude control inner loop. The results show that under nominal conditions, the SAC agent shows good tracking performance, keeping the tracking error low when achieving 40 bank climbing turns and 70 bank flat turns. Additionally, to demonstrate the fault-tolerance of this algorithm, it was tested against six different failure cases, such as a jammed rudder, 70% reduction in aileron effectiveness, structural failure, icing, and a shift in the centre of gravity. For example, Figure 2.5 shows the tracking response of the controller when 70% reduced aileron effectiveness occurs from the 30th second onwards, as can be seen, the controller (blue line) follows the reference signal (red line) effectively.

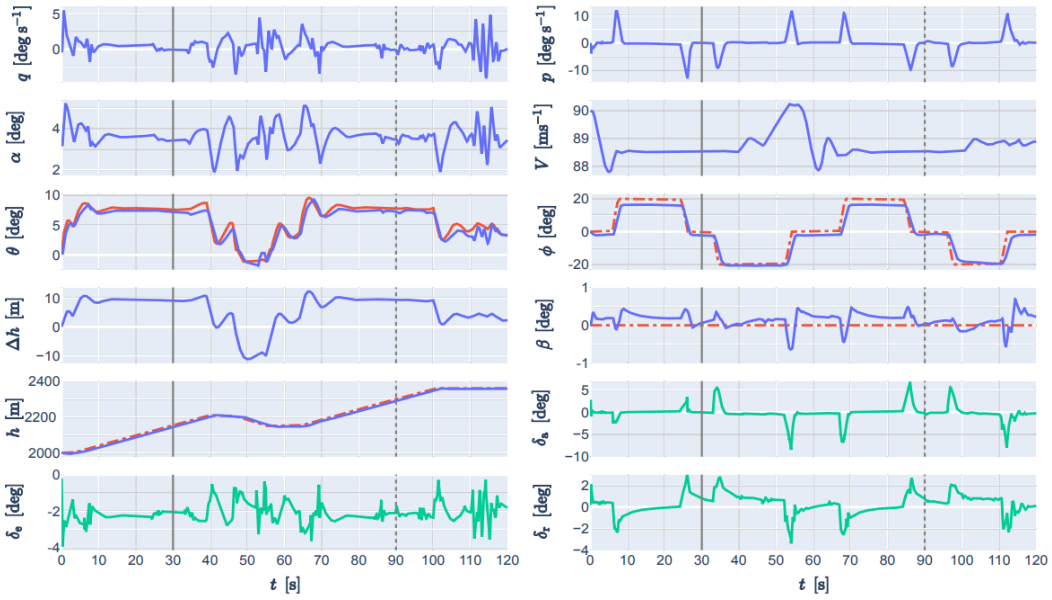


Figure 2.5: Caption

The performance when dealing with adverse flight conditions shows robustness, even when the agent had not experienced any of these conditions during training. This demonstrates that offline controllers can perform well, without risking the safety of the aircraft with any online exploration. SAC's stochastic policy and deep neural network allowed for better generalization and enhanced exploration during offline training, however, it must be noted that a drawback to this approach is the complexity of the learning procedure.

Online Adaptive IDHP Controller | Heyer, Kroezen, and Van Kampen (2020)

In contrast to the offline approach, the research paper by Jeyer, Kroezen, and van Kampen [19] explores a purely online and adaptive framework to develop a controller for the Delft University Aircraft Simulation

Model and Analysis Tool (DASMAT), a high-fidelity simulation model of the Cessna 550 Citation II aircraft. This framework is able to learn a near-optimal policy online, without any prior knowledge of the system dynamics and without an offline training phase.

The learning framework, shown in Figure 2.6 is composed of three parametric structures: an actor that approximates the control policy, a critic that estimates the partial derivative of the state-value function with respect to the state, and the incremental model of the plant. Additionally, a separate target critic is introduced to improve learning stability, at the expense of slower learning. The incremental plant model is a linear approximation of the plant using a first-order Taylor series expansion about an operating point.

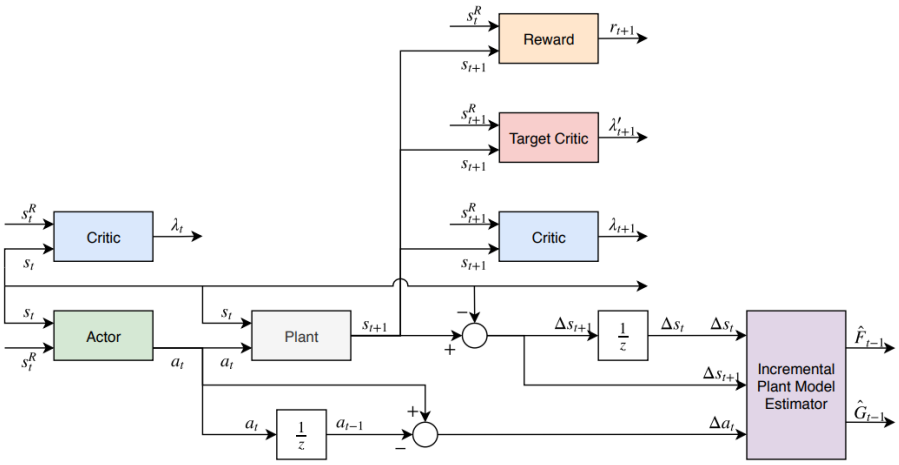


Figure 2.6: Schematic of the feed-fowrward signal flow of the learning framework [19]

The controller was structured with an inner loop for pithc and roll rate control, augmented by an outer loop consisting of PID controllers for altitude and roll angle tracking.

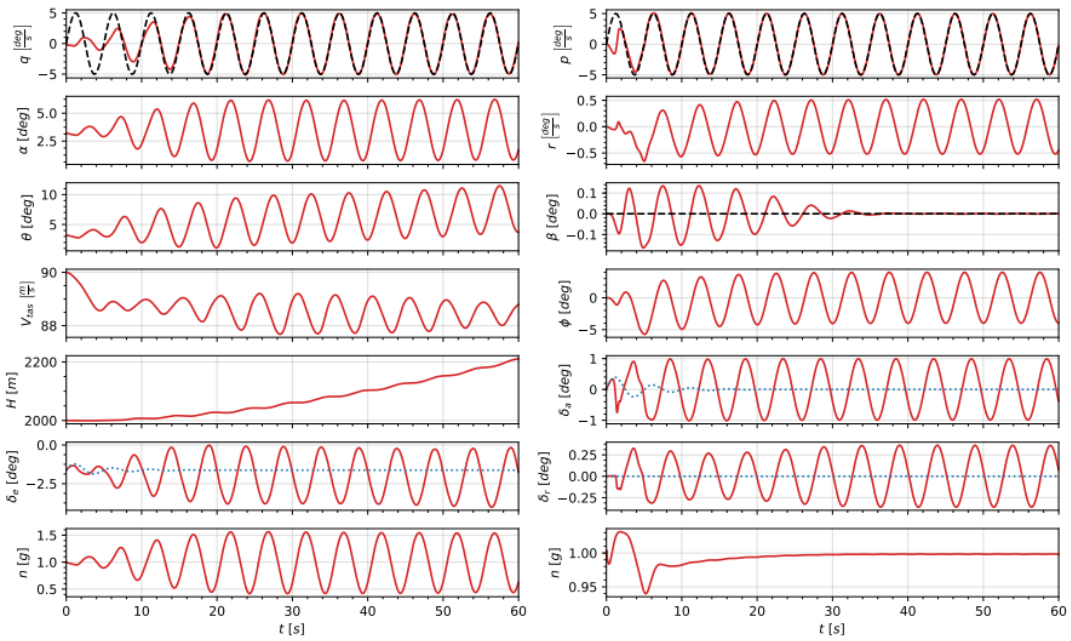


Figure 2.7: Caption

The results show that the iDHP controller is capable of learning near-optimal control policy with no system

knowledge and no offline training phase. Figure 2.7 shows that under nominal flight conditions, the online controller manages to follow reference signals in under 30 seconds. To test the fault-tolerance of the controller, it was tested under the conditions of aileron failure and a control disturbance. The controller can perform a right turn despite aileron failure and adapts to the control disturbance within 10 seconds.

This paper solidifies iDHP as a state-of-the-art method for online adaptive control. It can learn without a pre-existing model. However, a future recommendation would be to reduce the dependence of traditional PID controllers, making the system less dependent on the model. However, the research done by Lee and van Kampen [27] showed that iDHP alone does not outperform the structure that contains a pre-tuned PID controller in this paper.

Hybrid Cascaded SAC-iDHP Controller | Teirlinck (2022)

Teirlinck [49] combines the approaches in the previous two papers to make a novel hybrid SAC-iDHP adaptive controller for the Cessna 550 Citation II simulation. The author created a cascaded controller, where the inner loop uses the hybrid architecture to control the attitude of the aircraft, and the outer loop only uses the SAC framework for altitude tracking, as seen in Figure 2.8

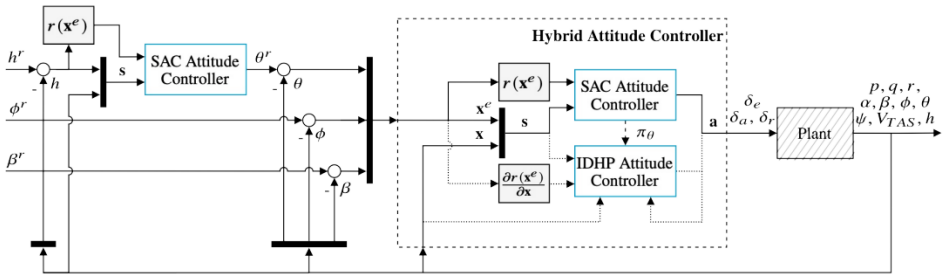


Figure 2.8: Structure of hybrid cascaded controller from Teirlinck [49]. Please note that the block in the outer loop should be called "SAC Altitude Controller"

The SAC altitude and attitude controllers are trained offline, and the iDHP controller was trained online. Traditional iDHP frameworks require initial excitation in order to train the signal, however, in the hybrid architecture, the pre-existing converged SAC policy can provide this initial excitation.

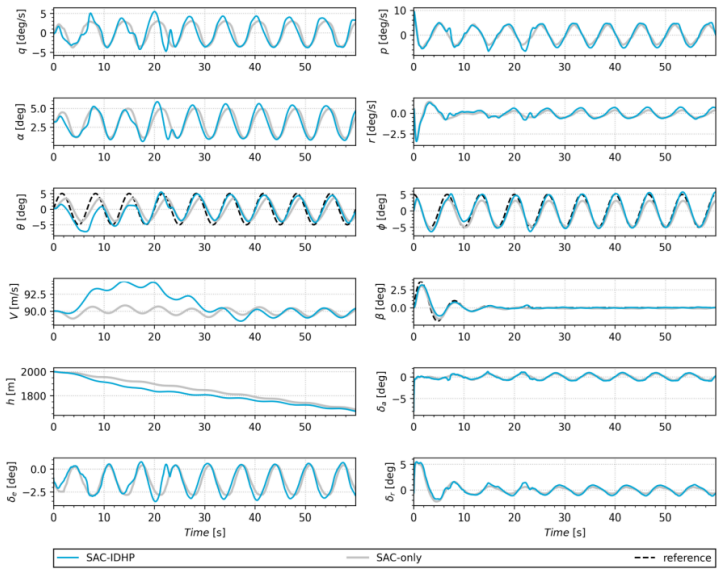


Figure 2.9: Comparison between SAC-iDHP and SAC-only controller for a tracking task under nominal flight conditions from Teirlinck [49]

Under nominal flight conditions, the hybrid SAC-iDHP framework outperforms an SAC-only framework in an attitude tracking task, improving performance by 2.9% as seen in Figure 2.9. The results also show a small improvement of 0.74% in an altitude tracking task. Additionally, under the conditions of 70% reduced elevator aileron effectiveness and 90% reduced aileron effectiveness, the hybrid controller showed an improvement of 5.76% and 0.82% in tracking performance. In addition to this, compared to the SAC-only controller, the hybrid controller displays more robustness to initial flight conditions (IFC).

Hybrid DSAC-iDHP Controller | Vieira dos Santos (2023)

Vieira dos Santos [51] directly improves on the work by Teirlinck [49] by using a distributional SAC framework for the offline component of a hybrid DSAC-iDHP controller for the simulation of the Cessna 550 Citation II. DSAC has shown to improve aircraft control due to its ability to create safer policies, as it learns complete distributions and provides deeper insight into the risk of different actions. Vieira dos Santos deemed this framework more suitable for robust control as it is risk-sensitive and displays a more stable learning performance than SAC.

The DSAC-iDHP hybrid framework was trained by first training the DSAC framework offline, then its output policy was used as the initial excitation to train the online iDHP. The performance of the hybrid DSAC-iDHP controller was compared against a hybrid SAC-iDHP controller, a DSAC-only offline controller and an SAC-only offline controller. The goal of this research was to evaluate the robustness and fault-tolerance of the novel framework.

	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%

Figure 2.10: Tracking performance of different controllers subject to three different tracking tasks, the mean and variance of the tracking nMAE is displayed from Vieira [51]

Figure 2.10 displays the performance of all four controllers under three different tracking tasks, showing that the DSAC-iDHP hybrid framework outperforms the offline controllers significantly, and the other hybrid framework. Under sensor noise and bias, the hybrid DSAC controller outperforms the hybrid SAC controller by 1.2%, displaying improved robustness. When faced with reduced elevator effectiveness of 70% and reduced aileron effectiveness of 90%, the hybrid-DSAC framework also outperforms the hybrid-SAC framework by 1.1% and 3% respectively. The novel DSAC-iDHP framework is an improvement to the SAC-iDHP framework implemented by Teirlinck [49], showing increased tracking performance, improved robustness, and fault-tolerance.

Uncertainty-Driven Distributional RL | Homola, Li, and van Kampen (2025)

The research by Homola, Li, and van Kampen [21] introduces a novel variant of the DSAC algorithm applicable to safety-critical systems. This method doesn't just use the mean of the learned return distribution like DSAC methods, but uses the uncertainty in the return to guide decisions, improving the safety of flight controllers. The Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC) algorithm is designed to combine sample efficiency and safe flight control, resulting in a better tracking performance, and improved fault-tolerance and robustness when favouring low-uncertainty state-action pairs.

The core mechanism of RUN-DSAC is the modification of the learning objective. Instead of optimizing the expected return $\Psi(Z) = \mathbb{E}[Z]$, RUN-DSAC optimizes a weighted sum between the expected return and the standard deviation $\Psi(Z) = \mathbb{E}[Z] + \mu\sqrt{\mathbb{V}[Z]}$, where $\mathbb{V}[\cdot]$ represents the variance and μ the uncertainty modulation factor dictating the agent's risk preference. When the uncertainty modulation factor is positive, the policy is driven towards actions with high variance, referred to as *Risky* exploration, and when μ is negative, lower-variance actions are favoured, referred to as *Conservative*. The policy update can thus be written as:

$$Q_\theta^V(s,a) = Q_\theta(s,a) + \mu\sigma_Q \tag{2.26}$$

In Equation 2.26 Q_θ represents the expected return and σ_Q the standard deviation. The architecture of this algorithm can be seen in Figure 2.11

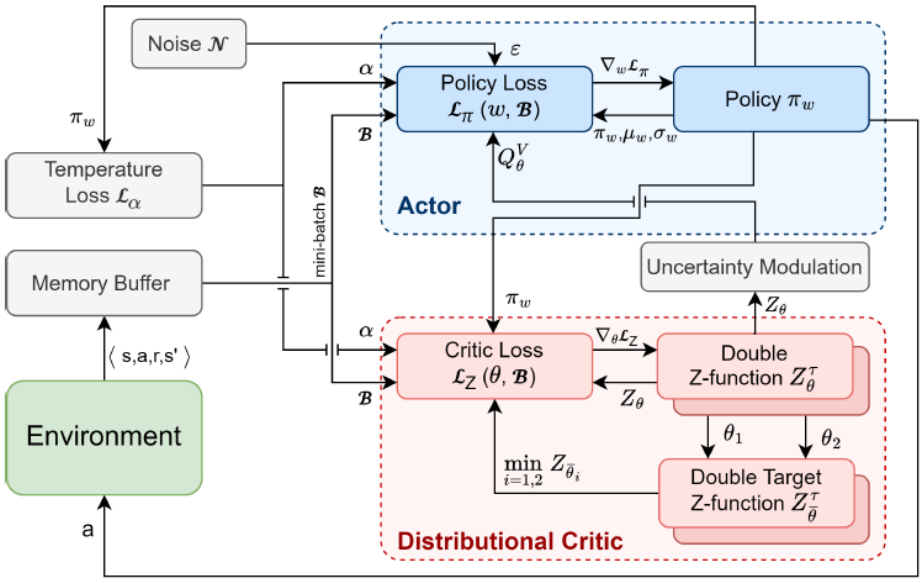
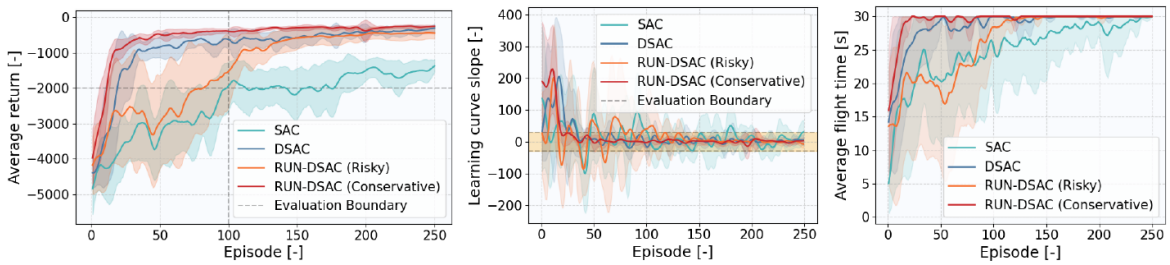


Figure 2.11: Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC) architecture from Homola, Li, and Kampen [21]

The first notable contribution of RUN-DSAC is the improvement in learning performance and stability. The Conservative RUN-DSAC agent exhibits a more efficient and stable learning process when compared to SAC, DSAC, and Risky RUN-DSAC. As shown in the learning curves in Figure 2.12, the Conservative agent converges faster and to a higher final reward. Additionally, the shaded area in the learning curve for Conservative RUN-DSAC is smaller than that of the other agents, indicating less variance across training runs compared to the different algorithms. This indicates a more reliable and predictable training process, addressing some of the inconsistencies encountered with SAC. The Conservative variant outperforms DSAC across all five learning metrics used, with a considerable 54.5% improvement in the gradient of the learning curve. It is worth noting that the Risky RUN-DSAC agent performed worse than the DSAC agent across all metrics, and shows a worse learning performance as seen in Figure 2.12.



(a) Learning curves. **(b) Learning curve slopes.** **(c) Episodic flight time.**

Figure 2.12: Comparison of learning performance between SAC, DSAC, Conservative RUN-DSAC, and Risky RUN-DSAC from Homola, Li, and Kampen [21]

Additionally, the Conservative RUN-DSAC agent demonstrated improved tracking performance in the nominal case, with an nMAE of 3.95±0.42%. However, its generalization capabilities are better displayed in Figure 2.13, where the Conservative RUN-DSAC agent displays a lower nMAE across five unseen flight conditions. The Conservative agent also displays a lower variance across all flight conditions, indicating the increased robustness of the agent. It is worth noting the Conservative agent's performance

in the near-stall conditions (G5), where choosing low-risk actions enables the agent to obtain higher rewards compared to the others.

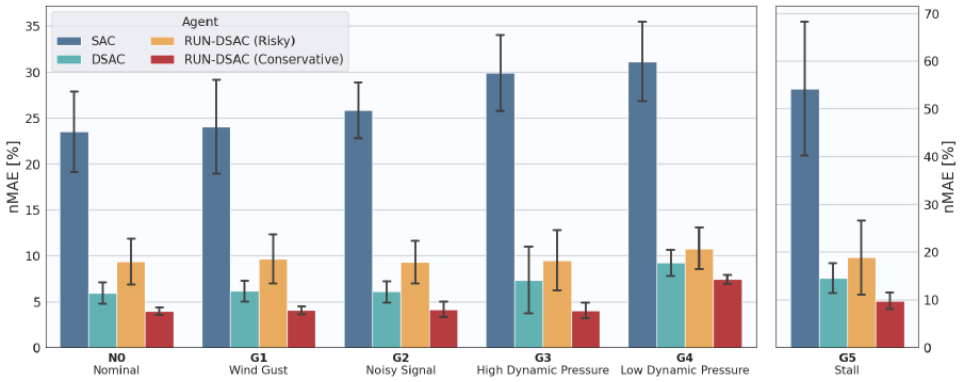


Figure 2.13: Attitude tracking nMAe and standard deviation across different generalization scenarios from Homola, Li, and Kampen [21]

Additionally, this paper tests the developed agents under seven different fault scenarios. The results show that Conservative RUN-DSAC had the best tracking performance across all these fault scenarios, with the lowest nMAE and variance, followed by the DSAC agent as seen in Figure 2.14. This superior fault-tolerance can be attributed to the risk-averse policy learned by the Conservative agent, which, by avoiding uncertainty, favours more stable and predictable regions of the state-action space. In contrast, the Risky agent behaves in an aggressive and sometimes unstable manner, resulting in lower rewards and thus lower tracking performance across all fault-tolerant scenarios.

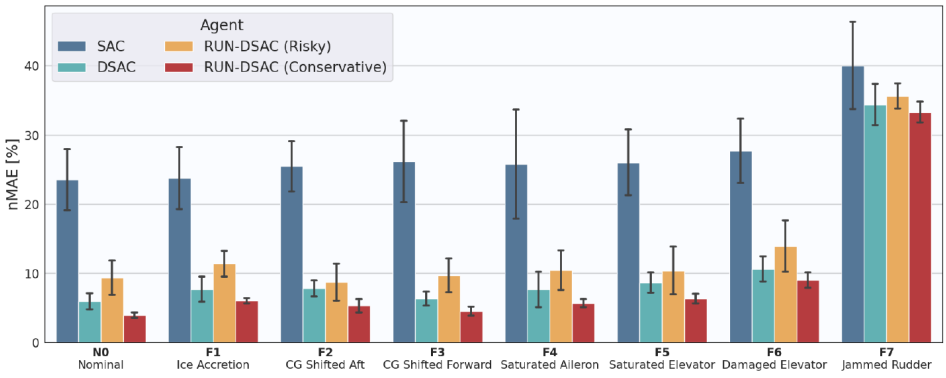


Figure 2.14: Attitude tracking nMAE and standard deviation across different fault scenarios from Homola, Li, and Kampen [21]

The work by Homola, Li, and van Kampen showcases the enhanced learning efficiency and attitude tracking accuracy of the Conservative RUN-DSAC algorithm. The algorithm's superior performance, reliability, and fault tolerance make it a suitable candidate to be integrated into a hybrid RL controller, combining its offline advantages with the adaptability of an online component such as iDHP.

2.8. Concluding Remarks

Ensuring fault-tolerant control is fundamental for the safety and reliability of modern aircraft. Traditional flight control methods, such as linear controllers with gain scheduling, nonlinear dynamic inversion, or backstepping, fail to adapt to unforeseen failures and adverse flight conditions, depending on robust control techniques to reduce the dependence on perfect models [41]. Incremental methods, such as INDI and IBS, have further reduced model dependence, yet these approaches are fairly limited, relying on accurate sensor measurements and meticulous tuning [1, 38]. These controllers fail to guarantee safe performance when flying beyond the modeled envelope or dealing with complex nonlinearities as a

consequence of failure.

This challenge has motivated the use of reinforcement learning to develop model-free, data-driven methods for flight control. By learning control policies through interaction with the environment, reinforcement learning allows controllers to adapt to highly nonlinear system dynamics, unexpected perturbations, and failures too complex to model. The extension of tabular reinforcement learning methods with function approximators has made it possible to apply these algorithms to high-dimensional systems with continuous control tasks, making them suitable for aerospace systems [48]. The combination of adaptability and scalability makes reinforcement learning methods an attractive alternative for fault-tolerant flight control systems.

Early applications in aerospace used Approximate Dynamic Programming (ADP) algorithms, such as IDHP, IHDP, and IGDHP, which showed real-time adaptation capabilities for tasks like attitude tracking and trajectory tracking [58, 57, 56]. These adaptive controllers learn online and can adapt to changing flight conditions, learning new behaviours by interacting with the environment. However, they have a lower sample efficiency when learning, and online exploration can lead the aircraft to unsafe flight conditions in the search for a better policy.

Deep reinforcement learning (DRL) methods address these limitations when applied offline. Algorithms like TRPO and PPO [35, 6] improved learning stability, and actor-critic methods like DDPG, TD3, and SAC, [15, 12, 30] achieved strong performance in continuous control tasks, including UAV navigation and aircraft attitude control [5, 3]. Distributional SAC introduced distributional value estimation to enhance robustness, and RUN-DSAC recently extended these advantages by incorporating mechanisms to handle uncertainty, making these algorithms suitable for safety-critical, fault-tolerant aerospace settings [21]. However, when applied offline, these methods lack the adaptability capabilities of online methods, while displaying a higher sample efficiency in training.

To overcome the limitations of offline-only and online-only reinforcement learning approaches, hybrid solutions have emerged as a promising alternative. These frameworks integrate the sample efficiency and safety of offline algorithms with the adaptability of online algorithms. Prior research has demonstrated the effectiveness of these frameworks in aerospace control applications, where safety and adaptability are necessary. Teirlinck [49] combined SAC with IDHP to form a hybrid controller for the Cessna 550 Citation II, which outperformed its online and offline components in tracking performance under nominal and fault conditions. Vieira [51] further extended this research using a DSAC-iDHP, improving tracking accuracy and fault-tolerance against actuator failures. These results demonstrate that hybrid reinforcement learning offers an ideal foundation for developing a robust and fault-tolerant flight control system.

Based on the outcomes of previous hybrid implementations, this research proposes the implementation of a controller consisting of an offline RUN-DSAC agent and an online IDHP agent. RUN-DSAC is selected due to its advantages when handling uncertainty, its improved tracking performance, and its improved learning performance. The IDHP agent complements this by enabling real-time policy improvement through incremental updates, allowing the control to adapt to evolving flight conditions and unforeseen faults.

Part III

Additional Results

3

Offline Agent Fine-Tuning

In order to compare the agents fairly, it is first necessary to tune each of the agents. First, the offline agents are tuned. Since these are trained over a large number of episodes, the agent's performance gradually improves as training progresses. Yet, the different hyperparameters strongly influence the control signal and the agents' learning curves.

3.1. Offline Tuning Methodology

Once the offline agents have been implemented, a dedicated fine-tuning procedure is required to ensure fair performance across all operating conditions. Given the high computational cost associated with training DRL agents from scratch, a transfer-based tuning strategy is implemented.

For each offline algorithm (SAC, DSAC, and RUN-DSAC), a fully trained agent is first selected. The configuration of each of these agents that yielded the best performance is then selected to initialize the tuning procedure. This baseline provides a well-performing starting policy, allowing the tuning process to focus on performance refinement rather than global convergence.

Fine-tuning is then performed by retraining the baseline agent under a different trimmed flight condition while varying a single hyperparameter being analysed. The remaining hyperparameter values remain fixed to those of the baseline agent. To ensure statistical relevance, each hyperparameter combination is evaluated across multiple random seeds. For each training run, the agent is trained for a limited number of timesteps, between 100,000 and 150,000, to reduce the computational cost of offline agent fine-tuning. Compared to the 500,000 to 1,000,000 timesteps used to train the offline agents fully, this training horizon is quite limited, but due to the strong initialization provided by the pretrained base, it was sufficient to achieve good tracking performance and allow comparison between hyperparameter values. The best checkpoint for each of the hyperparameter configurations is then evaluated, as well as the learning curves of the tuning procedure, in order to determine the effect of the varied hyperparameter on performance. This procedure is repeated across different hyperparameters under consideration.

This tuning process is applied across offline agents using the same task definitions, evaluation metrics, and training durations. Performance comparisons are based on tracking performance, measured using the nMAE of the tracked signals, and a qualitative assessment of the control signal behaviour. This ensures that improvement in tracking performance is not accompanied by aggressive and oscillatory control signals. The fine-tuned agents form the basis for hybrid agent initialization and fault-tolerant performance evaluation.

3.2. Learning Behaviour and Hyperparameter Sensitivity

3.2.1. Effect of Learning Rate

The learning rate is the most influential hyperparameter in the training of DRL agents, as it governs how strongly the policy parameters are updated in the direction of improvement. It affects both the convergence speed and training stability. To assess the effect of different learning rates, the procedure described in section 3.1 was performed, varying the learning rates while keeping all other hyperparameters constant.

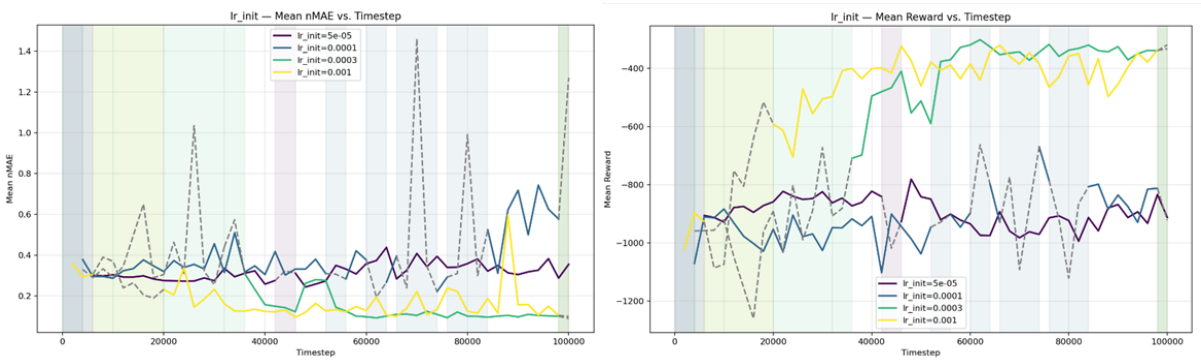


Figure 3.1: normalized Mean Absolute Error (nMAE) and Mean Reward Against Time for SAC Agents Trained with Varying Initial Learning Rates

Figure 3.1 shows the learning curves and tracking performance for the SAC agent trained with different learning rates. The figure indicates that the learning rate strongly influences the tracking error. When this value is too low (e.g., $lr = 0.00005$ or $lr = 0.0001$), the tracking error is higher, and the reward is lower. In addition, when the learning rate lies within an appropriate range, the agents converge to comparable final performance levels in tracking accuracy and reward, as evidenced by the nMAE troughs achieved by the agents with $lr = 0.001$ and $lr = 0.0003$ in the figure on the left. The primary effect of the learning rate is on the speed of convergence, rather than the achievable steady-state performance. Higher learning rates show faster convergence and a faster increase in accumulated reward.

These results show that if the learning rate lies within a suitable region, similar performance can be achieved, and the rate determines how quickly the agent approaches this optimum. However, if the rates are too low, optimal performance won't be achieved, and if they are too high, the variance increases during training, degrading tracking accuracy. Based on these observations, learning rates that balance convergence speed and tracking accuracy are selected for each offline agent.

3.2.2. Effect of Batch Size and Target Update Rule

In addition to the learning rate, the influence of other hyperparameters is examined using the tuning procedure described in section 3.1. The effects of varying batch size and critic soft-update parameter τ are shown in Figure 3.2 for the SAC agent.
The results in Figure 3.2 show that variations of the batch size or the soft-update parameter τ have no effect on learning, and all agents achieve similar tracking accuracy after 100,000 timesteps. Similar experiments were performed by varying the buffer size and the discount factor γ, yielding similar results. Although slight differences can be seen in the convergence speed, these hyperparameters have no significant effect on the tracking performance or learning of the offline agents. For this reason, default values for these were selected, shown in Table 3.4

3.3. Control Signal Quality and Regularization

The time response of the SAC agent subjected to reference signals in all attitude channels can be seen in Figure 3.3. It is clear that the controller dislays a high tracking accuracy; however, the control signals

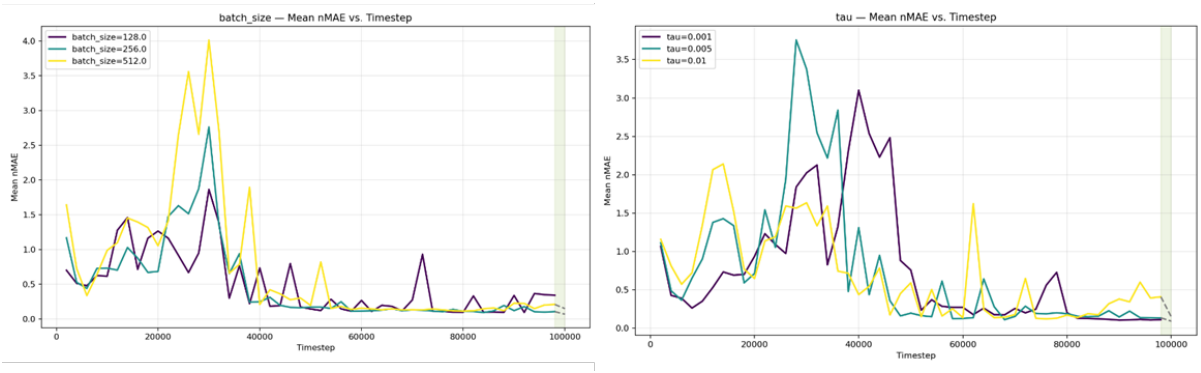


Figure 3.2: Enter Caption

for the actuators are highly oscillatory. Essentially, the agent is following the reference signal by making high-frequency and high-magnitude adjustments to each of the actuators.

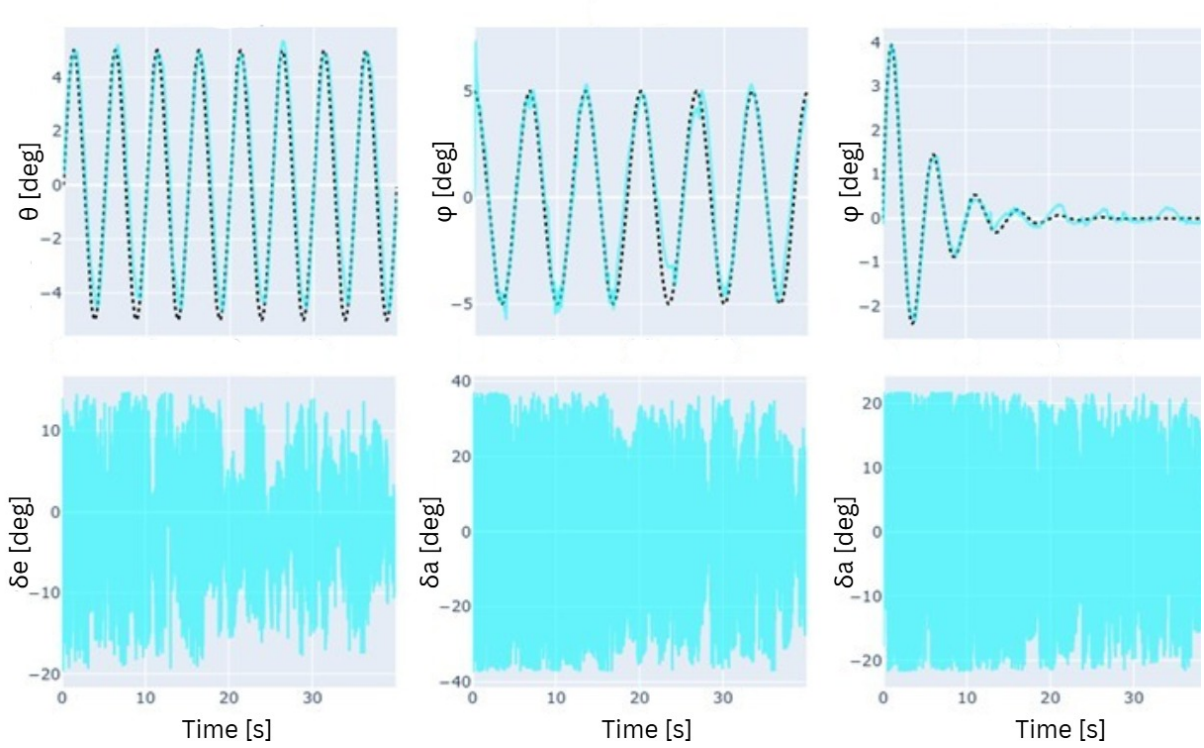


Figure 3.3: SAC Time Response with θ, ϕ, and β Reference Signals

The behaviour seen in the bottom three plots of Figure 3.3 is undesirable, as it could lead to excessive wear of the actuators and actuator saturation. To address this issue, additional regularization is necessary to penalize aggressive control actions during training. This is achieved with a technique called Control-Aware Policy Smoothing (CAPS) regularization.

3.3.1. CAPS Regularization

CAPS regularization augments the actor objective by introducing two penalty terms. A temporal regularization loss, defined in Equation 3.1, was used to encourage actions to be close to the previous action. Additionally, a spatial regularisation term, defined in Equation 3.2, was used to encourage actions to be near a chosen action from a Gaussian distribution, $\bar{s} \sim N(s, 0.05)$, around the current state.

$$\mathcal{L}_T = D(\pi(s_t, s_{t+1})) = || \pi(s_t) - \pi(s_{t+1}) ||_2 \quad (3.1) \qquad \mathcal{L}_S = D(\pi(s_t, \bar{s})) = ||\pi(s_t) - \pi(\bar{s})||_2 \quad (3.2)$$

The terms $\mathcal{L}_\mathcal{T}$ and $\mathcal{L}_f$ are incorporated into the actor's loss formulation, accompanied by two weights λ_T and λ_S respectively. These weights dictate the importance of each of the regularization terms and must be tuned appropriately.

While CAPS regularization introduces explicit penalties on aggressive control actions, it introduces a trade-off between tracking performance and control signal smoothness. Low values of the CAPS coefficients result in excellent tracking performance with highly oscillatory control signals, and vice versa. Consequently, CAPS coefficients must be tuned carefully to ensure that improvements in the quality of the control signal do not hinder the controller's ability to follow a reference signal. The following subsection outlines the procedure for systematically evaluating this trade-off and identifying suitable CAPS coefficients for each offline agent.

3.3.2. Evaluation Methodology

To tune these two hyperparameters, a grid search is performed over different values for λ_T, λ_S, and algorithm-specific hyperparameters. In the case of SAC, only the two regularization weights are used; DSAC includes the hyperparameter ξ, which influences Wang's risk distortion by making the algorithm risk-averse ($\xi < 0$), risk-neutral ($\xi = 0$), or risk-prone ($\xi > 0$); Finally, RUN-DSAC includes the hyperparameter μ_{init}, the algorithm's risk-modulation parameter, which also follows the same rules as ξ for DSAC. For each combination of regularization weights and algorithm-specific hyperparameter, the procedure described in section 3.1 is performed.

3.4. Summary of Offline Tuning Outcomes

The SAC agent is the first agent to be tuned. Before determining the grid with the CAPS weights values, different agents are trained with discrete values for $\lambda_T = \lambda_S$. The simulation of these trained agents is seen in Figure 3.4.

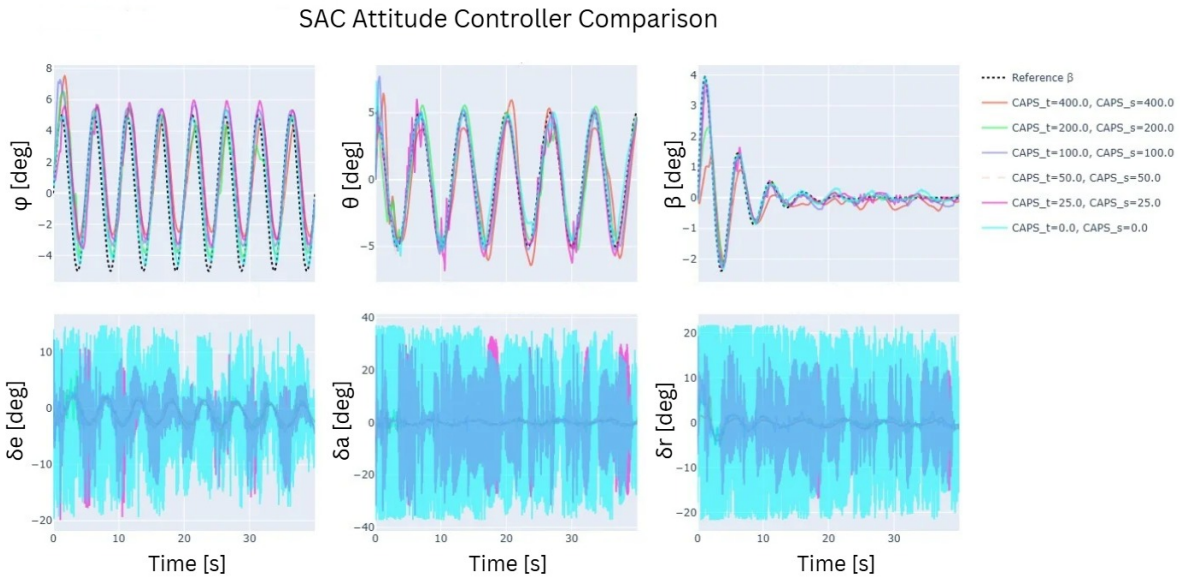


Figure 3.4: Time-Response Comparison Between Fully-Trained SAC Agents with Varying CAPS Weights

It becomes evident that increasing the weights on the CAPS regularization parameters smooths out the control signal, which starts very oscillatory when the coefficients are 0, but these oscillations decrease in magnitude and frequency as the CAPS coefficients are increased. This, unfortunately, is accompanied by a reduction in tracking performance seen in Table 3.1. It is also evident that values for $\lambda_t = \lambda_s = 50.0$

Table 3.1: Effect of CAPS Coefficients on Tracking Performance for SAC (nMAE in %)

CAPS Configuration	β_{nMAE} [%]	θ_{nMAE} [%]	ϕ_{nMAE} [%]	$\textbf{nMAE}_{\textbf{total}}$ [%]
$\lambda_T = 0.0,\ \lambda_S = 0.0$	2.53	13.95	12.15	9.54
$\lambda_T = 25.0,\ \lambda_S = 25.0$	3.29	33.97	11.97	16.41
$\lambda_T = 50.0,\ \lambda_S = 50.0$	3.97	24.86	9.16	12.66
$\lambda_T = 100.0,\ \lambda_S = 100.0$	3.95	24.99	12.87	13.94
$\lambda_T = 200.0,\ \lambda_S = 200.0$	4.23	26.18	19.02	16.48
$\lambda_T = 400.0,\ \lambda_S = 400.0$	8.85	37.34	26.42	24.20

result in the best performance.

For this reason, the hyperparameter grid search is performed with the following grid:

$$\begin{aligned} \lambda_T &\in \{30,\ 40,\ 50,\ 60\}, \\ \lambda_S &\in \{30,\ 40,\ 50,\ 60\}, \end{aligned} \tag{3.3}$$

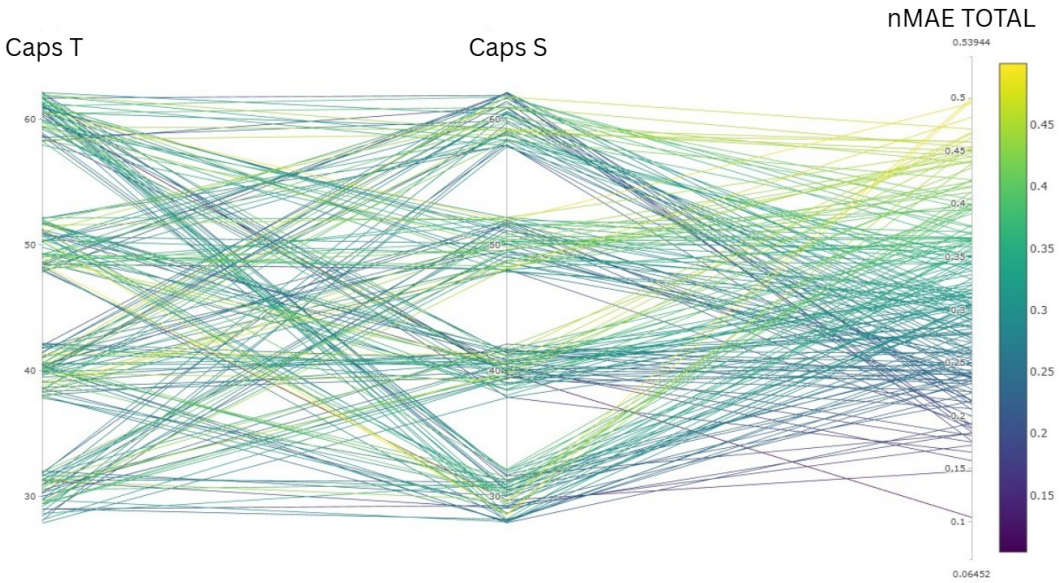


Figure 3.5: SAC Agent CAPS Coefficient Fine-Tuning Parallel Axis Plot

Figure 3.5 shows the results for the simulations across 8 different seeds. The figure shows little variation across combinations, and the same combinations can yield a wide range of results depending on the seed. Thus, the fully trained agent $\lambda_t = \lambda_s = 50.0$ is chosen as the base for the SAC agent. Its time response under nominal conditions is shown in Figure 3.6.

The same procedure is repeated for DSAC. First, fully trained agents with varying CAPS weights are compared to establish a baseline for the grid search. From the results in Table 3.2, it can be inferred that, similar to SAC, the higher the CAPS coefficients, the smoother the control signals at the expense of tracking accuracy. After a closer inspection of the responses of the compared agents, values of approximately 0.625 for λ_T and λ_S are selected to refine the search.

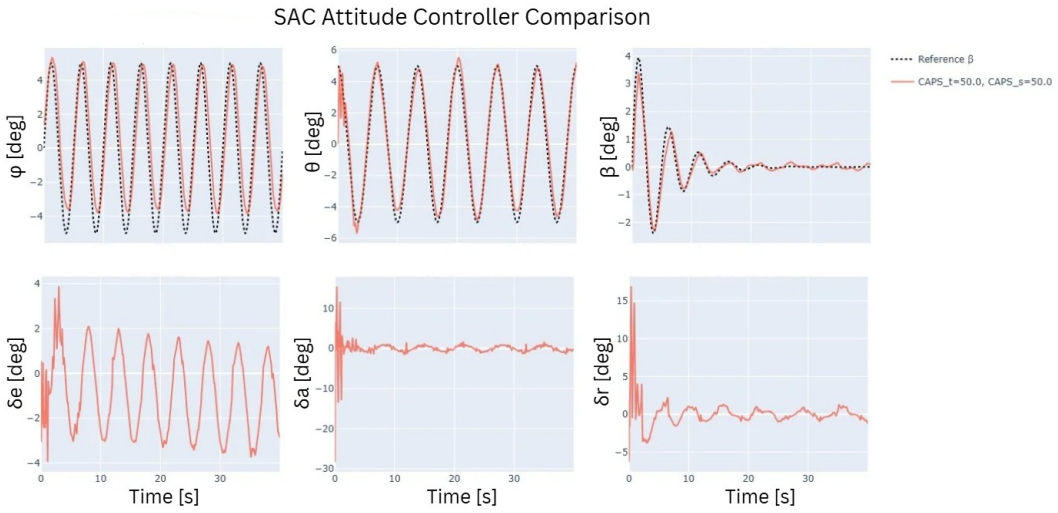


Figure 3.6: Tracked States and Control Signal Response for Tuned SAC under Nominal Conditions

Table 3.2: Effect of CAPS Coefficients on Tracking Performance for DSAC (nMAE in %)

CAPS Configuration	β_{nMAE} [%]	θ_{nMAE} [%]	ϕ_{nMAE} [%]	**nMAE$_{\text{total}}$ [%]**
$\lambda_T = 0.0,\ \lambda_S = 0.0$	1.81	10.55	7.96	6.77
$\lambda_T = 0.625,\ \lambda_S = 0.625$	7.21	25.34	18.67	17.07
$\lambda_T = 1.25,\ \lambda_S = 1.25$	3.51	37.23	27.31	22.68
$\lambda_T = 2.5,\ \lambda_S = 2.5$	8.43	40.35	30.05	26.27
$\lambda_T = 5.0,\ \lambda_S = 5.0$	5.63	34.22	32.88	24.24
$\lambda_T = 7.5,\ \lambda_S = 7.5$	8.23	53.32	25.00	28.85
$\lambda_T = 25.0,\ \lambda_S = 25.0$	15.87	58.02	72.31	48.73
$\lambda_T = 50.0,\ \lambda_S = 50.0$	30.52	151.69	66.47	82.90
$\lambda_T = 100.0,\ \lambda_S = 100.0$	16.78	106.11	94.69	72.53
$\lambda_T = 200.0,\ \lambda_S = 200.0$	15.14	65.78	117.56	66.16
$\lambda_T = 400.0,\ \lambda_S = 400.0$	50.75	79.32	50.34	60.14

Finally, like in the case of SAC, a grid search is performed over the following grid:

$$\begin{aligned} \lambda_T &\in \{0.3125,\ 0.625,\ 1.25,\ 2.5,\}, \\ \lambda_S &\in \{0.3125,\ 0.625,\ 1.25,\ 2.5,\}, \\ \xi &\in \{-2,\ -1,\ 0,\ 1\}. \end{aligned} \tag{3.4}$$

The hyperparameter ξ is included for DSAC agent, as it affects the risk-behaviour of the agent when used in Wang's risk distortion operator. The results of this grid-search are seen in Figure 3.7. This figure shows only 25% of the run simulations in order to identify trends more easily. It can be seen that negative, risk-averse χ values have higher variability in their tracking accuracy given different CAPS weights. The configuration chosen is the one yielding the lowest tracking error, where $\lambda_T = 0.3125$, $\lambda_S = 1.25$, and $\xi = -1$

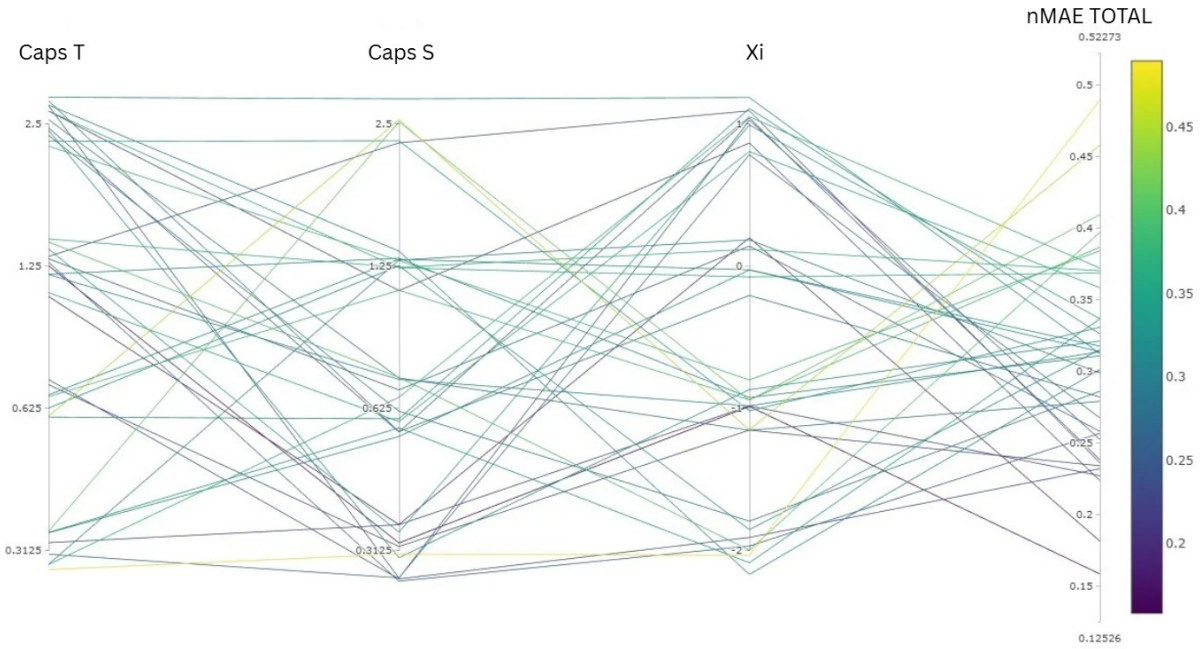


Figure 3.7: DSAC Agent CAPS Fine-tuning Parallel Axis Plot

Finally, the same procedure is repeated for the RUN-DSAC agent. This time the initial comparison shows that the best tracking accuracy is 25.32% achieved by the agent with $\lambda_T = \lambda_S = 1.25$ as seen in Table 3.3. Figure 3.8 shows that this agent does not display the highly oscillatory control signals seen in the previous Table 3.2 and Table 3.2. Additionally, the same trend as in the aforementioned figures is observed; The higher the CAPS coefficients, the lower the tracking accuracy.

Table 3.3: Effect of CAPS Coefficients on Tracking Performance for RUN-DSAC (nMAE in %)

CAPS Configuration	β_{nMAE} [%]	θ_{nMAE} [%]	ϕ_{nMAE} [%]	**nMAE$_{\text{total}}$ [%]**
$\lambda_T = 0.625,\ \lambda_S = 0.625$	4.76	58.24	17.07	26.69
$\lambda_T = 1.25,\ \lambda_S = 1.25$	7.31	37.46	31.19	25.32
$\lambda_T = 2.5,\ \lambda_S = 2.5$	10.56	31.84	43.21	28.53
$\lambda_T = 3.75,\ \lambda_S = 3.75$	10.52	59.35	28.80	32.89
$\lambda_T = 5.0,\ \lambda_S = 5.0$	5.16	49.40	38.71	31.09
$\lambda_T = 7.5,\ \lambda_S = 7.5$	16.25	148.43	41.63	68.77
$\lambda_T = 10.0,\ \lambda_S = 10.0$	18.67	44.35	102.44	55.15
$\lambda_T = 25.0,\ \lambda_S = 25.0$	11.90	49.04	141.22	67.38
$\lambda_T = 50.0,\ \lambda_S = 50.0$	9.99	86.23	32.09	42.77
$\lambda_T = 100.0,\ \lambda_S = 100.0$	24.16	59.44	64.05	49.22
$\lambda_T = 200.0,\ \lambda_S = 200.0$	22.05	78.85	118.85	73.25
$\lambda_T = 400.0,\ \lambda_S = 400.0$	16.87	149.06	349.21	171.71

$\lambda_T = \lambda_S = 1.25$ is selected as the baseline for the RUN-DSAC fine-tuning procedure. Our search grid is defined as follows:

$$\begin{aligned} \lambda_T &\in \{1.25,\ 2.5,\ 5.0\}, \\ \lambda_S &\in \{1.25,\ 2.5,\ 5.0\}, \\ \mu_{\text{init}} &\in \{-2,\ -1,\ -0.5,\ -0.25,\ 0,\ 0.5,\ 1,\ 2\}. \end{aligned} \tag{3.5}$$

The value for the weight of 0.625 is not included in the grid, as Figure 3.8 shows this agent to have a more oscillatory performance than 1.25 with worse tracking accuracy. Additionally, multiple values of μ_{init} are selected as this parameter has a strong influence on the behaviour of the agent. The results in

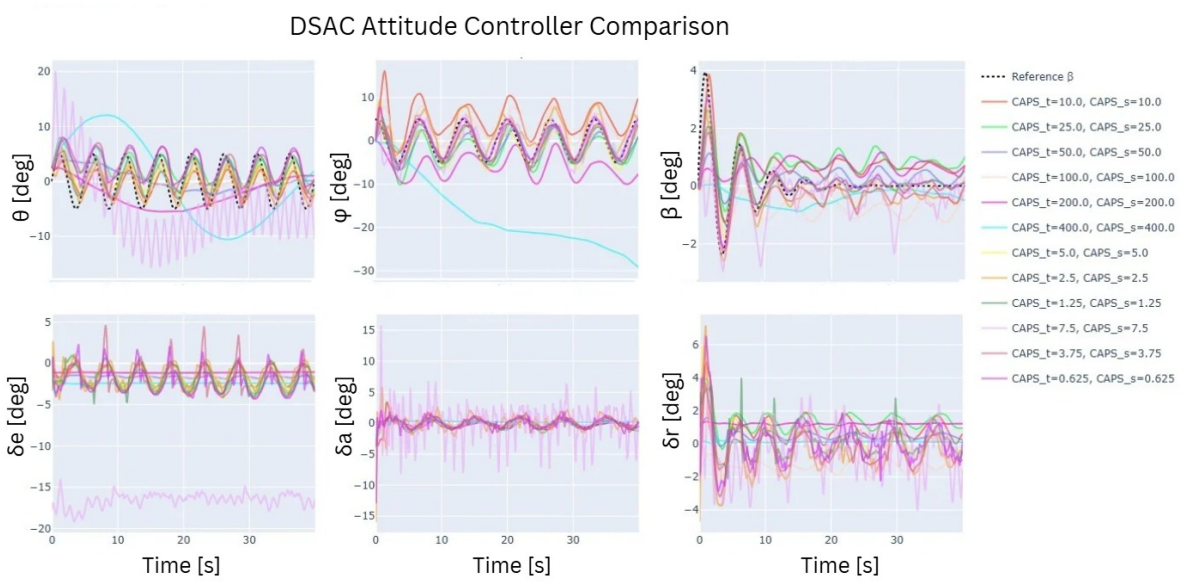


Figure 3.8: Time-Response Comparison Between Fully-Trained RUN-DSAC Agents with Varying CAPS Weights

Figure 3.9 show that positive values of μ_{init}, characterised by risk-prone behaviour, result in lower tracking errors than negative values of this parameter. The best configuration for the RUN-DSAC agent occurs when $\lambda_T = 5.0$, $\lambda_S = 1.25$, and $\mu_{\text{init}} = 1.0$, which is selected as the configuration of the RUN-DSAC offline agent.

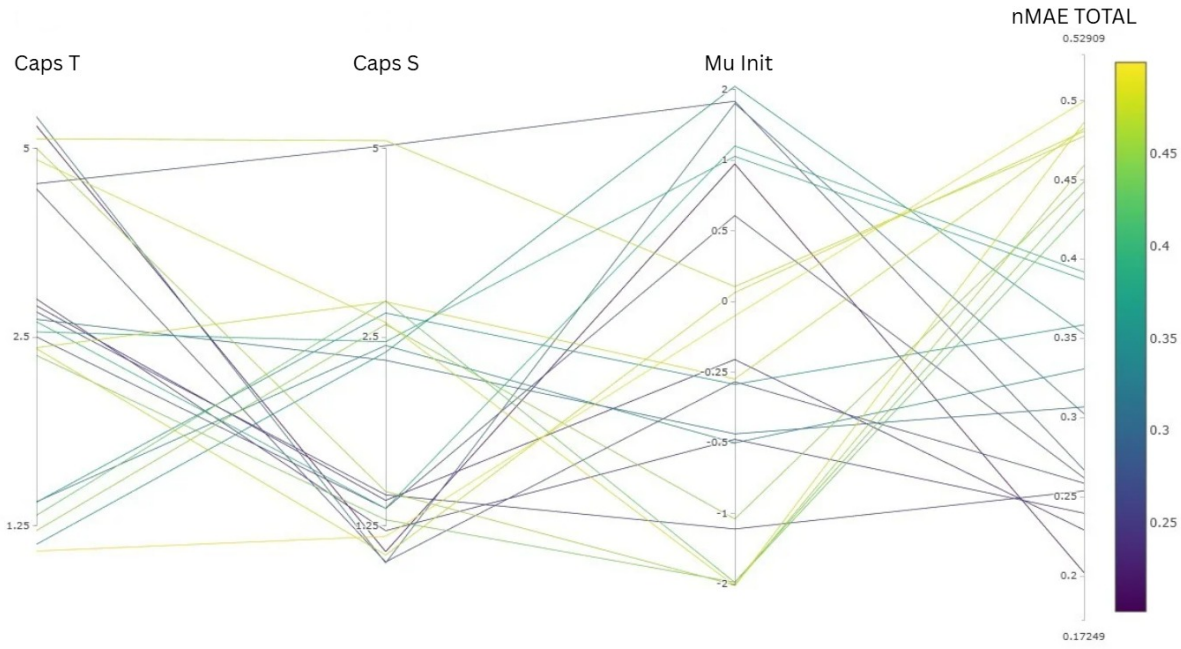


Figure 3.9: RUN-DSAC Agent CAPS Fine-tuning Parallel Axis Plot

3.5. Concluding Remarks

Based on the tuning procedures described in the previous sections, a final configuration is selected for each offline RL agent. These configurations represent a balance between tracking performance and control signal smoothness. Baseline agents for SAC, DSAC, and RUN-DSAC are fully-trained using these

configurations, further improving on the already achieved performance. The selected hyperparameters for these agents are described in Table 3.4.

Table 3.4: Final Configuration of Offline Reinforcement Learning Agents

Parameter	**SAC**	**DSAC**	**RUN-DSAC**
General Parameters			
State indices	$\{0, 1, 2\}$	$\{0, 1, 2\}$	$\{0, 1, 2\}$
State dimension	6	6	6
Discount factor γ	0.99	0.99	0.99
Learning rate	4×10^{-4}	3×10^{-4}	3×10^{-4}
Target update rate τ	0.005	0.005	0.005
Update frequency	1	1	1
Batch size	256	256	256
Replay buffer size	1×10^5	1×10^5	1×10^5
Initial entropy coefficient	1.0	1.0	1.0
Neural Network Architecture			
Actor hidden layers	[64, 64]	[64, 64]	[64, 64]
Critic hidden layers	[64, 64]	[64, 64]	[64, 64]
CAPS Regularization			
Temporal coefficient λ_T	50.0	0.3125	5.0
Spatial coefficient λ_S	50.0	1.25	1.25
CAPS noise std.	0.05	0.05	0.05
Distributional / Uncertainty Parameters			
Number of quantiles N	–	32	32
Huber loss parameter κ	–	1.0	1.0
Risk measure	–	Wang	–
Wang parameter ξ	–	-1.0	–
Initial uncertainty parameter μ_{init}	–	–	1.0
Final uncertainty parameter μ_{final}	–	–	0.0
Uncertainty decay steps	–	–	1×10^6

While most of the hyperparameters in Table 3.4 are identical across algorithms, differences arise in CAPS regularization coefficients, and the agent-specific parameters. These differences reflect the distinct learning mechanisms employed by each agent. These configurations are used through the remainder of the thesis. They are used to form the fully-trained offline agents, and as the offline baseline of the hybrid agents.

4

Online and Hybrid Agent Fine-Tuning

The performance of online and hybrid reinforcement learning controllers is also strongly affected by the hyperparameters selected, which influence stability, convergence speed, and fault-tolerant behaviour. Unlike offline agents, where training is performed over long horizons, and thus suboptimal hyperparameter configurations can still converge to high-performing agents, online and hybrid agents must operate reliably within a short time period and continuously adapt to changing conditions.

This chapter presents the tuning procedure used for the IDHP agent and the hybrid SAC-IDHP, DSAC-IDHP, and RUN-DSAC-IDHP agents. A structured tuning strategy was used to identify the stability regions, select tracking scale configurations and evaluate performance across the full range of fault scenarios. The same procedure is applied to all online and hybrid agents, ensuring fair comparison.

4.1. Hyperparameter Search for Online and Hybrid Agents

The tuning process begins with a broad exploration of the hyperparameters affecting the learning dynamics of the online and hybrid agents. These include the actor and critic learning rates, the discount factor γ, the initial covariance parameter cov_0 used by the critic, and the initial policy standard deviation.

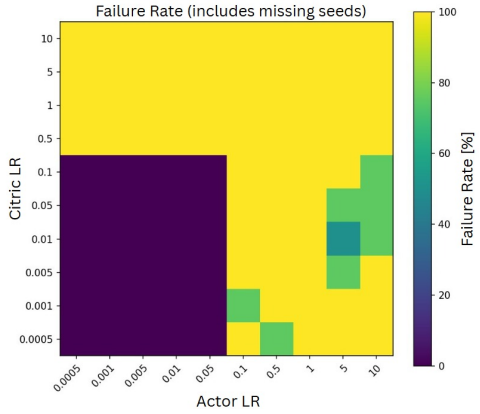
Due to the short duration of evaluation runs, an extensive search, using a large number of hyperparameter combinations, is possible. Each configuration is assessed over a 30-second interval, allowing rapid identification of unstable regions in the hyperparameter space. These configurations are discarded from further consideration.

4.1.1. Learning Rate Tuning and Stability Regions

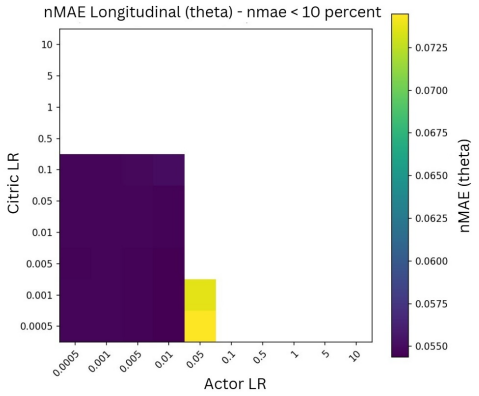
Among the considered hyperparameters, the actor and critic learning rates have the most influence on learning stability and closed-loop behaviour. If these are too high, the control responses are oscillatory or unstable, while if these are too small, the agent adapts slowly and is thus unable to adapt to the environment dynamics and mitigate faults.

To identify the suitable learning regions, a two-dimensional grid for actor and critic learning rates is defined. Then, simulations are run with each combination from the defined grid for a certain number of seeds for statistical relevance. The resulting performance landscape is summarized using a heatmap representation of the failure rate and the normalized mean absolute error as seen in Figure 4.1.

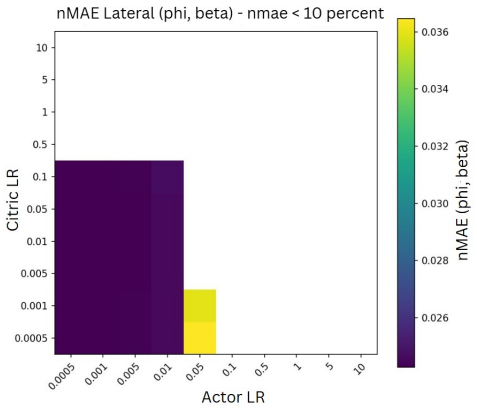
In Figure 4.1a, Figure 4.1a highlights the failure rate of each combination of actor and critic learning rates. When the actor learning rate exceeds 0.05 or the critic learning rate exceeds 0.1, the failure rate increases drastically. When examining the longitudinal tracking error, the best performance is observed with the critic learning rate below 0.1 and the actor learning rate below 0.05. Similarly, for the lateral case, the best performance is achieved when the actor learning rate is below 0.01, and the critic LR is



(a) DSAC-IDHP Actor and Critic Learning Rate Failure Heatmap



(b) DSAC-IDHP Actor and Critic Learning Rate Longitudinal nMAE Heatmap



(c) DSAC-IDHP Actor and Critic Learning Rate Lateral nMAE Heatmap

Figure 4.1: SAC CAPS Fine-tuning Parallel Axis Plot and Agent Response

under 0.1. This analysis allows one to identify the region of optimal performance for the critic and actor learning rates.

4.1.2. Secondary Hyperparameter Tuning

Having constrained the learning rates to stable regions, the remaining hyperparameters are assessed to determine their impact on performance. To do this, a grid with possible values for the model forgetting rate γ_{model}, the target network update rate τ, and the initial policy standard deviation is made. Simulations are run for each combination in this hyperparameter space.

The results from these simulations are shown in Figure 4.2, which shows that the hyperparameters have little to no influence on the tracking performance. In this example plot of the SAC-IDHP algorithm, one can observe a difference of $5e^{-5}$ between the highest and lowest tracking accuracies. This difference is insignificant, and although the figure shows that selecting a std_{init} of 0.5 typically results in lower tracking errors, the tracking performance across all combinations remains comparable. This observation indicates a clear hierarchy in the effect of hyperparameters; once the learning rates have been appropriately tuned, variations in other parameters have a minor influence on closed-loop tracking performance.

4.2. Tracking Scale Tuning

After stabilizing the learning dynamics, further performance improvements are found when varying the scaling of the tracking errors in the reward function. These tracking scales dictate the relative importance

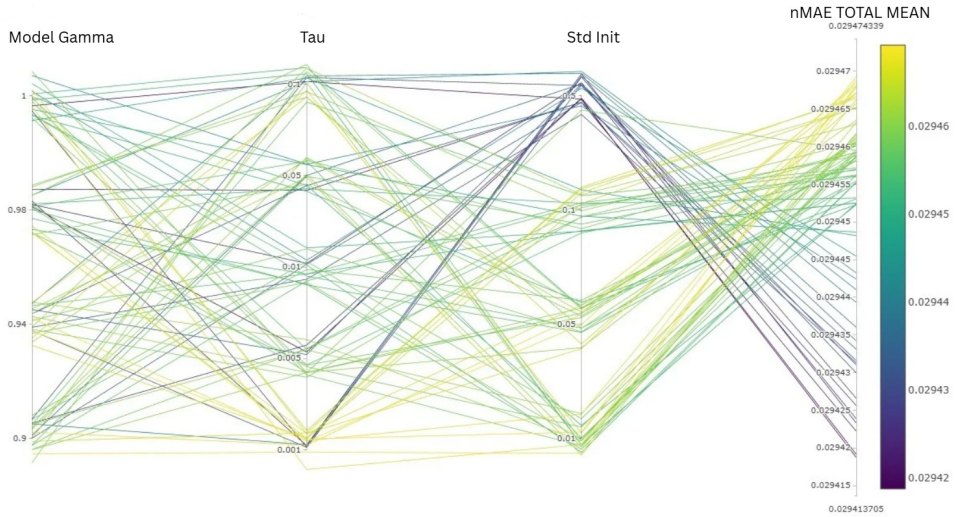


Figure 4.2: SAC-IDHP Parallel Axis Plot for γ_{model}, τ and std_{init}

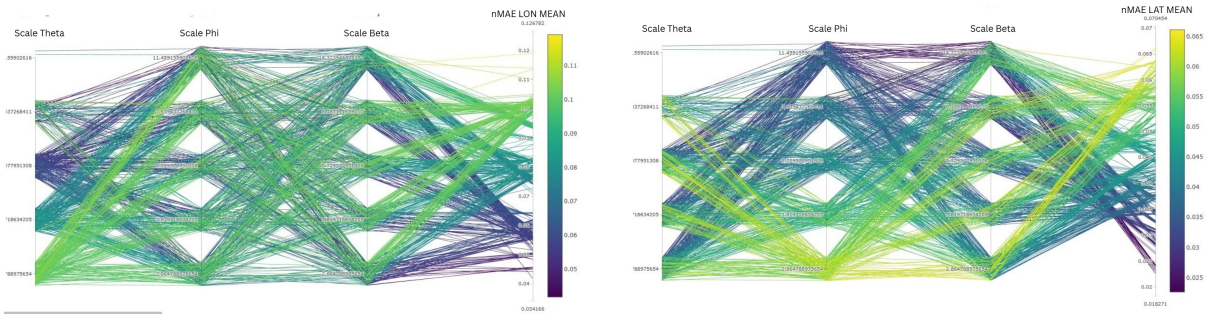
of the tracked variables, and thus shape the learned control strategy.

For example, increasing the tracking scale associated with the pitch angle θ increases the penalty for pitch-tracking errors, prompting more aggressive elevator activity to mitigate these errors and improve pitch accuracy. However, this prioritization can come at the expense of other states or instability when control authority for that actuator is reduced. Conversely, lower tracking scales reduce control aggressiveness, which can lead to poor tracking and degraded fault-mitigation capabilities.

All offline agents are initially trained using a baseline set of tracking scales derived from work by Teirlinck [49] and Vieira [51]. To improve the performance, a series of tuning experiments is realized by deviating from these.

4.2.1. Tracking Scale Tuning

To assess the influence of the tracking scales and select the best-performing combination for the hybrid and online algorithms, a similar experiment to the one described in subsection 4.1.2 is conducted. A simulation is run with each combination of tracking scales for a set number of seeds, and the results are displayed in a parallel-axis plot.



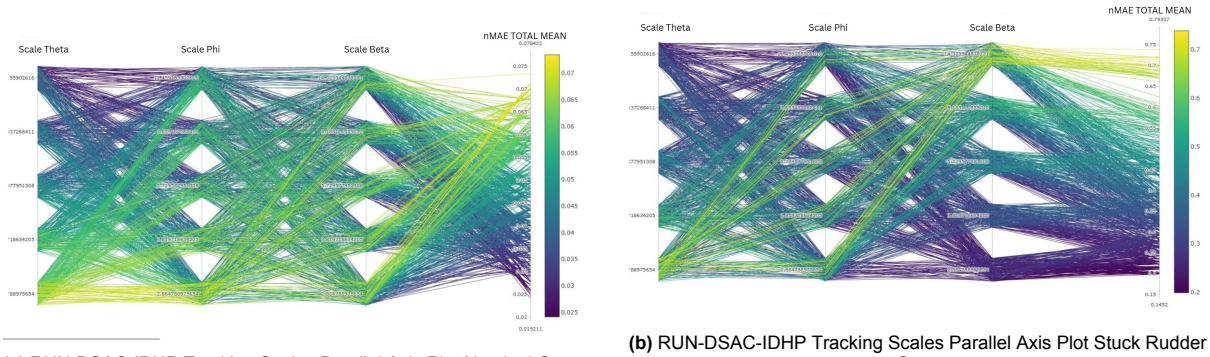
(a) DSAC-IDHP Tracking Scale Parallel Axis Plot Longitudinal nMAE

(b) DSAC-IDHP Tracking Scale Parallel Axis Plot Lateral nMAE

Figure 4.3: DSAC-IDHP Tracking Scales Parallel Axis Plots

Figure 4.3 shows the results of tuning the DSAC-IDHP agent tracking scales. In contrast to the earlier hyperparameters, clear patterns can be observed in this parameter space. In particular, the longitudinal plot in Figure 4.3a shows that increasing the θ tracking scale improves performance. Similarly, Increasing

ϕ and β improves lateral tracking performance in Figure 4.3b.



(a) RUN-DSAC-IDHP Tracking Scales Parallel Axis Plot Nominal Case

(b) RUN-DSAC-IDHP Tracking Scales Parallel Axis Plot Stuck Rudder Case

Figure 4.4: RUN-DSAC-IDHP Tracking Scales Parallel Axis Plot Nominal and Stuck Rudder Cases

Likewise, Figure 4.4a shows that increasing the tracking scales across all tracked variables results in the best tracking performance for the RUN-DSAC-IDHP agent under nominal conditions. However, under fault conditions, the optimal tracking scales are different, maximising β tends to result in worse performance. This can be observed in Figure 4.4b for the Stuck Rudder fault scenario. Maximizing tracking performance under nominal conditions often results in more aggressive control behaviour, which can hinder the controller's ability to mitigate actuator degradation or structural faults. This highlights the inherent trade-off between tracking performance under nominal flight conditions and fault-tolerance in hybrid reinforcement learning controllers.

4.2.2. Tracking Scale Selection Across Fault Scenarios

The analysis presented in subsection 4.2.1 shows that tracking scale configurations yielding optimal performance under nominal conditions do not necessarily generalize to the considered fault scenarios. Aggressive tracking scales can lead to increased tracking errors or loss of stability when faced with these conditions. Thus, selecting the tracking scales based on the nominal performance is insufficient for fault-tolerant control.

To address this limitation, a new procedure is adopted. A set of tracking scale configurations, identified from the nominal tracking scale tuning, is evaluated across all fault scenarios. This set of configurations is defined taking into account the nominal scenario optimal tracking scales, and by progressively loosening the tracked variables to form new configurations. For each configuration short evaluation runs are performed across all fault scenarios using multiple random seeds. Rather than optimizing the scales, this experiment aims to identify the configuration that exhibits improved performance across all fault conditions when compared to the offline agent.

Table 4.1: Multi-fault tracking performance of IDHP–RUN–DSAC configurations (nMAE in %)

Configuration	Success	nMAE_θ [%]	nMAE_ϕ [%]	nMAE_β [%]	nMAE_{tot} [%]
IDHP–RUN–DSAC A ($\theta = 1/5,\ \phi = 1/5,\ \beta = 1/5$)	10	8.93	3.43	8.20	6.85
IDHP–RUN–DSAC C ($\theta = 1/5,\ \phi = 1/5,\ \beta = 1/15$)	10	7.32	3.39	10.45	7.05
IDHP–RUN–DSAC D ($\theta = 1/5,\ \phi = 1/5,\ \beta = 1/20$)	10	6.95	3.09	11.17	7.07
IDHP–RUN–DSAC G ($\theta = 1/10,\ \phi = 1/5,\ \beta = 1/5$)	10	13.63	3.03	8.10	8.26
IDHP–RUN–DSAC J ($\theta = 1/10,\ \phi = 1/10,\ \beta = 1/20$)	10	10.07	5.03	10.84	8.65
Baseline (RUN-DSAC)	10	14.33	6.21	7.63	9.39
IDHP–RUN–DSAC B ($\theta = 1/5,\ \phi = 1/5,\ \beta = 1/10$)	11	28.00	40.07	10.65	26.24
IDHP–RUN–DSAC E ($\theta = 1/5,\ \phi = 1/10,\ \beta = 1/5$)	11	26.78	59.79	9.02	31.86
IDHP–RUN–DSAC I ($\theta = 1/10,\ \phi = 1/10,\ \beta = 1/5$)	11	31.14	112.34	14.10	52.53
IDHP–RUN–DSAC H ($\theta = 1/10,\ \phi = 1/5,\ \beta = 1/20$)	11	37.76	127.99	24.69	63.48
IDHP–RUN–DSAC F ($\theta = 1/5,\ \phi = 1/10,\ \beta = 1/20$)	11	29.35	336.65	17.59	127.86

Note: Tracking scale values are reported in inverse degrees for readability. In the implementation, angular tracking errors are expressed in radians and scaled as $k = 1/(\alpha \cdot \pi/180)$.

Table 4.1 summarizes the results of this experiment for the RUN-DSAC-IDHP agent. The selected configuration is configuration B. This seems counterintuitive, since when looking purely at the average, this configuration performs significantly worse than many others. However, this is due to the fact that this agent is able to maintain control in the Inverted Elevator fault scenario with high tracking errors. When each fault is evaluated independently, this agent surpasses the baseline RUN-DSAC tracking performance consistently, yet its average is skewed, since all agents above it are unable to tracking the reference signal in this fault-scenario, thus this error is not included in the average calculation.

4.3. Concluding Remarks

The tuning procedure presented in this chapter shows that the performance of online and hybrid reinforcement learning agents is governed by different hyperparameters. Algorithmic hyperparameters, particularly the learning rates, determine the learning stability and convergence reasons. Once these have been chosen within stable operating regions, their influence becomes secondary when attempting to improve tracking performance.

The relative scaling of the tracking errors for the tracked variables is found to dominate the controller behaviour. Tracking scales directly dictate the prioritisation of the task, and thus exert influence on both the nominal tracking performance and fault-mitigation capability. This observation motivated a tuning strategy in which tracking scale combinations are selected based on their performance across nominal and fault scenarios, ensuring the agent can mitigate faults at the expense of a slight reduction in nominal performance.

Using this approach, different tracking scale configurations are identified for each of the tuned controllers. These configurations are evaluated against all faults, and those that consistently outperform the offline agent performance and reduce tracking error across all fault conditions are selected. This ensures the final agent configuration is able to perform at a sufficient level under nominal conditions, but also mitigates unexpected fault conditions better than the offline agents. The final configurations are defined in Table 4.2 and these are used throughout the remainder of the thesis. d

Table 4.2: Final Online and Hybrid Controller Configuration

Parameter	**SAC-IDHP**	**DSAC-IDHP**	**RUN-DSAC-IDHP**
Common Online Learning Parameters			
Discount factor γ		0.6	
Target update rate τ		0.01	
Initial policy standard deviation		0.05	
Actor learning rate		0.05	
Critic learning rate		0.10	
Initial critic covariance cov_0		10^6	
Model forgetting factor γ_{model}		0.98	
Network activation		tanh	
Tracking Scale Configuration (degrees)			
Pitch θ	$1/7.5^\circ$	$1/7.5^\circ$	$1/5.0^\circ$
Roll ϕ	$1/7.5^\circ$	$1/5.0^\circ$	$1/5.0^\circ$
Sideslip β	$1/5.0^\circ$	$1/20.0^\circ$	$1/10.0^\circ$

5

Extended Fault Scenario Results

In addition to the fault scenarios discussed in the Part I, several other actuator faults and structural degradation cases are evaluated to analyse the behaviour of the hybrid controllers further. While Part I illustrates the behaviour and capabilities of the proposed RUN-DSAC-IDHP agent, through the use of the most significant examples, this section aims to complete the Results section by providing insight into the remaining fault types. The considered faults include:

- Center-of-gravity shift
- Aileron deflection limited to $\pm 5°$
- Aileron effectiveness reduced to 10%
- Elevator effectiveness reduced to 30%
- Elevator effectiveness reduced to 10%

5.1. Faults With Strong Mitigation

5.1.1. Center-of-Gravity Shift

This failure mode occurs when the center-of-gravity is shifted backwards by 0.25 m. The offline agents SAC and RUN-DSAC are able to maintain stable tracking behaviour when this fault occurs after 10 s, with only minor amplitude deviations from the reference signal. However, when combined with the IDHP agent, the hybrid controllers significantly reduce the minor tracking errors, as seen in Figure 5.1.

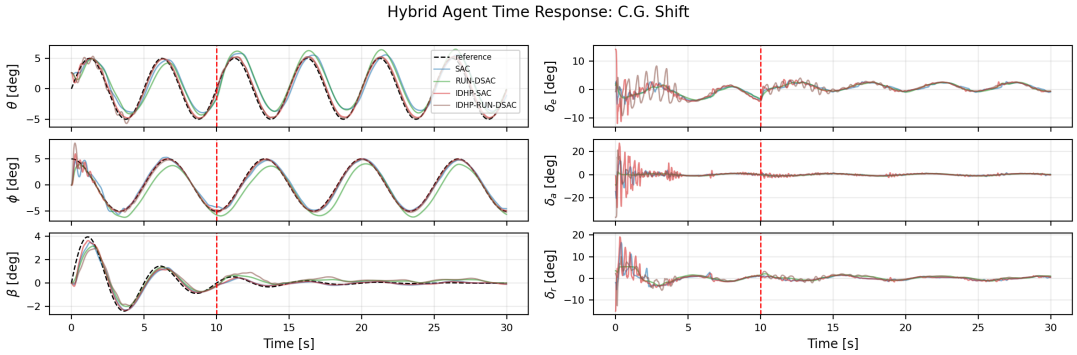


Figure 5.1: SAC, RUN-DSAC, SAC-IDHP, and RUN-DSAC-IDHP under c.g. Shift Fault Condition

In particular, the SAC-IDHP configuration achieves the highest tracking accuracy while introducing slight control oscillations at the beginning of the response. RUN-DSAC-IDHP performs similarly, achieving lower accuracy in the β reference signal, but displaying a smoother control that oscillates with a lower

frequency and magnitude.

This scenario demonstrates that hybrid reinforcement learning is particularly useful when full control authority remains available, and the fault is simply a parametric deviation. The IDHP component can effectively compensate for the shifted equilibrium by correcting any steady-state bias, thereby fully mitigating the fault.

5.1.2. Aileron Deflection Limit

The next fault considered is limiting the aileron deflection to $\pm 5°$. However, further inspection of the aileron control signal reveals that the commanded deflections always remain within these bounds. Thus, the response of the agents to this fault is identical to the nominal response examined in Part I.

Among the evaluated controllers, SAC-IDHP exhibits the best tracking, followed closely by RUN-DSAC-IDHP, and all hybrid agents improve upon and compensate for the small deviations in magnitude and phase that are observed with the purely offline agents. The response for the SAC and SAC-IDHP agents can be observed in Figure 5.2.

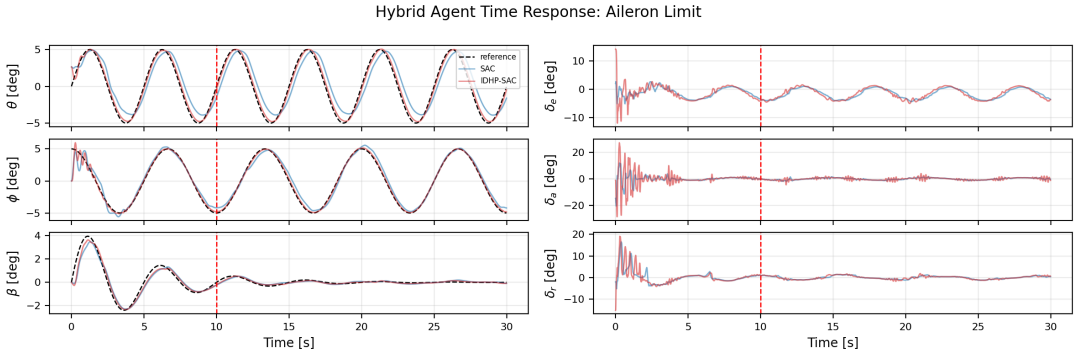


Figure 5.2: SAC and SAC-IDHP under Limited Aileron Fault

5.1.3. Aileron Effectiveness Reduction

Under the ”da_reduce” fault, the aileron effectiveness is reduced by 90%. This is a severe loss of control authority, yet it is handled reasonably well by the offline controllers and, subsequently, their hybrid counterparts. Figure 5.3 shows that SAC's ϕ tracking is only slightly affected by the fault, but quickly returns to tracking the signal, while Figure 5.4 shows that RUN-DSAC displays a higher tracking error. Yet both hybrid versions successfully mitigate the fault.

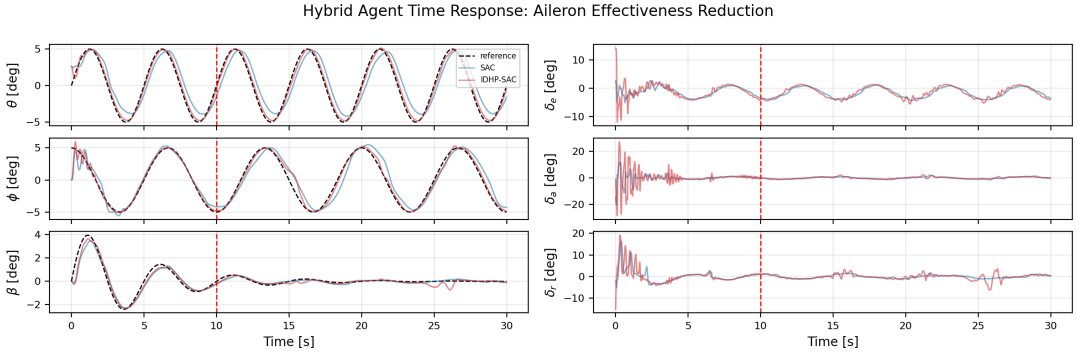


Figure 5.3: SAC and SAC-IDHP under Reduced Aileron Effectiveness Fault

The hybrid control effort, as seen in Part I, continues displaying the same behaviour; it is highly oscillatory in the first seconds of the response, and then those oscillations dampen out. SAC-IDHP's control signals

are characterized by high-magnitude, high-frequency oscillations at the beginning of the response, while RUN-DSAC-IDHP shows lower-frequency, lower-magnitude oscillations that persist throughout the entire simulation.

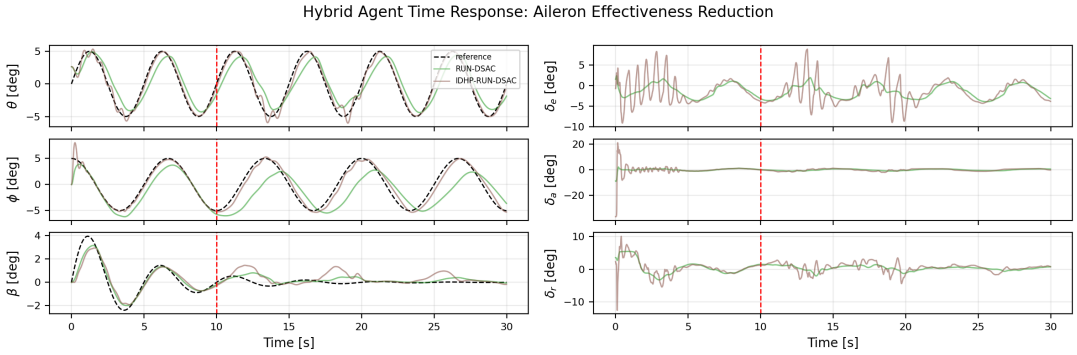


Figure 5.4: RUN-DSAC and RUN-DSAC-IDHP under Reduced Aileron Effectiveness Fault

5.2. Faults With Partial Mitigation

5.2.1. Moderate Elevator Effectiveness Reduction

The agents were subjected to two fault scenarios testing reductions in elevator effectiveness. The first reduced this value to 30% of its nominal value. Following the fault's occurrence, all offline agents show a noticeable reduction in pitch tracking, and SAC even experiences degradation in roll tracking.

The hybrid controllers, like the RUN-DSAC-IDHP shown in Figure 5.5, only show partial mitigation of the fault at hand. SAC-IDHP exhibits the most oscillatory behaviour, especially when correcting roll-angle errors. RUN-DSAC-IDHP, on the other hand, maintains a more stable tracking behaviour, and although it performs slightly worse when tracking θ than the former, it tracks all three reference signals better overall, with smoother control signals.

The hybrid agents in this case can correct tracking errors if the pitch angle has sufficient control authority over the elevator. Yet recovery to the nominal amplitude is limited by the actuator's available authority.

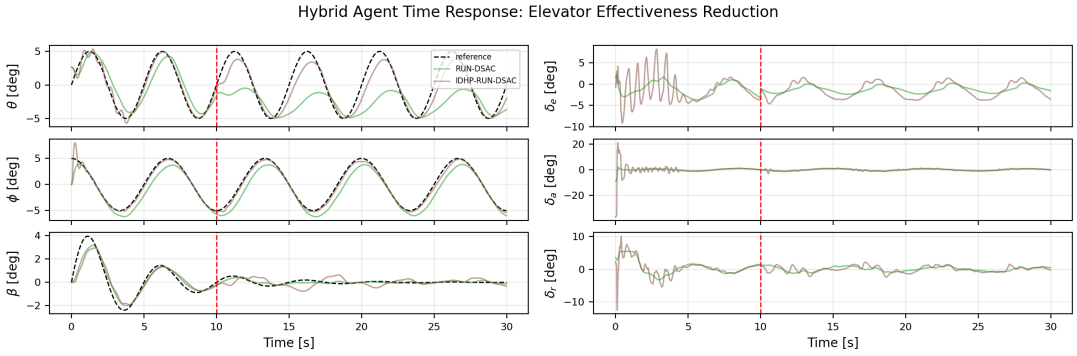


Figure 5.5: RUN-DSAC and RUN-DSAC-IDHP under Reduced Elevator Effectiveness Fault

5.2.2. Extreme Elevator Effectiveness Reduction

When the elevator's effectiveness is reduced by 90% performance across all agents drastically reduces. The offline agents follow the general shape of the pitch and sideslip signals; however, due to reduced elevator control authority, they fail to track the pitch signal, as seen in Figure 5.6.

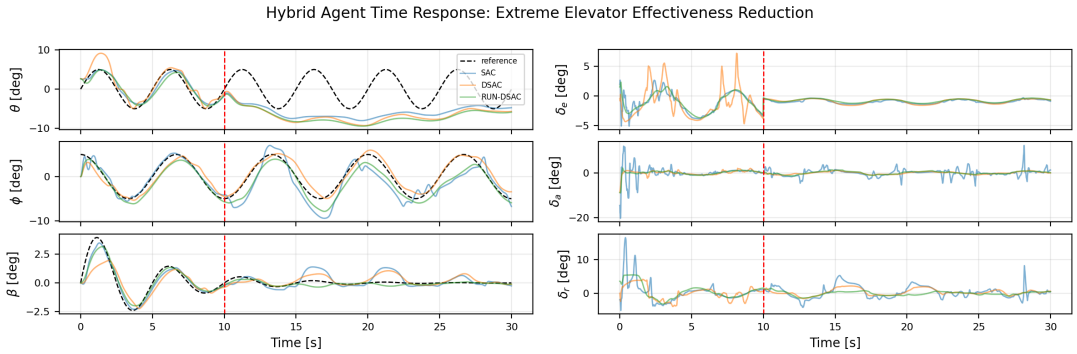


Figure 5.6: Offline Agents Under Extreme Reduced Elevator Effectiveness Fault

The hybrid agents, except for RUN-DSAC-IDHP, show lower tracking accuracy than their offline counterparts. SAC-IDHP introduces higher oscillations in the pitch, roll, and sideslip signals, deviating further from the reference signals in Figure 5.8. Additionally, its control signals are characterised by highly oscillatory behaviour of high magnitude, with the exception of δ_e whose magnitude is bounded by the fault being evaluated. DSAC-IDHP performs the worst of all agents, quickly diverging after the fault is introduced. The agent's control response reaches actuator limits as it tries to correct the instability caused by reduced elevator effectiveness.

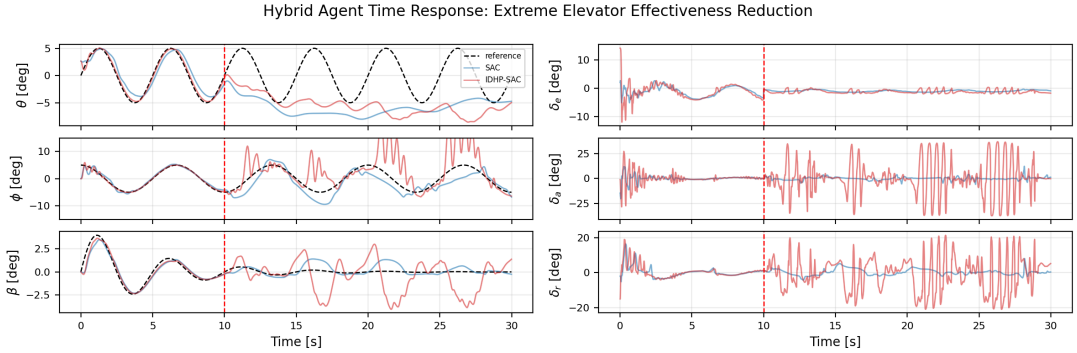


Figure 5.7: RUN-DSAC and RUN-DSAC-IDHP under Extreme Reduced Elevator Effectiveness Fault

Finally, Figure 5.7 shows that the RUN-DSAC-IDHP agent achieves the best tracking performance among the implemented agents. It attempts to correct pitch-angle tracking errors and partially follows the reference signal. Similar to the case of the limited elevator in Part I, the control signal for the elevator appears to reach its limit introduced by the fault, thus ensuring that only partial tracking is possible. For the remaining roll and sideslip channels, the agent introduces slightly more oscillation and degrades tracking performance, but remains within acceptable bounds.

5.3. Concluding Remarks

The additional fault scenarios presented in this chapter provide further insight into the behaviour of hybrid RL agents under various levels of actuator faults and structural changes. The results show a clear dependence of hybrid performance on both the severity of the fault and the quality of the algorithm's offline initialization.

For deviations in the dynamics of the aircraft, such as the shifted center-of-gravity fault or the horizontal tail reduction in Part I, hybrid reinforcement algorithms prove to be effective. In these situations, full actuator authority remains available, and the IDHP component corrects the tracking errors in the offline response due to the fault. SAC-IDHP is the hybrid controller that performs best among the three, and these results confirm that hybrid adaptation is well-suited to model mismatch, since the IDHP controller's

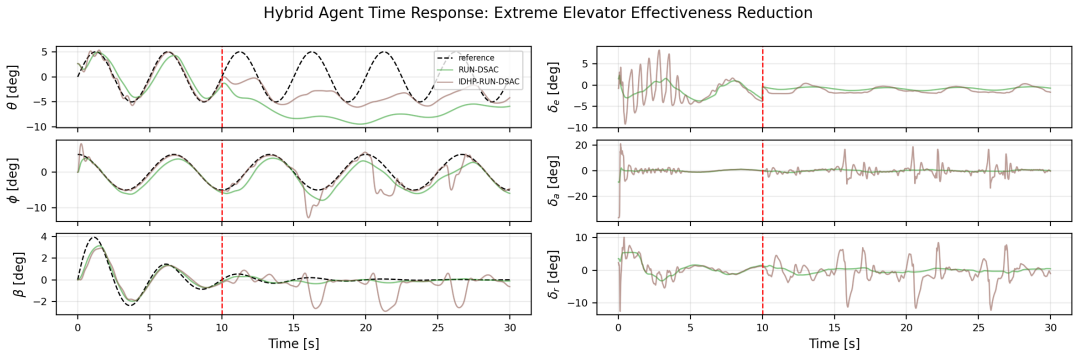


Figure 5.8: SAC and SAC-IDHP under Extreme Reduced Elevator Effectiveness Fault

incremental model can update its model of the system dynamics after the fault occurs.

In the fault scenarios where an actuator is bounded to a certain operational range, the agents' tracking performance is dependent on this operational range and the control effort required to follow the reference signal. In the case of the aileron deflection limit of $\pm 5^\circ$, the aileron control signal oscillates between $\pm 2.5^\circ$ approximately in order to track the reference signal. In this situation, the observed behaviour matches the nominal case, meaning the learned policy operates within the tested control margins. If the aileron was bounded to $\pm 1^\circ$, for example, then the agents would fail to track the roll angle reference. Additionally, the control effort for the aileron would reach its maximum and minimum value, as it oscillates, similar to the case of the elevator limit in Part I.

For moderate actuator degradations, such as reduced aileron and elevator effectiveness in subsection 5.1.3 and subsection 5.2.1, hybrid controllers continue to outperform stand-alone offline controllers. While the offline controllers exhibit larger deviations from the reference signal after the fault, the hybrid agents retain sufficient control authority to correct tracking errors and improve tracking performance. SAC-IDHP displays the highest tracking accuracy, due to both its better offline initialization and increased corrective behaviour. SAC-IDHP's control signals are characterized by higher-frequency oscillations than those of RUN-DSAC-IDHP's.

When the observed actuator degradation is extreme, such as a 90% reduction in elevator effectiveness, aggressive online adaptation can destabilise the closed-loop system. All offline agents exhibit large deviations from the pitch reference signal. When combined with IDHP to form a hybrid agent, the online component aims to compensate for large deviations in θ and smaller ones in the other references. In the cases of SAC-IDHP and DSAC-IDHP, this compensation introduces large high-frequency oscillations across all channels and, in the latter case, causes the system to diverge. On the other hand, RUN-DSAC-IDHP successfully reduces the tracking error in the pitch while bounded by the actuator constraints. This agent's uncertainty-prone policy displays increased elevator authority compared to the others, allowing RUN-DSAC-IDHP to partially mitigate this fault, while maintaining the frequency and magnitude of the control signal oscillations comparatively low.

These results, taken together, lead to the conclusion that hybrid reinforcement learning does not fully mitigate fault across all fault scenarios. Its effectiveness depends on the fault severity, the remaining actuator authority, and the offline policy used for initialization. SAC-IDHP performs best across all mild degradations, while RUN-DSAC-IDHP shows higher fault tolerance as it generalizes across more fault types, performing only slightly worse than the former. This analysis complements the results presented in Part I and further demonstrates that combining offline reinforcement learning with incremental online adaptation increases fault tolerance and attitude-tracking accuracy, while being limited by the available actuator authority and conditioned by the offline policy initialization.

6

Verification and Validation

The development of the different reinforcement learning-based flight controllers must go through a verification and validation (V&V) process. In this work, verification ensures both the aircraft simulation environment and the reinforcement learning agents are numerically correct and behave as they should. Validation, on the other hand, assesses whether the simulation environment successfully represents a Cessna 550 Citation II aircraft and responds in the same manner as the aircraft in literature. In this chapter, subsection 6.1.1 outlines the verification of the simulation environment, subsection 6.1.3 verifies the correctness of the algorithms through the use of a simplified problem, and finally **??** validates the simulation environment against real aircraft data.

6.1. Verification

Verification in this work is performed at two levels: the simulation environment and the reinforcement learning agents developed.

6.1.1. Simulation Model Verification

Model Origin and Implementation

The Cessna 550 Citation II model used in this work is based on the CitAST (Citation Analysis and Simulation Toolkit) model developed at TU Delft. The CitAST tool provides trimming, linearization, and simulation capabilities for the aircraft and has been used for controller design and analysis.

In this work, the CitAST model was compiled and translated into Python, since implementing RL agents is simpler in this language. The agents interact exclusively with the Python interface of the CitAST model; it is necessary to verify that this implementation reproduces the Matlab model with high fidelity.

Verification Methodology

To verify the Python implementation of the Citation model against the original Matlab/Simulink CitAST model, an identical time-domain simulation is run in parallel in both environments, trimmed at the same flight condition, and with identical simulation settings and control input signals. This procedure follows the verification approach used in previous TU Delft work [51].

The model was trimmed at an altitude of 2000 m and an airspeed of 90 ms^{-1}, and simulated with a sample time of 0.01 s for 60 s. After trimming, the control signal defined in Equation 6.1 was set as the simulation input, and the experiment was run.

$$\delta_e(t) = \begin{cases} 2^\circ & 1 \leq t < 3 \\ -2^\circ & 3 \leq t < 5 \\ 0 & \text{otherwise} \end{cases} \qquad \delta_a(t) = \begin{cases} 5^\circ & 6 \leq t < 7.5 \\ -5^\circ & 7.5 \leq t < 9 \\ 0 & \text{otherwise} \end{cases} \qquad \delta_r(t) = \begin{cases} 3^\circ & 10 \leq t < 11 \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

This sequence was selected to excite both the longitudinal and lateral dynamics, creating a full 12-state vector, the progression of which is used for verification.

6.1.2. Time Response Comparison

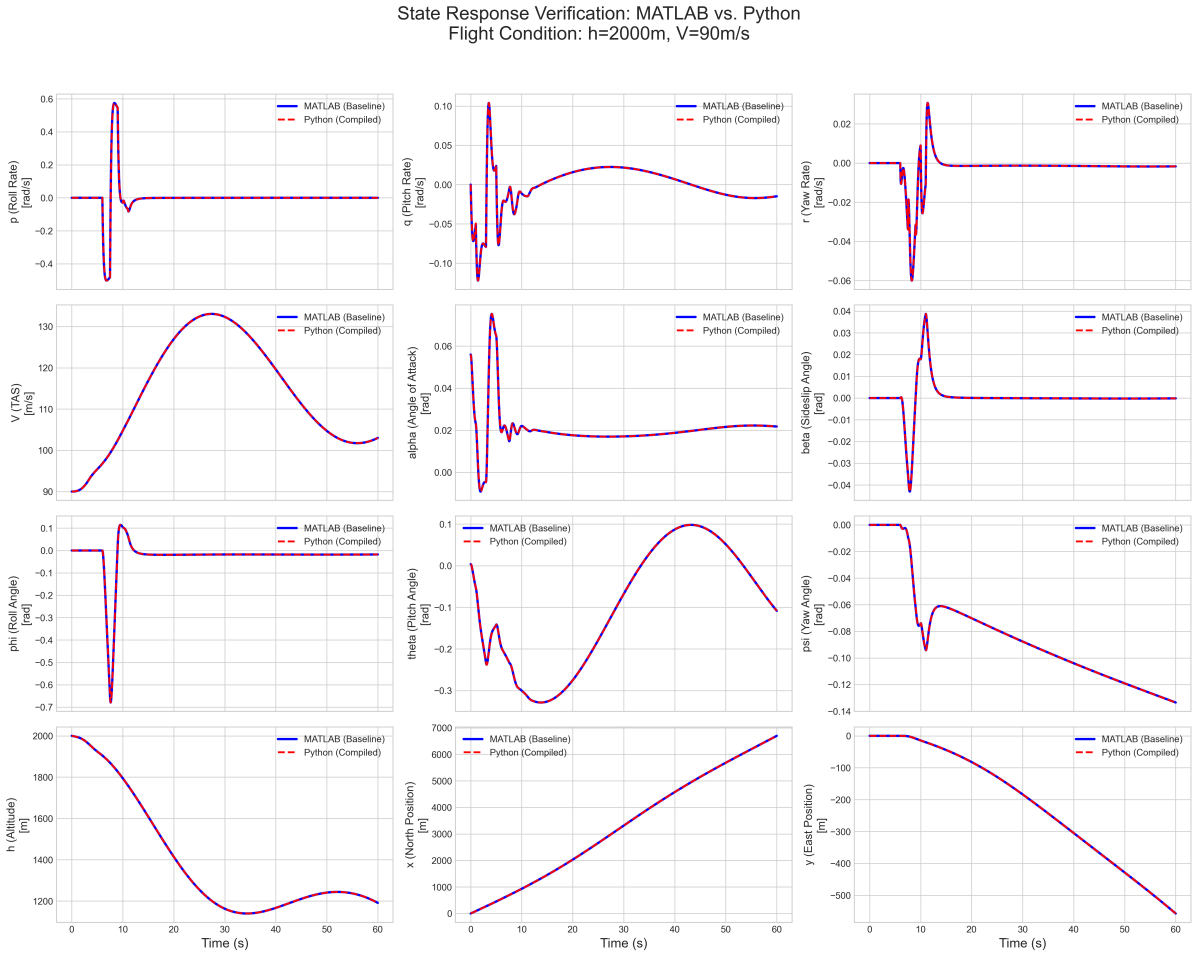


Figure 6.1: Python Model 12-state Time Response Verification

The time responses of all 12 states are compared for both the compiled Python model and the CitAST model. The results in Figure 6.1 show that both models' time responses are identical when subjected to the same input, confirming that the simulation model the agents interact with matches the high-fidelity simulation environment for the Cessna 550 Citation II in the CitAST framework. Minor differences in the state values may arise from numerical differences, rounding errors, and discretization, yet the near-perfect overlap confirms the correct implementation of the MATLAB model in Python.

6.1.3. Algorithm Verification Using Benchmark Problem

While the verification of the environment ensures the simulation is physically consistent, it does not guarantee that the RL algorithms were implemented correctly. Given the complexity of the algorithms, an independent verification stage is required to ensure these functions as intended.

Cart-Ball Problem Formulation

To verify the correctness of the algorithms, a simplified nonlinear benchmark problem, ”Cartball,” was introduced. This is a simple two-degree-of-freedom system consisting of a cart and a ball connected through a linear spring-damper pair. This problem, despite being simple, exhibits coupled dynamics and requires continuous control inputs, making it sufficiently complex to verify the actor-critic updates, offline learning, and online adaptation behaviour.

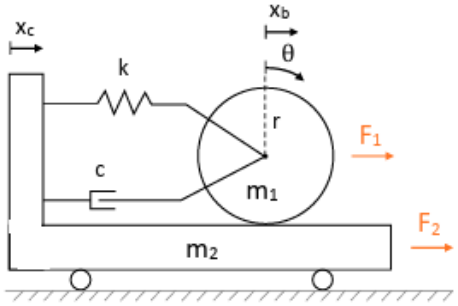


Figure 6.2: Cartball Problem from Dally[5]

The state vector in this problem is defined in Equation 6.2, where x_b and x_c represent the ball and cart horizontal positions respectively, and $\dot{x}_b$ and $\dot{x}_c$ the respective velocities.

$$\mathbf{x} = \begin{bmatrix} x_b & x_c & \dot{x}_b & \dot{x}_c \end{bmatrix}^T \tag{6.2}$$

$$\mathbf{u} = \begin{bmatrix} F_1 & F_2 \end{bmatrix}^T \tag{6.3}$$

Additionally, F_1 and F_2 in Equation 6.3 represent the two forces applied independently to the ball and cart.

The system dynamics are linear but coupled through the stiffness and damping coefficients, k and c respectively, as shown by the state space in Equation 6.4. Forward Euler integration with a timestep of 0.01 s is used to solve the ordinary differential equation. The outputs of the equation are the cart position x_c and the ball's angular position θ defined in Equation 6.5.

$$\begin{bmatrix} \dot{x}_b \\ \dot{x}_c \\ \ddot{x}_b \\ \ddot{x}_c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -k/m_1 & k/m_1 & -c/m_1 & c/m_1 \\ k/m_2 & -k/m_2 & c/m_2 & -c/m_2 \end{bmatrix} \begin{bmatrix} x_b \\ x_c \\ \dot{x}_b \\ \dot{x}_c \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m_1 & 0 \\ 0 & 1/m_2 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{6.4}$$

$$\begin{bmatrix} x_c \\ \theta \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1/r & -1/r & 0 & 0 \end{bmatrix} \begin{bmatrix} x_b \\ x_c \\ \dot{x}_b \\ \dot{x}_c \end{bmatrix} \tag{6.5}$$

Control Task and Reference Signal

The objective of this problem is to track reference trajectories for both x_c and θ, replicating the reference signal tracking tasks used for the Citation problem. During offline training, a series of randomized cosine steps form the reference signal. The offline agents are then evaluated with a similar random signal or a

sine trajectory. When trained online, the agent's reference signal is a sine curve.

The reward function follows Dally's formulation:

$$r_t = -(|e_{x_c}| + |e_\theta|) \tag{6.6}$$

Which can be clipped to ensure numerical stability. Tracking performance, like in Part I, is measured in terms of the normalized mean absolute error (nMAE) averaged between both tracked states.

Agents Verified

The following reinforcement learning agents were adapted to and verified with the cartball problem:

- Soft Actor-Critic (SAC)
- Distributional SAC (DSAC)
- Returns Uncertainty Navigated DSAC (RUN-DSAC)
- Incremental Dual Heuristic Programming (IDHP)
- Hybrid SAC-IDHP
- Hybrid DSAC-IDHP
- Hybrid RUN-DSAC-IDHP

The offline agents are trained with similar hyperparameters to those agents trained for the Citation problem. Meanwhile, the IDHP and hybrid variants learn online and are therefore more sensitive to hyperparameter values.

Results

All three offline agents learned to successfully track the reference trajectories in the cart displacement and ball angular rotation. Additionally, the control signals for all offline agents remain bounded and physically plausible.

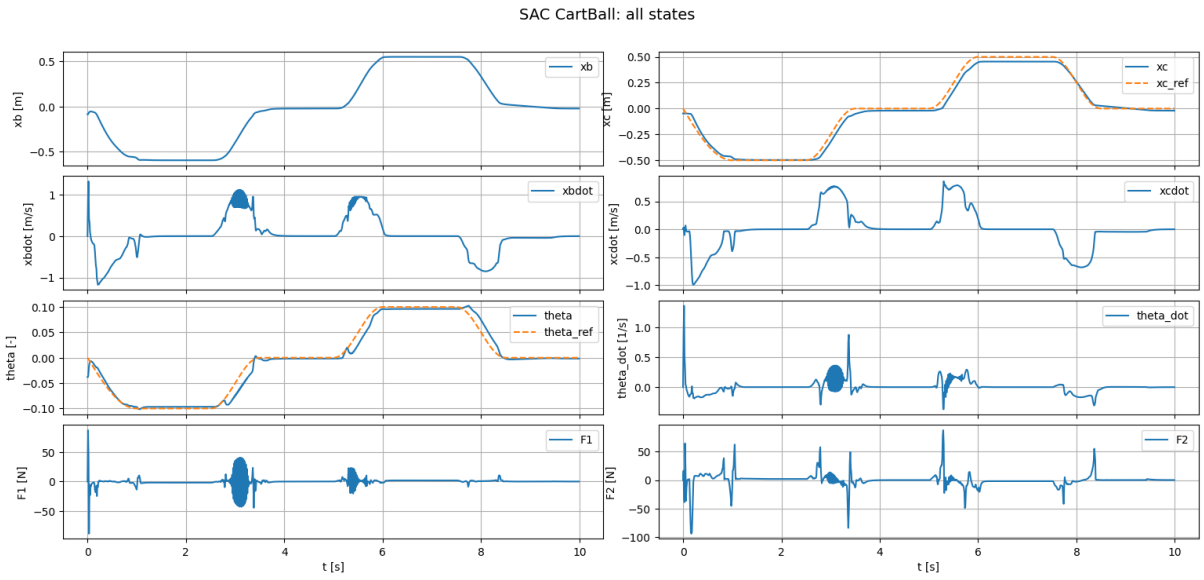


Figure 6.3: State and Input Time Response of SAC for CartBall Problem

SAC tracks the reference signals closely, but displays the most oscillatory behaviour, as seen in Figure 6.3. DSAC exhibits higher tracking errors, but a smoother control response. And RUN-DSAC shows the best behaviour, with improved stability, higher tracking accuracy, and reduced control oscillations. This aligns with the results found for the Citation problem, where RUN-DSAC and SAC were characterized by accurate reference tracking, and the former by smoother control signals.

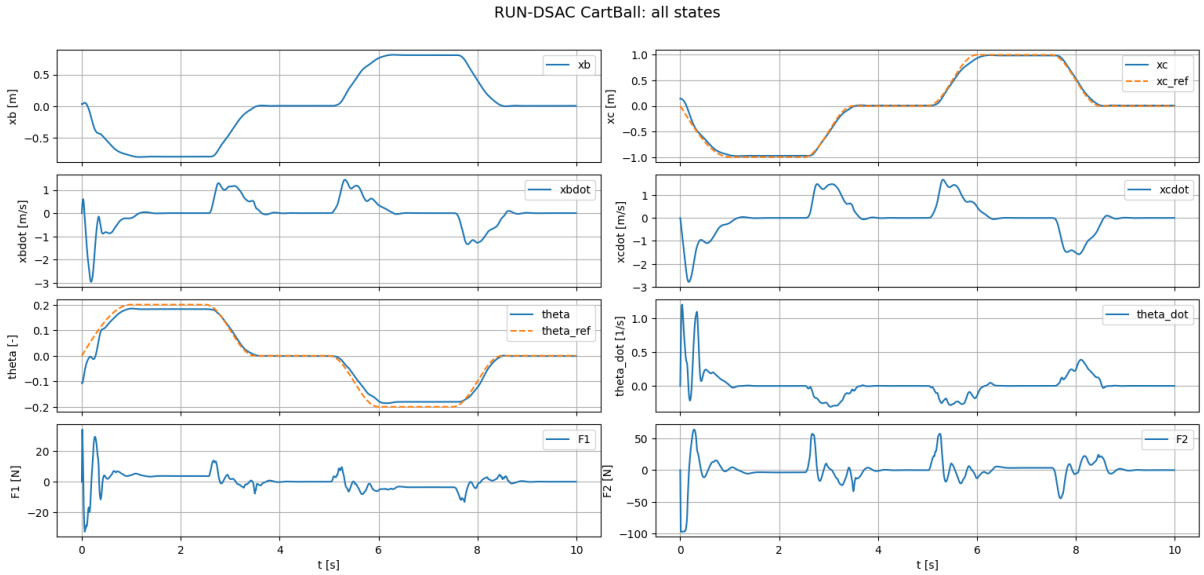


Figure 6.4: State and Input Time Response of RUN-DSAC for CartBall Problem

These offline controllers are used as the initialization for the hybrid controllers. Figure 6.5 shows the response of the hybrid RUN-DSAC-IDHP agent. The hybrid agent retains the baseline performance from the offline controller, but also exhibits modified behaviour. The agent's control signals show a higher magnitude than those of the offline agent, but the forces remain stable and bounded. Additionally, the hybrid controller does not degrade the performance from the offline policy, confirming the correct architecture of the hybrid, where the online adaptation complements the offline initialization.

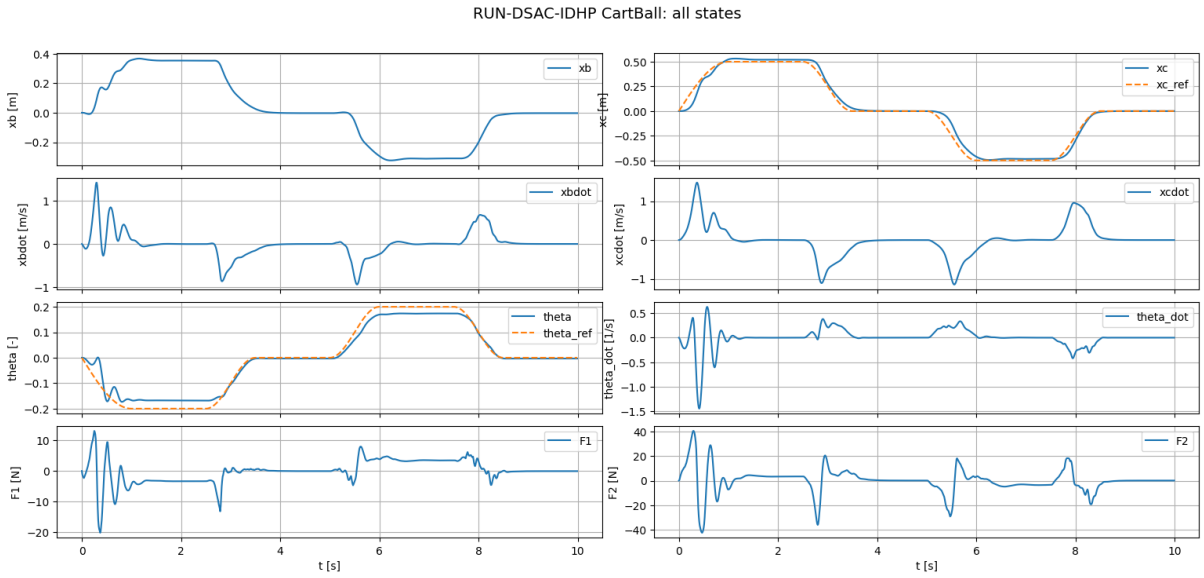


Figure 6.5: State and Input Time Response of RUN-DSAC-IDHP for CartBall Problem

The similarities in the offline and hybrid performances between the CartBall agent's response and the Citation agent's response verify the correctness of the hybrid algorithm.

Finally, the IDHP agent was also implemented for the CartBall problem. However, with the current configuration, the agent is unable to consistently track a reference signal. This is expected, as the problem presents coupled dynamics and simultaneous actuation of both control signals. The current

implementation would require extensive tuning to see if it is able to achieve an accurate tracking performance, but since the objective of this benchmark is verification of the implemented RUN-DSAC-IDHP algorithm, and not IDHP tuning, further optimization is not pursued. The fact that the hybrid agents work correctly suggests the online implementation is correct, and only faces convergence issues from incorrect hyperparameter initialization.

6.2. Validation of the Citation Environment

The Citation simulation environment used in this work is derived from the Delft University Aircraft Simulation Model and Analysis Tool (DSAMAT), which has been previously validated with real flight-test data from the TU Delft Cessna 550 Citation II aircraft, the PH-LAB. Van de Hoek, de Visser, and Pool [20] conducted a comprehensive aerodynamic model identification study and identified a new high-fidelity model for the aircraft from a dataset of dynamic flight maneuvers.

In this study, the flight path of the PH-LAB was reconstructed using an Unscented Kalman Filter (UKF), leading to accurate estimation of the aircraft states and sensor biases from the flight-test data. The aerodynamic force and moment coefficients were then identified using a Two-Step Method accompanied by orthogonal least-squares regression, which resulted in accurate models for the six forces and moments. When this model was validated against independent data, it showed clear improvements over the original DASMAT model [20]. Time-domain comparisons during 3-2-1-1 maneuvers also showed high fidelity between the simulated and measured states, confirming the enhanced accuracy of the new Citation model, and validating it.

Since the aircraft model used in this thesis is based on this validated aerodynamic framework, and the Python implementation was verified with the original Matlab CitAST model in subsection 6.1.1, it is not necessary to revalidate the model against real-flight data. Consequently, the aircraft simulation model used for these reinforcement learning studies is considered a validated representation of the Cessna 550 Citation II dynamics.

6.3. Concluding Remarks

This chapter establishes the accuracy of the simulation environment and the correctness of the reinforcement learning agent implementations employed in this work. First, the Python implementation of the Citation model was verified against the original Matlab/Simulink model, showing identical time-domain responses under identical trim conditions and inputs. Second, the developed reinforcement learning agents were verified using a simplified CartBall problem as a benchmark for performance. The offline agents, SAC, DSAC, and RUN-DSAC, and their hybrid counterparts showed stable learning and accurate tracking performance.

Additionally, the CartBall experiments confirm that the agents work as expected. The performance differences between the offline agents were consistent with those observed for the Citation attitude tracking problem. Additionally, the offline initialization of the hybrid agents affected the corrections of the online component in a similar manner in both the CartBall problem and the Citation problem. The consistency across the different environments further verifies the implementation of the reinforcement learning frameworks.

Finally, validation of the Citation model relied on previous validation conducted by van den Hoek et al. [20], demonstrating improved aerodynamic fidelity of the new identified model compared to the original DASMAT model. Since the present work employs this validated model, later verified when implemented in Python, the simulation environment used in this thesis is both verified and validated.

These validation and verification procedures ensure that the results in the previous chapter are not influenced by errors in the implementation, and thus reflect the actual behaviour of the proposed implementation.

Part IV

Closure

7

Reflection on Research Questions

This research developed a new hybrid RL flight controller composed of an offline Returns Uncertainty-Navigated Distributional SAC agent and an online Incremental Dual Heuristic Programming algorithm. The offline initialization of the agent provides the online agent with a strong policy to improve upon in order to reduce tracking errors.

This section revisits the research questions formulated in chapter 1 and reflects on how these were addressed throughout this work. This section showcases the development of the work, and how each of the research questions was answered through the literature review, methodology, verification, and the evaluation of experimental results.

The first stage of this work established a theoretical foundation. In chapter 2, the fundamentals of reinforcement learning are introduced, including policy and value-based methods, actor-critic architectures, entropy regularization, and approximate dynamic programming. This answered RQ1.1 directly, linking the fundamentals of RL with the requirements of continuous flight control.

Research Question 1

RQ1. What is the most suitable reinforcement learning framework for fault-tolerant flight control?

- **RQ1.1.** What are the key principles of reinforcement learning, and how are they relevant to flight control systems?
- **RQ1.2.** What are the current state-of-the-art Reinforcement Learning algorithms and architectures used for general aircraft flight control?
- **RQ1.3.** What defines a fault-tolerant flight control system, and to what extent can the state-of-the-art RL approaches be applied to achieve this capability?

This background information was followed by a review of state-of-the-art RL algorithms for flight control, including SAC, DSAC, and IDHP-based hybrid methods. This helped answer RQ1.2 and RQ2.1, as the theoretical differences, advantages, and disadvantages of hybrid reinforcement learning were explored in the same section. Additionally, throughout this analysis, fault tolerance was defined as the ability of a controller to maintain stability and sufficient tracking performance under actuator faults, answering RQ1.3. The work by Teirlinck [49] was also used to define the fault scenarios evaluated when testing the fault tolerance of the agents.

Research Question 2

RQ2. How can a hybrid offline-online reinforcement learning controller be designed and implemented?

- **RQ2.1.** What are the defining characteristics and theoretical advantages of a hybrid offline-online architecture for control?
- **RQ2.2.** Which specific offline and online RL algorithms are most suitable to serve as the components of the hybrid controller?
- **RQ2.3.** What is the methodology for designing and implementing the complete hybrid framework?

The conclusion in chapter 2 explored the theoretical advantages of hybrid reinforcement learning, highlighting the joint advantages of online adaptability and stable and robust offline learning. Based on the state-of-the art algorithms and previous implementations, RUN-DSAC was chosen as the offline component due to uncertainty handling, while IDHP was selected as the online component, answering RQ2.2.

The Methodology section of Part I highlighted the method used to implement the reinforcement learning agent. The agents were implemented progressively, starting with IDHP and SAC, until eventually reaching the RUN-DSAC-IDHP framework. The hybrid agent was developed by initializing a tuned IDHP agent with a frozen offline policy. The agents all interacted with a high-fidelity model of the Cessna 550 Citation II aircraft and are tasked with following attitude reference signals. Through the description of this implementation, RQ2.3 was fully answered.

Research Question 3

RQ3. How does the performance of the hybrid controller compare to non-hybrid baselines in fault-tolerant scenarios?

- **RQ3.1.** What metrics are used to evaluate the performance and flight-tolerance of an aircraft flight controller?
- **RQ3.2.** How does the hybrid controller's performance and learning stability compare against a purely offline-trained controller when subjected to nominal and failure modes?
- **RQ3.3.** How does the hybrid controller's performance and stability compare to a purely online-trained controller when adapting to failure modes?

Finally, the following stage of the research evaluated and analysed the experimental results obtained. First, the experimental settings were designed, and the performance metrics were selected and described. The normalized mean absolute error (nMAE) was selected to measure tracking performance, answering RQ3.1. Then, the series of nominal and fault-tolerant experiments was conducted, comparing the performance of the novel RUN-DSAC-IDHP agent against its offline and online baselines, as well as other hybrid and offline agents (SAC-IDHP, DSAC-IDHP, SAC, and DSAC). The hybrid framework demonstrated improved performance under nominal and fault conditions, and strong fault mitigation when subjected to actuator or aerodynamic degradation faults. These results successfully answered RQ3.2 and RQ3.3.

Additionally, the analysis derived from these results helped identify the strengths and limitations of the evaluated agents. The primary advantages include improved fault tolerance and strong initial performance, enabled by the offline policy's fault generalization capabilities. The agents are limited by the actuator limits, and if faults severely affect them, the agents can only partially mitigate the faults. These findings address the final research question, RQ3.4.

Overall Reflection

The progression of this research first established the theoretical foundations of RL and identified the most suitable framework for fault-tolerant flight control (RQ1). Then, the hybrid controller was designed

and implemented to track the attitude of the Cessna 550 Citation II environment (RQ2). And finally, the controller's performance was evaluated under nominal and fault conditions, and compared against a series of baseline agents, showcasing the capabilities and limitations of the proposed framework (RQ3). Through this work, the research objective of developing and evaluating a hybrid RL framework for fault-tolerant flight control of the Cessna Citation II was achieved.

While further work is necessary to address real-world implementation challenges and formalized analysis of the stability of the controller, the results showcase the strength of reinforcement learning based approaches for fault-tolerant flight control. The results also particularly emphasize the advantages of hybrid methods when faced with flight control problems.

8

Conclusion

Fault-tolerant flight control is a fundamental challenge of aerospace engineering. As aircraft become increasingly autonomous, the need for adaptive controllers capable of overcoming faults has become critical. Traditional control strategies rely on accurate modelling of system dynamics and have limited fault-mitigation capabilities. Reinforcement learning has surged as an alternative to traditional flight control. These methods learn directly from interaction with the environment and don't require high-fidelity models of aircraft dynamics to control flight.

Purely offline DRL algorithms, such as SAC or RUN-DSAC, are able to reach control strategies that generalize across fault scenarios and experience robustness when faced with varying flight conditions [49, 21]. However, these show reduced adaptability during flight and struggle to mitigate fault conditions. Online RL agents, by contrast, are known for their adaptability but raise concerns about the safety of in-flight learning. Previous research by Teirlinck [49] and Vieira [51] showed that combining frozen offline policies with adaptive IDHP learning superposed the strengths of both algorithm types. Under nominal conditions, the hybrid retains the stability of the offline-trained agent, and under fault scenarios, it can adapt thanks to its online component.

The hybrid framework implemented in this work combines a RUN-DSAC agent trained offline with an adaptive IDHP agent trained online. The offline component provides a frozen policy that can track reference signals accurately after being trained in a high-fidelity simulation of the Cessna 550 Citation II aircraft. The online component enhances the agent's adaptation to changing conditions and fault scenarios. RUN-DSAC was selected as the offline base for the novel implementation due to its improved use of the information encoded in the uncertainty of the distribution of returns. This allowed the agent to improve performance by choosing risk-prone policies,

The experimental results show that the hybrid controller improves upon the tracking performance of its offline counterpart under nominal conditions. Under fault conditions, RUN-DSAC-IDHP mitigates mild and moderate faults and improves upon the control efforts of the offline base, up to the actuator limits. Compared with the hybrid SAC-IDHP and DSAC-IDHP frameworks, the novel RUN-DSAC-IDHP is slightly outperformed by the latter. However, RUN-DSAC-IDHP handles more severe fault cases and exhibits smoother control, albeit with a slight reduction in tracking accuracy compared to SAC-IDHP. The results highlight the advantages of integrating offline and online learning within a single framework, particularly for fault-tolerant flight control. Offline DRL exhibits strong generalization across fault conditions, thereby providing a strong initialization for the adaptive agent that mitigates faults online.

This research demonstrates that hybrid reinforcement learning architectures provide a viable approach to adaptive, model-free flight control strategies. The RUN-DSAC offline framework has shown the ability to avoid risk for safety-critical policies [21], or, in this work, to seek risk to find a policy that improves tracking performance. The exploitation of the uncertainty of the return distribution allows the agent to learn a safer, more accurate control policy that, when combined with an online component, can ensure

fault-tolerant flight control. Although many challenges remain before solutions like these are implemented on an actual aircraft, this work showcases the capabilities of reinforcement-learning-based flight-control systems in an increasingly autonomous world.

9

Recommendations for Further Work

This chapter explains the recommendations for future research to broaden the analysis of the proposed framework.

Formal Stability and Robustness Analysis of the Implemented Framework

Even though the proposed RUN-DSAC-IDHP framework demonstrated stability and fault-mitigation capabilities across nominal and fault scenarios, it is necessary to analyze the agent's stability guarantees. The closed-loop dynamics arising from the interaction between the adaptive IDHP agent, initialized with a frozen RUN-DSAC policy, and the environment must be analysed. Future work should investigate the robustness margins under disturbances and actuator degradations. In this way, the agent's guarantees would be established, strengthening the agent's foundation as a safety-critical flight control system.

Systematic Hyperparameter Sensitivity Analysis and Adaptive Tuning

Actor-critic RL methods are subject to their learning rates, entropy coefficients, network hyperparameters, and reward scaling. The online IDHP controller is also subject to a set of hyperparameters that affect the agent's stability and performance when tracking errors arise. Hyperparameter tuning was performed in an arbitrary manner, but future work should analyze the sensitivity of the agents to different hyperparameter changes. Reducing this sensitivity would allow this implementation to be more easily reproduced.

Robustness Evaluation

The agent was evaluated under varying fault scenarios, yet the operating envelope explored was very small. A broader evaluation might be necessary under more extreme conditions, complex flight manoeuvres, and combined fault cases. Additionally, the controller's response to atmospheric wind disturbances and sensor noise must be examined. This would help assess the robustness of the proposed approach and the capabilities of the proposed framework to adapt to more demanding, realistic operational regimes.

References

[1] Paul Acquatella, E van Kampen, Qi Ping Chu, et al. “Incremental backstepping for robust nonlinear flight control”. In: *Proceedings of the EuroGNC* 2013 (2013).

[2] Gabriel Moraes Barros and Esther Luna Colombini. “Using soft actor-critic for low-level UAV control”. In: *arXiv preprint arXiv:2010.02293* (2020).

[3] 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)*. IEEE. 2019, pp. 523–533.

[4] Omar Bouhamed et al. “Autonomous UAV navigation: A DDPG-based deep reinforcement learning approach”. In: *2020 IEEE International Symposium on circuits and systems (ISCAS)*. IEEE. 2020, pp. 1–5.

[5] Killian Dally and Erik-Jan van Kampen. “Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control”. In: *AIAA SCITECH 2022 Forum*. American Institute of Aeronautics and Astronautics, Jan. 2022. DOI: 10.2514/6.2022-2078. URL: http://dx.doi.org/10.2514/6.2022-2078.

[6] Hao Dong, Zihan Ding, and Shanghang Zhang. *Deep Reinforcement Learning Fundamentals, Research and Applications: Fundamentals, Research and Applications*. Jan. 2020. ISBN: 978-981-15-4094-3. DOI: 10.1007/978-981-15-4095-0.

[7] Jay Farrell, Manu Sharma, and Marios Polycarpou. “Backstepping-based flight control with adaptive function approximation”. In: *Journal of Guidance, Control, and Dynamics* 28.6 (2005), pp. 1089–1102.

[8] Rick R. Feith. “Safe Reinforcement Learning in Flight Control: Introduction to Safe Incremental Dual Heuristic Programming”. Master’s thesis. Delft, The Netherlands: Delft University of Technology, 2020.

[9] Silvia Ferrari and Robert F Stengel. “Online adaptive critic flight control”. In: *Journal of Guidance, Control, and Dynamics* 27.5 (2004), pp. 777–786.

[10] Silvia Ferrari and Robert F. Stengel. “Model-Based Adaptive Critic Designs”. In: *Handbook of Learning and Approximate Dynamic Programming*. John Wiley Sons, Ltd, 2004. Chap. 3, pp. 65–95. ISBN: 9780470544785. DOI: https://doi.org/10.1002/9780470544785.ch3. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470544785.ch3. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470544785.ch3.

[11] FlyCraft Aviation. *Cessna Citation II Aircraft Page*. Accessed: 2026-03-16. 2024. URL: https://flycraft.com/aircraft/cessna-citation-ii/.

[12] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[13] Pieter van Gils et al. “Adaptive incremental backstepping flight control for a high-performance aircraft with uncertainties”. In: *AIAA guidance, navigation, and control conference*. 2016, p. 1380.

[14] 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, p. 0385.

[15] Tuomas Haarnoja et al. “Reinforcement learning with deep energy-based policies”. In: *International conference on machine learning*. PMLR. 2017, pp. 1352–1361.

[16] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. Pmlr. 2018, pp. 1861–1870.

[17] Dongchen Han and SN Balakrishnan. “Adaptive critic based neural networks for control-constrained agile missile control”. In: *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*. Vol. 4. IEEE. 1999, pp. 2600–2604.

[18] Lei He et al. “Deep reinforcement learning based local planner for UAV obstacle avoidance using demonstration data”. In: *arXiv preprint arXiv:2008.02521* (2020).

[19] 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*. 2020, p. 1844.

[20] 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: *Proceedings of the 4th CEAS Specialist Conference on Guidance, Navigation and Control (EuroGNC)*. Warsaw, Poland, 2017.

[21] Marek Homola, Yifei Li, and Erik-Jan van Kampen. “Uncertainty-Driven Distributional Reinforcement Learning for Flight Control”. In: *AIAA SCITECH 2025 Forum*. 2025, p. 2793.

[22] Erik-Jan van Kampen, Q.P. Chu, and J.A. Mulder. “Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics”. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*. DOI: 10.2514/6.2006-6429. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2006-6429. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2006-6429.

[23] Vijay Konda and John Tsitsiklis. “Actor-critic algorithms”. In: *Advances in neural information processing systems* 12 (1999).

[24] Tim Koning. “Low level quadcopter control using Reinforcement Learning: Developing a self-learning drone”. Master’s thesis. Delft, The Netherlands: Delft University of Technology, Apr. 2020. URL: https://resolver.tudelft.nl/uuid:0b9e0796-13b5-42ba-b231-fbb6aadd5233.

[25] Stephen H Lane and Robert F Stengel. “Flight control design using non-linear inverse dynamics”. In: *Automatica* 24.4 (1988), pp. 471–483.

[26] Stuart “Kipp” Lau. “AINsight: LOC-I Is a Killer in Aviation”. In: *Aviation International News* (July 2025). Expert Opinion – Business Aviation section.

[27] Jun H Lee and Erik-Jan van Kampen. “Online reinforcement learning for fixed-wing aircraft longitudinal control”. In: *AIAA Scitech 2021 Forum*. 2021, p. 0392.

[28] Sergey Levine et al. “Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems”. In: *arXiv preprint arXiv:2005.01643* (2020). arXiv: 2005.01643 [cs.LG]. URL: https://arxiv.org/abs/2005.01643.

[29] Yifei Li and Erik-Jan van Kampen. “Reinforcement Learning-based Intelligent Flight Control for a Fixed-wing Aircraft to Cross an Obstacle Wall”. In: *2024 European Control Conference (ECC)*. IEEE. 2024, pp. 1636–1641.

[30] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[31] Derong 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. DOI: 10.1109/TSMC.2020.3042876.

[32] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

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

[34] Gaurav Saini and SN Balakrishnan. “Adaptive critic based neurocontroller for autolanding of aircrafts”. In: *Proceedings of the 1997 American Control Conference (Cat. No. 97CH36041)*. Vol. 2. IEEE. 1997, pp. 1081–1085.

[35] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[36] Peter Seres. “Distributional Reinforcement Learning for Flight Control: A risk-sensitive approach to aircraft attitude control using Distributional RL”. Defended November 9, 2022. Master’s thesis. Delft, The Netherlands: Delft University of Technology, Nov. 2022.

[37] Jennie Si et al. *Handbook of learning and approximate dynamic programming*. John Wiley & Sons, 2004.

[38] S Sieberling, QP Chu, and JA Mulder. “Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction”. In: *Journal of guidance, control, and dynamics* 33.6 (2010), pp. 1732–1742.

[39] David Silver et al. “Deterministic policy gradient algorithms”. In: *International conference on machine learning*. Pmlr. 2014, pp. 387–395.

[40] Yuda Song et al. “Hybrid rl: Using both offline and online data can make rl efficient”. In: *arXiv preprint arXiv:2210.06718* (2022).

[41] Brian L Stevens, Frank L Lewis, and Eric N Johnson. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[42] Bo Sun and Erik-Jan van Kampen. “Incremental Model-Based Heuristic Dynamic Programming with Output Feedback Applied to Aerospace System Identification and Control”. In: *2020 IEEE Conference on Control Technology and Applications (CCTA)*. 2020, pp. 366–371. DOI: 10.1109/CCTA41146.2020.9206261.

[43] Bo Sun and Erik-Jan van Kampen. “Incremental Model-Based Global Dual Heuristic Programming for Flight Control”. In: *IFAC-PapersOnLine* 52.29 (2019). 13th IFAC Workshop on Adaptive and Learning Control Systems ALCOS 2019, pp. 7–12. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2019.12.613. URL: https://www.sciencedirect.com/science/article/pii/S2405896319325558.

[44] 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 (2020), p. 103425. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2019.103425. URL: https://www.sciencedirect.com/science/article/pii/S0952197619303288.

[45] Bo Sun and Erik-Jan van Kampen. “Intelligent adaptive optimal control using incremental model-based global dual heuristic programming subject to partial observability”. In: *Applied Soft Computing* 103 (2021), p. 107153. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2021.107153. URL: https://www.sciencedirect.com/science/article/pii/S1568494621000764.

[46] Sihao Sun et al. “Incremental nonlinear fault-tolerant control of a quadrotor with complete loss of two opposing rotors”. In: *IEEE Transactions on Robotics* 37.1 (2020), pp. 116–130.

[47] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems* 12 (1999).

[48] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[49] Casper Teirlinck. “Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance”. Master’s thesis. Delft, The Netherlands: Delft University of Technology, Sept. 2022. URL: https://resolver.tudelft.nl/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85.

[50] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30 (Sept. 2015). DOI: 10.1609/aaai.v30i1.10295.

[51] Lucas Vieira. “Evaluating Reinforcement Learning for Quadcopter Control: From Simulation to Real World”. Defended November 17, 2023. Master’s thesis. Delft, The Netherlands: Delft University of Technology, Nov. 2023.

[52] C. C. de Visser, X. Wang, and E. J.. van Kampen. *AE4311 Nonlinear and Adaptive Flight Control – Lecture 1: Introduction and NDI*. Lecture slides. 2024.

[53] Zhuang Wang et al. “A pretrained proximal policy optimization algorithm with reward shaping for aircraft guidance to a moving destination in three-dimensional continuous space”. In: *International Journal of Advanced Robotic Systems* 18.1 (2021), p. 1729881421989546.

[54] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2016. arXiv: 1511.06581 [cs.LG]. URL: https://arxiv.org/abs/1511.06581.

[55] Paul J. Werbos. “A Menu of Designs for Reinforcement Learning Over Time”. In: *Neural Networks for Control*. The MIT Press, Jan. 1991. ISBN: 9780262291293. DOI: 10.7551/mitpress/4939.003.0007. eprint: https://direct.mit.edu/book/chapter-pdf/2301870/9780262291293_cac.pdf. URL: https://doi.org/10.7551/mitpress/4939.003.0007.

[56] Y Zhou, E van Kampen, and Qi Ping Chu. “Incremental approximate dynamic programming for nonlinear flight control design”. In: *Proceedings of the EuroGNC 2015* (2015).

[57] Ye Zhou, Erik-Jan van Kampen, and Q.P. Chu. “Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control”. In: Oct. 2016.

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

[59] Ye Zhou, Erik-Jan van Kampen, and Q.P. Chu. “Nonlinear Adaptive Flight Control Using Incremental Approximate Dynamic Programming and Output Feedback”. In: *Journal of Guidance, Control, and Dynamics* 40 (Nov. 2016), pp. 1–8. DOI: 10.2514/1.G001762.