Uncertainty-Aware Reinforcement Learning for Flight Control

Mastering the Mystery of Flight

Marek Homola

TU Delft

Uncertainty-Aware Reinforcement Learning for Flight Control

Mastering the Mystery of Flight

Thesis report

by

Marek Homola

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on January 19, 2024 at 14:00

Thesis committee:	
Chair:	Dr. ir. Ewoud Smeur
Supervisors:	Dr. ir. Erik-Jan van Kampen
	Yifei Li
External examiner:	Dr. Alexei Sharpanskykh
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	March, 2023 - January, 2024
Student number:	4828542

Faculty of Aerospace Engineering · Delft University of Technology

Preface

We are at the dawn of a new era where the rapid evolution of Artificial Intelligence is revolutionizing the world. In the coming decades, the machine learning techniques like Reinforcement Learning (RL) are anticipated to tackle increasingly complex challenges in engineering, economics, medicine, climate sciences, and other fields. This development, coupled with my curiosity for bio-inspired algorithms and genuine love for anything that flies, fueled my inspiration throughout this thesis. The result? *RUN-DSAC* – an uncertainty-driven RL algorithm designed to enhance the data sample efficiency and safety of autonomous flight systems, bringing them yet another step closer to practical real-world application. In the spirit of scientific advancement, the code repository is made publicly available[1].

Completing this thesis also symbolizes a graceful touchdown, concluding my five-year flight through the skies of Aerospace Engineering at TU Delft. This unforgettable experience shaped me into the engineer and the individual I stand as today, and I am deeply grateful to the many inspiring people who journeyed with me through this remarkable chapter of my life.

First and foremost, Erik-Jan van Kampen deserves my deepest gratitude for his unwavering support, sage guidance, and being an inexhaustible source of inspiration. He patiently reviewed and discussed my often ridiculously extensive progress reports and even went above and beyond by offering invaluable advice on my future career paths, showcasing his extraordinary commitment to this research and his exceptional role as a supervisor. Special thanks also go to Yifei Li, whose participation in my progress meetings and insightful remarks consistently added value to my work. Additionally, I extend my sincere appreciation to the numerous professors at TU Delft who have provided consultations on specific details of my research. While I cannot name each individually, their collective expertise and guidance have been essential in shaping my work. Finally, I would like to specially acknowledge Marc G. Bellemare, Mark Rowland, and Will Dabney at Google, the pioneers of Distributional Reinforcement Learning. Their readiness to engage with my endless inquiries, despite their demanding schedules, has been a highlight of this journey. The opportunity to discuss my work with such leading figures in the RL research field was an incredibly stimulating and enriching experience.

This chapter of my life would not have been the same without the steadfast support of my friends from Delft and Slovakia. A special thanks to Doru and Alex – your humor has been a source of endless laughter, and memories we have created are ones I will cherish forever. I eagerly anticipate the times we have yet to share. Additionally, my gratitude extends to Lucas and Michaela, Aerospace Engineering graduates and wonderful friends of mine, whose guidance and tips on my thesis were invaluable. Your support was crucial in maintaining my sanity and confidence throughout this period. Last but definitely not least, a heartfelt thanks to my family, with a special mention to my parents. Their sacrifices and relentless support have been instrumental in my achievements. Equally, my gratitude goes to Patrik, my brother, whose well-timed jokes were the perfect remedy for stress.

Turning the final page at TU Delft, I carry forward a treasure trove of knowledge, profound gratitude, and lasting bonds. All of this is thanks to you! Ďakujem!

Marek Homola,
Delft, January 2024

[1]`https://github.com/mhomola/RUN-DSAC`

Contents

Nomenclature

List of Abbreviations

ACD	Adaptive Critic Designs
ADP	Approximate Dynamic Programming
AFCS	Automatic Flight Control System
AI	Artificial Intelligence
ANN	Artificial Neural Network
BQ	Bayesian Quadature
BRL	Bayesian Reinforcement Learning
CDF	Cumulative Distribution Function
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DOF	Degree of freedom
DP	Dynamic Programming
DPG	Deterministic Policy Gradient
DRL	Distributional Reinforcement Learning
DSAC	Distributional Soft Actor-Critic
EOM	Equations of Motion
FCSD	Flight Control System Design
GP	Gaussian Process
IID	Independent and identically distributed
INDI	Incremental Nonlinear Dynamic Inversion
IQN	Implicit Quantile Network
LOC	Loss of Control
LSTM	Long Short Term Memory
MAB	Multi-Armed Bandits
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
MSBE	Mean Squared Bellman Error
MSE	Mean Squared Error
NDI	Nonlinear Dynamic Inversion
NLP	Natural Language Processing
nMAE	Normalized Mean Absolute Error
NN	Neural Netwok
OBM	On-board Model
PID	Proportional-Integral-Derivative
POMDP	Partially Observable Markov Decision Process
QF	Quantile Function
QR	Quantile Regression
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RQ	Research Question
RTV	Right Truncated Variance
RUN-DSAC	Returns Uncertainty Navigated DSAC
RV	Random Variable
SAC	Soft Actor-Critic
SGD	Stochastic Gradient Descent
TD	Temporal Difference
TS	Thomson Sampling
UAV	Unmanned Aerial Vehicle
UCB	Upper Confidence Bound
UM-DSAC	Unconstrained Monotonic (NN) DSAC
UMNN	Unconstrained Monotonic Neural Network
VTOL	Vertical Take-off and Landing

List of Symbols

α	Angle of attack	[rad]
α_T	Entropy temperature factor	[-]
α_{TD}	TD learning rate	[-]
β	Angle of sideslip	[rad]
γ	Discount factor	[-]
δ_e	TD-error	[-]
$\underline{\delta}$	Control surface deflections	[rad]
δ_a	Aileron deflection	[rad]
δ_e	Elevator deflection	[rad]
$\dot{\delta}_e$	Elevator deflection rate	[rad/s]
δ_r	Rudder deflection	[rad]
ϵ	Exploration coefficient	[-]
η	Entropy Temperature	[-]
θ	Pitch angle	[rad]
θ	Critic parameters	[-]
$\bar{\theta}$	Target critic parameters	[-]
θ_{ll}	Orientation angle of the Lunar Lander	[rad]
$\dot{\theta}_{ll}$	Rotational velocity of the Lunar Lander	[rad/s]
κ	Huber-loss parameter	[-]
λ_S	Spatial smoothness parameter weight	[-]
λ_T	Temporal smoothness parameter weight	[-]
μ	Mean	[-]
π	Policy	[-]
ξ	Risk-distortion parameter	[-]
Σ	Covariance matrix	[-]
σ	Variance	[-]
τ_i	Quantile fraction	[-]
ϕ	Roll angle	[rad]
Ψ	Distortion risk-measure	[-]
ψ	Yaw angle	[rad]
Ω	Angular velocity	[rad/s]
$\mathcal{A}$	Action space	[-]
a	Action	[-]
a'	Subsequent action	[-]
A_s	Step signal amplitude	[deg]
$\mathcal{B}$	Batch of data	[-]
$\mathcal{D}$	Memory buffer	[-]
d	Terminal state indicator	[-]
D_C	Cramér distance	[-]
D_{KL}	KL divergence	[-]
$F_{Z,\theta}^{-1}$	Quantile function	[-]
$\mathcal{H}$	Entropy	[-]
J	Cost function	[-]
L_S	Spatial smoothness parameter	[-]
L_T	Temporal smoothness parameter	[-]
$\mathcal{L}_W$	Wasserstein distance	[-]
N_q	Number of quantile fractions	[-]
$\mathcal{P}$	State-transition space	[-]
q	Pitch rate	[rad/s]
Q^π	State-action value function	[-]
$\mathcal{R}$	Reward function	[-]
R	Return	[-]
r	Reward	[-]
$\mathcal{S}$	State space	[-]
s	State	[-]
s'	Subsequent state	[-]
$\mathcal{T}$	Trajectory function	[-]
t	Time (step)	[s]
Δt	Simulation time step	[s]
T_{lateral}	Lateral engine thrust value	[-]
T_{main}	Main engine thrust value	[-]
$\underline{u}$	Control input vector	[-]
u	Forward velocity component	[m/s]
v	Sideways velocity component	[m/s]

V_{tas}	True airspeed	[m/s]
V^{π}	Value function	[-]
w	Vertical velocity component	[m/s]
w	Policy parameters	[-]
w_s	Step signal width	[s]
$\underline{x}$	Dynamic state vector	[-]
x_{ll}	Horizontal coordinate of the Lunar Lander	[m]
$\dot{x}_{ll}$	Horizontal velocity of the Lunar Lander	[m/s]
y_{ll}	Vertical coordinate of the Lunar Lander	[m]
$\dot{y}_{ll}$	Vertical velocity of the Lunar Lander	[m/s]
Z	Value distribution function	[-]

List of Figures

List of Tables

1 Introduction

In the ever-evolving world of aviation, ensuring the safety and efficiency of flight operations remains a paramount concern. For decades, traditional automatic flight control systems (AFCS) have been the cornerstone of aviation, employing established control algorithms and techniques. These AFCS have undoubtedly played a vital role in enabling safe and reliable flight operations. However, as aerospace systems become increasingly sophisticated and diverse, conventional AFCS are struggling to keep up with the demands of modern aviation, thereby exposing their inherent limitations.

Current AFCS heavily depend on accurate model descriptions of system dynamics [1], leading to a time-consuming and iterative design process. This costly approach involves careful system identification through processes such as Computational Fluid Dynamics (CFD) computations, scaled wind tunnel measurements, and eventually flight tests [2]. The reliance on detailed models not only extends the development timeline but also necessitates extensive verification and validation activities. Moreover, developing, validating and verifying high-fidelity models is becoming even more difficult and expensive with increased complexity in the design and control of novel aerospace system configurations, such as V-shaped flying wings [3, 4], vertical take-off and landing (VTOL) systems [5, 6, 7], aircraft with morphing wings [8, 9], ornithopters [10, 11], and others. To overcome this limitation of current AFCS, there is a growing need for model-free controller design approaches that streamline the development process and reduce reliance on extensive model identification and validation.

Furthermore, as the traditional AFCS controllers are designed based on predetermined assumptions, they lack the flexibility and adaptability to effectively handle situations outside their original design parameters. Unforeseen scenarios encompass a wide range of challenging situations, including adverse weather conditions, system failures, unexpected obstacles, and others. The inability of AFCS to adapt and respond appropriately to these scenarios can lead to compromised flight control, jeopardizing the safety of the aircraft. This is examplified by the crash of Air France Flight 447 in 2009, caused by the loss of accurate airspeed readings and inadequate pilot training to handle such scenarios [12]. Consequently, this highlights the demand for highly intelligent and adaptable autonomous aerospace systems that can safely operate in partially observable and uncertain environments, without strict reliance on accurate models.

Substantial research effort was devoted to address the need for reduced model fidelity requirements and more autonomous systems. Robust control techniques, such as H_∞-synthesis [13, 14], offer assurances of closed-loop stability and performance even in the presence of model uncertainties and disturbances, but often at the expense of very conservative control performance [15]. Another popular approach is the Nonlinear Dynamic Inversion (NDI), which involves inverting the dynamics of a nonlinear system to design a control law [16]. However, its limitations include sensitivity to modeling errors and uncertainties, as well as the challenge of accurately estimating the system's states. Sensor-based approaches, such as Incremental NDI (INDI) [17] and Incremental Backstepping (IBS) [18], offer a promising alternative by reducing the dependence on global models and instead utilizing incremental control effectiveness models based on sensor measurements. INDI and IBS have demonstrated adaptability and fault-tolerance, leading to enhanced robustness against modeling errors and uncertainties [19, 20, 21]. However, challenges related to sensor synchronization and filtering arise with these methods. This research proposes an alternative solution based on bio-inspired Artificial Intelligence (AI) - *Reinforcement Learning*.

1.1. Reinforcement Learning for Flight Control

Reinforcement Learning (RL) is a subfield of Machine Learning that focuses on decision-making in dynamic and uncertain environments. It offers a promising avenue for designing autonomous flight control systems that can learn to operate effectively without explicit a-priori knowledge of the underlying system dynamics. RL provides a unique framework for training control systems by enabling them to learn through interactions with their environment [22]. Unlike traditional control algorithms that require explicit models and predefined control strategies, RL algorithms learn to make decisions by optimizing a cumulative reward signal.

In the context of flight control, RL agents can learn to control an aircraft by continuously exploring and adapting their actions based on the feedback received from the environment in order to achieve a certain goal, such as following a desired trajectory or minimizing fuel consumption. This would not be feasible with traditional RL algorithms, including SARSA and Q-learning since they were designed for problems with finite and discrete state-action spaces [22]. To address this, recent advancements in RL, particularly Deep Reinforcement Learning, have harnessed deep neural networks (DNN) as function approximators. This enables the application of RL to complex flight control scenarios.

Despite the potential benefits of RL in flight control, there are specific challenges that need to be addressed. First, RL algorithms necessitate a substantial number of data samples to converge, which becomes more challenging due to the complexity and high dimensionality of flight control tasks. Secondly, flight control is safety-critical, raising concerns about the safety of RL agents during the training phase, where exploration may lead to unexplored and potentially hazardous states. Furthermore, the performance and safety of algorithms trained under specific conditions may degrade under varying conditions, highlighting robustness as a significant concern.

Several methods have been proposed to approach these challenges. For instance, online incremental Approximate Dynamic Programming (ADP) methods, such as Incremental Dual-Heuristic Programming (IDHP), have demonstrated sample-efficient online learning and adaptive control [23, 24, 25]. However, these algorithms have limited generalization power and are only suitable for low-dimensional control tasks. Alternatively, Hierarchical RL (HRL) flight controllers were proposed to address dimensionality. This improved the sample efficiency but also led to difficulties in policy learning, inconsistent outcomes, and increased number of hyperparameters. To address the safety concern, a Shielded RL controller was applied to control the flight path angle, which integrates a *shield* controller to monitor the safety of actions [26]. However, the shield relies on predefined safety rules instead of learning them, which limits its ability to adapt to new environments or dynamic scenarios. Another study showed that the Soft Actor-Critic (SAC) algorithm can achieve robust flight control and adaptation to unforeseen failures but had inconsistent results and a subpar success rate during training [27]. To improve SAC's learning characteristics, Distributional SAC (DSAC) was proposed, which learns the entire probability distribution of rewards instead of a point value [28]. Despite its limitations, such as ill-defined distribution estimates in the early stages of learning, the improved sample efficiency and robustness of DSAC motivated this research.

DSAC has highlighted the potential benefits of learning entire distributions instead of point values, addressing both sample efficiency and safety challenges. By explicitly capturing the uncertainty in the form of probability distributions, RL algorithms can make more effective use of limited samples, significantly enhancing data efficiency. For example, uncertainty estimates might allow agents to focus exploration in regions of the state-action space that are uncertain or have higher potential for learning. This not only accelerates the learning process but also enables RL systems to handle complex and uncertain environments more robustly. Furthermore, probabilistic distributions enable safety-aware decision-making, empowering agents to assess risk and avoid hazardous or unknown states. Consequently, this research is committed to exploring the potential of uncertainty-aware RL methods applied in flight control.

1.2. Research Formulation

As stated earlier, the goal of this research is to investigate the possibilities and benefits of utilizing uncertainty-aware RL techniques in the domain of flight control. More specifically, this work investigates how complementing a RL algorithm with uncertainty information can enhance its sample efficiency, while evaluating its robustness to various unforeseen phenomena.

Before formulating a precise research objective and delineating the research scope through research questions, it is essential to establish several fundamental concepts. Firstly, in this context, *uncertainty-aware* methods pertain to approaches that acquire supplementary statistical insights about unknown variables beyond their mean value, including aspects like variance, probability distribution, or other relevant measures of uncertainty. Additionally, *sample efficiency* denotes the quantity of samples required by an algorithm to attain a given performance level, such as a specific tracking error in relation to a reference trajectory. Subsequently, the research objective can be precisely outlined.

Research Objective

This research aims to improve the sample efficiency of state-of-the-art RL flight controllers in reference-tracking tasks by integrating uncertainty information into the learning process. Additionally, it seeks to assess the robustness of such a method to unforeseen in-flight faults, sensor noise, and atmospheric disturbances, in comparison to uncertainty-agnostic approaches.

In order to achieve the research objective and define the research scope, a comprehensive set of research questions and subquestions was formulated and categorized into three phases: *Methodology*, *Implementation*, and *Evaluation*.

Research Questions - Methodology

RQ-M-1 What is the state-of-the-art for RL in flight control?
RQ-M-2 What are the sources of uncertainty in RL and how can they be leveraged?
RQ-M-3 What is the state-of-the-art in *uncertainty-aware* RL methods?
- **RQ-M-3a** How do these methods model the uncertainty information?
- **RQ-M-3b** Which of these methods is the most suitable for flight control in terms of implementation feasibility, uncertainty prediction accuracy, and sample efficiency?

A wide array of RL algorithms is available, each tailored to specific applications. Therefore, **RQ-M-1** is essential because it shall provide a comprehensive understanding of the current RL advancements and approaches that are suitable for flight control in terms of their ability to handle high-dimensional continuous action spaces and safety-sensitive problems. Subsequently, **RQ-M-2** guides the research towards its core. It leads to an examination of how incorporating uncertainty can improve the sample-efficiency and other performance metrics of RL algorithms in the context of flight control. However, there are different sources of uncertainty and they can affect different variables in an RL algorithm, which is addressed by **RQ-M-3**. Last but not least, there are numerous RL methods that incorporate the uncertainty information in learning, which is why **RQ-M-3** was added. It was split into two subquestions - **RQ-M-3a**, which encourages to investigate the underlying principles of these uncertainty-aware RL methods, and **RQ-M-3b**, which navigates towards evaluating the most promising method for the flight control application among all the studied approaches.

Research Questions - Implementation

RQ-I-1 How can the proposed uncertainty-aware RL controller be implemented in a Cessna Citation II simulation model?
- **RQ-I-1a** At which control level should the proposed RL controller be implemented?
- **RQ-I-1b** Which simplifications can be implemented in the simulation model, while maintaining its relevant characteristics?

RQ-I-2 How can in-flight faults, sensor noise, and atmospheric disturbances be appropriately implemented in the simplified model?

Once the method is specified, the research can proceed with the implementation part, which is captured by the *Implementation* set of research questions. First, **RQ-I-1** was included to investigate how the uncertainty-aware RL method selected in the methodology part can be implemented in the controller of Cessna Citation II simulation model, which will serve as a test environment. To answer this question comprehensively, it was divided into two subquestions. Subquestion **RQ-I-1a** aims to determine the control level used for evaluating the proposed methods, encompassing the selection of Degrees-of-Freedom (DOF) to be controlled, the inputs provided to the controller, and the outputs generated by the controller. On the other hand, subquestion **RQ-I-1b** aims to address the inherent complexity of full flight control by exploring a simplified aircraft model and trim conditions to strike a balance between model complexity and computational efficiency, while delivering meaningful research outcomes. Furthermore, the inclusion of **RQ-I-2** aims to ensure the algorithm's readiness for robustness evaluation.

Research Questions - Evaluation

RQ-E-1 How can the performance of the proposed uncertainty-aware RL controller be evaluated with a Cessna Citation II simulation model?

- **RQ-E-1a** Which metrics can be used to compare the reference-tracking accuracy of flight controllers?
- **RQ-E-1b** Which reference-tracking signals should be selected to evaluate the controller?
- **RQ-E-1c** Which specific fault conditions should be considered during the testing phase?

RQ-E-2 How does the proposed uncertainty-aware RL architecture perform in reference-tracking flight tasks in terms of sample efficiency and accuracy?

RQ-E-3 How robust is the selected uncertainty-aware RL flight controller to unforeseen system faults, sensor noise and atmospheric disturbances?

The final phase of the research involves evaluating the effectiveness of the proposed uncertainty-aware RL method in flight control. First of all, the evaluation process needs to be set up, which is addressed by **RQ-E-1**. Specifically, **RQ-E-1a** focuses on identifying appropriate metrics to effectively evaluate the tracking performance of the proposed controller, while **RQ-E-1b** aims to determine the optimal selection of trajectories for reference-tracking to ensure accurate assessment. Furthermore, **RQ-E-1c** aims to explore various fault scenarios that should be included in the robustness analysis. Finally, the performance of the proposed uncertainty-aware algorithm can be assessed based on sample efficiency and reference-tracking accuracy, as indicated by **RQ-E-2**, and its robustness, as addressed by **RQ-E-3**.

1.3. Structure of the Report

The purpose of this report is to describe the methodology applied in this research project, present the findings, and answer the research questions specified earlier. It is structured into four parts, each addressing distinct aspects of the research:

Part I presents the research paper derived from this study, which encapsulates the research motivation, background, applied methodology, and primary findings.

Part II summarizes the literature study that lays the foundation for this research. An overview of the fundamental concepts of (Deep) RL is presented in Chapter 2, followed by a discussion in Chapter 3 on the current state-of-the-art Deep RL methods and their applicability to flight control. Next, Chapter 4 explores various sources of uncertainty in reinforcement learning and examines several Uncertainty-Aware RL methods. This is followed by a preliminary analysis in Chapter 5.

Part III encompasses both supplementary findings and detailed expansions of results presented in the paper, offering a deeper scientific exploration and analysis. First, the aircraft's simulated time traces under in-flight faults and unseen flight conditions are analyzed in Chapter 6, followed by the discussion on hyperparameter optimization strategies in Chapter 7. Sensitivity analysis of the RL-based control systems to the environment design is then explored in Chapter 8, while Chapter 9 provides an evaluation of the architectural and computational complexity of the algorithms used in this research. Lastly, verification and validation procedures are presented in Chapter 10.

Part IV concludes the research project by reflecting on the research questions and offering a comprehensive conclusion in Chapter 11, and suggesting avenues for future research in Chapter 12.

Part I

Scientific Article

UNCERTAINTY-DRIVEN DISTRIBUTIONAL REINFORCEMENT LEARNING FOR FLIGHT CONTROL

Marek Homola[*]
Faculty of Aerospace Engineering
Delft University of Technology
P.O. Box 5058, 2600GB Delft, The Netherlands

ABSTRACT

In the rapidly evolving aviation sector, the quest for safer and more efficient flight operations has historically relied on traditional Automatic Flight Control Systems (AFCS) based on high-fidelity models. However, such models not only incur high development costs but also struggle to adapt to new, complex aircraft designs and unexpected operational conditions. As an alternative, deep Reinforcement Learning (RL) has emerged as a promising solution for model-free, adaptive flight control. Yet, RL-based approaches pose significant challenges in terms of sample efficiency and safety assurance. Addressing these gaps, this paper introduces Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC). Designed to enhance the learning efficiency, adaptability, and safety of flight control systems, RUN-DSAC leverages the rich uncertainty information inherent in the returns distribution to refine the decision-making process. When applied to the attitude tracking task on a high-fidelity non-linear fixed-wing aircraft model, RUN-DSAC demonstrates superior performance in learning efficiency, adaptability to varied and unforeseen flight scenarios, and robustness in fault tolerance that outperforms the current state-of-the-art SAC and DSAC algorithms.

Keywords Autonomous Flight Control · Deep Distributional Reinforcement Learning · Uncertainty-Driven Control · Safety-Critical Systems · Sample Efficient Machine Learning

1 Introduction

In the dynamically advancing world of aviation, ensuring the safety and efficiency of flight operations remains paramount. Traditional automatic flight control systems (AFCS) have been the cornerstone of aviation for decades, leveraging established control algorithms and techniques. Yet, these systems are increasingly limited by their dependence on intricate, high-fidelity models of system dynamics [62]. Developing, validating and verifying these models is resource-intensive, involving careful system identification through computational simulations, scaled wind tunnel measurements, and flight tests [51]. Moreover, this challenge escalates with growing complexity in the design and control of novel aerospace configurations, such as V-shaped flying wings [7, 21], vertical take-off and landing (VTOL) systems [71, 4], aircraft with morphing wings [70, 47], ornithopters [14, 18], and designs with high-aspect-ratio wings introducing significant dynamics nonlinearity due to increased structural flexibility [68, 69]. Hence, there is an intensifying urgency for model-free control techniques that streamline the development cycle and minimize the reliance on extensive model identification and validation. Furthermore, traditional AFCS, constrained by predetermined design assumptions, are susceptible to failure in unforeseen situations like adverse weather or system malfunctions [28, 38]. Incidents, such as the 2009 Air France Flight 447 crash [23], emphasize the critical need for more intelligent and adaptable autonomous aerospace systems that can ensure safe operation under uncertain circumstances.

Extensive research has focused on reducing model fidelity requirements to enable more autonomous systems. Robust control techniques like H_∞-synthesis [39, 12] ensure closed-loop stability even in the presence of model uncertainties and disturbances, but at the expense of very conservative control performance [57, 59]. Nonlinear Dynamic Inversion (NDI) offers a popular alternative by inverting system dynamics to formulate control laws [10]. However, its limitations lie in sensitivity to modeling errors, uncertainties, and the accuracy of the system's state estimates. Sensor-based approaches, such as Incremental NDI (INDI) [42] and Incremental Backstepping (IBS) [2], mitigate the dependence on global models and instead rely on incremental control models based on sensor measurements. This approach improves robustness to model imperfections, increasing adaptability and fault tolerance [60, 63, 13]. Nevertheless, INDI and IBS methods face challenges related to sensor synchronization and filtering. This research proposes an alternative solution based on bio-inspired Artificial Intelligence - *Reinforcement Learning (RL).*

[*]MSc Student in the Control and Simulation Division

Deep RL, leveraging deep neural networks (DNN) to approximate complex functions [27], offers a promising avenue for designing autonomous flight control systems capable of learning without explicit a-priori system dynamics knowledge. These Deep RL agents adaptively refine their actions through continuous interaction with the environment in order to achieve objectives such as trajectory tracking or fuel economy [64]. Yet, implementing RL in flight control presents notable challenges. First, RL algorithms necessitate extensive data samples to converge, which becomes even more challenging due to the complexity and high dimensionality of flight control tasks [35]. Secondly, the safety-critical nature of flight control raises concerns regarding the reliability and safety of the converged RL control policies [30].

Several methods have been proposed to approach these challenges. Online incremental Approximate Dynamic Programming (ADP) methods like Incremental Dual-Heuristic Programming (IDHP) demonstrate sample-efficient online learning and adaptive control but face generalization and dimensionality limits [33, 22]. Hierarchical RL flight controllers offer a solution to dimensionality but at the cost of complicating policy learning and added hyperparameter complexity [35, 37]. In the domain of safety, Shielded RL introduces a specialized 'shield' controller to regulate flight path angles, yet its reliance on predefined safety rules restricts adaptability [26]. Alternatively, the Soft Actor-Critic (SAC) algorithm has shown potential for robust flight control and adaptability to unforeseen failures but suffers from training inconsistencies and low convergence success rates [17]. Building upon SAC, Distributional SAC (DSAC) was introduced, learning the full returns distribution instead of solely the expected value. DSAC's improved sample efficiency and robustness in flight control [56, 66] serve as the impetus for this research.

In conventional DSAC algorithms, action selection is governed by the mean of the learned return distribution [5, 15, 16]. However, this underutilizes the rich information embedded in the distribution. To fill this void, previous studies have explored using these distributions to distort the expectations and synthesize risk-sensitive policies, improving the safety of RL-based flight controllers [43, 56]. Building on these foundations, we introduce *Returns Uncertainty-Navigated DSAC* (RUN-DSAC), a novel variant designed to exploit the returns uncertainty for more informed decision-making. Our methodology offers an alternative to other uncertainty-aware approaches, such as Bayesian methods, which suffer from scalability issues [11] and rely on an accurate selection of a prior distribution [24, 9, 53].

This research advances intelligent flight control systems with four key contributions: Firstly, it extends previous research [56] by validating DSAC's superior learning efficiency and stability over SAC in flight control, while exploring its robustness in previously untested generalization scenarios and fault conditions. More importantly, we introduce the novel uncertainty-driven RUN-DSAC algorithm, enabling the synthesis of sample-efficient and safe flight control policies. Thirdly, the study reveals that prioritizing low-uncertainty state-action pairs during learning markedly improves both learning performance and exploration safety, resulting in superior tracking performance with improved fault tolerance and robustness to unseen flight conditions. Lastly, we show that while favoring high-uncertainty may hinder learning performance, the resulting policies exhibit heightened resilience to perturbations in unexpected flight conditions.

2 Background

First, the core principles of RL need to be defined, including the notions of Maximum Entropy RL and Distributional RL. Additionally, the flight control task is formulized within the context of RL.

2.1 Fundamentals of Deep Reinforcement Learning

RL represents a bio-inspired machine learning methodology that relies on an iterative trial-and-error mechanism for deriving optimal control policies. In this framework, an autonomous agent incrementally refines its decision-making proficiency in a specific task domain through continuous agent-environment interactions [64]. This sequential decision-making process is conceptualized as a Markov Decision Process (MDP), defined by the set $\mathcal{M} \sim \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$. Here $S \subset \mathbb{R}^n$ defines the state-space, $\mathcal{A} \subset \mathbb{R}^m$ the action-space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward mapping, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the stochastic state transition dynamics, and $\gamma \in (0, 1)$ the discount factor. At each discrete time-step t, the agent selects an action $a_t \in \mathcal{A}$ according to the policy $a \sim \pi(\cdot|s)$ depending on the current state s_t. It then observes the state-transition tuple $T_t = \langle s, a, r, s' \rangle$, with the instantaneous reward $r = R(s, a)$ and subsequent state s'.

The agent's objective is to identify the optimal policy π^* that maximizes the expected *return*, defined as the expected sum of discounted rewards across a decision sequence. This objective is facilitated by employing $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which estimates the expected return from the given state s upon selecting action a and thereafter following policy $\pi \in \Pi$, as given by Equation 1.

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right] \mid a_t \sim \pi(\cdot|s_t),\ s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t),\ s_0 = s,\ a_0 = a \tag{1}$$

To enable systematic convergence through iterative algorithms, Q^π can be expressed using the contractive Bellman equation [8], as given by Equation 2.

$$\mathcal{T}^\pi Q(s, a) = \mathbb{E}[R(s, a)] + \gamma \mathbb{E}_{\mathcal{P},\pi} [Q(s', a')] \tag{2}$$

Deep RL integrates DNNs for function approximation in both value-based and policy-based RL frameworks, as well as in hybrid actor-critic architectures. Value-based methods like DQN use DNNs to estimate the action-value function $Q_\theta(s,a) \approx Q(s,a)$, thus implicitly defining the policy, predominantly in discrete action spaces [49, 48]. Conversely, policy-based approaches directly approximate the optimal policy $\pi_w(s) \approx \pi^*(s)$ and are suited for continuous action spaces, albeit at the cost of higher variance and slower convergence [58]. Nonetheless, recent advancements such as TRPO and PPO have partly mitigated these limitations [55, 54]. Actor-critic methods synergize the strengths of both approaches - the critic evaluates the action-value function, providing an estimate of the expected return that aids in reducing the variance of the policy optimized by the actor [46]. This enables efficient handling of complex, high-dimensional, continuous control tasks. State-of-the-art actor-critics, such as SAC, have demonstrate robust performance and fast convergence in intricate control environments [32].

2.2 Soft Actor-Critic (SAC)

In contrast to traditional RL that solely focuses on maximizing cumulative reward, *Maximum Entropy RL* methods like SAC aim to optimize a modified objective function that balances both reward accumulation and policy entropy maximization, as given by Equation 3. Maximizing the entropy term $\mathcal{H}(\pi_w(\cdot|s_t))$ promotes more effective exploration of the state space and increases policy robustness.

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \left[R(s_t,a_t) + \alpha_T \mathcal{H}(\pi_w(\cdot|s_t)) \right] \right] \quad \textit{with} \quad \mathcal{H}(\pi_w(\cdot|s)) = \mathbb{E}_{a\sim\pi_w}\left[-\log(\pi_w(a|s)) \right] \tag{3}$$

This leads to the formulation of a *soft Bellman equation*, as given by Equation 4. Here, α_T is a temperature parameter that controls the trade-off between maximizing the expected return and maximizing the entropy, thereby influencing the degree of stochasticity and exploration in the policy π.

$$\mathcal{T}_S^\pi Q(s,a) = \mathbb{E}\left[R(s,a)\right] + \gamma \mathbb{E}_{\mathcal{P},\pi}\left[Q(s',a') - \alpha_T \log \pi(a'|s')\right] \tag{4}$$

Haarnoja et al. [31] introduce dynamic adaptation of α_T to attain a specified target entropy $\bar{\mathcal{H}}$, thereby improving exploration efficiency and reducing hyperparameter sensitivity. The target entropy $\bar{\mathcal{H}}$ is typically set in correspondence with the action space dimension, $\bar{\mathcal{H}} = -m$. The loss function for optimizing this adaptive α_T is detailed in Equation 5. In this formulation, $\mathcal{B}$ represents a mini-batch of transitions $\mathcal{T}$ drawn from the experience replay buffer $\mathcal{D} = \{\mathcal{T}_0, \mathcal{T}_1, ...\}$, which is characteristic for off-policy RL methods.

$$\mathcal{L}^{\mathcal{B}}(\alpha_T) = \mathbb{E}_{\mathcal{B}}\left[\alpha_T \bar{\mathcal{H}} - \alpha_T \log \pi_w(a|s)\right] \tag{5}$$

To mitigate action-value function overestimation, SAC employs a double-critic architecture [25]. This entails training two parallel Q-functions, $Q_{\theta_{1,2}}(s,a)$, and computing the temporal-difference (TD) error δ using their minimum (Equation 6). Alongside these *behavioral* Q-networks, SAC also maintains *target* Q-networks, parameterized by $\bar{\theta}$. These networks undergo slower updates, achieved through incremental adjustments via Polyak averaging with a step size ζ, thereby enhancing learning stability.

$$\mathcal{L}_Q^{\mathcal{B}}(\theta_i) = \mathbb{E}_{\mathcal{B}}[\delta_i^2] \quad \textit{with} \quad \delta_i = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\bar{\theta}_i}(s',a') - \alpha \log \pi_w(a'|s')\right) - Q_{\theta_i}(s,a), \;\; a' \sim \pi_w(\cdot|s') \tag{6}$$

In SAC, the actor employs a stochastic policy represented by an m-dimensional multivariate Gaussian distribution with diagonal covariance, where the mean vector $\mu_w \in \mathbb{R}^m$ and covariance diagonal $\sigma_w \in \mathbb{R}^m$ are modeled by a DNN with parameters w. Furthermore, the *reparameterization* trick is employed to facilitate backpropagation through the stochastic policy [31], where the action a is reparametrized as a deterministic function of policy parameters and an independent Gaussian noise variable ϵ. To bound action domain, a *tanh* squashing function is applied post-sampling, as in Equation 7. While the actor is stochastic during training, it opts for deterministic actions in the evaluation phase by selecting the mean vector μ_w.

$$a_w(s) = \tanh\left(\tilde{a}_w(s)\right) \quad \textit{with} \quad \tilde{a}_w(s) \sim \mathcal{N}(\mu_w(s), \sigma_w(s)) \tag{7}$$

Finally, the policy loss function used to update the actor's parameters is detailed in Equation 8.

$$\mathcal{L}_\pi^{\mathcal{B}}(w) = \mathbb{E}_{\mathcal{B}}\left[\alpha \log \pi_w(a_w(s)|s) - \min_{i=1,2} Q_{\theta_i}(s, a_w(s))\right] \tag{8}$$

2.3 Distributional Reinforcement Learning

Unlike classical RL's focus on optimizing the expected sum of rewards, *Distributional RL* aims to learn the full probability distribution over returns. This is formalized as $Z : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$, where $\mathcal{Z}$ denotes the action-value distribution space with finite moments for each state-action pair, as defined by Equation 9 [6].

$$\mathcal{Z} := \left\{ Z : \mathcal{S} \times \mathcal{A} \to P(\mathbb{R}) \mid \mathbb{E}\Big[\|Z(s,a)\|_p\Big] < \infty \quad \forall (s,a),\; p \geq 1 \right\} \tag{9}$$

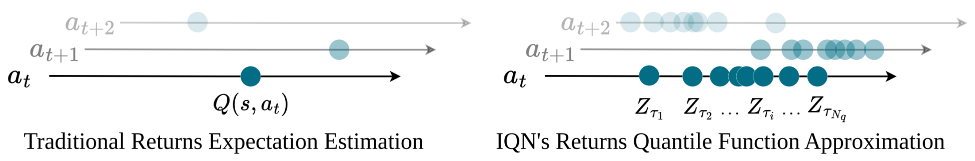

a_{t+2}
a_{t+1}
a_t
$Q(s, a_t)$
Traditional Returns Expectation Estimation
a_{t+2}
a_{t+1}
a_t
Z_{τ_1} Z_{τ_2} … Z_{τ_i} … $Z_{\tau_{N_q}}$
IQN's Returns Quantile Function Approximation

Figure 1: Comparison of traditional (left) and distributional (right) approaches in representing the returns for a particular state s and selected action a at discrete time-step t, inspired by Dabney et al. [16]

This leads to the formulation of the *distributional Bellman operator* $\mathcal{T}_D^\pi : \mathcal{Z} \to \mathcal{Z}$ [5], as articulated in Equation 10. Here, the notation $X \overset{\mathcal{D}}{=} Y$ signifies that random variables X and Y follow identical distributions.

$$\mathcal{T}_D^\pi Z(s, a) \overset{\mathcal{D}}{=} R(s, a) + \gamma Z(s', a') \tag{10}$$

The $\mathcal{T}_D^\pi$ operator is established to be contractive under a p-Wasserstein metric [5], given in Equation 11. This metric quantifies optimal displacement costs between probability distributions within a specified metric space [45]. Here, the quantile functions $F_X^{-1}(\tau)$ and $F_Y^{-1}(\tau)$ are formulated as $F_Z^{-1}(\tau) = \inf\{z \in \mathbb{R} : \tau \leq F_Z(z)\}$, where $F_Z(z)$ represents the cumulative distribution function and τ the quantile fraction.

$$W_p(F_X, F_Y) = \left(\int_0^1 \left\| F_X^{-1}(\tau) - F_Y^{-1}(\tau) \right\|_p d\tau \right)^{1/p} \tag{11}$$

Various techniques for parameterizing return distributions have been proposed, such as categorical distributions [5] and quantile regression [15]. The present study employs Implicit Quantile Networks (IQN), selected for their capacity to flexibly model complex, multi-modal return distributions while maintaining parametric and computational efficiency [16]. IQN implicitly approximate the continuous quantile function $F_Z^{-1}(\tau)$ by propagating quantile fractions, stochastically sampled from a uniform distribution ($\tau \sim U([0, 1])$), through a DNN to derive corresponding quantile values. The comparison between the traditional return expectation estimation and the IQN's approximation of $F_Z^{-1}(\tau)$ for randomly sampled τ is depicted in Figure 1.

2.4 Distributional Soft Actor-Critic (DSAC)

DSAC extends SAC through the incorporation of an IQN critic $Z_\tau(s, a; \theta)$ to approximate the return distribution function [44, 20]. This fusion of maximum entropy learning with distributional critics has been demonstrated to be effective in learning high-dimensional continuous control tasks [44]. Modelling the returns distribution also facilitates the generation of risk-sensitive policies, which was shown to enhance the safety of RL-based flight control in a quadcopter [43] and a business jet [56].

In accordance with the $\mathcal{T}_D^\pi$ operator (Equation 10), the TD-error δ_{ij}^k for a given pair of quantile fractions $\hat{\tau}_i$ and $\hat{\tau}_j$ is specified by Equation 12. Here, $k \in \{1, 2\}$ represents the index of each Z_τ-network within the double-critic framework, with $\hat{\tau}_{i,j} = (\tau_{i,j} + \tau_{i+1,j+1})/2$, and $\bar{\theta}_k$ and $\bar{w}$ denoting parameter vectors for the target quantile and policy networks, respectively.

$$\delta_{ij}^k = r + \gamma \left[\min_{k=1,2} Z_{\hat{\tau}_i}(s', a'; \bar{\theta}_k) - \alpha \log \pi(a'|s'; \bar{w}) \right] - Z_{\hat{\tau}_j}(s, a; \theta_k) \quad \textit{with} \quad a' \sim \pi_w(\cdot|s'),\ \tau_i, \tau_j \sim U([0, 1]) \tag{12}$$

Parameter vector θ is optimized via quantile regression, employing the weighted pairwise Quantile Huber Loss as given by Equation 13 [36], where $\mathbb{1}$ represents the indicator function. This loss function transitions from a quadratic to a linear form at a specified threshold κ, conferring robustness to outliers compared to conventional mean squared error.

$$\rho_\tau^\kappa(\delta) = \left| \tau - \mathbb{1}_{\{\delta<0\}} \right| \mathcal{L}_\kappa(\delta) \quad \textit{with} \quad \mathcal{L}_\kappa(\delta) \begin{cases} \frac{1}{2}\delta^2 & \textit{if}\, |\delta| \leq \kappa \\ \kappa(|\delta| - \frac{1}{2}\kappa) & \textit{otherwise} \end{cases} \quad \textit{and} \quad \mathbb{1}_{\{\delta<0\}} \begin{cases} 1 & \textit{if}\, \delta < 0 \\ 0 & \textit{otherwise} \end{cases} \tag{13}$$

Consequently, for a set N of independent quantile fractions, the critic loss function is formalized as in Equation 14.

$$\mathcal{L}_Z^{\mathcal{B}}(\theta) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (\tau_{i+1} - \tau_i) \rho_{\hat{\tau}_j}^\kappa (\delta_{ij}^k) \tag{14}$$

The policy π_w in DSAC adheres to the parametrization structure inherent to SAC. However, DSAC's critics output a distribution of returns, hence a transformation function $\Psi : \mathcal{Z} \to \mathbb{R}$ is employed to convert this distribution into a scalar action-value function Q. While this *risk measure function* Ψ can be implemented as a risk-neutral expectation $\mathbb{E}[\cdot]$, it may also be realized through alternative functions such as a Wang transform [67], enabling the modulation of risk sensitivity by skewing the critic's estimated returns distribution towards either more *risk-averse* or *risk-seeking* behaviors [16]. The resultant distorted action-value Q^r is incorporated into the distributional policy loss function, as specified by Equation 15.

$$\mathcal{L}_\pi(w) = \mathbb{E}_{\mathcal{B}} \left[\alpha \log \pi_w(a_w(s)|s) - Q_\theta^r(s, a_w(s)) \right] \quad \textit{with} \quad Q_\theta^r(s, a_w(s)) = \Psi \left[\min_{i=1,2} Z_{\theta_i}(s, a_w(s)) \right] \tag{15}$$

2.5 Reinforcement Learning for Flight Control

The fixed-wing aircraft flight control poses a highly complex problem with non-linear and highly-coupled transition dynamics with 6 degrees of freedom (DOF). In general terms, the flight control problem can be defined by Equation 16.

$$\dot{x} = f(x, u, t) + w \approx f(x, u) + w \tag{16}$$

Here, f symbolizes the non-linear transition dynamics governed by aircraft's equations of motion, with the state vector $x \in \mathbb{R}^n$ and control input vector $u \in \mathbb{R}^m$ as its arguments. To concentrate on the system's inherent dynamics, quasi-stationarity is assumed, thus excluding explicit time dependence from f. The additive term $w \in \mathbb{R}^n$ accounts for stochastic disturbances, modeling the system's inherent noise and aerodynamic perturbations. For analytical tractability, the dynamics are often decoupled into longitudinal and lateral components, each with its subset of DOF.

To formulate a *flight trajectory tracking* control task as an MDP, the dynamic state vector x must be augmented with tracking errors with respect to desired flight trajectories τ_{ref}. Furthermore, some flight states may be unobservable, uncertain or distorted by measurement noise, leading to a Partially Observable MDP (POMDP). Hence, it is imperative to distinguish between the RL state vector s, which reflects the agent's observations, and the true dynamic state vector x. Actions a may either directly correspond to control inputs u or represent incremental control commands to refine input smoothness, with the actual control inputs u incorporated into the state vector [17]. The reward function is usually designed to penalize deviations from the reference trajectory τ_{ref} proportionally to the absolute $R(s, a) \propto |(\tau_{\text{ref}} - x)|$ or the squared tracking error $R(s, a) \propto (\tau_{\text{ref}} - x)^2$.

Applying RL to partially observable flight control systems introduces significant challenges. The Markov property assumption, where the future state transition $\mathcal{P}(s'|a, s)$ is fully predictable from the observation vector s, is invalidated by incomplete state observability, which impedes the formulation of predictive and reliable control policies. Additionally, the high dimensionality of the state-action space requires highly sample-efficient algorithms that can generalize across diverse flight conditions. Flight control's safety-critical nature further constrains the use of exploratory methods, as suboptimal actions during online in-flight learning or in response to unobserved states encountered post-training risk catastrophic outcomes. All these factors culminate in a pronounced *simulation-to-reality gap*, where RL agents trained in simulations may falter in real-world flight scenarios, emphasizing the necessity for safety-prioritizing agents that are both sample efficient and robust to environmental uncertainties. The proposed RUN-DSAC algorithm addresses these challenges by integrating uncertainty quantification into the decision-making framework of the RL agent, enhancing sample efficiency and safety.

3 Methodology

This section details the RUN-DSAC framework and the implementation of the aircraft control task addressed in this study.

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

RUN-DSAC extends DSAC by embedding uncertainty quantification into policy decisions, enhancing control strategies in complex and uncertain environments.

3.1.1 The Principle of RUN-DSAC

Building upon DSAC, the RUN-DSAC algorithm presents an innovative approach to adaptive control in uncertain environments. While DSAC distorts the value distribution to encode risk preferences implicitly, RUN-DSAC refines this method by explicitly adjusting policy decisions based on quantified uncertainty. This approach draws inspiration from Liu et al. [43], who estimated right truncated variance (RTV) using DSAC's quantile functions to modulate the distortion function for risk-adaptive flight control of a quadrocopter. In contrast, RUN-DSAC computes the variance from the full distribution of returns for a comprehensive understanding of returns variability, while it explicitly channels this information into the actor's policy derivation.

Research in *risk-sensitive* RL recognizes variance $\mathbb{V}[\cdot]$ as an intuitive and comprehensive metric for uncertainty, fluctuation, and decision robustness [61, 19, 1]. Additionally, Bellemare et al. [5] show that the distributional Bellman operator $\mathcal{T}_D^\pi$ contracts in the second moment of the discounted returns, asserting that convergence in value distribution space concomitantly yields accurate variance estimations. To incorporate variance into the learning objective, RUN-DSAC shifts the focus from solely optimizing the expected return, $\Psi(Z) = \mathbb{E}[Z]$, to balancing the mean against the standard deviation, $\Psi(Z) = \mathbb{E}[Z] + \mu\sqrt{\mathbb{V}[Z]}$. The Q-function is thus modified as in Equation 17, where Q_θ is the expectation. This value then guides the policy update.

$$Q_\theta^V(s, a) = Q_\theta(s, a) + \mu\sigma_Q \tag{17}$$

Here, σ_Q is the standard deviation that is approximated with Equation 18, where $Q(s, a)$ represents the returns expectation, and μ is the *uncertainty modulation factor* that balances between the mean and the standard deviation in the objective.

$$\sigma_Q = \sqrt{\sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) \left[Z_{\hat{\tau}_i,\theta}(s, a) - Q(s, a)\right]^2} \tag{18}$$

The coefficient μ modulates the agent's risk preference. A positive μ inclines the policy towards actions with high variance in returns, promoting risk-affine exploration, which will be referred to as *Risky*. Conversely, a negative μ induces risk-averse tendencies by favoring lower-variance actions, thereby enhancing the predictability and safety of the policy, and will be referred to as *Conservative*. As the agent's familiarity with the environment evolves, the initial high variance due to exploratory actions naturally diminishes. To reflect this maturation, μ is programmed to linearly decay to zero across N_{RUN} learning episodes.

The architecture of the RUN-DSAC algorithm is depicted in Figure 2, with the corresponding pseudocode detailed in Appendix A.

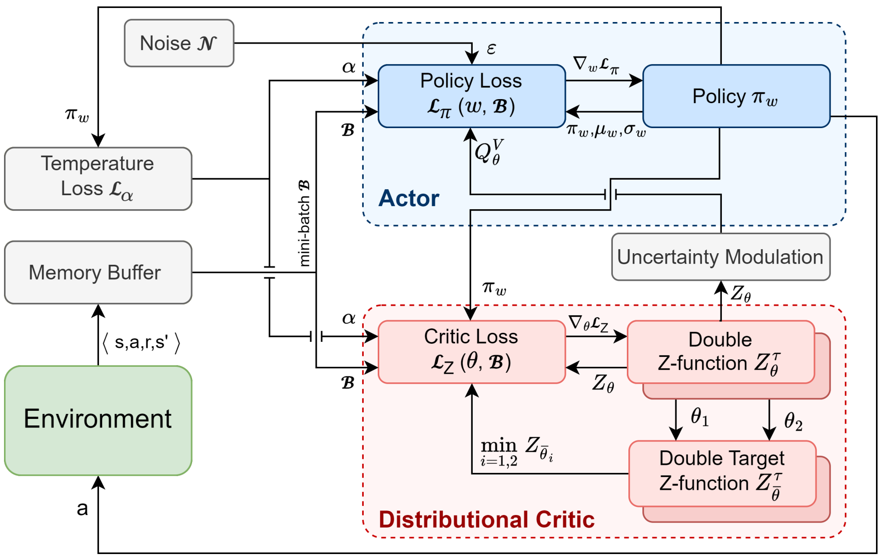

Noise $\mathcal{N}$
ε
α
Policy Loss
$\mathcal{L}_\pi$ (w, $\mathcal{B}$)
$\nabla_w \mathcal{L}_\pi$
Policy π_w
π_w
$\mathcal{B}$
π_w, μ_w, σ_w
Q^V_θ
Temperature Loss $\mathcal{L}_\alpha$
mini-batch $\mathcal{B}$
Actor
Uncertainty Modulation
Memory Buffer
π_w
Z_θ
α
$\nabla_\theta \mathcal{L}_Z$
Critic Loss
$\mathcal{L}_Z$ (θ, $\mathcal{B}$)
Double Z-function Z^τ_θ
$\langle$ s,a,r,s' $\rangle$
$\mathcal{B}$
Z_θ
Environment
θ_1
θ_2
$\min_{i=1,2} Z_{\bar{\theta}_i}$
Double Target Z-function $Z^\tau_{\bar{\theta}}$
a
Distributional Critic

Figure 2: Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC) architecture.

3.1.2 Actor and Critic Design

The actors used in this work are modeled as DNNs with a sequence of hidden layers that approximate a stochastic multivariate $\tanh$-Gaussian policy with state inputs s, as depicted in Figure 3. The final hidden layer's output diverges into two streams, one estimating the mean μ and the other computing the logarithm of the standard deviation $\log \sigma$, which is then exponentiated to derive σ. Actions a are produced by sampling from $\mathcal{N}(\mu, \sigma)$ and applying a $\tanh$ function to ensure boundedness within the action space. However, during deterministic policy evaluation, such as in post-training assessments, the network defaults to $\tanh(\mu)$ for consistent, non-stochastic output. Finally, the network provides reparametrization to facilitate back-propagaion.

The DSAC critics employ IQNs to estimate the return distribution Z_θ, as depicted in Figure 4. Here, states s and actions a are embedded in the Ψ layer. Concurrently, N randomly sampled quantile fractions $\tau \sim U([0, 1])$ undergo cosine embedding Φ, as given by Equation 19, with parameters w_{ij}, b_j, and *Sigmoid* activation σ. These embeddings are elementwise-multiplied through Hadamard product [16], followed by layer normalization for consistent output scaling and numerical stability [3], before progressing through hidden layers H to the output layer F. The parameters are optimized iteratively with *Adam* stochastic gradient descent (SGD) [40] using the quantile Huber loss (Equation 13).

$$\phi_j(\tau) = \sigma\left(\sum_{i=0}^{N-1} \cos(\pi i \tau) w_{ij} + b_j\right) \tag{19}$$

3.1.3 Policy Regularization

Deep RL controllers often learn policies that lack action smoothness, leading to high-frequency control signal oscillations. In aviation control systems, such oscillatory actions compromise tracking performance and precipitate overheating, increased power consumption, and actuator failure. This issue can be addressed using incremental control strategies, where the agent's actions are cumulative adjustments [17]. Such an approach, however, escalates the dimensionality, negatively impacting sample efficiency. Instead, the present paper employs the Conditioning for Action Policy Smoothness (CAPS) regularization technique to achieve smoother control policies without increasing problem complexity [52].

The CAPS methodology uses regularization to minimize the policy function's Lipschitz constants both temporally and spatially, as reflected in the temporal smoothness term $\mathcal{L}_T$ (Equation 20) and the spatial smoothness term $\mathcal{L}_S$ (Equation 21). Here, $||.||_2$

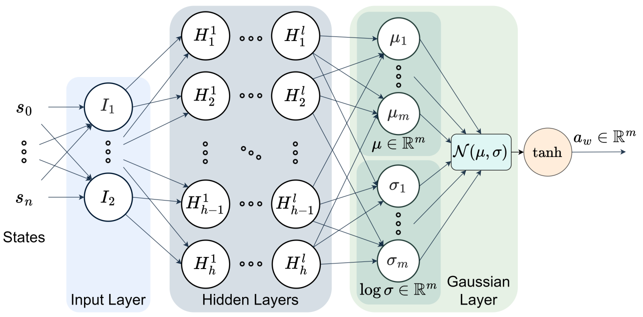


Figure 3: Multivariate tanh-Gaussian policy.

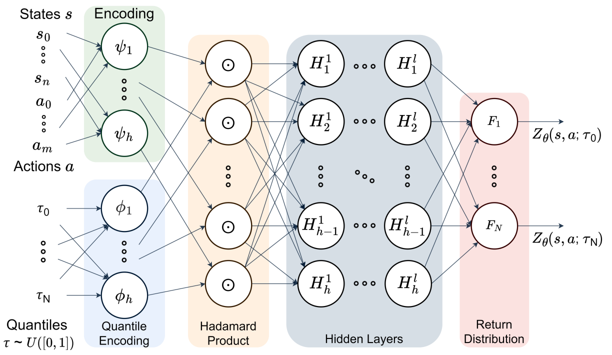


Figure 4: Implicit quantile network (IQN) architecture, inspired by Dabney et al. [16].

represents the Euclidean norm, with $\mathcal{L}_T$ enforcing action consistency across successive time steps and $\mathcal{L}_S$ term promoting action equivalence for analogous states.

$$\mathcal{L}_T = ||\pi(s_t) - \pi(s_{t+1})||_2 \quad (20) \qquad \mathcal{L}_S = \|\pi(s) - \pi(\bar{s})\|_2 \quad with \quad \bar{s} \sim N(s, \sigma_{\text{CAPS}}) \quad (21)$$

The hyperparameters λ_S and λ_T balance action smoothness and policy improvement, as given in Equation 22, leading to a smoother control policy when integrated with the loss function, as in Equation 23.

$$\mathcal{L}_\pi^{CAPS} = \lambda_T \mathcal{L}_T + \lambda_S \mathcal{L}_S \quad (22) \qquad \mathcal{L}_\pi(w) = \mathbb{E}_\mathcal{B}\left[\alpha \log \pi_w(a_w(s)|s) - Q_\theta^V(s, a_w(s)) + \mathcal{L}_\pi^{CAPS}\right] \quad (23)$$

3.2 Flight Attitude Control

This subsection delves into the attitude control of a Cessna Citation II, employing the DASMAT simulation model.

3.2.1 Simulation Environment

This study examines the attitude control of a Cessna Citation II using DASMAT, a validated high-fidelity aircraft simulation model featuring fully-coupled, non-linear dynamics [34]. DASMAT encompasses 12 dynamic states $x \in \mathbb{R}^{12}$, as defined in Equation 24. These include three airspeed components (true airspeed V, angle of attack α, and angle of sideslip β), three angular velocities (roll p, pitch q, and yaw r rates), and three Euler angles for angular orientation (roll ϕ, pitch θ, and yaw ψ). Also, translational positions in a local tangent plane are quantified through horizontal coordinates X_e and Y_e, and altitude h.

$$x = [p, q, r, V, \alpha, \beta, \phi, \theta, \psi, h, X_e, Y_e]^T \in \mathbb{R}^{12} \quad (24)$$

The agent's control input $u \in \mathbb{R}^3$, defined in Equation 25, exclusively controls aerodynamic surface deflections: elevator δ_e, aileron δ_a, and rudder δ_r. The corresponding actuator deflection limits are given by Equation 26 [41], which define the control policy's action space. Thrust is regulated by a separate inner loop tasked with velocity regulation, thereby reducing overall control problem complexity. These dynamic states and control inputs are illustrated in Figure 5.

$$u = [\delta_e, \delta_a, \delta_r]^T \in \mathbb{R}^3 \quad (25) \qquad \mathcal{A} = \underbrace{[-17°, 15°]}_{\delta_e} \times \underbrace{[-19°, 15°]}_{\delta_a} \times \underbrace{[-22°, 22°]}_{\delta_r} \in \mathbb{R}^3 \quad (26)$$

The simulation operates at a refresh rate of f_s = 100 Hz. It assumes ideal sensors and models actuators through low-pass filter dynamics, coupled with predetermined deflection saturation limits.

3.2.2 Controller Architecture

The control diagram in Figure 6 shows the feedback loop between the non-linear RL agent and the controlled plant. The agent directly controls actuator deflections based on the observation vector s defined in Equation 27, which is a subset of the state vector x augmented with tracking error specified in Equation 28.

$$s = [\theta_e, \phi_e, \beta_e, p, q, r, \alpha]^T \in \mathbb{R}^7 \quad (27) \qquad e = [\theta_e, \phi_e, \beta_e] = [\theta_{\text{ref}} - \theta, \phi_{\text{ref}} - \phi, \beta_{\text{ref}} - \beta] \in \mathbb{R}^3 \quad (28)$$

The reward function r, detailed in Equation 29 and inspired by Dally and van Kampen [17], penalizes the $L1$ norm of attitude tracking error, where a scaling parameter c_r compensates for the lower magnitude of sideslip error by proportionally weighting

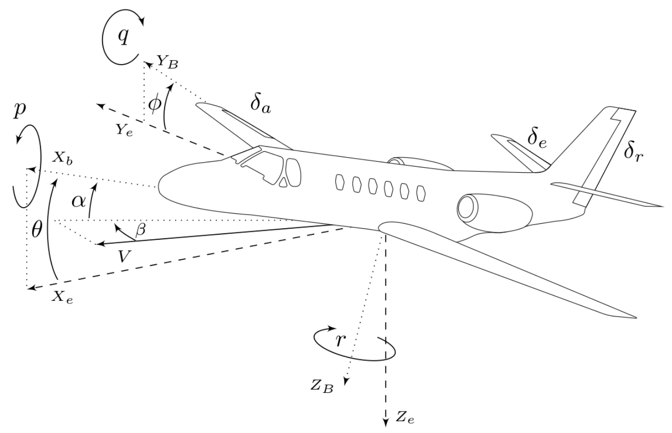


Figure 5: Cessna Citation II's dynamic states and flight control surfaces, showing the body frame (X_b, Y_b, Z_b) at an attitude defined relative to the inertial frame (X_e, Y_e, Z_e). Yaw (ψ) is set to zero for clarity. (Adopted from Seres and van Kampen [56])

its influence.

$$r(s,a) = -\frac{1}{3}\left\|\text{clip}\left[c_r \odot e, -1, 1\right]\right\|_1 \quad \textit{with} \quad c_r = \frac{6}{\pi}[1,1,4]^T \in \mathbb{R}^3 \tag{29}$$

Additionally, three soft constraints were implemented to enforce the flight envelope, specified as $h \geq 100$ m, $|\theta| \leq 60°$ and

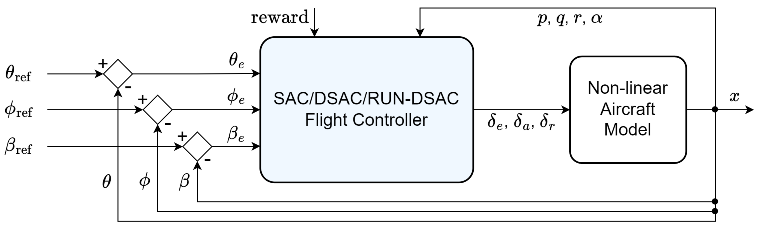


Figure 6: RL-based flight attitude controller architecture.

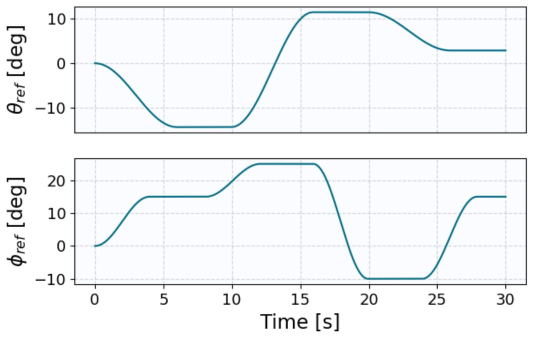


Figure 7: Random cosine-smoothed step sequences, applied as training reference signals for pitch θ_{ref} and roll ϕ_{ref} during a single episode.

$|\phi| \leq 75°$. Constraint violations trigger premature episode termination and incur a high sparse negative reward r_p, proportional to the remaining time frames as outlined in Equation 30. A positive constant c_p scales the penalty for early termination, with an empirically optimized value of $c_p = 2$. These constraints were observed to guide the agent towards desired behaviors, effectively improving the sample efficiency.

$$r_p = -c_p f_s (t_{\max} - t) \tag{30}$$

3.3 Experiment Design

This paper compares the uncertainty-driven RUN-DSAC with the original DSAC in flight attitude control tasks. Additionally, these algorithms are benchmarked with SAC, the state of the art in RL that previously demonstrated fault-tolerant flight control capabilities, but showed inconsistencies in offline training convergence and heightened sensitivity to stochastic processes and hyperparameters [17]. Essentially, three approaches to uncertainty management are contrasted: SAC, which operates without uncertainty information; DSAC, which models uncertainty; and RUN-DSAC, which actively uses the modeled uncertainty.

The agents are trained in 30-second episodes using randomly generated pitch θ_{ref} and roll ϕ_{ref} reference signals, starting from a trimmed flight at h = 2,000 m and V = 90 m/s. These reference signals are formulated as cosine-smoothed step functions, with their amplitudes uniformly sampled from 15 discrete levels within [-20°, 20°] for θ_{ref} and [-35°, 35°] for ϕ_{ref}. Figure 7 illustrates an instance of such a signal set, while the sideslip reference β_{ref} is set to zero. Post each episode, *average returns* are evaluated by averaging the rewards accumulated over 10 independent trajectories $\mathcal{T}$ collected under the current policy.

Table 1: Optimized hyperparameters for the SAC, DSAC and RUN-DSAC agents

Shared			**DSAC (Additional)**		
Hidden layers structure	$\bar{h}$	64x64	Num. of quantiles	N_q	32
Actor learning rate	-	$1{\cdot}10^{-3}$	Num. of cosine neurons	$\|\phi\|$	64
Critic learning rate	-	$4.4{\cdot}10^{-4}$	τ embedding activation	-	Sigmoid
Discount factor	γ	0.98	Huber regression threshold	κ	1.0
Entropy target	$\bar{\mathcal{H}}$	-m = -3	**RUN-DSAC (Additional)**		
Batch size	$\|\mathcal{B}\|$	256	Initial uncertainty modulation factor	μ_{init}	1 or -1
Memory buffer size	$\|\mathcal{D}\|$	10^6	Uncertainty modulation decay horizon	N_μ	125
Non-linear activation	-	ReLU			
Polyak step size	ζ	0.995			
Policy regularization	$\mathcal{L}_T, \mathcal{L}_S,$	400			

Table 2: Evaluation scenarios for generalization capabilities and fault-tolerance.

Identifier	Scenario	Description
N0	Nominal	Trim condition consistent with training (h = 2000 m, V = 90 m/s).
G1	Wind Gust	15 ft/s vertical wind gust sustained for 3 seconds.
G2	Noisy Signal	Gaussian noise sensor models applied to the observations (Table 3).
G3	High Dynamic Pressure	Trim condition changed to h = 2000 m, V = 150 m/s.
G4	Low Dynamic Pressure	Trim condition changed to h = 10000 m, V = 90 m/s.
G5	Near-stall	The pitch angle reference θ_{ref} set to 40° at t = 35 s.
F1	Ice Accretion on Wings	Lift curve decreased by 30%, drag coefficient incremented by 0.06.
F2	Center of Gravity Shifted Aft	Center of gravity shifted aft by 0.25 m.
F3	Center of Gravity Shifted Forward	Center of gravity shifted forward by 0.25 m.
F4	Saturated Aileron	Aileron deflection constrained to $\pm 1°$.
F5	Saturated Elevator	Elevator deflection constrained to $\pm 2.5°$.
F6	Damaged Elevator	Elevator effectiveness coefficient decreased by 70%.
F7	Jammed Rudder	Rudder immobilized at a 15° deflection.

Table 3: Sensor models characterized by Gaussian noise distribution based on the PH-LAB aircraft in-flight data [29].

Signal	Unit	Noise (σ^2)	Bias
p, q, r	rad/s	$4.0{\times}10^{-7}$	$3.0{\times}10^{-5}$
θ, ϕ	rad	$1.0{\times}10^{-9}$	$4.0{\times}10^{-3}$
α	rad	$4.0{\times}10^{-10}$	-

The hyperparameters configured for the agents are listed in Table 1. The first column lists parameters refined through rigorous tuning, with $\bar{\mathcal{H}}$, ζ, $\mathcal{L}_T$, and $\mathcal{L}_S$ adopted from literature [17, 65, 56]. These proved optimal for all three agents. DSAC's additional parameters are inspired by Ma et al. [44], who also observed that employing *Sigmoid* over *ReLU* activation improves the smoothness of quantile fractions embedding. For RUN-DSAC, μ_{init} was selected such that uncertainty is effectively captured during learning, while N_μ was determined through iterative experimentation.

After training, the agents' *learning performance* is contrasted in terms of sample efficiency, return convergence, and consistency. Quantifying sample efficiency necessitates caution. Relying on a single metric, such as samples required to reach a certain return threshold, can lead to misleading interpretations as different threshold choices may lead to varied outcomes. To mitigate bias and ambiguity, a diverse set of metrics is employed, encompassing: average returns at the final episode (M1), average returns post-100 episodes (M2), sample count to achieve average returns of -2000 in filtered learning curves (M3), area between learning curves and the x-axis (M4), and the sample count required for the gradients of filtered learning curves to remain within ±30 (M5).

Additionally, the agents' policy robustness is assessed across various unseen flight scenarios, as detailed in Table 2. First, agents' *tracking performance* is evaluated under the baseline scenario $N0$, which replicates the training flight conditions and establishes a benchmark. Subsequent scenarios, inspired from literature [17], were selected for their relevance to flight attitude control, complexity, and compatibility with the DASMAT framework. *Generalization capability* scenarios (G) evaluate the agents' adaptability to untrained flight regimes, such as variations in dynamic pressure. From these, $G1$ simulates a wind gust with velocity derived from MIL-F-8785C standards [50], while $G2$ evaluates robustness to noisy sensors simulated by realistic sensor models as outlined in Table 3 [29]. *Fault tolerance* scenarios (F) probe fault-robustness by introducing variations in the controlled plant to simulate in-flight failures.

Each agent is evaluated on identical tasks, using extended episodes of $t_{\max}$ = 80 s with predefined cosine-smoothed step sequences in pitch and roll reference maneuvers, mirroring training signals. For a fair comparison of tracking proficiency across all controllers, the normalized mean absolute error (nMAE) metric is employed. This involves normalizing θ_e and ϕ_e tracking errors against the peak amplitudes of evaluation reference signals, and β_e is normalized the range of [-5°, 5°].

4 Results and Discussion

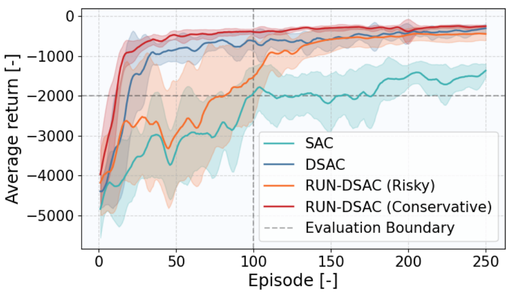
This section presents findings concerning learning efficiency, fault tolerance, and robustness under various flight conditions, assessed for SAC and three distributional agent variants: DSAC, *Conservative* RUN-DSAC, and *Risky* RUN-DSAC. Unless stated otherwise, both the mean and standard deviation of the results are calculated from ten independent evaluations. Line graphs feature a bold line for the average across policies and a shaded area for the standard deviation. In general, the SAC and RUN-DSAC agents are benchmarked against the baseline DSAC agent, where statistical significance in performance variation is assessed using a t-test, assuming normal distribution of samples.

4.1 Learning Performance

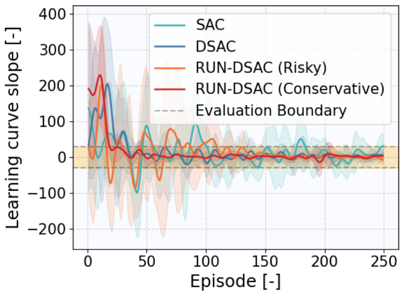
The learning curves for each policy are illustrated in Figure 8a. Distributional agents consistently converge to return levels in line with those documented in prior research [17, 56, 66]. Conversely, SAC underperforms, yielding returns over threefold lower than DSAC and demonstrating greater variance. These deficiencies are attributable to SAC's lower convergence success rate and pronounced sensitivity to stochastic processes like initialization, as noted by Dally and van Kampen [17]. Nevertheless, contrary to their selective reporting of outcomes from only converged SAC policies, this paper presents unfiltered results.

The learning curve of *Conservative* RUN-DSAC highlights the effectiveness of integrating Q-value uncertainty into decision-making for enhanced sample efficiency, marked by its rapid ascent. Furthermore, prioritizing predictable policies results in reduced variance in learning curves and the highest average converged returns, as indicated by *M1* in Table 4. This suggests that *Conservative* RUN-DSAC learned to track the training attitude reference signals more accurately and consistently compared to other agents. Comparative analysis in Table 4 also shows *Conservative* RUN-DSAC outperforming DSAC across all five learning performance metrics, with four displaying statistically significant differences. The largest improvement of 54.5% is observed in the *M5* metric, which is visualized in Figure 8b. While the *Conservative* RUN-DSAC's learning curve slope is the fastest to converge to the metric boundary, it also shows the least oscillations, which suggest a consistent and predictable learning performance that is highly desirable in the safety-focused flight control domain.

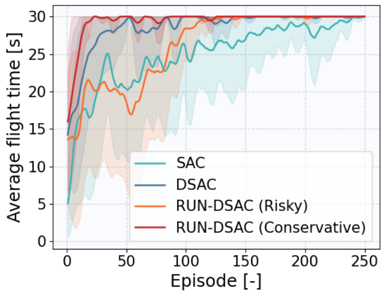
Conversely, *Risky* RUN-DSAC's slow convergence and initially high variance suggest that promoting uncertainty impedes learning performance. This variance arises from the algorithm's exploratory nature and the reward function design, which terminates the training flight prematurely and imposes high penalties when flight envelope is violated (Equation 30). Figure 8c compares the mean flight duration per episode, revealing that *Risky* RUN-DSAC initially exhibits more frequent and earlier violations, but achieves "crash-free" flight after approximately 125 episodes (coinciding with μ decay to 0). Conversely, *Conservative* RUN-DSAC achieves this stability in about 25 episodes, emphasizing its safety advantage. SAC trails significantly, requiring the entirety of 250 episodes to achieve crash-free flight. Finally, although this study focuses on offline training, the characteristics of *Conservative* RUN-DSAC render it potentially suitable for online flight control, particularly for in-flight fault management, where sample-efficient, predictable and safe learning is paramount.



(a) Learning curves comparison.



(b) Learning curve slopes comparison.



(c) Episodic flight time comparison.

Figure 8: Comparison of learning performance, with curves smoothed using a Gaussian kernel with a length of 10. Dashed lines and yellow shading indicate performance metric thresholds.

Table 4: Assessment of agent learning efficiency using four metrics shows mean scores with standard deviations, percent difference from DSAC, and p-values for significance (bold for $p < 0.05$). Green indicates improvement, red a decline.

		M1 (Final)	M2 (Returns)	M3 (# Episodes)	M4 (Area)	M5 (Gradient)
DSAC	Value	-308.1	-626.4 ± 254.4	16.8 ± 6.0	(20.8 ± 5.3)·10^4	183.2 ± 56.7
SAC	Value	**-1369.5±187.9**	**-1926.0 ± 676.4**	**69.1 ± 47.1**	**(59.0 ± 7.3)·10^4**	**230.9 ± 27.0**
	Difference	**-344.5%**	**-207.5%**	**+311.3%**	**+183.7%**	**+26.0%**
	p-value	**$1{\cdot}10^{-10}$**	**$2{\cdot}10^{-4}$**	**$9{\cdot}10^{-3}$**	**$6{\cdot}10^{-10}$**	**$4{\cdot}10^{-2}$**
RUN-DSAC (Risky)	Value	-447.7±167.0	**-1528.2 ± 998.8**	37.8 ± 32.8	**(36.0 ± 12.4)·10^4**	162.1 ± 48.0
	Difference	-45.3%	**-144.0%**	+125.0%	**+73.1%**	-11.5%
	p-value	$5{\cdot}10^{-2}$	**$2{\cdot}10^{-2}$**	$7{\cdot}10^{-2}$	**$3{\cdot}10^{-3}$**	$4{\cdot}10^{-1}$
RUN-DSAC (Conserv.)	Value	-253.1±28.0	**-398.5 ± 88.6**	**9.1 ± 4.0**	**(12.9 ± 1.6)·10^4**	**83.3 ± 55.1**
	Difference	+17.9%	**+36.4%**	**-45.8%**	**-37.9%**	**-54.5%**
	p-value	$2{\cdot}10^{-1}$	**$3{\cdot}10^{-2}$**	**$5{\cdot}10^{-3}$**	**$4{\cdot}10^{-4}$**	**$1{\cdot}10^{-3}$**

4.2 Tracking Performance

Learning curves provide an initial assessment of training effectiveness but do not fully represent control capabilities. Hence, the trained policies are evaluated against reference signals mirroring training conditions (Figures 9-12). The observed attitude tracking performance of the agents aligns with the learning curves presented in Figure 8a. SAC shows limited tracking accuracy with high variance in policies, reflected in a nMAE of 23.53±4.40%. This, along with large oscillations in its response, renders it aerodynamically and structurally unsuitable for flight control. In comparison, DSAC substantially outperforms SAC with smoother actions and a nearly four times lower nMAE of 5.95±1.15%, marking a significant improvement ($p = 2 \cdot 10^{-6} < 0.05$).

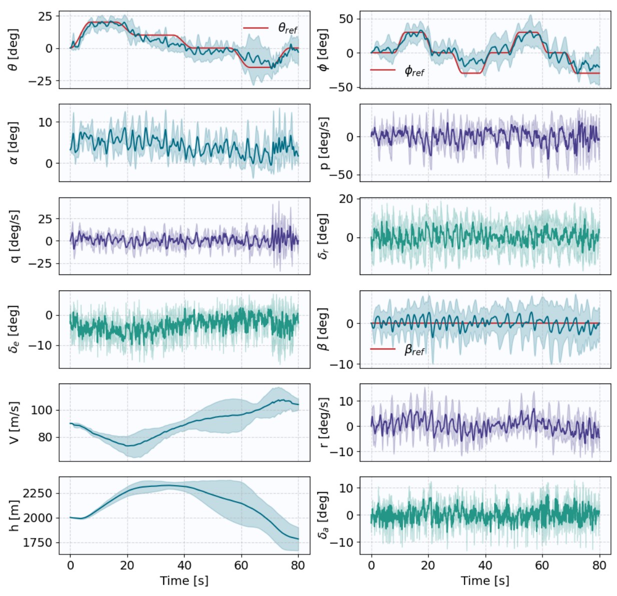


Figure 9: State time-traces of the SAC policy in the **nominal** scenario.

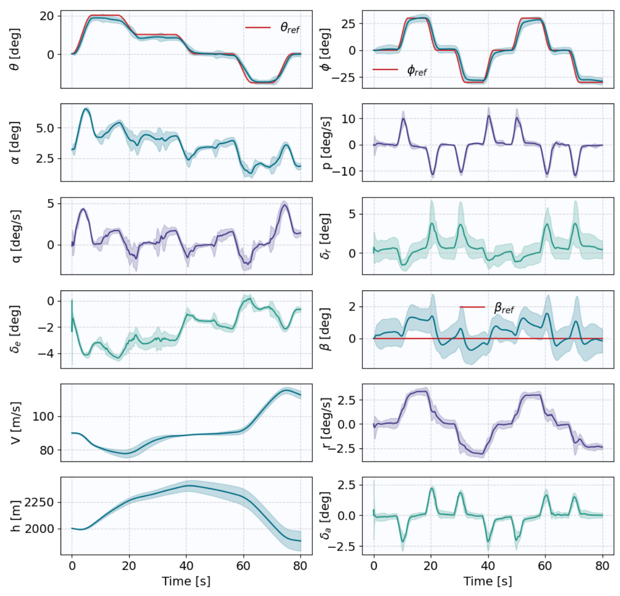


Figure 10: State time-traces of the DSAC policy in the **nominal** scenario.

Most notably, *Conservative* RUN-DSAC achieves the highest signal tracking accuracy and the least variability, evidenced by a reduction in nMAE to 3.95±0.42%, surpassing DSAC by 33% ($p = 4 \cdot 10^{-4} < 0.05$). This improved tracking and response predictability affirm the benefits of integrating uncertainty information into decision-making to enhance the efficiency and safety of intelligent flight controllers. Conversely, *Risky* RUN-DSAC shows higher tracking error and state variability, with a 57% higher nMAE of 9.36±2.48% compared to DSAC ($p = 3 \cdot 10^{-3} < 0.05$). These distinct results of the RUN-DSAC variants validate theoretical expectations: the *Conservative* RUN-DSAC's focus on predictability yields lower policy variability, while the exploration-driven *Risky* variant exhibits increased variability.

The distributional agents' state time-traces also reveal dynamic couplings between the principal axes, particularly where ϕ maneuvers influence θ and β tracking. The agents have adapted to exploit this coupling for enhanced control, reflected in the

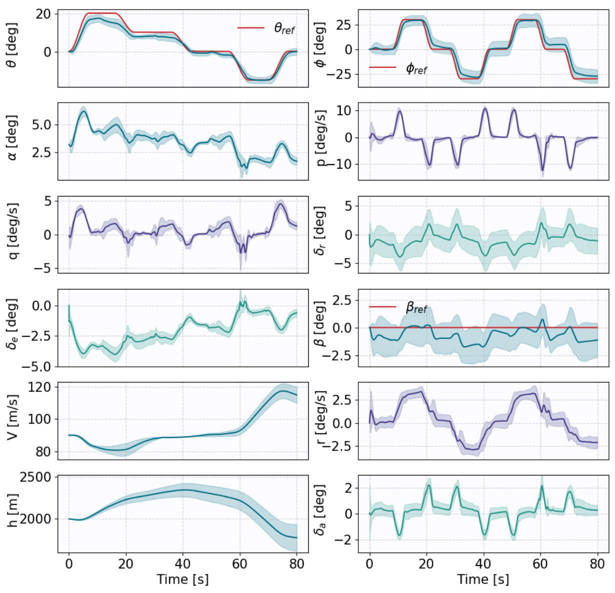


Figure 11: State time-traces of the *Risky* RUN-DSAC policy in the **nominal** scenario.

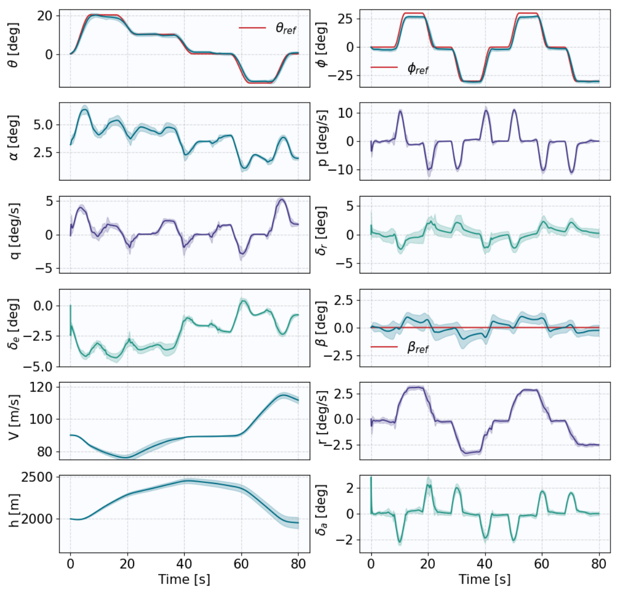


Figure 12: State time-traces of the *Conservative* RUN-DSAC policy in the **nominal** scenario.

coordinated control surface deflections. However, this learned coupling can lead to undesired outcomes when actuator inputs are correlated unintentionally, which is especially notable in the *Conservative* RUN-DSAC agent. Here, the agent's initial sharp elevator deflection (for trim adjustment) inadvertently triggers corresponding aileron and rudder deflections, despite zero roll and yaw reference. This causes transient roll and yaw rates that the agent swiftly neutralizes, yet a residual 2° roll offset and about 3° under-correction in the 30° ϕ_{ref} steps remain. In contrast, DSAC and *Risky* RUN-DSAC, less impacted by unintentional coupling, reach full roll reference but with compromised pitch precision.

The under-correction observed in *Conservative* RUN-DSAC is attributed to its inclination towards familiar, albeit more conservative, actions, thus favoring strategies with proven past efficacy even when they are not optimal for the given situation. Additionally, this agent might implicitly prioritize precise pitch control over roll due to the greater risks associated with pitch deviations, such as their impact on aerodynamic forces and stall potential. This explains its superior performance in pitch tracking compared to other distributional agents at the cost of under-correcting roll.

4.3 Generalization Capability

The preceding section highlights the *Conservative* RUN-DSAC's reliance on the confidence with pre-learned strategies. However, this potentially impairs its adaptability to novel scenarios, which inspires an evaluation of the agents' robustness in untrained flight conditions. While prior research demonstrated enhanced safety in a near-stall scenario through distorting DSAC returns [56], this study investigates the impact of embedding return uncertainty into policy training on stall robustness, and expands the scope with four additional case studies simulating unforeseen common flight conditions.

Figure 13 presents the agents' generalization performance, employing a separate ordinate for *G5* due to its higher nMAE scores and a distinct evaluation reference signal. In all scenarios, *Conservative* RUN-DSAC demonstrates the highest attitude tracking accuracy and the lowest result variance among agents, with statistical significance ($p < 0.05$). Conversely, the *Risky* variant underperforms compared to DSAC in all scenarios and exhibits higher variance (except in *G3*), with statistically significant differences in *G1* and *G2*. SAC, consistent with its learning curve shown in Figure 8a, records significantly the lowest performance and highest variance across policies.

The *Stall* (*G5*) scenario is selected to showcase the generalization capability of distributional agents in flight control. This scenario is particularly relevant due to the substantial uncertainty inherent in near-stall conditions, arising from the agents' lack of exposure to dynamics altered by unobserved airspeed and altitude states, or abrupt loss of lift. This uncertainty is directly correlated with increased flight safety risks, potentially leading to loss-of-control (LOC) incidents. The *G5* task entails a sustained high pitch-up maneuver to induce near-stall conditions, with zero roll and sideslip references. Figures 14 and 15 respectively illustrate the longitudinal response of distributional agents and analyze the average rewards and return distributions these agents experience during flight.

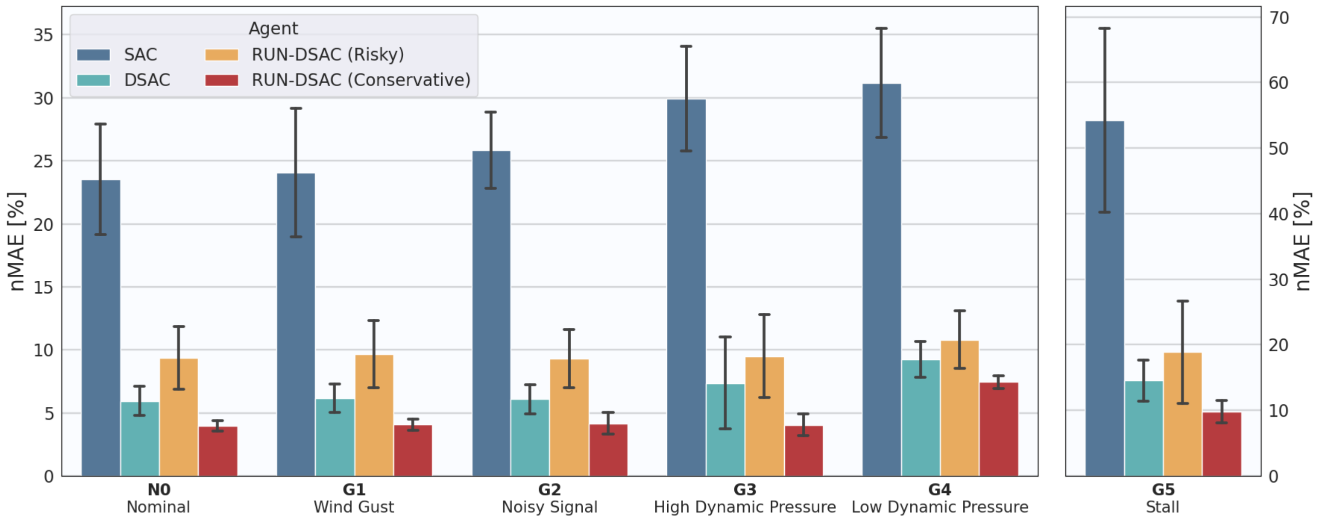


Figure 13: Attitude tracking nMAE and its standard deviation across various generalization scenarios.

High pitch angles significantly affect aircraft aerodynamics, resulting in decreased airspeed and increased altitude. At $t = 40\ s$, the aircraft is commanded to reach a $40°\ \theta_{\text{ref}}$, extending beyond the training conditions. As Figure 15 shows, this unfamiliar state is marked by a decline in expected returns and heightened variance, indicating increased uncertainty in agents' performance. Notably, the *Conservative* RUN-DSAC agent, with the highest expected returns and lowest variance, demonstrates the greatest confidence and tracking performance (reflected in the highest rewards) in this *Stall* scenario. Contrarily, although the *Risky* variant exhibits the lowest confidence based on its expected returns and variance, it outperforms DSAC in managing the high pitch angle reference at $t = 40\ s$, as evidenced by higher rewards. This suggests that while *Risky* RUN-DSAC's exploratory strategy is less effective in nominal conditions, it may be beneficial in unexpected scenarios.

Attempting to attain the high pitch angle reference, the agents encounter aerodynamic instabilities and stall-induced oscillations. Interestingly, the distributional agents display varying oscillation intensities in their responses: DSAC exhibits the most significant oscillations, followed by the *Conservative* RUN-DSAC, while *Risky* RUN-DSAC demonstrates the least oscillatory behavior. This pattern is consistent across other near-stall flight scenarios, including those affected by wind gusts (*G1*) or ice-accretion faults leading to maximal lift reduction (*F1*).

The oscillation differences among agents can be explained by correlating each agent's actor DNN to a *gain parameter* K in classical control theory. K influences a system's reaction to error signals: a high gain K triggers aggressive responses, potentially causing instability or oscillations, whereas a low gain K yields conservative reactions. *Risky* RUN-DSAC seems to adopt a policy with a higher K, enhancing responsiveness to counteract the diminished aerodynamic efficiency and damping under decreased dynamic pressure in near-stall scenarios (resulting from lower velocity and increased altitude). Conversely, DSAC, with a lower K, demonstrates less responsive behavior with larger, possibly delayed corrections. This results in inadequate error compensation, leading to more pronounced oscillations. *Conservative* RUN-DSAC exhibits intermediate characteristics.

To validate the proposed *gain analogy*, a similar response pattern is anticipated in the low dynamic pressure scenario (*G4*), whereas the opposite is expected under conditions of increased dynamic pressure (*G3*) due to enhanced aerodynamic effectiveness. In the latter case, the aggressive, high-K control of the *Risky* variant might exacerbate oscillations, while the low-K control of DSAC and *Conservative* RUN-DSAC could more effectively stabilize the system. This hypothesis is supported by Figure 16, which contrasts the pitch rate responses of the distributional agents under both low and high dynamic pressure conditions.

While the *gain analogy* partially explains the heightened oscillations of the *Conservative* RUN-DSAC relative to the *Risky* variant for low dynamic pressure scenarios, it falls short in explaining similar response disparities in scenarios with nominal dynamic pressure but altered aircraft dynamics (e.g., *F5* or *F7*). Hence, an alternative hypothesis is that the *Conservative* RUN-DSAC's focus on predictability may predispose it to overfit to training scenarios, leading to suboptimal and instability-prone policies under altered conditions. In contrast, the *Risky* RUN-DSAC's emphasis on uncertainty promotes exposure to a broader range of states and actions, bolstering robustness to novel scenarios and reducing instability susceptibility, albeit with trade-offs in tracking precision and increased policy variance. Supporting this hypothesis, a Gaussian kernel-smoothed density estimation of α distributions experienced by each agent during training is illustrated in Figure 17. This analysis reveals that the *Risky* RUN-DSAC has encountered a wider range of α, including extremes, which better prepares it for scenarios like $G5$. In

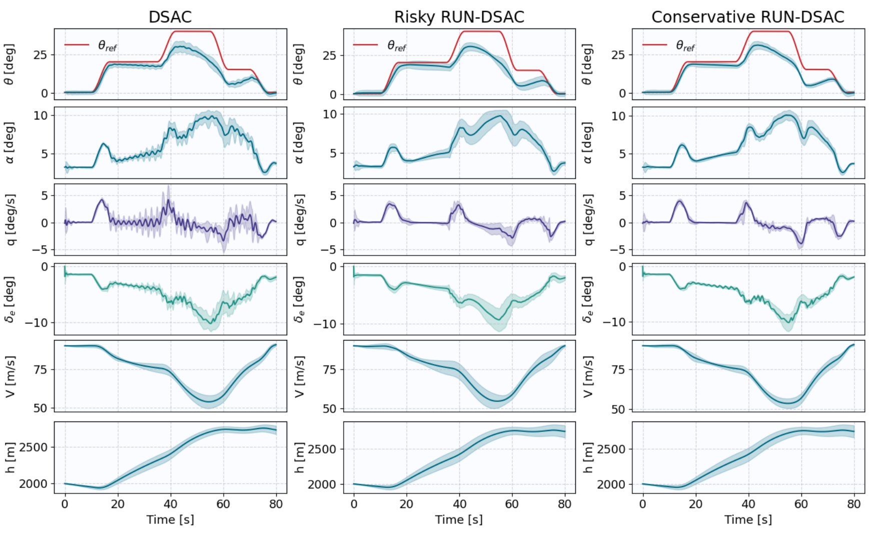


Figure 14: Longitudinal state time traces of DSAC (left), *Risky* RUN-DSAC (middle), and *Conservative* RUN-DSAC (right) policies in the **stall** scenario.

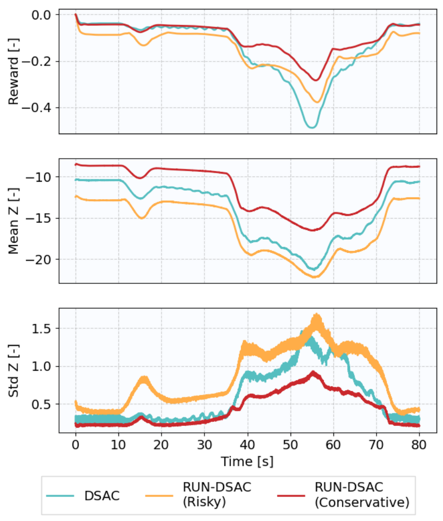


Figure 15: Average rewards, state means and variances collected by distributional agents during flight in the **stall** scenario.

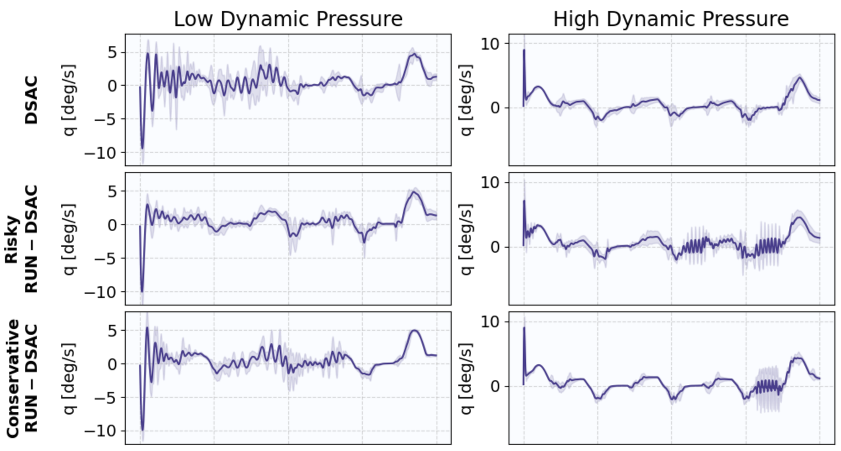


Figure 16: Pitch rate time traces for DSAC (top row), *Risky* RUN-DSAC (middle row), and *Conservative* RUN-DSAC (bottom row) for the **low dynamic pressure** (left column) and **high dynamic pressure** (right column) scenarios.

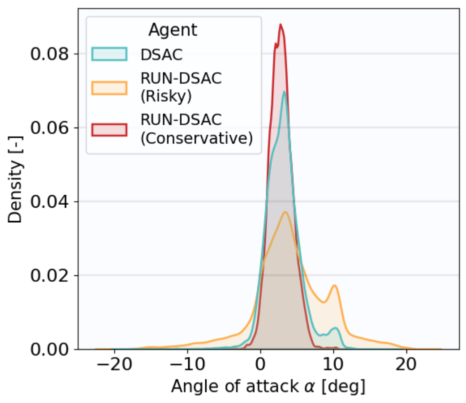


Figure 17: Kernel density estimation of α experienced by various distributional agents during a single training run.

contrast, the *Conservative* RUN-DSAC's experience is more restricted, with less frequent encounters of extremes than DSAC, underpinning its propensity to overfitting and increased vulnerability to instabilities in unforeseen scenarios.

4.4 Fault Tolerance

Assessing agents in diverse in-flight fault scenarios is imperative for ensuring safety, given the impracticality of modeling every conceivable failure for offline training. Figure 18 illustrates the performance of these agents in such evaluations. The *Conservative* RUN-DSAC consistently outperforms other agents in all seven fault scenarios, with statistically significant improvements over DSAC ($p < 0.05$) in all but *F7*. Conversely, the *Risky* variant underperforms compared to DSAC in every scenario, with statistically significant disparities in *F1*, *F3*, *F4*, and *F6*. SAC exhibits the lowest performance and highest policy variance.

Despite the superior tracking performance and reduced policy variance of *Conservative* RUN-DSAC, it was observed to exhibit heightened sensitivity to aircraft dynamic changes due to failures, especially those affecting control surface functionality, compared to the other distributional agents. Such sensitivity leads to marked disturbances in attitude states, contrasted by minimal oscillations in *Risky* RUN-DSAC, with DSAC exhibiting intermediate behavior. This reinforces the hypothesis of *Conservative* RUN-DSAC's overfitting tendency: policies refined during training become compromised under substantially altered dynamics.

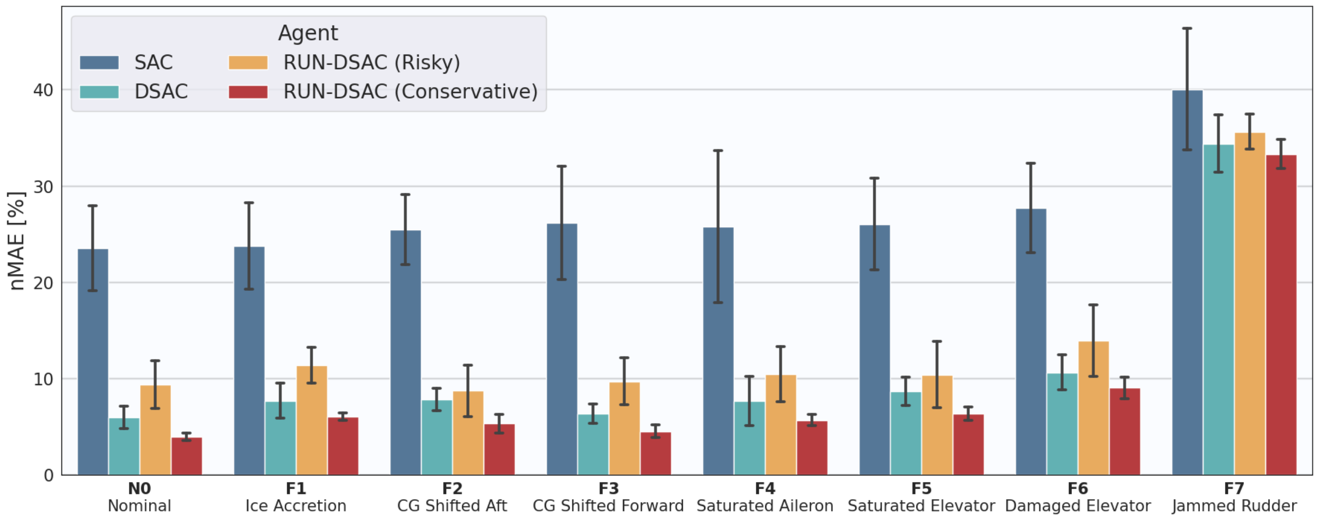


Figure 18: Attitude tracking nMAE and its standard deviation across various in-flight fault scenarios.

The disparity in disturbance patterns are particularly evident in the *F7* scenario, which is characterized by high nMAE scores due to persistent sideslip bias. Figures 19 and 20 compare the state trajectories of *Risky* respectively *Conservative* RUN-DSAC variants. Here, both agents unsuccessfully attempt to counteract the stuck rudder by commanding deflections in the opposite direction. However, the rudder control signal triggers aileron deflections due to the learned roll-yaw coupling, counterbalancing the negative roll induced by the dihedral effect. This stabilizes β at $\sim 10°$ and arrests further roll, but introduces a negative roll angle offset, subsequently impacting pitch tracking through pitch-roll coupling. This unusual flight dynamics causes pronounced oscillations in *Conservative* RUN-DSAC's response, in contrast to the minimal disturbances in *Risky* RUN-DSAC. Thus, *Conservative* RUN-DSAC's policies, while highly optimized for trained scenarios, prove inadequate under substantially altered control conditions.

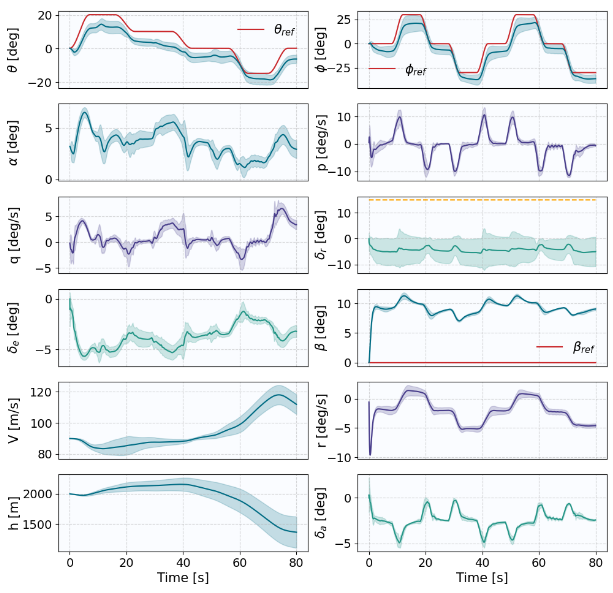


Figure 19: State time-traces of the *Risky* RUN-DSAC policy in the **immobilized rudder** scenario.

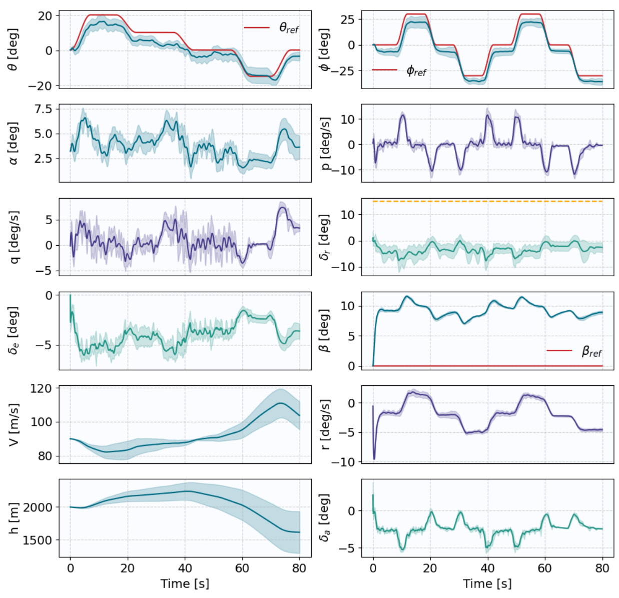


Figure 20: State time-traces of the *Conservative* RUN-DSAC policy in the **immobilized rudder** scenario.

5 Conclusion

This paper presents the Returns Uncertainty-Navigated Distributional Soft Actor-Critic (RUN-DSAC) algorithm to advance the model-free flight controllers for non-linear, fully-coupled aerospace dynamics. The integration of uncertainty quantification within the RL decision-making framework contributes to the synthesis of intelligent, adaptive, and safety-centric autonomous flight controllers.

We show that RUN-DSAC, particularly its *Conservative* variant, significantly enhances learning efficiency and attitude tracking accuracy in trained flight conditions over state-of-the-art algorithms like SAC and DSAC. The improvement is attributed to the algorithm's ability to leverage the full distribution of returns, prioritizing actions with minimal uncertainty. This approach leads to more predictable and stable policies, which are crucial in the safety-critical domain of aviation.

In scenarios involving unforeseen flight conditions and in-flight system failures, *Conservative* RUN-DSAC continues to showcase superior tracking performance. However, an inclination towards overfitting to training conditions is observed, causing increased sensitivity and heightened oscillations under significantly altered flight dynamics. In contrast, the *Risky* RUN-DSAC variant, which encourages exploration by favoring actions with higher uncertainty, displays robustness in novel and unexpected scenarios at the cost of reduced precision and increased variance in trained policies. This dichotomy between the *Conservative* and *Risky* variants of RUN-DSAC highlights a fundamental trade-off in RL-based flight control: the balance between the tracking accuracy and predictability of learned policies in familiar scenarios, and their adaptability and robustness in unforeseen circumstances.

In conclusion, while RUN-DSAC demonstrates significant potential in model-free flight control, its real-world applicability necessitates comprehensive validation. The reliability of these results is contingent on the simulation model's fidelity and the breadth of scenarios tested. Critical model parameters, particularly those governing aircraft dynamics and environmental interactions, may not fully capture the complex and variable conditions of actual flight. The performance of RUN-DSAC in more extreme or untested scenarios also remains uncertain, and any unforeseen interactions or emergent behaviors could pose substantial safety risks. Therefore, the algorithm's deployment in a real Cessna Citation II would require rigorous validation under a broader spectrum of conditions, more nuanced modeling of aircraft and sensor dynamics, and robust safeguards against the unpredictable nature of real-world aviation environments.

5.1 Significance of Contributions

Improving learning performance and robustness of offline RL algorithms by leveraging uncertainty quantification in the decision-making, this research stands as a valuable advancement in AI-driven model-independent aerospace control technologies. RUN-DSAC paves the way for efficient, robust, and adaptive intelligent flight controllers requiring minimal human-domain knowledge, contributing to bridging the simulation-to-reality gap. This work not only enhances the safety and reliability of fault-tolerant autonomous flight controllers but also sets a foundation for the application of AI in various safety-critical domains.

5.2 Recommendations

Online Flight Control: Given the promising attributes observed in *Conservative* RUN-DSAC, future research should explore its application in online flight control settings, assessing its real-time adaptability and effectiveness in dynamically adjusting to changing flight conditions and emergencies.

Hyperparameters optimization: Further optimization of RUN-DSAC's hyperparameters, particularly the newly introduced μ_{init} and N_μ, could enhance its performance. Adjusting μ_{init} to decay from a high positive value to a negative value across certain N_μ could balance necessary exploration for robust performance and conservatism for policy predictability.

6-DOF Flight Control: Leveraging the learning efficiency of *Conservative* RUN-DSAC, its application in full 6-DOF flight control is promising. A hierarchical approach, integrating uncertainty assessments at different control levels, could effectively tackle the *curse of dimensionality* inherent to high-dimensional control tasks.

Flight-Test Validation: With RUN-DSAC showing potential for robust real-system control, validation in rigorous flight tests is crucial to bridge the simulation-to-reality gap. Initial efforts should focus on employing techniques like domain randomization and robustness training in high-fidelity simulations, followed by trials on scaled-down models.

Application Beyond Flight Control: The successful application of RUN-DSAC in flight control suggests its potential in other complex domains, such as advanced Air Traffic Management. Its predictability and safety-centric approach could improve airspace efficiency by safely reducing aircraft separation standards.

References

[1] Prashanth L. A. and Mohammad Ghavamzadeh. *Variance-Constrained Actor-Critic Algorithms for Discounted and Average Reward MDPs*. arXiv:1403.6530 [cs, math, stat]. Mar. 2015. DOI: `10.48550/arXiv.1403.6530`. URL: `http://arxiv.org/abs/1403.6530` (visited on 11/03/2023).

[2] Paul Acquatella, Erik-Jan Van Kampen, and Q.P. Chu. *Incremental backstepping for robust nonlinear flight control*. Apr. 2013.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. July 2016. DOI: `10.48550/arXiv.1607.06450`. URL: `http://arxiv.org/abs/1607.06450` (visited on 05/13/2023).

[4] Alessandro Bacchini and E. Cestino. "Electric VTOL Configurations Comparison". In: *Aerospace* 6 (Feb. 2019), p. 26. DOI: `10.3390/aerospace6030026`.

[5] Marc G. Bellemare, Will Dabney, and Rémi Munos. *A Distributional Perspective on Reinforcement Learning*. July 2017. DOI: `10.48550/arXiv.1707.06887`. URL: `http://arxiv.org/abs/1707.06887` (visited on 04/08/2023).

[6] Marc G. Bellemare, Will Dabney, and Mark Rowland. *Distributional Reinforcement Learning*. MIT Press, 2023. URL: `http://www.distributional-rl.org`.

[7] Justus Benad. *The Flying V - A new Aircraft Configuration for Commercial Passenger Transport*. Nov. 2015. DOI: `10.25967/370094`.

[8] D.P. Bertsekas and J.N. Tsitsiklis. "Neuro-dynamic programming: an overview". In: *Proceedings of 1995 34th IEEE Conference on Decision and Control*. Vol. 1. ISSN: 0191-2216. Dec. 1995, 560–564 vol.1. DOI: `10.1109/CDC.1995.478953`. URL: `https://ieeexplore.ieee.org/document/478953` (visited on 10/12/2023).

[9] Jacob Buckman. *A Sober Look at Bayesian Neural Networks*. en. URL: `https://jacobbuckman.com/2020-01-17-a-sober-look-at-bayesian-neural-networks/` (visited on 05/24/2023).

[10] Ryan James Caverly et al. "Nonlinear Dynamic Inversion of a Flexible Aircraft". en. In: *IFAC-PapersOnLine*. 20th IFAC Symposium on Automatic Control in AerospaceACA 2016 49.17 (Jan. 2016), pp. 338–342. ISSN: 2405-8963. DOI: `10.1016/j.ifacol.2016.09.058`. URL: `https://www.sciencedirect.com/science/article/pii/S2405896316315300` (visited on 05/28/2023).

[11] Alex J. Chan and Mihaela van der Schaar. *Scalable Bayesian Inverse Reinforcement Learning*. Mar. 2021. DOI: `10.48550/arXiv.2102.06483`. URL: `http://arxiv.org/abs/2102.06483` (visited on 05/31/2023).

[12] Dongyan Chen. "Reliable H control for uncertain affine nonlinear systems". In: *Proceedings of the 29th Chinese Control Conference*. July 2010, pp. 1933–1938.

[13] Rafael A. Cordeiro, José R. Azinheira, and Alexandra Moutinho. "Robustness of Incremental Backstepping Flight Controllers: The Boeing 747 Case Study". In: *IEEE Transactions on Aerospace and Electronic Systems* 57.5 (Oct. 2021), pp. 3492–3505. ISSN: 1557-9603. DOI: `10.1109/TAES.2021.3082663`.

[14] G.C.H.E. de Croon et al. "Design, Aerodynamics, and Vision-Based Control of the DelFly". en. In: *International Journal of Micro Air Vehicles* 1.2 (June 2009), pp. 71–97. ISSN: 1756-8293. DOI: `10.1260/175682909789498288`. URL: `https://doi.org/10.1260/175682909789498288` (visited on 05/28/2023).

[15] Will Dabney et al. *Distributional Reinforcement Learning with Quantile Regression*. Oct. 2017. DOI: `10.48550/arXiv.1710.10044`. URL: `http://arxiv.org/abs/1710.10044` (visited on 04/08/2023).

[16] Will Dabney et al. *Implicit Quantile Networks for Distributional Reinforcement Learning*. June 2018. DOI: `10.48550/arXiv.1806.06923`. URL: `http://arxiv.org/abs/1806.06923` (visited on 04/08/2023).

[17] Killian Dally and Erik-Jan van Kampen. "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control". In: *AIAA SCITECH 2022 Forum*. Jan. 2022. DOI: `10.2514/6.2022-2078`. URL: `http://arxiv.org/abs/2202.09262` (visited on 05/15/2023).

[18] G.C.H.E. De Croon et al. *The DelFly*. en. Dordrecht: Springer Netherlands, 2016. ISBN: 978-94-017-9208-0. DOI: `10.1007/978-94-017-9208-0`. URL: `http://link.springer.com/10.1007/978-94-017-9208-0` (visited on 05/28/2023).

[19] Dotan Di Castro, Aviv Tamar, and Shie Mannor. *Policy Gradients with Variance Related Risk Criteria*. arXiv:1206.6404 [cs, math, stat]. June 2012. DOI: `10.48550/arXiv.1206.6404`. URL: `http://arxiv.org/abs/1206.6404` (visited on 11/03/2023).

[20] Jingliang Duan et al. "Distributional Soft Actor-Critic: Off-Policy Reinforcement Learning for Addressing Value Estimation Errors". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.11 (Nov. 2022). arXiv:2001.02811 [cs, eess], pp. 6584–6598. ISSN: 2162-237X, 2162-2388. DOI: `10.1109/TNNLS.2021.3082568`. URL: `http://arxiv.org/abs/2001.02811` (visited on 10/20/2023).

[21] Francesco Faggiano et al. *Aerodynamic Design of a Flying V Aircraft*. June 2017. DOI: `10.2514/6.2017-3589`.

[22] Rick Feith. "Safe Reinforcement Learning in Flight Control: Introduction to Safe Incremental Dual Heuristic Programming". en. In: (2020). URL: `https://repository.tudelft.nl/islandora/object/uuid%3A07f53daa-b236-4bb9-b010-bc6654383744` (visited on 05/29/2023).

[23] *Final Report: Accident to Airbus A330-203 registered F-GZCP, Air France AF 447 Rio de Janeiro - Paris, 1st June 2009 \textbar AAIU.ie*. URL: `http://www.aaiu.ie/node/687` (visited on 05/30/2023).

[24] Vincent Fortuin et al. *Bayesian Neural Network Priors Revisited*. Mar. 2022. DOI: `10.48550/arXiv.2102.06571`. URL: `http://arxiv.org/abs/2102.06571` (visited on 05/25/2023).

[25] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. Oct. 2018. DOI: 10.48550/arXiv.1802.09477. URL: http://arxiv.org/abs/1802.09477 (visited on 05/17/2023).

[26] Giulia Gatti. "Shielded reinforcement learning for flight control". en. In: (2023). URL: https://repository.tudelft.nl/islandora/object/uuid%3A1c4f10ef-c279-4969-8943-41c46e6b2d8f (visited on 05/30/2023).

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. en-US. Publication Title: MIT Press. Nov. 2016. URL: https://mitpress.mit.edu/9780262035613/deep-learning/ (visited on 05/30/2023).

[28] P.C. Gregory and United States Air Force Wright Air Development Center. *Proceedings of the Self Adaptive Flight Control Systems Symposium, Wright Air Development Center, 13 and 14 Jan., 1959*. WADC technical report. WADC, 1959. URL: https://books.google.nl/books?id=0COgnQEACAAJ.

[29] Fabian Grondman et al. "Design and flight testing of incremental nonlinear dynamic inversion based control laws for a passenger aircraft". en. In: *AIAA Guidance, Navigation, and Control* 210039 (2018). Publisher: American Institute of Aeronautics and Astronautics Inc. (AIAA). DOI: 10.2514/6.2018-0385. URL: https://repository.tudelft.nl/islandora/object/uuid%3A964e1942-5c83-45e7-8ace-507a33ea3146 (visited on 10/23/2023).

[30] Shangding Gu et al. *A Review of Safe Reinforcement Learning: Methods, Theory and Applications*. arXiv:2205.10330 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2205.10330. URL: http://arxiv.org/abs/2205.10330 (visited on 09/30/2023).

[31] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. Jan. 2019. DOI: 10.48550/arXiv.1812.05905. URL: http://arxiv.org/abs/1812.05905 (visited on 05/18/2023).

[32] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. Aug. 2018. DOI: 10.48550/arXiv.1801.01290. URL: http://arxiv.org/abs/1801.01290 (visited on 05/18/2023).

[33] Stefan Heyer. "Reinforcement Learning for Flight Control: Learning to Fly the PH-LAB". en. In: (2019). URL: https://repository.tudelft.nl/islandora/object/uuid%3Adc63cae7-4289-47c7-889e-253f7abd7c72 (visited on 05/20/2023).

[34] 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". en. In: *Advances in Aerospace Guidance, Navigation and Control*. Ed. by Bogusław Dołęga et al. Cham: Springer International Publishing, 2018, pp. 259–277. ISBN: 978-3-319-65283-2. DOI: 10.1007/978-3-319-65283-2_14.

[35] Jonathan Hoogvliet. "Hierarchical Reinforcement Learning for Model-Free Flight Control: A sample efficient tabular approach using Q(lambda)-learning and options in a traditional flight control structure". en. In: (2019). URL: https://repository.tudelft.nl/islandora/object/uuid%3Ad66efdb7-d7c7-4c44-9b50-64678ffdf60d (visited on 05/19/2023).

[36] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics* 35.1 (Mar. 1964), pp. 73–101. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177703732. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-35/issue-1/Robust-Estimation-of-a-Location-Parameter/10.1214/aoms/1177703732.full (visited on 05/11/2023).

[37] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. "Hierarchical Reinforcement Learning: A Survey and Open Research Challenges". en. In: *Machine Learning and Knowledge Extraction* 4.1 (Mar. 2022). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, pp. 172–221. ISSN: 2504-4990. DOI: 10.3390/make4010009. URL: https://www.mdpi.com/2504-4990/4/1/9 (visited on 09/30/2023).

[38] Dmitry I. Ignatyev, Hyo-Sang Shin, and Antonios Tsourdos. "Two-layer adaptive augmentation for incremental backstepping flight control of transport aircraft in uncertain conditions". In: *Aerospace Science and Technology* 105 (Oct. 2020), p. 106051. ISSN: 1270-9638. DOI: 10.1016/j.ast.2020.106051. URL: https://www.sciencedirect.com/science/article/pii/S1270963820307331 (visited on 09/30/2023).

[39] P.P. Khargonekar, I.R. Petersen, and K. Zhou. "Robust stabilization of uncertain linear systems: quadratic stabilizability and H/sup infinity / control theory". In: *IEEE Transactions on Automatic Control* 35.3 (Mar. 1990), pp. 356–361. ISSN: 1558-2523. DOI: 10.1109/9.50357.

[40] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 05/13/2023).

[41] Ramesh Konatala, Erik-Jan Van Kampen, and Gertjan Looye. "Reinforcement Learning based Online Adaptive Flight Control for the Cessna Citation II(PH-LAB) Aircraft". In: *AIAA Scitech 2021 Forum*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2021. DOI: 10.2514/6.2021-0883. URL: https://arc.aiaa.org/doi/10.2514/6.2021-0883 (visited on 11/13/2023).

[42] Yu Li et al. “Angular acceleration estimation-based incremental nonlinear dynamic inversion for robust flight control”. en. In: *Control Engineering Practice* 117 (Dec. 2021), p. 104938. ISSN: 0967-0661. DOI: 10.1016/j.conengprac.2021.104938. URL: https://www.sciencedirect.com/science/article/pii/S096706612100215X (visited on 05/28/2023).

[43] Cheng Liu, Erik-Jan van Kampen, and Guido C. H. E. de Croon. *Adaptive Risk-Tendency: Nano Drone Navigation in Cluttered Environments with Distributional Reinforcement Learning*. Sept. 2022. DOI: 10.48550/arXiv.2203.14749. URL: http://arxiv.org/abs/2203.14749 (visited on 04/12/2023).

[44] Xiaoteng Ma et al. *DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning*. June 2020. DOI: 10.48550/arXiv.2004.14547. URL: http://arxiv.org/abs/2004.14547 (visited on 05/31/2023).

[45] By James McCaffrey and 08/16/2021. *Wasserstein Distance Using C# and Python -*. en-US. Publication Title: Visual Studio Magazine. URL: https://visualstudiomagazine.com/articles/2021/08/16/wasserstein-distance.aspx (visited on 05/31/2023).

[46] W. Thomas Miller, Richard S. Sutton, and Paul J. Werbos. “A Menu of Designs for Reinforcement Learning Over Time”. In: *Neural Networks for Control*. Conference Name: Neural Networks for Control. MIT Press, 1995, pp. 67–95. URL: https://ieeexplore.ieee.org/document/6300641 (visited on 10/13/2023).

[47] Tigran Mkhoyan et al. “Morphing wing design using integrated and distributed trailing edge morphing”. In: *Smart Materials and Structures* 31 (Nov. 2022). DOI: 10.1088/1361-665X/aca18b.

[48] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. en. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236 (visited on 03/03/2023).

[49] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. Dec. 2013. DOI: 10.48550/arXiv.1312.5602. URL: http://arxiv.org/abs/1312.5602 (visited on 03/08/2023).

[50] David J. Moorhouse and Robert J. Woodcock. *Background Information and User Guide for MIL-F-8785C, Military Specification - Flying Qualities of Piloted Airplanes*. en. Tech. rep. Section: Technical Reports. Air Force Wright Aeronautical Laboratories, July 1982. URL: https://apps.dtic.mil/sti/citations/ADA119421 (visited on 11/20/2023).

[51] Eugene A. Morelli and Vladislav Klein. *Aircraft System Identification: Theory and Practice*. English. 1st edition. Reston, VA: American Institute of Aeronautics & Astronautics, Sept. 2006. ISBN: 978-1-56347-832-1.

[52] Siddharth Mysore et al. *Regularizing Action Policies for Smooth Control with Reinforcement Learning*. May 2021. DOI: 10.48550/arXiv.2012.06644. URL: http://arxiv.org/abs/2012.06644 (visited on 08/22/2023).

[53] Ibai Roman et al. “Evolution of Gaussian Process kernels for machine translation post-editing effort estimation”. en. In: *Annals of Mathematics and Artificial Intelligence* 89.8 (Sept. 2021), pp. 835–856. ISSN: 1573-7470. DOI: 10.1007/s10472-021-09751-5. URL: https://doi.org/10.1007/s10472-021-09751-5 (visited on 05/24/2023).

[54] John Schulman et al. *Proximal Policy Optimization Algorithms*. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347 (visited on 03/09/2023).

[55] John Schulman et al. *Trust Region Policy Optimization*. Apr. 2017. DOI: 10.48550/arXiv.1502.05477. URL: http://arxiv.org/abs/1502.05477 (visited on 05/15/2023).

[56] Peter Seres. “Distributional Reinforcement Learning for Flight Control: A risk-sensitive approach to aircraft attitude control using Distributional RL”. en. In: (2022). URL: https://repository.tudelft.nl/islandora/object/uuid%3A6cd3efd1-b755-4b04-8b9b-93f9dabb6108 (visited on 05/30/2023).

[57] S. Sieberling, Q. P. Chu, and J. A. Mulder. “Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction”. In: *Journal of Guidance, Control, and Dynamics* 33.6 (Nov. 2010), pp. 1732–1742. ISSN: 0731-5090. DOI: 10.2514/1.49978. URL: https://arc.aiaa.org/doi/10.2514/1.49978 (visited on 05/28/2023).

[58] David Silver et al. “Deterministic Policy Gradient Algorithms”. en. In: *Proceedings of the 31st International Conference on Machine Learning*. ISSN: 1938-7228. PMLR, Jan. 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html (visited on 10/12/2023).

[59] Sigurd Skogestad and I Postlethwaite. “Multivariable Feedback Control: Analysis and Design”. In: vol. 2. Jan. 2005.

[60] Ewoud J. J. Smeur, Qiping Chu, and Guido C. H. E. de Croon. “Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles”. In: *Journal of Guidance, Control, and Dynamics* 39.3 (Mar. 2016), pp. 450–461. ISSN: 0731-5090. DOI: 10.2514/1.G001490. URL: https://arc.aiaa.org/doi/10.2514/1.G001490 (visited on 05/28/2023).

[61] Matthew J. Sobel. “The variance of discounted Markov decision processes”. en. In: *Journal of Applied Probability* 19.4 (Dec. 1982). Publisher: Cambridge University Press, pp. 794–802. ISSN: 0021-9002, 1475-6072. DOI: 10.2307/3213832. URL: https://www.cambridge.org/core/journals/journal-of-applied-probability/article/abs/variance-of-discounted-markov-decision-processes/AA4549BFA70081B27C0092F4BF9C661A (visited on 11/03/2023).

[62] Brian L. Stevens, Frank L. Lewis, and Eric N. Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. en. John Wiley & Sons, Oct. 2015. ISBN: 978-1-118-87097-6.

[63] 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 (Feb. 2021), pp. 116–130. ISSN: 1941-0468. DOI: 10.1109/TRO.2020.3010626.

[64] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 978-0-262-03924-6.

[65] Casper Teirlinck. “Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance”. en. In: (2022). URL: https://repository.tudelft.nl/islandora/object/uuid%3Adae2fdae-50a5-4941-a49f-41c25bea8a85 (visited on 05/31/2023).

[66] Lucas Vieira dos Santos. “Safe & Intelligent Control: Fault-tolerant Flight Control with Distributional and Hybrid Reinforcement Learning using DSAC and IDHP”. en. In: (2023). URL: https://repository.tudelft.nl/islandora/object/uuid%3A10f5fa68-f934-414a-9067-988f51f098cb (visited on 10/01/2023).

[67] Shaun S. Wang. “A Class of Distortion Operators for Pricing Financial and Insurance Risks”. In: *The Journal of Risk and Insurance* 67.1 (2000). Publisher: [American Risk and Insurance Association, Wiley], pp. 15–36. ISSN: 0022-4367. DOI: 10.2307/253675. URL: https://www.jstor.org/stable/253675 (visited on 10/20/2023).

[68] Xuerui Wang. *Flexible Aircraft Gust Load Alleviation with Incremental Nonlinear Dynamic Inversion*. en-us. Feb. 2019. URL: https://xueruiwangtud.github.io/publication/flexiblegla/ (visited on 11/20/2023).

[69] Xuerui Wang. *Nonlinear Incremental Control for Flexible Aircraft Trajectory Tracking and Load Alleviation*. en-us. Aug. 2021. URL: https://xueruiwangtud.github.io/publication/glider/ (visited on 11/20/2023).

[70] Terrence A. Weisshaar. “Morphing Aircraft Systems: Historical Perspectives and Future Challenges”. In: *Journal of Aircraft* 50.2 (Mar. 2013), pp. 337–353. ISSN: 0021-8669. DOI: 10.2514/1.C031456. URL: https://arc.aiaa.org/doi/10.2514/1.C031456 (visited on 05/28/2023).

[71] Lyu Ximin et al. “Design and implementation of a quadrotor tail-sitter VTOL UAV”. In: May 2017. DOI: 10.1109/ICRA.2017.7989452.

A RUN-DSAC Pseudocode

The RUN-DSAC is summarized in algorithm 1. First, the replay buffer $\mathcal{D}$, network parameters and the temperature parameter α_T are initialized. Furthermore, initial (pre-training) $\langle s, a, r, s', d\rangle$ transitions are collected using a randomly initialized policy and stored in $\mathcal{D}$. The algorithm is run for N_{eps} episodes, each executing $N_{\text{transitions}}$ transitions using the current policy $\pi_w(\cdot|s)$.

Algorithm 1 RUN-DSAC algorithm

```
Hyperparameters: $N_q$, $|\mathcal{B}|$, $\zeta$, $\bar{\mathcal{H}}$, $\lambda_S$, $\lambda_T$, $\sigma$, $\kappa$, $\mu_{init}$, $\mu_{fin}$, $N_{RUN}$
Input: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\rangle$, $N_{eps}$, $N_{trans}$
Result: $Z_{\theta_k}(s,a)$, $\pi_w(s)$
Initialize replay buffer $\mathcal{D}$
Initialize actor and critic parameters $\theta_1$, $\theta_2$, $w$, target parameters $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$, $\bar{w} \leftarrow w$
Initialize temperature $\alpha_T$
Collect initial trajectories $\mathcal{T}$ and store in the replay buffer $\mathcal{D}$
for $episode \leftarrow 1$ to $N_{eps}$ do
    for $transition \leftarrow 0$ to $N_{trans}$ do
        Collect a $\langle s, a, r, s', d\rangle$ transition using $\pi_w(\cdot|s)$
        Sample a mini-batch $\mathcal{B} \overset{i.i.d.}{\sim} \mathcal{D}$
        Generate quantile fractions $\tau_i, i = 0, \dots, N_q$, $\tau_j, j = 0, \dots, N_q$
        Sample $a' \sim \pi_{\bar{w}}(\cdot|s')$
        for $i \leftarrow 0$ to $N_q - 1$ do
            $\hat{\tau}_i \leftarrow \frac{\tau_i + \tau_{i+1}}{2}$
            for $j \leftarrow 0$ to $N_q - 1$ do
                $\hat{\tau}_j \leftarrow \frac{\tau_j + \tau_{j+1}}{2}$
                $y_i \leftarrow \min_{k=1,2} Z_{\hat{\tau}_i,\bar{\theta}}(s', a')$
                $\delta_{ij}^k \leftarrow r + \gamma\,[y_i - \alpha_T \log \pi_{\bar{w}}(a'|s')] - Z_{\hat{\tau}_j,\theta_k}(s,a), k = 1,2$
            end
        end
        $J_Z(\theta_k) \leftarrow \frac{1}{N_q}\sum_{i=0}^{N_q-1}\sum_{j=0}^{N_q-1}(\tau_{i+1} - \tau_i)\rho_{\hat{\tau}_j}^{\kappa}(\delta_{ij}^k), k = 1,2$
        Update $\theta_k$ with $\nabla J_Z(\theta_k)$, $k = 1,2$ (using ADAM)
        Update $\bar{\theta}_k \leftarrow \zeta\theta_k + (1-\zeta)\bar{\theta}_k$, $k = 1,2$
        Sample new actions $\tilde{a} \sim \pi_w(\cdot|s)$ with reparametrization trick
        $Q(s,\tilde{a}) \leftarrow \sum_{i=0}^{N-1}(\tau_{i+1} - \tau_i)\,\min_{k=1,2} Z_{\hat{\tau}_i,\theta_k}(s,\tilde{a})$
        $\sigma_Q \leftarrow \sqrt{\sum_{i=0}^{N-1}(\tau_{i+1} - \tau_i)\,[Z_{\hat{\tau}_i,\theta}(s,a) - Q(s,a)]^2}$
        $\mu \leftarrow \max(\mu_{init} - (\mu_{init} - \mu_{fin})\frac{episode}{N_{RUN}}, 0)$
        $\mathcal{L}_T \leftarrow \|\pi_w(s) - \pi_w(s')\|_2$ $\mathcal{L}_S \leftarrow \|\pi_w(s) - \pi_w(\tilde{s})\|_2$ with $\tilde{s} \sim \mathcal{N}(s,\sigma)$
        $J_\pi(w) \leftarrow \alpha_T \log(\pi_w(\tilde{a}|s)) - (Q(s,\tilde{a}) + \mu\sigma_Q) + \lambda_T\mathcal{L}_T + \lambda_S\mathcal{L}_S$
        Update $w$ with $\nabla J_\pi(w)$ (using ADAM)
        Update $\bar{w} \leftarrow \zeta w + (1-\zeta)\bar{w}$
        $J_{\mathcal{H}}(\alpha_T) \leftarrow \alpha_T\bar{\mathcal{H}} - \alpha_T \log(\pi_w(\tilde{a}|s))$
        Update $\alpha_T$ with $\nabla J_{\mathcal{H}}(\alpha_T)$ (using ADAM)
    end
    Store the collected trajectories $\mathcal{T}$ in the replay buffer $\mathcal{D}$
end
```

For critic network training, two quantile fraction sets are generated, one for the current and the other for the target value distribution estimates. New actions are then sampled from the target policy $\pi_{\bar{w}}(\cdot|s')$, followed by the computation of the temporal difference error δ_{ij}^k for each quantile fraction pair. The critic loss function $J_Z(\theta_k)$ is subsequently evaluated and used to update the critic networks parameters. The target parameters are softly updated with a mixing coefficient ζ. Next, reparametrized action samples $\tilde{a}$ are drawn to calculate the expected return $Q(s,\tilde{a})$ and the return variance σ_Q is derived. Additionally, the uncertainty modulation factor μ is linearly decayed, and the smoothness losses $\mathcal{L}_T$ and $\mathcal{L}_S$ are computed. These components are used to formulate the policy loss function $J_\pi(w)$ and its gradient $\nabla J_\pi(w)$ is used to update the policy network parameters w and to softly update the target parameters $\bar{w}$. Furthermore, the algorithm adaptively adjusts the temperature parameter α_T by minimizing a loss function designed to align the current policy's entropy with a predefined target level $\bar{\mathcal{H}}$. Finally, the newly acquired $\langle s, a, r, s', d\rangle$ transitions, collectively known as trajectories, are added to the replay buffer $\mathcal{D}$.

Part II

Literature Study and Preliminary Analysis

*The following part has been evaluated as a constituent of the *AE4020 Literature Study* course.

2 Fundamentals of Reinforcement Learning

Before diving into the core of this research, an explanation of the foundational aspects of RL is necessary. Therefore, this chapter aims to explain the fundamental components of RL. First, the insights into the RL origins is discussed in Section 2.1. Next, Section 2.2 provides a comprehensive overview of the fundamental concepts and techniques in RL. Finally, Section 2.3 will present how these RL concepts can be combined with artificial neural networks to generate RL algorithms that are able to handle high dimensionality.

2.1. Origins of Reinforcement Learning

The roots of RL can be traced back to three major threads, which came together in 1980s - (1) learning by trial and error, (2) optimal control approached with value functions and dynamic programming, and (3) temporal difference (TD) learning, which interrelated the former two threads to some extent [22]. The theory behind the *trial-and-error learning* stems from the psychology of animal learning. Thorndike's 1898 *Law of Effect* highlighted that behaviors followed by satisfying consequences are more likely to be repeated, while those followed by discomforting consequences are less likely to be repeated. The strength of the association between the behavior and the situation depends on the magnitude of the satisfaction or discomfort experienced. Skinner's 1938 theory of *operant conditioning* builds on Thorndike's work, emphasizing that behavior is influenced by its consequences and that reinforcement is a powerful tool for shaping behavior [29]. Moreover, in 1948, Alan Turing proposed the idea of implementing trial-and-error learning in a computer through a ”pleasure-pain system” [30]. Turing's system involves a computer making a random choice when faced with an unknown situation. If the result is positive (pleasurable), it is saved as a permanent solution, but if it is negative (painful), the choice is discarded. This idea by Turing is regarded as one of the earliest and most influential concepts in the realm of AI.

The *optimal control* thread refers to designing a controller to optimize the performance of a dynamical system over time. Richard Bellman and others approached this class of problems by combining the concepts of a system state and a value function to form the Bellman Equation [31]. The methods that involve solving this equation to solve optimal control problems became known as Dynamic Programming (DP). DP suffers from ”the curse of dimensionality”, meaning its computational requirements grow exponentially with the number of state variables. Despite this, DP is still considered the only feasible way of solving general stochastic optimal control problems. Connections between optimal control, DP, and learning were slow to be recognized, but they are now extensively developed by many researchers [22].

Temporal difference (TD) learning, which is inspired by the concept of *secondary reinforcers* from the animal psychology, is a unique approach to RL that involves estimating a certain quantity in a temporally successive manner. Arthur Samuel was the first to implement this learning method in his checkers-playing program [32]. Klopf brought the ideas of trial-and-error learning and TD learning together in 1972 with the generalized reinforcement principle. Sutton, Barto and Anderson further developed these ideas and created *the actor-critic architecture* [33], and in 1988, they separated TD learning from control, treating it as a general prediction method [34]. Finally, the earliest publication of a TD learning rule was discovered in a paper by Ian Witten in 1977 [35]. He proposed a specific TD learning algorithm, known as TD(0), which updates the expected reward of an action based on the difference between the observed reward and the expected reward of the next state.

2.2. Primer on Reinforcement Learning

Reinforcement learning refers to a bio-inspired machine learning method whereby an agent learns to make optimal decisions in an uncertain and dynamic environment [22]. The agent's task is to learn a mapping between states of the environment and actions that maximize a scalar reward signal. The agent's goal is to maximize the cumulative reward over time.

In RL, the learner does not have prior knowledge or information about the consequences of taking an action in a particular state. Instead, it interacts with the environment by taking actions, observing the resulting state, and receiving a reward signal. The learner must use this feedback to iteratively update its behavior until it can identify the actions that lead to the highest reward. By recursively iterating on its behavior, the learner can eventually converge to an optimal policy, which specifies the best action to take in each state. This process involves a trade-off between exploration and exploitation, where the learner must explore different actions to learn about the environment, while also exploiting its current knowledge to maximize reward [22]. These and more concepts will be elaborated on in the following sections. The theory found in this chapter is primarily based on the perspective of the RL proposed by Sutton and Barto [22].

2.2.1. Elements of Reinforcement Learning

RL is a distinct form of machine learning as it involves direct interaction between *agent* and an *environment*, which is not present in supervised nor unsupervised learning approaches. At each time-step t, the agent makes a decision to execute an *action* $a_t \in \mathcal{A}$(s) on the environment, and receives feedback in the form of a change in the environment from the current state $s_t \in \mathcal{S}$ to another state $s_{t+1} \in \mathcal{S}$ and a reward $r_t \in \mathcal{R}(s_t, a_t, s_{t+1})$ (hence $\mathcal{R} : \mathcal{S}$ x $\mathcal{A}$ x $\mathcal{S} \rightarrow \mathbb{R}$). These are then used to update the *policy* π, which is a mapping from the current environment observation to the actions to be taken. This interaction between an agent and environment is visualized in Figure 2.1, which depicts a one-step transition $\mathbb{T}_t = \langle s,\ a,\ r,\ s_{t+1} \rangle$. Note that another common notation found in literature uses the symbol s for the current state and s' for a future state.

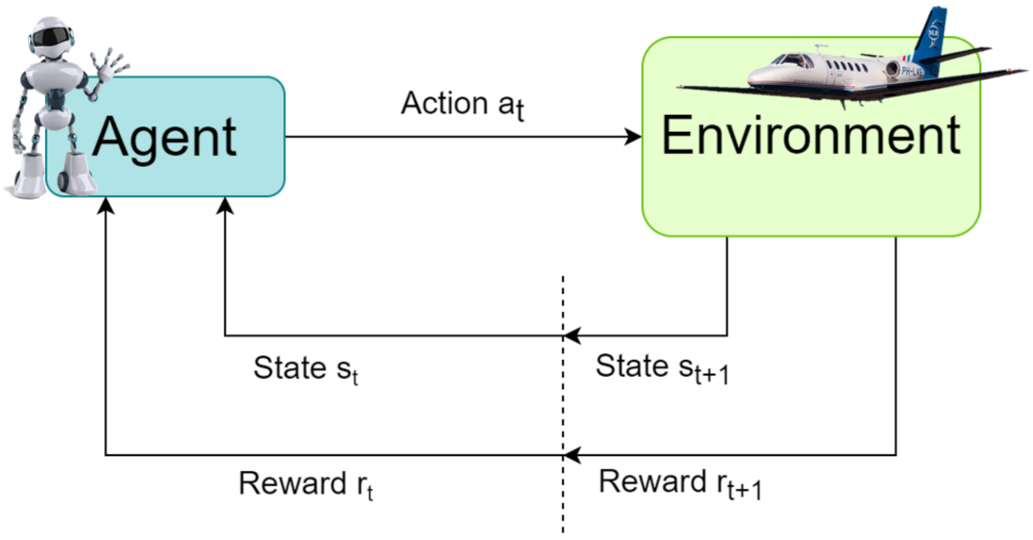


Figure 2.1: The agent-environment interaction in RL

An agent in RL consists of several components, including a *policy* π (deterministic $a = \pi(s)$ or stochastic $a \sim \pi(a|s)$ mapping from state to action), a *value function* $V(s)$ (describing the expected total reward for specific states), and possibly a model representation of the environment.

2.2.2. Policy and Trajectory

As mentioned, the policy can be understood as a control law that the agent follows to determine the next actions. Following this definition, a trajectory $\mathcal{T}$ is a sequence of n state-action pairs that an agent experiences as it follows a policy, formally given in Equation 2.1.

$$\mathcal{T}_n = (s_0, a_0, s_1, a_1, ..., s_{n-1}, a_{n-1}) \tag{2.1}$$

2.2.3. Cumulative Reward

The rewards shall be provided in such a way that maximizing them leads to the agent achieving the desired goal. The reward signal should indicate what is to be accomplished, not how to achieve it. As noted previously, the value of a reward depends on the current state s_t, the action taken a_t and the resulting state s_{t+1}. However, the agent's goal is not to maximize this immediate reward, but the cumulative reward in the long run. The cumulative reward (or "return") R_t over a trajectory starting at a time signature t is given by Equation 2.2. Moreover, since the flight control task researched in this work is an example of a *continuing task*, i.e. a task that goes on continuously without a limit, the discount rate γ (with $0 \leq \gamma \leq 1$) was introduced to prevent the cumulative reward from growing towards infinity.

$$R_t = \lim_{n\to\infty}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^n r_{t+n}) = \lim_{n\to\infty}\sum_{k=0}^{n}\gamma^k r_{t+k+1} \tag{2.2}$$

The discount rate γ is used to determine the importance of future rewards relative to immediate rewards. For example, a value close to 0 leads to a *myopic* agent, as in the agent is concerned mainly with maximizing the immediate rewards. In general, prioritizing actions that result in immediate rewards can diminish opportunities to attain potentially better future rewards, leading to a decrease in overall return. In contrast, a discount rate close to 1 signifies a more farsighted agent, for which future outcomes are given more weight.

Bounding γ to $0 \leq \gamma < 1$ ensures that the series in Equation 2.2 is convergent, as $\lim_{n\to\infty}\gamma^n r_{t+n} \to 0$. The opposite of continuing tasks are *episodic* tasks where the cumulative reward does not approach infinity as the number of time steps for every episode is finite, thus the discount rate can be set to 1. Nevertheless, episodic tasks with long episodes still commonly make use of discounting [36].

2.2.4. Markov Decision Process

At the fundamental level, every method employed in RL is based on the mathematical postulate of a fully-observable Markov Decision Process (MDP). MDPs represent a classical formulation of sequential decision-making, in which actions exert an impact on not only immediate rewards, but also on subsequent states, and consequently on future rewards. Before continuing with the discussion, it is important to define the state-transition probability function $\mathcal{P}(s'|s,a)$, which determines the probability of transitioning into a new state s' from the current state s by executing action a.

Within the realm of optimal control, a system is deemed to be a Markov Decision Process (MDP) if and only if it adheres to the Markov property, whereby the immediate transition relies exclusively on the latest state and action. Essentially, the information about the history of transitions is entirely captured in the present state and it can be used to make future choices, as described by Equation 2.3.

$$\mathcal{P}\{s_{t+1}, r_{t+1}|s_t, a_t\} = \mathcal{P}\{s_{t+1}, r_{t+1}|s_t, a_t, ..., s_0, a_0\} \tag{2.3}$$

If the agent is unable to accurately determine its full state based solely on observation, the system is partially observable. The MDP can be generalized to such cases by defining Partially Observable MDP (POMDP). Nevertheless, this chapter will continue to focus on the fully observable cases. Hence, the full MDP can be characterized by a structured set $\mathcal{M} = \langle\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R},\gamma\rangle$. Figure 2.2 combines all the concepts defined thus far by depicting a sequential MDP following a certain policy $\pi(a|s)$.

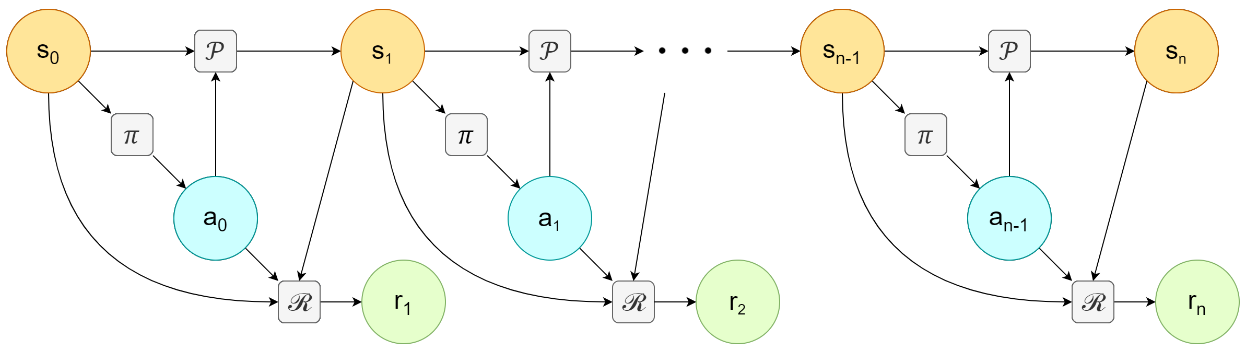


Figure 2.2: The sequential MDP following a policy $\pi(a|s)$

.

2.2.5. Value Functions

A state value function is defined in Equation 2.4. It determines the expected return obtained by choosing actions according to a certain policy π as we start in a certain state s.

$$V^{\pi}(s) = \mathbb{E}\left[R_t \mid s_t = s,\ a_t = a\right] \tag{2.4}$$

Alternatively, a state-action value function can be defined with Equation 2.5. This function determines the expected return obtained when the agent starts in a certain state s, executes a certain first action a, and then proceeds to act according to policy π. Hence, the difference between state-value function and the state-action value function is that the state value function represents the value of being in a particular state s, whereas the state-action value function represents the value of taking a particular action a in state s.

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_t \mid s_t = s,\ a_t = a\right] \tag{2.5}$$

These functions can then be used to identify an optimal policy, as by following this policy, the expected (discounted) return R_t is maximized. As a results, the optimal value functions $V^*(s)$ and $Q^*(s,a)$ can be established, as given in Equation 2.6 and Equation 2.7, respectively.

$$V^*(s) = \mathrm{max}_{\pi}\mathbb{E}\left[R_t \mid s_t = s,\ a_t = a\right] \tag{2.6}$$

$$Q^*(s,a) = \mathrm{max}_{\pi}\mathbb{E}\left[R_t \mid s_t = s,\ a_t = a\right] \tag{2.7}$$

Moreover, once the optimal policy for each state s and action a pairs is known from the Q^* function, the optimal action $a^*(s)$ can be determined, as given in Equation 2.8. Performing this action in state s leads to the maximum (discounted) return, hence it is called a *greedy action*.

$$a^*(s) = \mathrm{argmax}_a\, Q^*(s,a) \tag{2.8}$$

2.2.6. Bellman Equation

Realizing that both Equation 2.6 and Equation 2.7 can be defined recursively, the *Bellman Expectation Equation* defines the state value function (see Equation 2.9) and the state-action value function (see Equation 2.10) as the expected sum of the immediate reward and the discounted value of the expected next state, while following the optimal policy π^*.

$$V^*(s) = E_{s'\sim\mathcal{P}}[r(s,\pi^*(s) + \gamma V^*(s')] \tag{2.9}$$

$$Q^*(s,a) = E_{s'\sim\mathcal{P}}[r(s,a) + \gamma\mathrm{max}_{a'}\, Q^*(s',a')] \tag{2.10}$$

Richard Bellman's discovery of the recursive consistency allowed for the determination of optimal policies for MDP and has become a cornerstone of all RL algorithms [22].

2.2.7. Temporal Difference Learning

Before introducing the temporal difference update, it is essential to mention that more classes of methods exist. In fact, Markov Decision Processes can be solved by three distinct approaches - *dynamic programming* (DP), *Monte Carlo methods* (MC) and *temporal difference learning* (TD). All of these methods use the Bellman equation as a fundamental principle to iteratively update value functions or policies, but they differ in their approach to estimating the values and their reliance on a model of the MDP. For example, DP and TD learn at each time step during the episode, whereas MC methods wait until the end of the episode to update the value function or policy. Furthermore, DP differs from TD methods due to its reliance on a state-transition model, while TD is model-free [22]. The MC methods can be applied in both model-based and model-free architectures. Nevertheless, TD learning forms the basis for many of the algorithms discussed in this chapter, hence this subsection will focus on this approach.

TD combines ideas from both DP and MC methods. It enables agents to learn through bootstrapping, which means that they iteratively use past estimates or observations as a basis for further estimation. The principle of the TD update is to evaluate the difference between the target and the current estimate of the value function and to update the current estimate based on this error (scaled by a learning rate α_{TD}). This principle is summarized for the value function V and the action-value function Q in Equation 2.11 and Equation 2.12, respectively. The mentioned TD error is captured in the square brackets. Finally, the ultimate target values are the optimal value functions V^* and Q^*, respectively.

$$V(s_t) \leftarrow V(s_t) + \alpha_{TD}\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{2.11}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_{TD} \left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{2.12}$$

In discrete settings, such update rules are used by Q-learning, which learns the Q function, or by SARSA, which learns the V function [22]. These methods focus on learning the value function rather than directly learning a policy, while the policy is derived by selecting the action with the highest value for a given state. Similarly, they are used by actor-critic methods in continuous settings, which are further elaborated on in Section 3.2.

2.2.8. Exploration vs Exploitation

The trade-off between *exploration* and *exploitation* is another aspect of RL, where choosing to explore may lead to greater long-term rewards despite short-term losses. With no exploration, the algorithm simply chooses to perform the action with the highest estimated value $Q_t(a)$. In other words, such algorithm always exploits the current knowledge to maximize the immediate reward. This *greedy* action selection method is formulated in Equation 2.13.

$$a_t \doteq \arg\max_a Q_t(a) \tag{2.13}$$

The trade-off between exploration and exploitation is a fundamental aspect of RL. While choosing to exploit the current knowledge and select the action with the highest estimated value can lead to immediate rewards, exploration plays a crucial role in uncovering potentially better actions in the long run. One common approach is the ϵ-greedy method, where the algorithm mostly selects the greedy action but occasionally explores by choosing a random action with a small probability ϵ. Exploration is especially important during the early stages of learning when the agent's knowledge of the environment is limited. It also proves beneficial in noisy environments where rewards may fluctuate. Even in deterministic cases, exploration remains advantageous if the environment is non-stationary [22], ensuring that potentially superior actions are not overlooked.

RL identifies two types of methods to handle the exploration-exploitation trade-off, and any algorithm is based on either one. The essential concepts underlying these two approaches are outlined below:

- **On-policy learning** refers to algorithms that learn from the policy they use to interact with the environment. In simpler words, the agent modifies its policy based on the experiences it gains while following that same policy. This type of learning is usually employed when the objective is to enhance the existing policy.
- **Off-policy learning** refers to algorithms that gain knowledge from another policy than the one used to interact with the environment. Specifically, the agent modifies its policy based on the experiences garnered by following a different policy. Off-policy learning is typically employed when the objective is to learn about different policies or when there is a need to estimate the value of different policies.

On-policy RL algorithms update the policy with the *policy gradient update* rule, which involves the computation of the gradient of a performance metric, such as the expected cumulative reward, with respect to the policy parameters. Subsequently, this gradient is used to determine the direction of update for the policy to improve its performance. Examples of on-policy algorithms include Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) [37]. These concepts will be discussed in more detail in Section 3.1.

Off-policy RL algorithms utilize replay buffers to store a diverse range of past experiences, allowing the model to learn from them. These stored experiences are sampled from the *behavior* policy, which is responsible for the agent's actions during its interaction with the environment. Furthermore, importance weights are used to adjust for any differences between the behavior policy and the *target* policy, which represents the policy the agent aims to learn. In short, the replay buffer enables off-policy learning in RL by allowing to learn from experiences collected from any policy [38]. Section 3.2 will provide more detail about these methods.

2.2.9. Model-based vs. Model-free Reinforcement Learning

Finally, it is important to make a distinction between *model-based* and *model-free* RL methods. *Model-based* reinforcement learning involves building a model of the environment to plan and make decisions, while *model-free* reinforcement learning directly learns the policy or value function from interactions with the environment. Model-based methods rely on accurate modeling but can offer efficient exploration, while model-free methods are more flexible but may require more interactions to converge. The choice depends on the problem and available resources.

2.3. Deep Reinforcement Learning

Traditional RL techniques like SARSA are suitable for small problems featuring discrete state and action spaces. However, in real-world continuous scenarios, generalization is necessary due to the large and complex state and action spaces. Learning the value of each state-action pair individually is impractical and often infeasible (due to both, time and memory constraints). Technically, the continuous state and actions spaces could be discretized, but the resulting discretization can introduce errors. Therefore, for an adequate approximation, a very fine discretization would be required. However, the resulting number of state-action pairs increases exponentially due to *the curse of dimensionality*. To overcome this challenge, recent advancements in other machine learning methods can be leveraged, including deep neural networks used as function approximators. This integration of Deep Learning with RL is referred to as Deep RL.

Different function approximation techniques can be employed to generalize the value function and/or the policy, including radial basis functions, fuzzy logic systems, least-squares linear regressors, and artificial neural networks (ANN) [39]. Among these, ANNs have become the modern state-of-the-art in RL for function approximations due to their generalization power in high-dimensional control tasks and differentiability, which enables the use of gradient descent methods in estimating their parameters θ [39]. Therefore, this work will focus on this type of function approximation.

2.3.1. Artificial Neural Network

An ANN is a black-box function approximator that mimics the structure and function of biological neurons. ANNs consist of interconnected nodes, called neurons, that are organized into layers. Each neuron receives input from other neurons, performs a computation, and produces an output that is transmitted to other neurons. Furthermore, the connections between neurons are strengthened by weights and the outputs are produced by activation functions, imitating the firing behaviour of biological neurons. This gives ANNs the ability to learn complex, non-linear relationships between inputs and outputs. A single neuron is visualized in Figure 2.3, where $w_k \in \mathbb{R}^m$ represent the weights. To enable the activation of a neuron to be shifted by a constant, an extra bias parameter b is used. Furthermore, an ANN with neurons organized in different layers is depicted in Figure 2.4. Using many hidden layers renders ANN a Deep Neural Network (DNN).

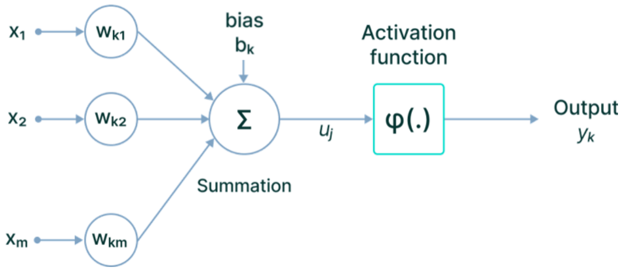


Figure 2.3: A single neuron [40].

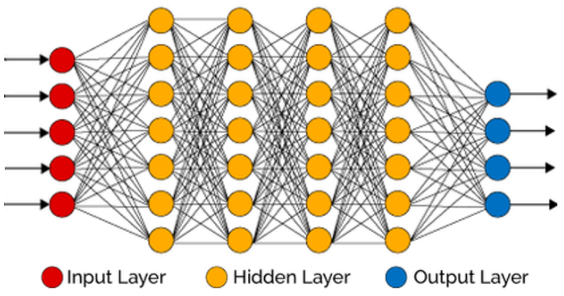


Figure 2.4: An Artificial Neural Network (ANN) [41].

2.3.2. Gradient Descent

During training, an ANN is presented with a set of input-output pairs and the weights of its neuron connections are adjusted in order to minimize the difference between the predicted output and the actual output. This process is often accomplished using an optimization algorithm, such as gradient descent. The gradient of an objective function $J(\theta)$ (with θ being the parameters to be optimized) is defined by Equation 2.14.

$$\nabla_\theta J(\theta) = \left[\frac{\partial J(\theta)}{\theta_1}, ..., \frac{\partial J(\theta)}{\theta_n}\right]^T \tag{2.14}$$

For an ANN, the objective function is usually formulated in terms of loss $\mathcal{L}(\theta)$, as given by Equation 2.15. In essence, the total loss function employed for ANN training is the summation of individual losses across all data samples. The function $f_\theta(x) = \hat{y}$ in this equation represents the ANN approximation using an input vector x, while $F(x) = y$ are the true output values. In contrast to supervised learning, RL techniques lack access to true labels, and instead learn from labeled samples y obtained by interacting with the environment. Finally, $|B|$ denotes the number of input-output pairs in a *batch* B, which is used to compute the loss.

$$J(\theta) = \mathcal{L}(\theta) = \frac{1}{|B|}\sum_{x\in B}(F(x_t) - f_\theta(x_t)) \tag{2.15}$$

After the parameters are initialized randomly with small values (to avoid saturation of activation functions) or by using alternative initialization algorithms, such as Xavier initialization [42], they are trained via gradient descent. The objective of the gradient descent is to find a local minimum of $\mathcal{L}(\theta)$. A fast and memory-efficient method to compute the gradient is through the use of the *backpropagation algorithm*, which recursively applies the chain rule of calculus to compute the derivative of the loss function with respect to each layer of the neural network. The obtained gradient is then used to optimize the parameters according to Equation 2.16.

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{|B|} \sum_{i=1}^{|B|} \nabla_\theta \mathcal{L}(\theta) \tag{2.16}$$

Here, α is a hyperparameter specifying the learning rate and t denotes the current epoch (i.e. the iteration step). The learning rate α needs to be chosen carefully as setting it too low results in too slow learning, whereas setting it too high might cause divergence. Often, α is scheduled to decay with the learning progress to guarantee improved convergence characteristics.

Modern ANN architectures utilize advanced *stochastic gradient descent* (SGD) methods, such as Adam [43], which utilize stochastic gradient methods along with additional enhancements such as *root mean square propagation* (RMSP) and *gradient descent with momentum*, to achieve better performance. In order to apply these automated SGD methods in practice for RL, it is necessary for the model to be differentiable with respect to the parameter vector.

2.3.3. Activation Functions

Activation functions are used in ANNs to introduce non-linearity into the output of a neuron. Some of commonly used activation functions are summarized in Equation 2.17 to Equation 2.20. For example, *Rectified Linear Unit* (ReLU, see Equation 2.17) passes the positive components of its input. It became very popular due to its computational efficiency and the ability to avoid and rectify vanishing gradient problems [44]. However, it suffers from ”dying ReLU” as the slope is 0 for negative inputs, which impedes learning. Therefore, variations of ReLU such as Leaky ReLU (Equation 2.18) were introduced to allows a small, non-zero output for negative inputs. The Sigmoidal function (Equation 2.19) is used to map any input to a value between 0 and 1, making it useful for binary classification problems. Alternatively, the output of Softmax function (Equation 2.20) is used to transform a vector of arbitrary real values into a probability distribution in order to represent the likelihoods of different categories. This is because each of its output vector components $y_i \in [0, 1]$ and their total sum is $\sum_i y_i = 1$ [45]. Hence, this particular activation function might be useful for the probabilistic RL approaches discussed in this work.

$$\text{ReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{2.17}$$

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$

$$\text{Leaky ReLU}(x) = \begin{cases} 0.01x & x \leq 0 \\ x & x > 0 \end{cases} \tag{2.18}$$

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \quad x \in \mathbb{R}^k \tag{2.20}$$

2.3.4. Layer Architectures

Numerous layer types exist, where each type has a specific purpose. For example, *Fully-connected linear layers* are the simplest type of layer, in which every neuron in one layer is connected to every neuron in the next layer. They are widely used as they are easily differentiable, simple and flexible, allowing them to be easily applied to a variety of input shapes and sizes.

Convolutional layers are designed to apply a sliding window (called a *kernel*) over the input data to extract local features. They are particularly useful in image processing and natural language processing (NLP) tasks, where there are spatial relationships between the inputs [46]. Closely related *pooling layers* are used to reduce the size of feature maps by summarizing sub-regions (such as sub-sampling image data in convolutional networks) [47]. The most common type of pooling layer is max-pooling, which outputs the maximum value of a sub-region.

Recurrent layers are used to handle sequential data by allowing information to persist across time steps. These layers are commonly used in NLP tasks, such as language translation and speech recognition [48]. Long Short-Term Memory (LSTM) is a popular type of recurrent layer that is designed to learn long-term dependencies in the data [49].

Normalization layers standardize the inputs to a layer (by correcting the mean and variance of each hidden unit) to improve training stability and speed [50]. Batch normalization is a commonly used normalization technique that normalizes the inputs to a layer over a batch of samples.

For an RL-based flight controller, a relatively shallow feed-forward ANN with fully-connected linear (possibly normalized) layers is believed to be the most suitable architecture, as it provides effective approximation for non-linear action-value models. Furthermore, the benefits of other layers, such as those used for processing large data like images, are not relevant for the flight controller's data size and requirements, as the flight controllers discussed in this work are not based on processing visual or long time-sequences data. Also, recurrent neural networks are constrained in RL due to the Markov Process assumption. Finally, one more ANN architecture will be discussed, which is particularly applicable for approximating probabilistic functions.

2.3.5. Unconstrained Monotonic Neural Networks

Functions expressing the probability functions, such as *CDF* and *QF* (inverse of CDF), are strictly monotonic. Therefore, when ANNs are used to approximate such functions, the monotonicity shall be preserved. However, this is not always guaranteed. For example, the non-monotonically increasing QF in [28] leads to an ill-defined return distribution, thereby potentially deteriorating the performance of the RL algorithm. Therefore, novel *Unconstrained Monotonic NN* (UMNN) are proposed as a solution [51]. A parametric continuous monotonic function $G(\cdot;\theta):\mathbb{R}\to\mathbb{R}$ is defined by UMNN as given in Equation 2.21.

$$G(x|c;\theta) = \int_0^x g(t,c;\theta_g)dt + \beta(c;\theta_\beta) \tag{2.21}$$

Here $g(\cdot;\theta):\mathbb{R}\to\mathbb{R}^+$ is a free-form ANN with positive outputs (enforced by an appropriate activation function, such as ReLU), $\beta\in\mathbb{R}$ is another ANN, c is a conditioning variable (composed of, for example, states s and actions a), and $\theta=\theta_g\cup\theta_\beta$ are trainable parameters. This structure is visualized in Figure 2.5.

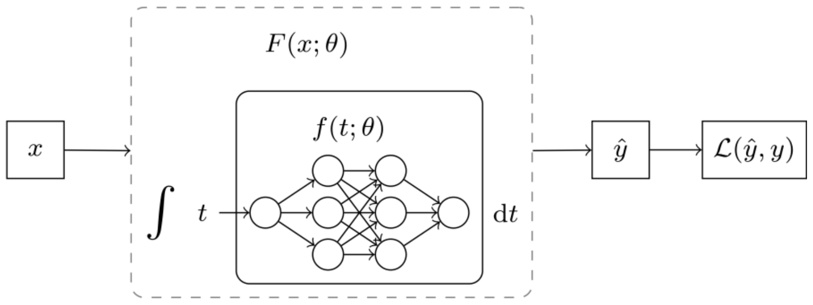


Figure 2.5: Unconstrainned Monotonic Neural Network, adopted from [51].

The forward evaluation of $G(x|c;\theta)$ requires solving an integral, which can be efficiently approximated numerically using Clenshaw-Curtis quadrature. To train the neural network g, it is necessary to calculate the gradient of G with respect to its parameters. Although this gradient could be obtained by back-propagating through the integral solver, doing so would lead to a memory usage that increases proportionally with the number of integration steps. Alternatively, the derivative of an integral with respect to θ can be expressed with the Leibniz integral rule, which results in Equation 2.22.

$$\begin{aligned}\nabla_\theta G(x|c;\theta) &= g(x|c;\theta)\nabla_\theta(x|c) - g(0;\theta)\nabla_\theta(0) + \int_0^x \nabla_\theta g(t;\theta)dt + \nabla_\theta\beta(c;\theta)\\ &= \int_0^x \nabla_\theta g(t;\theta)dt + \nabla_\theta\beta(c;\theta)\end{aligned} \tag{2.22}$$

The UMNN was applied in the UMDQN Distributional RL algorithm to improve its prediction of the random return distribution [52].

2.4. Conclusion

This chapter has provided a comprehensive overview of the origins, concepts, and advancements in the field of RL. One of its key contributions was the exploration of how RL methods can be effectively extended to tackle high-dimensional tasks through the utilization of ANNs. This understanding of RL and its integration with ANNs provides sufficient knowledge to delve into the state-of-the-art methods in RL and their applications in flight control settings, which will be discussed in Chapter 3.

3 State-of-the-art Deep RL Methods

This chapter explores the most prominent state-of-the-art Deep RL algorithms. Considering the continuous flight control task at hand, the focus is put specifically on algorithms designed for continuous state and action spaces. Many of these algorithms are captured in the RL taxonomy presented in Figure 3.1. However, it must be noted that this tree-like structure is by far non-exhaustive, which is highlighted by an empty branch. Furthermore, many alternative taxonomies exist [53, 54].

Model-based methods learn an explicit model of the environment dynamics to make informed decisions and optimize the agent's policy. Two particularly interesting algorithms from this category are the AlphaZero and the MuzRTO, which popularized deep RL by excelling in complex strategic games like GO, Shogy, and Starcraft3. To ensure safety in-flight, the RL-based flight controller developed in this work shall be as robust as possible to faults or other unexpected scenarios. However, it is impractical to create models for every possible failure [27]. Therefore, model-based algorithms are not considered further.

The objective of this work is to develop a sample efficient algorithm though uncertainty estimation. For this purpose, the off-policy methods appear the most suitable, because they are generally more sample efficient as they can reuse past experiences or data collected from different policies [55]. This allows them to learn from a wider range of experiences and make more efficient use of available data. Furthermore, leveraging past data and exploring a wider range of actions, off-policy methods can better quantify the uncertainty associated with their chosen actions. Hence, while a brief mention will be given to on-policy algorithms in Section 3.1 (as concepts from these algorithms are applied in actor-critic approaches), the subsequent subsections will primarily emphasize off-policy methods in Section 3.2. Furthermore, the application of these Deep RL algorithms for flight control will be discussed in Section 3.3. Therefore, this chapter aims to address **RQ-M-1**.

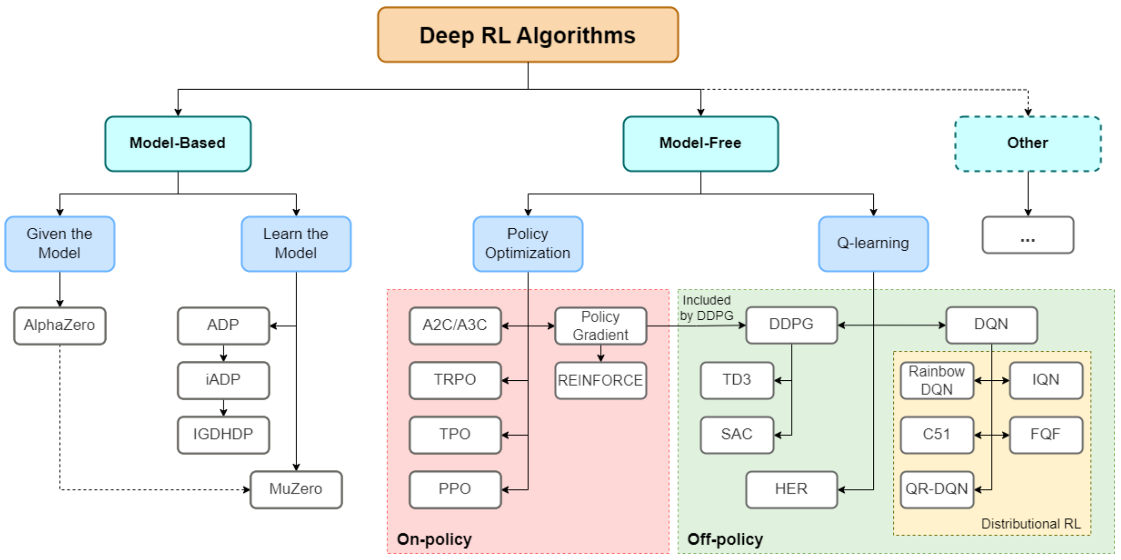


Figure 3.1: Taxonomy of modern Deep RL methods, adapted from [56]

3.1. Policy Optimization Methods

Policy-based methods in RL directly learn a policy without explicitly estimating the value function. By iteratively updating the parameters of a parametrized policy $\pi(s, a; w) = \mathbb{P}[a_t = a|s_t = s; w]$ through gradient descent, these methods aim to find the optimal policy based on observed rewards. This approach is particularly effective for handling high-dimensional and continuous action spaces, unlike value-based methods that require computationally demanding optimization processes or discretization of action spaces [57]. Moreover, the policy-based methods avoid the problem with infinitesimal changes in the value function estimates, which can have a disproportionate impact and cause discontinuities in the implicitly defined policy, adversely affecting overall performance [58]. Finally, policy-based methods also offer the advantage of accommodating stochastic policies, which can be beneficial in environments with aliasing or partial observability.

Policy gradient methods aim to optimize a parameter vector w to find a local or global maximum of the objective function $J(w)$. The objective is to maximize the expected cumulative rewards received from the environment, which is represented by the value function $V_\pi(s_t)$ (with s_t being the state s at time-step t). The policy gradient theorem utilizes the log derivative trick to define the policy gradient in terms of the score function $\nabla_w \text{log } \pi_w$ [56]. Putting all the concepts together, the policy gradient ascent step can be defined by Equation 3.1.

$$w_{t+1} = w_t + \alpha \underbrace{\nabla_w V_\pi(s_t)}_{\text{Policy gradient}} = w_t + \alpha \mathbb{E}_{\pi_w} \left[\sum_{t=0}^{T} \nabla_w \text{log } \pi_w(a|s) V_t \right] \tag{3.1}$$

The simplest policy gradient is the Monte-Carlo policy gradient method, also known as REINFORCE [59]. REINFORCE estimates the policy gradient by sampling trajectories through interactions with the environment, calculating the return for each trajectory, and using these returns to update the policy parameters. While it exhibits good learning stability and convergence properties, it suffers from high variance and slow learning.

As described, policy gradient methods are less efficient, requiring many samples to converge. Moreover, regular approaches are sensitive to the step-size hyperparameter α when training a parameterized policy using DNNs. TRPO tackles this sensitivity by constraining the policy update step within *a trust region* [60]. By using Kullback-Leibler (KL) divergence to limit the magnitude of policy changes, TRPO ensures that each update doesn't deviate too far from the current policy, which improves the stability. PPO, on the other hand, optimizes the step-size by using multiple samples efficiently. It introduces a clipped surrogate objective function that limits the policy update to a range around the old policy, ensuring that updates are performed in a controlled manner. PPO strikes a balance between exploiting the current policy and exploring new policies, allowing for stable and more sample-efficient learning compared to TRPO [37]. It mitigates the sensitivity to the choice of the trust region and achieves more consistent improvements in policy performance. Alternatively Tree-search Policy Optimization (TPO) improves TRPO by incorporating tree search algorithms to enhance exploration and efficiently search the action space, leading to better policy updates and improved learning performance in complex domains [61].

3.2. Q-learning

Q-learning methods focus on estimating the action-value function using deep learning. It is a model-free method that does not require prior knowledge of the environment's dynamics. By estimating the expected future rewards for different actions in a given state, Q-learning allows the agent to make informed decisions and improve its performance over time through trial and error.

3.2.1. Deep Q-Networks (DQN)

Deep Q-Networks (DQNs) are a type of off-policy RL algorithm that uses ANN to approximate the Q-function and make decisions based on the maximum expected cumulative reward [38]. The Q-function is approximated by a parameterized approximator $Q_\theta(s, a)$, where θ is a parameter vector. This architecture has demonstrated super-human capability in Atari gaming environments using convolutional feed-forward ANNs to estimate the action-value function from pixel-information [62]. These games are now a standard benchmark for deep RL algorithms [63].

To estimate the parameter vector θ of the Q-network, a loss function $\mathcal{L}(\theta)$ based on mean-squared Bellman error (MSBE) can be defined by Equation 3.2, which an SGD optimizer can use to train θ and progressively approximate the actual Q-function. The action a' is selected from the *behaviour distribution* $a' \sim \mu(s, a)$, as the architecture is off-policy. In practical applications, the behavior distribution $\mu(s, a)$ is commonly implemented as an ϵ-greedy action selection based on the estimated Q-function. Moreover, parameter d takes the value 1 (true) when the state s' is the final state (and 0 otherwise), indicating that the agent will not receive any further rewards in the future. The MSBE is calculated over the collected buffer of past transitions $\mathcal{D}$.

$$\mathcal{L}_\theta = \mathbb{E}\left[\left(\underbrace{r + \gamma(1-d)\max_{a'} Q_{\bar{\theta}}(s', a')}_{\text{Target Q-network}} - \underbrace{Q_\theta(s, a)}_{\substack{\text{Behavioural} \\ \text{Q-network}}}\right)^2\right], \quad (s, a, r, s', d) \sim \mathcal{D} \tag{3.2}$$

To ensure stability in the SGD optimization process, it is essential to incorporate two enhancements. The first one is already hinted in Equation 3.2 - *Experience Replay*. RL methods deal with time-series data characterized by strong correlations. However, Deep Learning methods rely on independently and identically distributed (IID) data, which necessitates the decorrelation of sampled experience in each learning step. To address this issue, the off-policy characteristic of DQNs enables the stabilization of learning through collecting experienced transitions into a memory buffer $\mathcal{D}$ and sampling a random mini-batch $\mathcal{B}$ of transitions to update the Q-function estimate using SGD. Furthermore, it is vital to maintain some exploration rate throughout the whole learning process to prevent *catastrophic forgetting* of DNNs. This occurs particularly when the policy has settled into a local optimum without any exploration, which saturates the replay buffer with highly correlated data. Consequently, the DNN ”forgets” previously visited states that lie outside the buffer.

As for the second enhancement, the learning process can be stabilized in off-policy algorithms by freezing the target Q-network parameters and updating them only after every N update steps. This provides a more stable and consistent target for the Q-value estimates during training. Subsequently, the target can be updated via *hard update*, where the parameters of the target Q-network are directly synchronized with the parameters of the primary Q-network. Alternatively, *soft update* (also known as Polyak -averaging) can be applied by taking a weighted average of the target network's parameters and the primary network's parameters ($\bar{\theta}_{i+1} = \zeta\bar{\theta}_i + (1-\zeta)\theta_i$ with $\zeta \in [0, 1]$).

3.2.2. Rainbow

The Rainbow algorithm combines several key enhancements to improve the stability and sample efficiency of DQNs [64] by integrating concepts like prioritized experience replay, dueling architectures, Double DQN, distributional value estimation, and multi-step learning. The combined architecture of Rainbow yields a state-of-the-art off-policy RL approach that exhibits superior performance in handling continuous state spaces and making decisions among a discrete action set.

Prioritized Experience Replay (PER) enhances the efficiency and learning performance of DQN by prioritizing experiences during replay [65]. Instead of uniformly sampling experiences from the replay buffer, PER assigns priorities based on TD error, with higher TD errors indicating greater importance. This selective replay improves the Q-network's performance. However, prioritization can lead to diversity issues, where low TD-error transitions visited early may be underrepresented. Stochastic prioritization addresses this by balancing greedy prioritization and random sampling.

Dueling architectures enhance DQN by explicitly separating the estimation of the state value $V(s)$ and the advantages of each action $A(s, a)$, representing the additional value gained by taking a specific action in that state. Combining these two components results in the Q-values for each action. This separation enables the agent to discern which states are not valuable without the necessity of learning the effect of each action in those states, which makes it particularly advantageous in applications with large action spaces.

Multi-step Q-learning enables more accurate value estimation by incorporating future rewards over multiple time steps. *Noisy Networks* introduce random perturbations to the network's parameters, enhancing exploration and preventing over-reliance on deterministic actions. Furthermore, *Distributional RL* captures the full distribution of possible returns, allowing for a more comprehensive understanding of uncertainty and improving the agent's ability to handle complex environments. Distributional methods wills be further addressed in Section 4.3.

Double DQN is a variant of the DQN that addresses the issue of overestimation in Q-value estimation. In standard Q-learning, the action selection and Q-value estimation are based on the same set of parameters, which can lead to overoptimistic value estimates. Double DQN decouples these estimations by using two

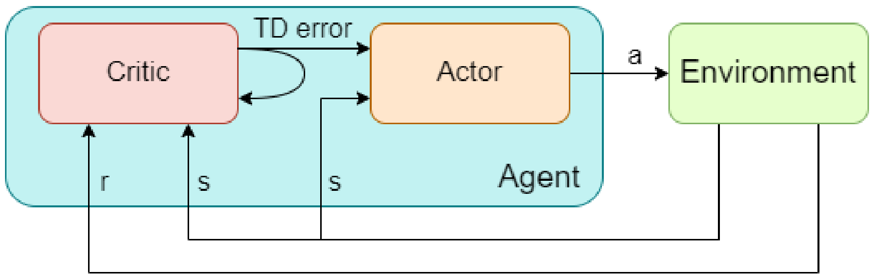


Figure 3.2: The agent-environment interaction in an actor-critic architecture

separate ANNs. The core idea behind Double DQN is to use the target network ($Q_{\bar{\theta}}$) to determine the action to be taken while using the primary network (Q_θ) to estimate the Q-values for the selected action. The Q-value update equation in Double DQN is then modified as given in Equation 3.3. By decoupling the action selection and value estimation using separate networks, Double DQN reduces overestimation biases, which results in more accurate Q-value estimates and increased efficiency.

$$Q_{\bar{\theta}}(s,a) = r + \gamma(1-d)Q_{\bar{\theta}}\left(s', \underset{a'}{\text{argmax}}\ Q_\theta(s',a')\right) \tag{3.3}$$

3.2.3. (Deep) Deterministic Policy Gradient ((D)DPG)

DPG is the first example of actor-critic methods considered in this work. Actor-critics combine policy-based and Q-learning approaches, with the actor approximating the policy and the critic estimating the value function [66]. A new step performed by an actor allows the critic to be updated through TD bootstrapping. As a result, the updated critic is used to update the actor's policy, and this iterative cycle continues at each iteration. By incorporating the actor, the process of optimizing action selection is eliminated, as the actor directly approximates the policy based on the current state, while the critic provides performance insights. The critic's Q-function estimate also reduces gradient variance, accelerating learning and enhancing convergence. The ability to generalize across both state and action spaces has made actor-critic methods prevalent in RL, with several algorithms considered state-of-the-art for solving real-world challenges. The agent–environment interaction with an actor-critic is depicted in Figure 3.2.

In contrast to the stochastic policy gradients, DPG defines the policy deterministically as a parameterized function of the state ($\pi_w(s)$). In other words, for a given state, the DPG algorithm directly outputs a specific action rather than sampling from a probability distribution. This brings significant improvements in computational and sample efficiency compared to stochastic policy gradient methods [66].

DPG utilizes off-policy data and the Bellman equation to acquire knowledge of the Q-function, and subsequently leverages the Q-function to learn the policy, hence it relies on the critic's value function approximation. Specifically, the policy network is updated by performing *gradient ascent* with respect to the policy parameters, utilizing the Q-network estimate from the buffer $\mathcal{D}$ as the objective function. Furthermore, the behavioral policy is still augmented with added Gaussian noise to enhance exploration [66]. In fact, DPG shares similarities with the Q-learning approach used in Deep Q-Networks (DQN), but it differs in that it employs a continuous policy instead of a discrete one [67].

DDPG then combines ideas from DPG and DQN [68]. It uses experience replay and fixed target networks from DQN to improve stability, and it is based on DPG (as it also uses the Q-function for off-policy learning and employs a deterministic actor that aims to maximize this Q-function) [68]. This extends the ability of the original DPG to generalize also to high-dimensional and complex continuous action spaces [56]. However, the interplay between the deterministic actor and the Q-function in DDPG can make the algorithm challenging to stabilize and sensitive to hyperparameter tuning. Furthermore, the most common limitation for DDPG is the significant overestimation of Q-values, which arises from the greedy maximization of a noisy Q-function estimate [69] and ultimately results in the failure of the exploited policy and divergence [70]. To address these issues, two independent improvements have been introduced: twin-delayed deep deterministic policy gradient (TD3) and soft actor-critic (SAC) algorithms.

3.2.4. Twin-delayed DDPG

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an extension of the DDPG designed to address the overestimation bias and improve the training stability [70]. TD3 improves upon DDPG by employing three key features: twin critics, target policy smoothing, and delayed policy updates.

1. *Twin Critics*: TD3 introduces the use of two Q-value estimators (i.e critics) instead of a single critic as in DDPG. This helps mitigate the overestimation bias problem by taking the minimum Q-value estimate from the two critics during both policy evaluation and target Q-value estimation, which reduces the impact of overestimation [70].
2. *Target Policy Smoothing*: TD3 mitigates the policy overfitting by applying clipped Gaussian noise to the target actions during the computation of target Q-values. This noise, controlled by a smoothing parameter, helps in regularizing the policy updates and prevents excessive determinism. This encourages exploration, which helps the policy avoid converging to a suboptimal deterministic behavior.
3. *Delayed Policy Updates*: In TD3, the policy updates are performed less frequently than the critic updates. Delaying the policy updates allows the Q-value estimates to stabilize and reduces the interference caused by the noisy estimates [70]. This results more stable and reliable training.

These modifications allow TD3 to provide more accurate Q-value estimates and avoids the collapse of the exploited policy. The target policy smoothing and delayed policy updates further enhance the training process, preventing overfitting and promoting exploration, which leads to more robust and effective policies.

3.2.5. Soft Actor-Critic

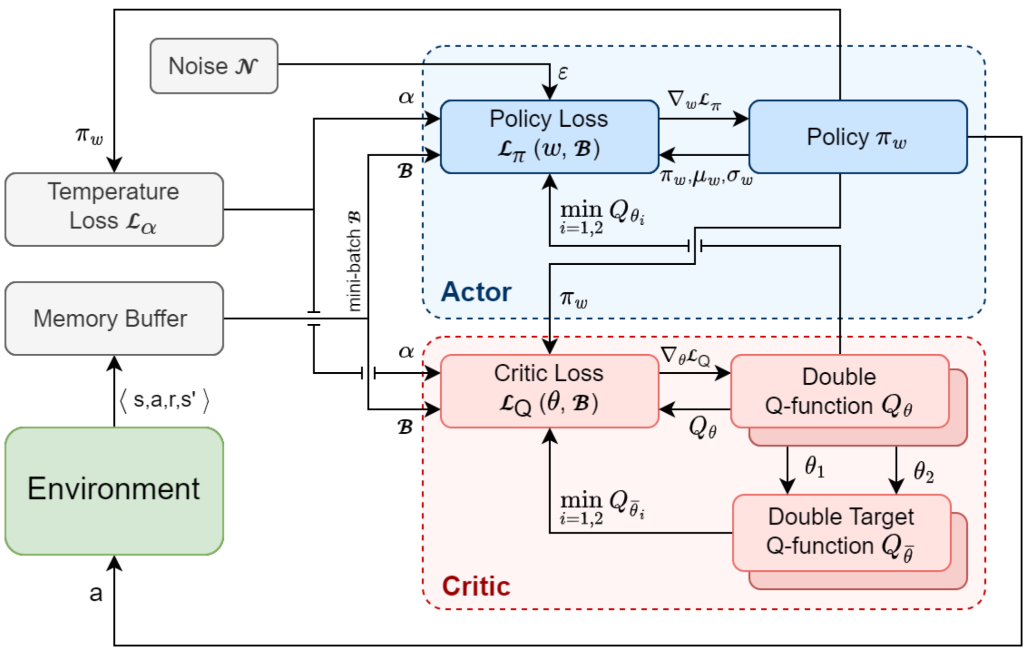

Noise $\mathcal{N}$
ε
α
Policy Loss $\mathcal{L}_\pi\ (w, \mathcal{B})$
$\nabla_w \mathcal{L}_\pi$
Policy π_w
π_w
$\mathcal{B}$
π_w, μ_w, σ_w
Temperature Loss $\mathcal{L}_\alpha$
$\min_{i=1,2} Q_{\theta_i}$
mini-batch $\mathcal{B}$
Memory Buffer
Actor
π_w
α
$\nabla_\theta \mathcal{L}_Q$
Critic Loss $\mathcal{L}_Q\ (\theta, \mathcal{B})$
Double Q-function Q_θ
⟨ s,a,r,s' ⟩
$\mathcal{B}$
Q_θ
θ_1
θ_2
Environment
$\min_{i=1,2} Q_{\bar{\theta}_i}$
Double Target Q-function $Q_{\bar{\theta}}$
a
Critic

Figure 3.3: Architecture of Soft Actor-Critic, adapted from [27].

Soft actor-critic (SAC) is an enhanced version of DDPG designed for handling large continuous action spaces [71], which was developed around the same period as TD3. Similar to DDPG and TD3, SAC operates off-policy and utilizes experience replay and fixed (double) Q-networks to mitigate overestimation bias. Furthermore, all three learn the Q-functions by MSBE minimization and update the target Q-networks by Polyak averaging during the training. However, in contrast to deterministic policies of DDPG and TD3, SAC incorporates stochastic policies to facilitate more effective exploration and prevent early fixation on local optima. Moreover, the noise from the stochastic actor provides sufficient target smoothing, hence no addition of clipped Gaussian noise is necessary.

To train the stochastic policy, SAC introduces the concept of *maximum entropy*, which involves optimizing a trade-off between the expected return and entropy [71]. The entropy quantifies the level of randomness in the policy and is defined by Equation 3.4, where a random variable x is retrieved from probability distribution $\mathcal{P}$. This trade-off closely relates to the exploration-exploitation dilemma: higher entropy promotes increased exploration. This potentially accelerates learning and helps to avoid prematurely converging to an unfavorable local optimum.

$$\mathcal{H}(\mathcal{P}) = \underset{x \sim \mathcal{P}}{\mathbb{E}} [-\log \mathcal{P}(x)] \tag{3.4}$$

In *entropy-regularized* RL, the agent receives a *bonus reward* at each time step proportional to the entropy of the policy at that time step. This additional bonus required the Bellman equation from Equation 2.10 to be updated to Equation 3.5, where $\alpha > 0$ is a *temperature factor* that determines the relative importance of the entropy regularisation and thus controls the exploration-exploitation trade-off.

$$\begin{aligned} Q^{\pi}(s,a) &= \underset{\substack{s' \sim \mathcal{P} \\ a' \sim \pi}}{\mathbb{E}} [R(s,a,s') + \gamma(Q^{\pi}(s',a') + \alpha\mathcal{H}(\pi_w(\cdot|s'))] \\ &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathbb{E}} [R(s,a,s') + \gamma(Q^{\pi}(s',a') - \alpha \log\ (\pi_w(a'|s'))] \end{aligned} \tag{3.5}$$

The right-hand side of Equation 3.5 represents an expectation over future states based on the samples from the replay buffer, and future actions, which are obtained from the current policy rather than the replay buffer. To highlight this, the future actions will be denoted by $\tilde{a}'$, instead of a'.

Critic Learning

Once the samples of entropy-regularized Q-function are obtained, the MSBE loss for each Q-function Q_{θ_i} can be set up as given by Equation 3.6.

$$\mathcal{L}_Q(\theta_i, \mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathbb{E}} \left[(Q_{\theta_i}(s,a) - y(r,s',d))^2 \right] \tag{3.6}$$

Here, the target value $y(r, s', d)$ is estimated by Equation 3.7, which considers the minimum value between the two clipped target networks, while also incorporating the entropy term.

$$y(r,s',d) = r + \gamma(1-d) \left(\min_{i=1,2} Q_{\bar{\theta}_i}(s', \tilde{a}') - \alpha \log \pi_w(\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi_w(\cdot|s') \tag{3.7}$$

In the original SAC algorithm, the temperature factor in Equation 3.7 was kept constant, which was observed to result in brittle behavior [71]. Therefore, techniques that adapt this factor dynamically were suggested [72], where the temperature factor is optimized by Equation 3.8 (with $\bar{\mathcal{H}}$ representing the target entropy).

$$\mathcal{L}(\alpha, \mathcal{D}) = -\mathbb{E} \left[\alpha(\log\ \left(\pi_w(a|s) + \bar{\mathcal{H}}\right) \right] \tag{3.8}$$

Policy Learning

The SAC policy aims, in each state, to take actions that maximize the Q-function including both the expected future rewards and the expected future entropy (see Equation 3.5). However, as stochastic policies introduce randomness in the action selection process, it is difficult to compute gradients with respect to the policy parameters. Therefore, SAC uses a *reparametrization trick*, which decouples the randomness from the policy parameters and transforms the sampling process into a differentiable operation. Specifically, a sample from $\pi_\theta(\cdot|s)$ is drawn by calculating a deterministic function of state, policy parameters, and independent noise [71]. For example, the original work on SAC uses a squashed Gaussian policy, which means that samples are obtained according to Equation 3.9. In practical implementation, reparameterization is facilitated by automatic gradient calculation tools like PyTorch [73].

$$a(s, \epsilon) = \tanh(\mu_w(s) + \sigma_w \odot \epsilon), \quad \epsilon \sim \mathcal{N}(0, I) \tag{3.9}$$

The tanh function here acts as a squashing function, which means that it ensures that actions are bounded to a finite range. The operator $\odot$ represents an element-wise product of two vectors.

Merging the loops for training the critic, the policy and the temperature loss function defines the architecture of SAC, which is visualized in Figure 3.3. A similar architecture was successfully applied to a continuous flight control tasks, which will be further elaborated on in Section 3.3. SAC was also demonstrated to outperform TD3 in terms of sample efficiency across various complex control tasks in a benchmark by [56]. Furthermore, its ability to model the distribution of actions makes it an appealing architecture for uncertainty-aware RL, thus it is particularly interesting for this research.

3.3. Deep Reinforcement Learning for Flight Control

The idea of applying RL to flight control is not novel. Already in 2002, an adaptive critic controller was used to control the full 6-DOF simulation of a business jet aircraft [74]. In 2006, a RL controller was applied to perform complex maneuvers with an acrobatic RC helicopter [75]. This section provides a comprehensive exploration of further applications of RL techniques in the context of flight control.

3.3.1. Definition of the Flight Control Problem

The fixed-wing aircraft flight control poses a highly complex problem with non-linear and highly-coupled transition dynamics with 6 DOF. In general terms, the flight control problem can be defined by Equation 3.10.

$$\dot{\underline{x}} = f(\underline{x}, \underline{u}, t) + \underline{w} \approx f(\underline{x}, \underline{u}) + \underline{w} \tag{3.10}$$

Here, the equation is simplified by assuming stationary dynamics within a short time frame, resulting in the omission of the time variable t. The non-linear state transition function of the aircraft system is represented by function f, which is modeled by *equations of motion* (EOM). To simplify the problem, function f can be decoupled to separately represent the longitudinal and lateral dynamics with respective DOF. It takes the *aircraft state vector* $\underline{x} \in \mathbb{R}^n$ and the *control inputs vector* $u \in \mathbb{R}^m$ as inputs. Furthermore, the system noise vector $w \in \mathbb{R}^n$ was included to capture the unpredictable fluctuations or disturbances present in a system.

The aircraft state vector is represented by a position vector $\underline{p} = [x, y, z]^T \in \mathbb{R}^3$, a velocity vector $\underline{v} = [u, v, w]^T \in \mathbb{R}^3$, an attitude vector described by Euler angles $[\phi, \theta, \psi]^T \in \mathbb{R}^3$ or quaternions $[q_1, q_2, q_3, q_4] \in \mathbb{R}^4$, and angular velocity vector $\omega = [p, q, r]^T \in \mathbb{R}^3$. The velocity vector is usually represented by a polar notation, with a magnitude of $V = |\underline{v}|$ and aerodynamic angles $\alpha = \text{atan}\left(\frac{w}{u}\right)$ and $\beta = \text{atan}\left(\frac{v}{\sqrt{u^2+w^2}}\right)$.

The primary control of a 6 DOF fixed-wing aircraft is usually provided by control surface deflections and thrust settings. Therefore, the control input $\underline{u}$ in its simplest form can be represented as $\underline{u} = [\delta_e, \delta_a, \delta_r, \delta_t]^T$, where the respective components correspond to the elevator, ailerons and rudder deflections, and the thrust setting. In a high-fidelity model, these inputs should be complemented by secondary control inputs, such as flaps, slats, spoilers, and other subsystems affecting the aerodynamics of the aircraft.

Finally, a few adjustments need to be made to formulate the flight control task as an MDP. First, the trajectory-tracking task requires to augment the system states with tracking errors with respect to desired trajectories. Furthermore, not all states might be observable in flight, which results in a partially observable MDP (POMDP). The actions a can correspond to the aircraft control inputs $\underline{u}$, or can represent the command control increment $\delta_e \underline{u}$ to smooth out the control input (while including the actual control input $\underline{u}$ in the state vector) [27]. Last but not least, the reward function can be formulated to be proportional to absolute (i.e. $R(s, a) \propto |\tau_{ref} - \underline{x}|$) or squared tracking error (i.e. $R(s, a) \propto (\tau_{ref} - \underline{x})^2$), where τ_{ref} represents the reference trajectory.

3.3.2. Challenges and Potential Solutions

The application of RL to flight control comes with several challenges that need to be addressed. This work identifies three main challenges, each of which will be discussed separately together with the previous attempts to resolve them. The contents of this section are condensely summarized in Table 3.1.

Table 3.1: Challenges in RL applied to flight control tasks and potential solutions.

Challenge	Source of Concern	Potential Solutions
Safety	Uncontrolled exploration, simulation gap	SHERPA, SAC, DSAC, Shielded RL, ...
Efficiency	Curse of dimensionality	HRL, SAC, ...
Explainability	Black-box nature of RL	DRX, SHAP, ...

Safety

Safety is a paramount concern in aviation and integrating RL into flight control systems introduces unique challenges that must be addressed to ensure safe operation. First of all, RL algorithms rely on exploration to discover optimal control strategies. However, free exploration can lead to physical damage or catastrophic failures if the environment is dangerous [76]. Furthermore, while a trained RL algorithm might perform adequately in scenarios that were well represented by the set of training data and environments, real-world scenarios may present novel and unforeseen situations. The learned policies may not generalize well to these new scenarios, potentially leading to unsafe behavior [76]. For this reason, robust RL algorithms are required. Furthermore, given the prevailing safety concerns associated with RL applications in control settings, an entire research field known as *Safe RL* has emerged with the aim of addressing these challenges [77]. This field can be divided into two different approaches. The first class of methods involve modifying the optimality criterion with a safety factor, while the other class focuses on modifying the exploration process to enhance safety through incorporating external knowledge or through the guidance of a risk metric.

The safety aspect of RL applied to flight control was aproached in several works. For example, in *Safety Handling Exploration with Risk Perception Algorithm* (SHERPA) [76], a safety filter monitors a designated fatal state space and guarantees that the agent avoids states with elevated associated risk. Over time, the set of safe states expands as exploration progresses. This approach has proven effective in enabling safe learning for the low-level flight control of UAVs. Alternatively, another work has shown that the SAC algorithm can achieve robust flight control of fixed-wing aircraft and is capable of generalizing to unforeseen failure scenarios [27]. For example, Figure 3.4 shows the response of a system experiencing a rudder failure at a fixed angle of -15 degrees from the 10-second mark onwards. Despite the significant failure, the robust controller manages to maintain stability by adapting its elevator and aileron positions, effectively neutralizing the impact of large pitch and roll angle errors and exhibiting a performance similar to the non-failed case, except for a persistently large sideslip angle error. The adaptive response (after 60s), trained on the failed system, replicates non-failed tracking performance in pitch and roll angles, despite a large sideslip angle error. These results were supported by another study, which has demonstrated the effective application of SAC and TD3 in suppressing the occurrence of dangerous roll oscillations at high angles of attack in flying-wing configurations, which reduces the risk of loss of control (LOC) [78].

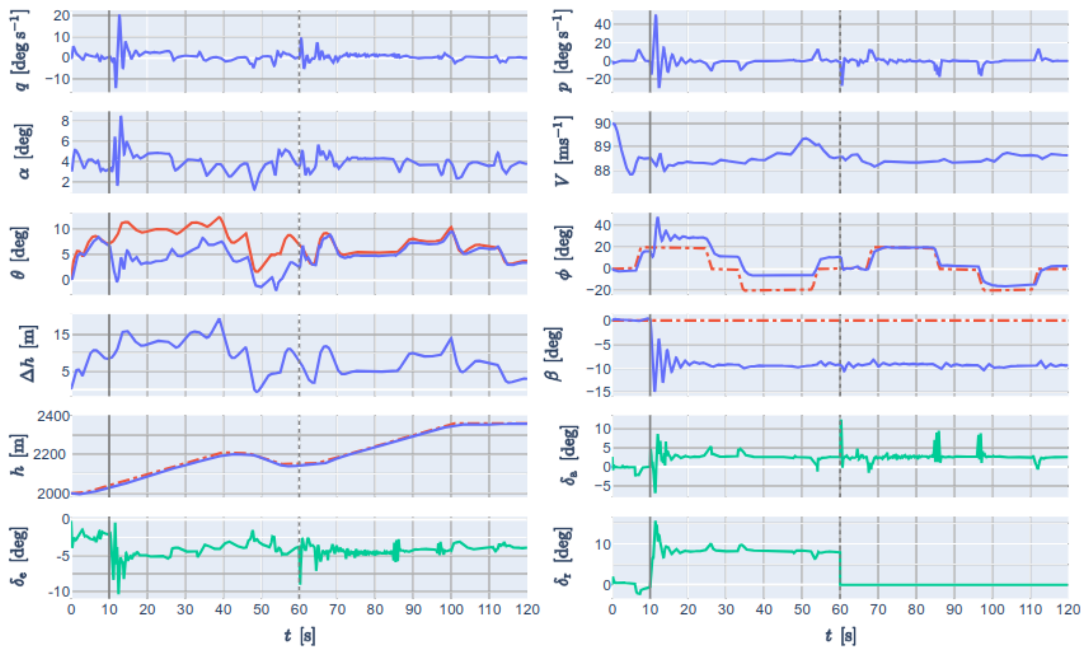


Figure 3.4: Altitude tracking with rudder stuck at -15° from t=10s. Red dashed and solid lines show external and self-generated references, and green solid lines show control inputs. Robust control until t=60s, adaptive control follows, adopted from [27].

Building upon SAC, a Distributional SAC algorithm was proposed for flight control tasks [28]. This method was demonstrated to improve learning characteristics of traditional SAC by estimating the entire probability distribution, which also allows for controlling the risk-taking behaviour. In a different approach a supplementary controller, known as a *shield*, is integrated with an offline DDPG controller [26]. The shield controller is responsible for monitoring the flight path angle and providing recommendations for safe actions when the system approaches a risky state space. Finally, Safety-informed Evolutionary RL (SERL) integrates Deep RL with neuro-evolutionary mechanism, addresses safety by incorporating a safety-informed mutation operator and utilizing a critical buffer, directing exploration towards less sensitive regions [79].

Another safety challenge arises from the *simulation gap*. This refers to the disparity between the simulated environment used for training RL agents and the real-world environment where these algorithms are deployed [80]. Closing this gap by full online learning seems out of question due to the high safety risks posed by exploration. Therefore, the RL controllers are typically trained offline before transferring them to the real-world system. The simulation gap may result from various factors such as model errors, delays in sensor measurements and control inputs, noisy or biased sensor outputs, and others. Therefore, robust flight controllers are sought in research, which could handle such discrepancies. For example, a hybrid SAC-IDHP was developed, which combines the previously discussed SAC, which offers high complexity generalization power, with Incremental Dual Heuristic Programming (IDHP), which provides adaptive online learning [81]. The SAC-IDHP was demonstrated to provide more adaptive control with lower tracking error across all tested nominal and fault scenarios. For example, the tracking performance comparison between SAC-IDHP and SAC only is displayed in Figure 3.5 for an aircraft with 90% reduced aileron effectiveness.

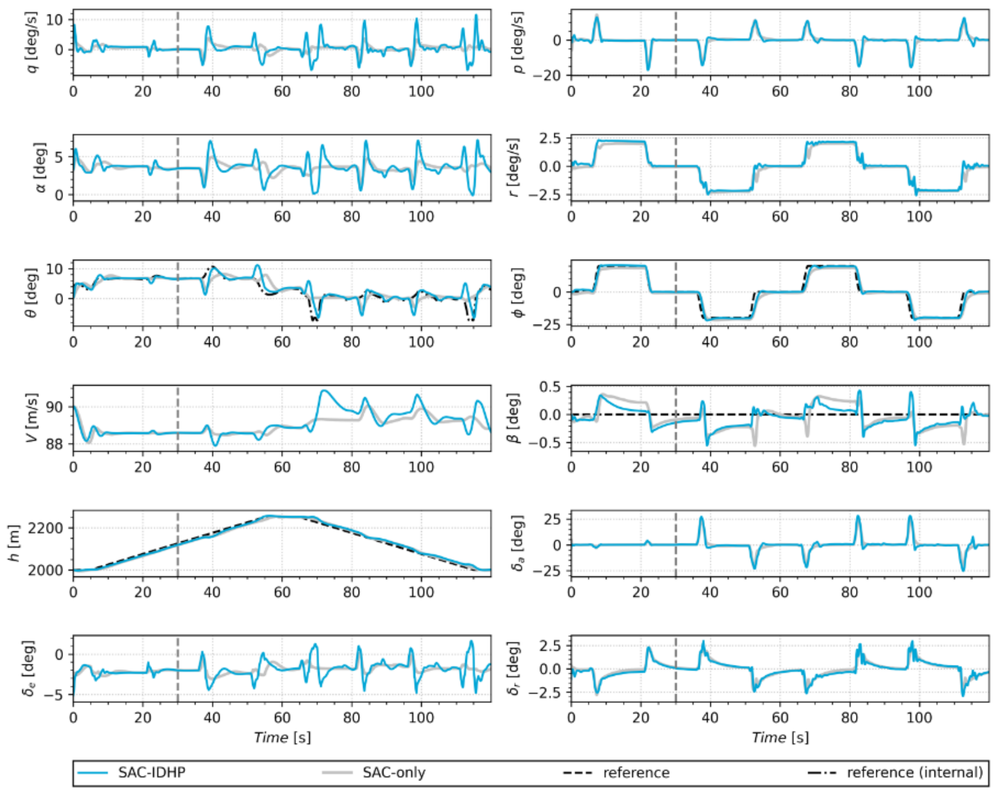


Figure 3.5: Altitude tracking with 90% reduced aileron effectiveness from t=30s. Comparison of SAC-IDHP (nMAE = 2.46%) and SAC-only (nMAE = 3.28%), adopted from [81].

Efficiency

High dimensionality of the state-action space for flight control tasks poses a challenge for efficient learning. What is more, exploring high-dimensional states becomes exponentially difficult due to the curse of dimensionality. Therefore, sample efficiency is essential for successful RL flight controllers.

To mitigate the effects of the curse of dimensionality, Hierarchical RL (HRL) methods were proposed. HRL decomposes the original complex learning task into multiple smaller RL problems. Subsequently, a cascade of agents is implemented, where each agent is responsible for a specific problem [82]. This is feasible through the concept of *time-scale separation*, where the dynamics of a system can be divided into subsystems operating at different time scales based on their respective speeds of change. For example, higher-frequency dynamics, such as fast rotations or transient behaviors, are typically considered to have a shorter time scale, while slower dynamics, such as attitude control, have a longer time scale. In such architectures, inner-loop controllers focus on faster dynamics and regulate specific aspects, while outer-loop controllers handle slower dynamics and provide overall control and coordination.

The applicability of HRL to flight control is evident, given the presence of both faster and slower dynamics within the aircraft's overall system. In fact, HRL has been used to improve the sample efficiency of an F-16 RL-controller by defining a three-level learning structure of an F-16 controller [83]. It uses the *options* algorithm, which uses predefined sub-policies that extend the agent's action space, allowing it to perform temporally extended actions instead of individual primitive actions. Another approach builds upon this work by incorporating an auxiliary safety modification layer (SML) equipped with prior knowledge of the system dynamics that corrects the actions to enhance safety [84]. A cascaded controller is applied also in the aforementioned SAC, which uses separate controllers to control altitude and altitude of Cessna Citation 500. The cascaded structure of this controller is visualized in Figure 3.6. HRL methods were shown to improve the sample efficiency, but also led to difficulties in policy learning, inconsistent outcomes, and increased number of hyperparameters.

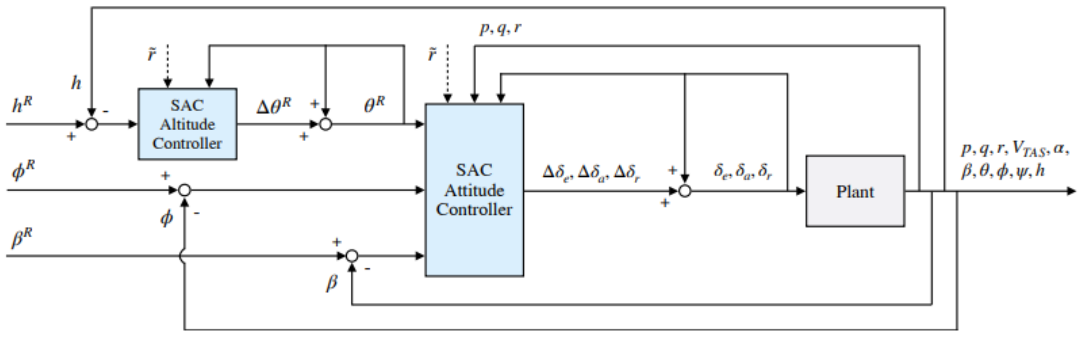


Figure 3.6: Cascaded SAC controller structure for altitude and attitude control, adopted from [27].

Explainability

Deep RL algorithms often operate as black boxes, where the internal decision-making processes are complex and not easily interpretable. The high-dimensional representations and non-linear mappings make it difficult to understand how specific inputs lead to particular actions or decisions. Furthermore, interpreting the learned policies becomes more difficult as the dimensionality and complexity increase. This opaqueness hampers the ability to explain why a certain control action was chosen by an RL agent. In contrast, traditional control systems often rely on explicit, human-readable rules or models that can be easily understood and interpreted to ensure the system's safety and reliability. This highlights the importance of explainability in the context of verifying and validating the behavior of flight systems.

Addressing the explainability challenge in RL for flight control is an active research area. For example, a Dominant Reward eXplanations (DRX) technique was proposed and applied to a longitudinal state-space model of the Cessna Citation II to produce straightforward and intuitive insights about the controller's behaviour [85]. This reward decomposition technique attempts to attribute the deep RL agent's behavior to the specific components of the reward function that are most responsible for shaping its actions. Alternatively, the method of SHapley Additive exPlanations (SHAP) was used to analyze the strategy learnt by an Actor-Critic IDHP controller of the same aircraft [86]. SHAP assigns importance scores to input features, quantifying their influence on the RL agent's predictions or actions.

3.3.3. Alternative Dynamic Programming Flight Controllers

While classical RL is the main focus of this work, it is important to investigate alternative flight control approaches rooted in dynamic programming (DP). This allows for comparison of different techniques that can lead to identification of further benefits and limitations of the discussed RL methods, while it can inspire novel ideas and techniques that can be incorporated into RL algorithms to enhance their performance.

Adaptive Critic Designs (ACD) are Approximate DP (ADP) methods that are perhaps the most prevalent DP control alternative to approach autonomous flight controllers with the actor-critic architecture. These methods aim to solve the DP problem by providing approximate solutions to deal with the curse of dimensionality. ACD differ from traditional RL in terms of their emphasis on the separation of the critic and the actor [87]. The ACD's critic's role is to provide an evaluation or guidance to the actor, rather than directly influencing the policy or value updates. The online adaptive nature of ACD and their focus on optimal control of continuous dynamic systems makes them attractive for flight control applications.

ACD encompass three main classes: Heuristic Dynamic Programming (HDP), Dual Heuristic Dynamic Programming (DHP), and Global Dual Heuristic Dynamic Programming (GDHP). HDP is the simplest form of ADP, where the critic estimates the state-value function [88]. To simplify the actor training, DHP improves HDP by outputting the gradient of the value function instead of the value function itself, which renders the back-propagation step unnecessary [89]. GDHP then combines these approaches by evaluating both the value function and its derivative with the goal of enhancing the approximation accuracy [89, 90]. One of the disadvantages of these methods is their dependence on accurate models of system dynamics. To mitigate the model dependence, incremental counterparts of these algorithms (IHDP, IDHP and IGDHP) were developed, which incrementally update and refine the model based on observed data and system interactions. In addition to the aforementioned SAC-IDHP, the application of ADPs in flight control has been explored in various other studies. For instance, one study investigated the use of a combination of PID and IDHP controllers for simultaneous altitude and roll angle reference tracking [23], while another study focused on a full IDHP controller for altitude control in the presence of measurement noise and atmospheric gusts [24].

Although incremental ADP (iADP) approaches perform well in terms of online learning and adaptation to in-flight changes in the system, they often face challenges in generalization due to their reliance on local linearization. Furthermore, the demonstrated effectiveness of iADP methods is limited to scenarios with few DOF and small state-action spaces. These two reasons hinder their ability to handle the full 6-DOF flight control involving highly coupled dynamics and complex high-dimensional state-action spaces. This is confirmed by the aforementioned research on full IDHP controller for altitude control, which resulted in high failure rate of 24% [24]. Finally, the continual online learning behavior of iADP methods poses safety concerns, as they require continuous and varied range of inputs. Prolonged periods of low excitation can lead to continually growing weights, which can destabilize the learning process and compromise the stability and safety of the flight control system.

3.4. Conclusions

This chapter outlined the most prevalent state-of-the-art deep RL algorithms used in practice. The emphasis was on model-free methods, as they are deemed to be more applicable for flight control tasks due to their superior robustness to unforeseen scenarios as compared to their model-based alternatives. These model-free algorithms were further divided into on-policy and off-policy classes. Policy optimization methods learn a policy without explicitly evaluating the value function. However, they tend to require a large number of samples to converge. This is solved by the off-policy Q-learning approaches, which enable efficient reuse of experience data and allow for more sample-efficient learning. Actor-critic architectures merge the favorable aspects of both classes and leverage the critic's value estimate to enhance the learning capabilities of the actor. Algorithms such as SAC and TD3 have demonstrated exceptional efficiency and performance when facing complex problems with high-dimensional continuous state and action spaces, which makes them potentially suitable for flight control. This was proved by the previous applications of RL-based flight controllers, as discussed in this chapter, hence **RQ-M-1** was answered. Additionally, an alternative research direction called Approximate Dynamic Programming (ADP) was reviewed, but it will not be further explored in this work due to limitations such as low generalization power, limited scalability, and safety concerns related to persistent excitation requirements. However, the online learning capabilities of ADP methods make them intriguing for bridging the simulation gap in RL flight control problems.

Furthermore, the challenges of safety, efficiency and explainability were discussed, along with their potential solutions in the flight control domain. Nevertheless, except for DSAC, these solutions did not explicitly consider or model uncertainties in the decision-making. However, considering uncertainty can enhance sample efficiency by enabling more informed decisions and improved robustness to outliers, which is the aim of this research. Additionally, incorporating uncertainty information enables risk assessment and safer control, while also facilitating explainability by providing justifications based on the agent's confidence level or uncertainty. Hence, the focus of the rest of this work will be on studying the uncertainty-aware algorithms.

4

Uncertainty in Reinforcement Learning

Traditional RL methods often overlook the inherent uncertainties present in real-world environments, which can hinder their performance, safety, and interpretability. This chapter delves into the emerging field of uncertainty-aware reinforcement learning, which aims to address these limitations by explicitly considering and modeling uncertainties in the decision-making process. First, sources of uncertainty will be examined in Section 4.1. Subsequently, Uncertainty-Aware RL methods will be explored in Section 4.2, where the Distributional RL (Section 4.3) and Bayesian RL (Section 4.4) will be discussed in more detail. Finally, one of these approaches will be selected for the following research phases in Section 4.5.

4.1. Taxonomy of Uncertainty

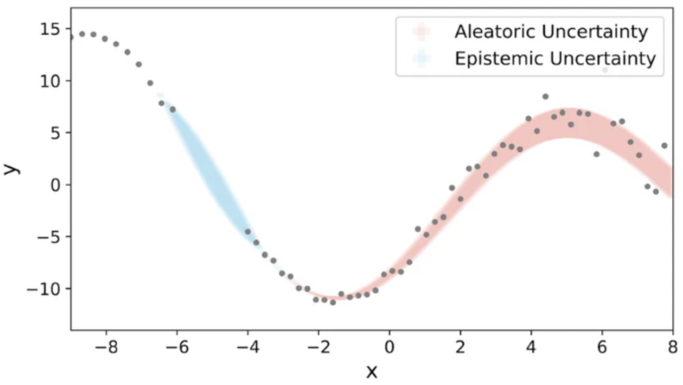


Figure 4.1: The comparison of aleatoric and epistemic uncertainties is illustrated by the plot, with the data points plotted as dots (adopted from [91]).

Uncertainty can be divided into two categories: aleatoric and epistemic. The difference between them is summarized by Figure 4.1. Aleatoric uncertainty accounts for the fluctuating levels of inherent noise in the data, whereas epistemic uncertainty represents the knowledge gap resulting from insufficient data.

Aleatoric uncertainty arises from the stochastic nature of the environment and the interactions within it [92]. Although it can be modelled and evaluated, it cannot be reduced. For instance, games like Chess have no aleatoric uncertainty while games like Poker involve significant aleatoric uncertainty. In the context of flight control, aleatoric uncertainty could include factors such as turbulence, wind gusts, sensor noise, and system failures.

There exist three primary sources of *aleatoric* uncertainty in RL, one for each element of the MDP: stochastic rewards, stochastic observations, and stochastic actions [92]. Stochastic rewards create irreducible uncertainty in the true value function. Stochastic observations can arise from incomplete observations or stochastic transition dynamics. Finally, when actions themselves are stochastic, the next state becomes uncertain. Good examples are the stochastic policy algorithms such as PPO or SAC, which randomly select actions from a distribution. While these sources of aleatoric uncertainty can be estimated and possibly measured, they cannot be reduced like epistemic uncertainties. Nonetheless, it is important to be aware of them. In particular, it is beneficial to distinguish between areas of high epistemic uncertainty and high aleatoric uncertainty when training an agent. For example, an area of high uncertainty in the environment may need to be explored, but if the uncertainty arises entirely from aleatoric sources, it will not be effective to keep visiting that area because there is no additional information to be gained.

Epistemic uncertainty (also called model or systematic uncertainty) arises from incomplete knowledge or information. In contrast to aleatoric uncertainty, it can be reduced through acquiring more data for learning or improving models [92]. In flight control, examples of epistemic uncertainty include inaccurate or incomplete models of the aircraft's dynamics, insufficient data to estimate the RL parameters or to train the neural network that approximates the value or policy function, and more. Epistemic uncertainty is naturally connected to the challenge of effective exploration. Specifically, this uncertainty can be used to balance the exploration-exploitation trade-off, allowing the agent to explore more in regions where it is uncertain about the optimal policy and exploit more in regions where it is confident about the optimal policy. Nevertheless, decreasing epistemic uncertainty is not always a straightforward task that can be achieved by acquiring more data or through further training [92]. For instance, deep Q-Networks may encounter a decrease in their performance over extended periods of training due to catastrophic forgetting. This refers to a situation in which the agent forgets how to perform well in the early stages of the environment because it has been primarily training on observations from later stages [93]. Therefore, it is not always the case that increasing the training time will result in a reduction of epistemic uncertainty.

4.2. Uncertainty-Aware Reinforcement Learning Methods

Once various sources of uncertainty have been identified, the next step is to investigate how they are tackled in the field of RL. While there are many different techniques for uncertainty-aware RL, this work will focus on the most commonly used ones. Specifically, this paper will discuss the following techniques: Count-based methods, Monte-Carlo dropout, Bootstrapping, Distributional RL, and Bayesian RL. Figure 4.2 provides a concise summary of these techniques.

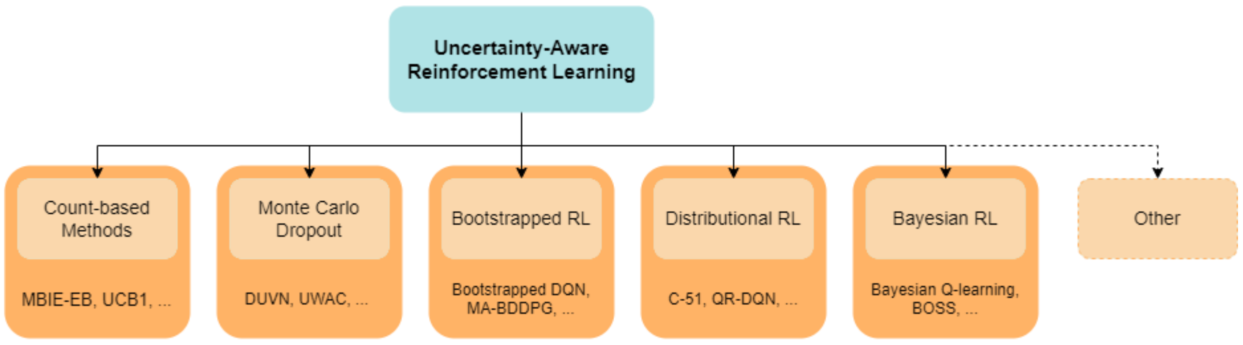


Figure 4.2: Summary of Uncertainty-Aware RL techniques

4.2.1. Count-Based Methods

Count-based methods are a class of exploration techniques that model uncertainty by keeping track of how often each state or state-action pair has been visited [94]. The idea is to encourage the agent to explore less-visited areas of the state or state-action space, which can lead to improved learning. For example, in tabular RL, the visitation count for each state or state-action pair $N(s,a)$ is maintained. Subsequently, the exploration bonuses can be added to the Q-value or the estimated value of a state or state-action pair, based on its visitation count. A common exploration bonus is the inverse square root of the count, as given in Equation 4.1.

$$\text{bonus}(s,a) = \beta \frac{1}{\sqrt{N(s,a)}} \tag{4.1}$$

Here, β represents a positive scalar controlling the magnitude of the exploration bonus. This bonus is then incorporated into the Bellman equation. For example, Model-based Interval Estimation with Exploration Bonuses (MBIE-EB) [95] solve the augmented Bellman equation as given in Equation 4.2

$$V(s) = \max_{a\in\mathcal{A}} \left[R(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}}\left[V(s')\right] + \beta \frac{1}{\sqrt{N(s,a)}} \right] \tag{4.2}$$

Related to these algorithms are the Upper Confidence Bound (UCB) algorithms. UCB algorithms are an exploration strategy originally developed for multi-armed bandit problems, but they can be adapted for RL by treating each state-action pair as an arm. For example, the UCB1 algorithm [96] adds an exploration bonus based on the visitation count and a confidence parameter $c > 0$ (which controls the exploration-exploitation trade-off), as defined in Equation 4.3. Here, t represents the current time step. The agent then selects actions according to the maximum UCB Q-value.

$$Q_{\text{UCB}}(s,a) = Q(s,a) + c\sqrt{\frac{2\log t}{N(s,a)}} \tag{4.3}$$

Count-based methods perform near-optimally in conjunction with tabular RL methods for solving small discrete MDPs, but they cannot be applied in high-dimensional continuous state space scenarios (such as flight control), since most states will occur only once [97]. Therefore, more recent work attempts to generalize these methods to continuous and high-dimensional MDPs using function approximators. For instance, in pseudo-count methods, the approximator is designed to model the underlying distribution of the environment and to estimate the probability of a particular state being visited based on the agent's past experience [98]. The estimated probability is then used to calculate a pseudo-count value that is added to the actual count for each state. An alternative solution is *hashing*, where state-action pairs are mapped to a fixed number of counter slots [97]. More specifically, the hashing function takes a state-action pair as input and produces a *hash value*, which is used to index into a fixed-size array of counters. When a state-action pair is visited, its corresponding counter is incremented.

4.2.2. Monte Carlo Dropout

Monte Carlo Dropout (MC-Dropout) is a technique used to estimate uncertainty in ANNs, including those used in Deep RL. Srivastava et al. introduced MC-Dropout in 2014 as a regularization method [99], and in 2016, Gal and Ghahramani showed that dropout can be also seen as an approximation of Bayesian inference in deep Gaussian processes (see Section 4.4) [100].

Let an ANN approximate a Q-function $Q(s, a; \theta)$, where θ is a vector of ANN parameters. To incorporate MC-Dropout into the ANN, a fraction p of the neurons is randomly deactivated during training, effectively creating a different sub-network for each forward pass. This resulting model of the Q-function can be then represented as $Q(s, a; \theta \odot m)$, where m is a *binary mask* with the same shape as θ and $\odot$ signifies an element-wise multiplication.

To use MC-dropout for uncertainty estimation, N forward passes with different binary masks m are performed, and the resulting predictions are used to estimate the variance of the Q-value or policy probabilities. This is summarized by Equation 4.4, where $\bar{Q}(s, a)$ is the mean predicted Q-value across all passes. The variance can then be used as a measure of epistemic uncertainty in the Q-value estimate [100], and can be incorporated into the learning process to encourage exploration and prevent overconfidence in uncertain regions of the state-action space.

$$\sigma^2(s,a) = \frac{1}{N}\sum_{t=1}^{N}(Q(s,a,\theta_t) - \bar{Q}(s,a))^2 \tag{4.4}$$

Employing dropout techniques in ANNs for RL is often discouraged, due to the non-stationary nature of the target distributions and the inherent risk of overfitting [92], yet it was used for uncertainty estimates [100]. Similarly, MC-dropout was used in a Double Uncertain Value Networks (DUVN) algorithm to estimate the epistemic uncertainty while simultaneously predicting the aleatoric uncertainty over the return distribution by having an ANN output the average of the Q-function ($\mu = Q(s, a)$) and its variance σ [101]. However, they achieved only modest empirical results. Alternatively, Uncertainty Weighted Actor-Critic (UWAC) used MC-dropout to mitigate the effect of out-of-distribution (OOD) data in the actor-critic architecture by scaling each the actor and critic errors by the inverse variance of the Q functions (i.e., by $\frac{\beta}{\sigma^2(Q(s,a))}$) [102], so that estimates with higher variance contribute less to the learning process. This approach showed improvements mainly in the expert-replay experiments.

Bootstrapping

Bootstrapping aims to approximate a population distribution using a sample distribution [92]. It typically involves an estimator ψ and a data set $\mathcal{D}$ as input. To generate a sample from the bootstrapped distribution,

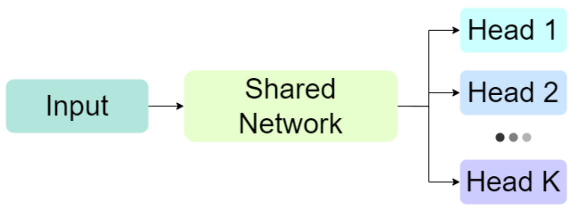


Figure 4.3: Bootstrapped DQN Neural Network architecture (adapted from [103])

a new data set $\tilde{\mathcal{D}}$ is drawn uniformly from $\mathcal{D}$ with replacement until $\tilde{\mathcal{D}}$ and $\mathcal{D}$ are equal in cardinality. $\mathcal{D}$ is then input to ψ to generate a bootstrap sample. The estimation of uncertainty is then straightforward. First, an ensemble of N estimates (for example, of Q-value function) is generated by drawing N data sets $\tilde{\mathcal{D}}$ from $\mathcal{D}$, each of which is used to train the estimator ψ individually. Due to the difference in training data, the output of each of this estimators might be different. Hence, an equation similar to Equation 4.4 can be used to evaluate the variance representing the uncertainty.

Bootstrapping from heads entails multiple forward passes of only a subset of layers (i.e. the heads), whereas performing MC-Dropout requires a full network pass for each repetition used to estimate the variance of predictions. This results in significant differences in computational requirements, which might be the reason why bootstrapping has received more attention in the RL literature [92]. For example, Bootstrapped DQN was introduced as a method for efficient exploration [103], where bootstrapping is introduced by modifying the traditional DQN ANN architecture. More specifically, this ANN consists of two parts - a shared architecture and K bootstrapped *heads* that branch off the shared component independently. Each of this heads is trained separatly on bootstrapped sub-sample $\tilde{\mathcal{D}}$. The shared network acquires a unified feature representation encompassing the entire data set $\mathcal{D}$, which can bring notable computational benefits while sacrificing some variety among its components. This architecture is visualized in Figure 4.3. As the heads are selected for training in a uniform manner, they should ideally only coincide with each other once they have all reached the optimal Q-function. Hence, the variance of the prediction of different heads is a measure of epistemic uncertainty, which decreases over time [103].

Building upon bootstrapped DQN (BDQN), another approach replaced the vanilla deep-Q networks with bootstrapped DQNs in model-based DDPG [104]. Their *Model-assisted Bootstrapped DDPG* demonstrated efficiency gains of up to 15 times and generated significantly larger rewards on continuous control tasks such as Reacher [105]. This exemplifies a prevailing pattern where incorporating available uncertainty-aware techniques can produce tangible improvements in empirical results. Furtermore, improving upon both BDQN and DDQN, a variant of DQN that utilizes K independent ANNs was introduced [106]. Unlike BDQN, which only uses separate heads for each prediction, this study employs completely separate networks and uses them for action selection (unlike DDQN, which uses separate networks only for target prediction). This simple modification led to performance comparable to Rainbow (see Section 3.2).

Distributional Reinforcement Learning

Traditionally, RL frameworks rely on estimating the expected future rewards for a given state and action $Q(s,a)$, which are used to learn a policy that maximizes the cumulative reward over time. However, in many real-world scenarios, the outcomes of actions are not deterministic, and there is inherent randomness in the environment. Distributional RL (DRL) is a recent development in the field that aims to address this limitation. Instead of only considering the expected value of future rewards, DRL use ANNs to learn the entire distribution of possible rewards $Z(s,a)$, providing a more complete and nuanced representation of value [107]. This can help DRL agents make better decisions when facing uncertain environments and risks associated with different actions, leading to more effective and robust behavior [108]. Moreover, a recent study revealed links between DRL and the human brain, where the authors suggest that a brain comprehends the potential future rewards as a complete distribution and not as a point value representing the mean of stochastic outcomes [109]. This finding serves as an encouraging factor for the use of DRL in various applications.

Several studies have demonstrated that using DRL can significantly improve learning performance and increase sample efficiency [107, 110, 111, 112], although the exact reasons for these observations are not fully understood yet [113, 114]. This improvement was empirically observed especially in control settings with non-linear approximators [115]. Moreover, DRL allows for the use of risk-sensitive policies, by applying risk-distortions that can adjust the risk-tendency of the agent [116]. This characteristic makes them interesting for the autonomous flight control applications, for which safety is imperative. Overall, the use of DRL in flight control can help improve learning performance and enhance safety, making it a promising area for future research. However, estimating a full distribution of returns instead of just a scalar value can result in a more computationally expensive algorithm compared to traditional RL methods. Additionally, the choice of the distributional approximation and the distance measure used to compare distributions can have a significant impact on the performance of DRL algorithms [52].

Several algorithms have been proposed based on DRL, such as Categorical DQN (C-51), Quantile Regression DQN (QR-DQN), and Implicit Quantile Networks (IQN). These algorithms differ in the way they represent and update the distribution $Z(s, a)$. For example, in *Categorical DQN*, the distribution of returns is represented as a discrete histogram with a fixed number of bins (categories), where each bin corresponds to a possible value of the return [107]. The probability of each bin is learned through interactions with the environment. In contrast, *QR-DQN* represents the distribution of returns using *quantiles* [110]. A quantile is a threshold value that divides the distribution into equal parts. For example, the median is the 0.5-quantile, which divides the distribution into two equal parts. Finally, IQN learns implicit quantiles rather than explicitely estimating pre-defined quantile values [111]. This enables IQN to adaptively capture the shape of the underlying distribution, resulting in a more accurate representation.

Bayesian Reinforcement Learning

Bayesian RL (BRL) refers to any method of RL that incorporates the Bayesian statistics. All the other methods discussed so far predict the underlying uncertainty solely based on the obtained data, without making any *prior* hypothesis about it. In contrast, BRL takes a probabilistic approach by explicitly modeling and updating beliefs about uncertain quantities. While the other approaches see the probability as the limit in the frequency of occurrence as the number of samples goes towards infinity, the Bayesian approach sees the probability as the measure of belief in the occurrence of events [117]. The Bayesian approach can be summarized in the following four steps:

1. Specify a *prior* probability distribution.
2. Obtain data.
3. Update the prior distribution with the data by applying the Bayes' theorem to obtain the posterior.
4. Analyze the posterior distribution, apply the results and return to step 1.

The *prior* mentioned in the first step is a subjective belief about a studied parameter. This prior distribution can be either informative or uninformative [118]. *Informative priors* provide relevant information that biases the estimated parameter towards specific values. In contrast, *uninformative priors* do not offer any meaningful information and are generally used when there is little prior knowledge available about the distribution of an estimated parameter. Informative priors help to constrain the range of possible values for a parameter, while uninformative priors allow the data to drive the analysis with minimal influence from prior beliefs. Common choice for the uninformative priors are normal distributions centered at 0 with some standard deviation σ [119].

The Bayes' theorem from the third step is formally stated in Equation 4.5. Here, h represents a hypothesis about some unknown elements (e.g. the model parameters, value function, policy, etc.), about which there is some prior belief. $\mathcal{D}$ are some data (e.g. the history of previous state-action pairs) that are used to update the belief about h. Following these definitions, $P(h)$ represents the *prior* and $P(\mathcal{D}) = \int_h P(\mathcal{D}, h')dh'$ is the *evidence*. The probability distribution $P(\mathcal{D}|h)$ is termed the *likelihood*, which captures the *aleatoric* uncertainty.Finally, the resulting $P(h|\mathcal{D})$ is called the *posterior*.

$$P(h|\mathcal{D}) = \frac{P(\mathcal{D}|h)P(h)}{P(\mathcal{D})} = \frac{P(\mathcal{D}|h)P(h)}{\int_h P(\mathcal{D}|h')P(h')dh'} = \frac{P(\mathcal{D}, h)}{\int_h P(\mathcal{D}, h')dh'} \tag{4.5}$$

Lastly, in the fourth step, the posterior distributions can be analyzed. This means that quantities such as mean, median, variance, quantiles, and others can be derived and used, for example, for obtaining the action performed by the flight controller. In fact, the statistics from the posterior allow for a principled optimization of the exploration/exploitation trade-off, which can lead to higher sample efficiency [120]. Subsequently, the posterior becomes the new prior and the steps are repeated. With more iterations and observed data, the variance in the prior distribution is expected to decrease, indicating greater confidence in its mean value.

In general, updating the posterior $P(h|\mathcal{D})$ is typically a challenging task, as it involves computing the normalizing constant given in Equation 4.6, which assumes that $P(h)$ is parameterized by an unknown parameter vector θ in certain parameter space Θ. Nonetheless, this computation can be alleviated by using *conjugate* distributions, which are a class of distributions where the prior and the posterior belong to the same parametric family [121]. Therefore, updating the posterior probability can be done using closed-form expressions, as only the distribution parameters need to be updated [120]. Moreover, using conjugate priors improves the interpretation, as it is easy to see how the parameters of the prior change after the Bayesian update [121]. Examples of conjugate distribution classes include Beta and Dirichlet distributions and Gaussian Processes.

$$\int_{\Theta} P(\mathcal{D}|h')P_{\theta}(h')d\theta \tag{4.6}$$

BRL can be split into two threads - model-based and model-free [120]. Model-free BRL focuses on directly learning an optimal policy π^* or value function V^* without explicitly modeling the environment's dynamics. It relies on trial and error by collecting data from interactions with the environment and updating beliefs about the optimal policy or value function using Bayesian inference. On the other hand, Model-based BRL involves constructing a model of the environment's dynamics, which captures the transition probabilities $\mathcal{T}$ and rewards $\mathcal{R}$ associated with state-action pairs. It incorporates Bayesian inference to update beliefs about the model parameters and then uses these beliefs to plan and optimize actions, aiming to find an optimal policy based on the model's predictions.

One notable model-free BRL approach is Bayesian Q-learning [122], which systematically optimizes the exploration and exploitation trade-off by representing Q-values as probability distributions. This method explores states more efficiently compared to standard model-free algorithms, albeit with higher computational requirements. On the other hand, BOSS is a model-based BRL algorithm that maintains a distribution over transition probabilities and rewards [123], using candidate policies sampled from the posterior distribution. BOSS is sample-efficient and effective in complex environments, although its performance may be limited by the size of the hypothesis space, the complexity of the environment model, and computational costs.

Other Methods
While this sub-section highlights some of the widely researched uncertainty-aware RL algorithms, it is important to note that this list is not exhaustive. Many other approaches exist, such as the method of Mavor-Parker et al. [124], which uses an ANN with two heads to produce the predicted mean and variance of the next state. The network's variance is a learned estimate of the aleatoric uncertainty, which is updated using the Maximum Likelihood Estimation loss function. This method yields significant enhancements in highly variable and stochastic environments. Moreover, the previously presented methods can be combined. For instance, Hiraoka et al. have merged dropout and bootstrapped Q-values by incorporating several heads with dropout layers [125], resulting in a more accurate approximation of epistemic uncertainty. The updated Q-networks are used as critics with SAC, leading to much faster and significantly improved performance.

Synthesis
Each of these methods holds great promise in enhancing RL techniques for flight control. Unfortunately, the limited scope of this work prevents a thorough evaluation of all of them, thus two methods were selected for further assessment - *Distributional RL* and *Bayesian RL*.

The advantage of Distributional and Bayesian RL is that they explicitly model the whole uncertainty distribution by probabilistic models, while the other methods only approximate uncertainty through variance or visit counts. Furthermore, Distributional RL became popular recently mainly due to its state-of-the-art sample efficiency and asymptotic performance on a wide range of tasks and the ease of implementing risk-sensitive policies [111]. Moreover, a distributional version of SAC was shown to improve the learning consistency when applied to a flight control task [28]. On the other hand, the Bayesian method offers a principled solution to the exploration-exploitation problem by optimizing actions considering uncertainty. Both Distributional and (model-free) Bayesian RL can be utilized to approximate the uncertainty of the value function. However, the techniques employed by each method to achieve this objective are distinct, offering an intriguing opportunity for comparison and contrast.

4.3. Distributional Reinforcement Learning

Any DRL algorithm can be characterized by two aspects - the parameterization of the return distribution, and the distance metric or loss function being optimized [52]. Moreover, the algorithm is often complemented with a risk-sensitivity mechanism. This section was written to explain the theory behind all of these components and to analyze the suitability of DRL methods for autonomous flight control.

DRL replaces scalar value estimates with *return (probabilistic) distribution functions*, which requires a redefinition of RL concepts like the Bellman Equation. Furthermore, when using return distribution functions, it is crucial to choose an appropriate probability metric to measure the difference between distributions. The choice of metrics is essential, as the distributional Bellman operator should be a contraction under the chosen distance metric. This section highlights the core distributional RL methods that result from the different ways the return distribution can be parameterized, the type of distance metric used, and the loss function applied to the Bellman updates.

4.3.1. Distributional Bellman Equation

As discussed in Section 2.2, the Bellman equation provides a compact and efficient way of calculating the expected value of a state in an MDP. The recursive nature of the MDP is leveraged to update the value of a state based on the expected value of its successor states. This allows for a more efficient and computationally feasible way of estimating the value of a state. In the DRL case, the Bellman Equation needs to be adjusted to account for the distribution of the Q-function. In literature, the action-value distribution function is usually denoted with Z and is defined by Equation 4.7.

$$Z(s_t, a_t) \overset{\mathcal{D}}{=} \sum_{k=t}^{\infty} \gamma^{k-t} R(s_k, a_k) \tag{4.7}$$

where $\overset{\mathcal{D}}{=}$ indicates the *equality by distribution*, i.e. the probability distribution of the random variables on both sides of the equation is identical.

Based on Equation 4.7, the translation of the Bellman Equation into distributional settings can be defined as given in Equation 4.8. The distributional Bellman equation states that the distribution of the return $Z(s,a)$ is characterized by three sources of randomness - the reward $R(s,a)$, the transition $(s,a) \rightarrow (s',a')$ and the return of the following state-action pair $Z(s',a')$. In the context of the theoretical outcomes, it is presumed that these three measures are independent.

$$\begin{gathered} Z(s,a) \overset{\mathcal{D}}{=} R(s,a) + \gamma Z(s',a') \\ s' \sim \mathcal{P}(\cdot|s,a),\ a \sim \pi(\cdot|s),\ a' \sim \pi(\cdot|s') \end{gathered} \tag{4.8}$$

Equation 4.8 maps $\mathcal{S} \times \mathcal{A}$ into the space of action-value distributions with finite moments, $\mathcal{Z}$, i.e. $Z : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{Z}$. The space $\mathcal{Z}$ is formally described in Equation 4.9.

$$\mathcal{Z} := \left\{ Z : \mathcal{S} \times \mathcal{A} \rightarrow \mathscr{P}(\mathbb{R}) \mid \mathbb{E}\left[\|Z(s,a)\|^p \right] < \infty \quad \forall (s,a), p \geq 1 \right\} \tag{4.9}$$

Finally, following the optimal policy π^* results in the definition of the distributional Bellman optimality equation, as given in Equation 4.10. This equation can be applied to create DRL algorithms that utilize generalized policy iteration to discover the best policy in model free settings.

$$\begin{gathered} Z(s,a) \overset{\mathcal{D}}{=} R(s,a) + \gamma Z\left(s', \underset{a'}{\text{argmax}}\, \mathbb{E}_{\mathcal{P},R}[Z(s',a')] \right) \\ s' \sim \mathcal{P}(\cdot|s,a),\ a \sim \pi^*(\cdot|s),\ a' \sim \pi^*(\cdot|s') \end{gathered} \tag{4.10}$$

4.3.2. Probability distribution distance metrics

Probability distance metrics are critical in measuring the similarity between probability distributions, which is fundamental for DRL. Moreover, they can considerably influence the performance of the DRL algorithm [52]. First, several mathematical definitions are required. Let the transition operator $P^\pi : \mathcal{Z} \rightarrow \mathcal{Z}$ be defined with Equation 4.11.

$$P^\pi Z(s,a) \overset{\mathcal{D}}{=} Z(s',a') \tag{4.11}$$

Subsequently, using Equation 4.8 and Equation 4.11, the Bellman operator $\mathcal{T}^{\pi} : \mathcal{Z} \rightarrow \mathcal{Z}$ can be defined with Equation 4.12.

$$\mathcal{T}^{\pi} Z(s,a) \overset{\mathcal{D}}{=} R(s,a) + \gamma P^{\pi} Z(s,a) \tag{4.12}$$

Moreover, the *contraction mapping* is defined as follows:

Definition 1. Let (M, d) be a metric space (where M is a set and d is a metric on M) and let $f : X \rightarrow X$ be a mapping. Then, f is called a *contraction* if there exists a fixed constant 0 $\leq h$ < 1 such that

$$d(f(x), f(y)) \leq hd(x,y), \text{ for all } x, y \in M$$

Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence (also known as the relative entropy) is a non-symmetric measure of the dissimilarity between two probability distributions P and Q. It quantifies the expected amount of information loss when using an approximation distribution Q to represent the true distribution P. Mathematically, the KL divergence of two discrete probability distributions P and Q is defined in Equation 4.13, while Equation 4.14 shows the definition for the continuous case. The $p(x)$ and $q(x)$ are the probability density functions of P and Q, respectively.

$$D_{KL}(P \mid\mid Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{4.13}$$

$$D_{KL}(P \mid\mid Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \tag{4.14}$$

The KL divergence is not a true distance metric, as it does not satisfy the triangle inequality and is not symmetric: $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ [126]. However, it possesses several desirable properties, such as being non-negative and equal to zero if and only if P and Q are the same distribution.

The Cramér Distance

The Cramér distance (CD), also known as the Energy Distance, corresponds to the L_p distance for p = 2 [127]. This is mathematically given in Equation 4.15, where F and G are *cumulative distribution functions*.

$$D_C(F,G) = \left(\int_{-\infty}^{\infty} (F(x) - G(x))^2 \, dx\right)^{1/2} \tag{4.15}$$

The distributional Bellman operator $\mathcal{T}^{\pi}$ is proved to be a contraction mapping in CD. Moreover, this metric is the only L_p distance metric that has unbiased sample gradients [127], meaning that the gradient estimates computed using a subset of the samples are not systematically biased relative to the true gradients that would be computed using the entire data set.

The Wasserstein Distance

The Wasserstein distance (WD), also known as the Earth Mover's Distance, quantifies the "minimal work" required to transform one distribution into another [128]. Here the "work" is the product of the distance $d(x,y)$ and the amount of mass transported from x to y. In contrast to the Kullback-Leibler divergence, which solely quantifies the variation in probability, the Wasserstein metric considers the underlying geometry between outcomes.

Given two *cdf* functions F and G defined on a metric space (M, d), the p-Wasserstein distance $W_p(F,G)$ between F and G is the L_p metric on inverse cumulative distribution functions, which are often referred to as *quantile functions* [111]. The formal description of the p-Wasserstein metric is provided in Equation 4.16.

$$W_p(F,G) = \left(\int_0^1 \left|F^{-1}(x) - G^{-1}(x)\right|^p dx\right)^{1/p} \tag{4.16}$$

The distributional Bellman operator $\mathcal{T}^{\pi}$ was proved to be a contraction mapping in the p-Wasserstein distance [107].

Synthesis

All the studied probability metrics are summarized and formally defined in Table 4.1.

Table 4.1: Formal definition of probability metrics used in DRL, adpopted from [52]

Probabilistic metric	Mathematical Definition	$\mathcal{T}^{\pi}$ contraction
KL divergence	$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$	×
Cramér distance	$D_C(F, G) = \left(\int_{-\infty}^{\infty} (F(x) - G(x))^2 \, dx \right)^{1/2}$	✓
Wasserstein distance	$W_p(F, G) = \left(\int_0^1 \left\| F^{-1}(x) - G^{-1}(x) \right\|^p dx \right)^{1/p}$	✓

4.3.3. Distributional RL Algorithms

This subsection will provide insights into the most renowned DRL algorithms.

Categorical DQN

The categorical approach to the distributional representation follows upon the prior research on DRL, which utilized a Gaussian parameterization for the return distribution and implemented the Q-learning or SARSA frameworks to estimate distributions of value functions. The new approach allowed for the integration of Distributional RL with deep RL principles, which was used by Bellemare et al. [107] by joining the categorical representation approach with DQN architecture.

The value distribution function can be represented as a random variable with support on the set of possible returns. The Categorical DQN algorithm discretizes this distribution into a fixed number $N \in \mathbb{N}$ of atoms within the predefined domain $[V_{min}, V_{max}]$. For instance, $C51$ is a version of the Categorical *DQN* algorithm which uses 51 atoms [107]. Let $\{z_i\}_{i=1}^{N}$ be a set of equally spaced atoms, where each z_i represents a possible return value. Mathematically, these supports are defined by Equation 4.17. Subsequently, the value function is evaluated using an ANN architecture to obtain a vector of N logits. This network is formally described by a parametric model $\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^N$. In order to obtain probabilities, the softmax function is applied to the final output of the network, as given by Equation 4.18.

$$z_i = V_{min} + i \cdot \Delta z, \quad \text{where } \Delta z = \frac{V_{max} - V_{min}}{N - 1} \tag{4.17}$$

$$Z_\theta = z_i \quad \text{w.p.} \quad q_i(s, a) := \frac{e^{\theta_i(s,a)}}{\sum_j^N e^{\theta_j(s,a)}} \tag{4.18}$$

Discretizing the approximate distribution is a computationally efficient approach, but it requires an extra step to align the supports, since they become disjointed between two Bellman updates. Hence, both the Bellman update and the projection step are performed for each atom value. The Categorical DQN approach also employs an ϵ-greedy policy, similar to DQN [38, 62]. Furthermore, despite the availability of distributional data, the C51 algorithm maximizes only the expected value, much like the nominal DQN approach. Nevertheless, C51 surpassed the previous state-of-the-art by a large margin in a number of Atari games, specifically on the Arcade Learning Environment (ALE). Consequently, the authors concluded that learning distributions matters in the presence of approximations, as it can lead to higher sample efficiency.

Another significant theoretical advancements introduced in the C51 study was the verification that the distributional Bellman operator is a maximal contraction of the p-Wasserstein metric between probability distributions. However, the C51 algorithm did not apply this metric directly to establish the loss function. Rather, it utilized the KL-divergence to measure the distance between the updated Bellman distribution, as stochastic gradient methods generally cannot minimize the Wasserstein metric [127].

Quantile Regression

Dabney et al. [110] designed an improved framework that bridged the gap between Wasserstein-metric theory and practical concerns, as described in the categorical approach. Their QR-DQN algorithm employs *quantile regression* (QR) to estimate the Q-value distributions for each action at each state in RL tasks by adjusting the return distribution stochastically, aiming to minimize the Wasserstein distance to the target distribution. While the categorical approach parameterizes the distribution by optimizing N variable probability values at fixed locations, the QR approach optimizes the variable location of N fixed, uniform probability values (i.e. quantile fractions). The discrete cumulative probabilities of the *cdf* are denoted by $\tau_i = \frac{i}{N}$ for $i = 1, \ldots, N$. QR-DQN outperforms C51 by not being constrained to a predefined value range, while it also does not require any projection. The probability representation used in QR-DQN is discretized by N Dirac δ-function expressing the quantile targets, as expressed by Equation 4.19.

$$Z_\theta(s,a) := \frac{1}{N}\sum_{i=1}^{N}\delta_{\theta_i(s,a)} \tag{4.19}$$

Z_θ can be interpreted as a regular Q-Value that is calculated by averaging the estimated Dirac $\delta(z)$ functions (defined at locations $z \in \mathbb{R}$) generated by an ANN. The $Z_\theta \in \mathcal{Z}_Q$ function maps each state-action pair (s, a) to a uniform probability distribution supported on $\{\theta(s,a)\}$.

Finally, QR is employed to minimize the 1-Wasserstein distance between the distribution at a given time step t and the distribution that results from updating the Bellman equation. However, the *quantile regression loss* is generally not smooth at zero, which could deteriorate the performance when using non-linear function approximation [110]. Therefore, Dabney et al. propose to use a modified quantile loss, named *quantile Huber loss*, given by Equation 4.20. This quantile Huber loss is simply the asymmetric variant of the *Huber loss*, defined by Equation 4.21 [129].

$$\rho_\tau^\kappa(u) = \left|\tau - \delta_{\{u<0\}}\right| \mathcal{L}_\kappa(u) \tag{4.20}$$

$$\mathcal{L}_\kappa(u) \begin{cases} \frac{1}{2}u^2 & \text{if } |u| \leq \kappa \\ \kappa(|u| - \frac{1}{2}\kappa) & \text{otherwise} \end{cases} \tag{4.21}$$

Here, κ determines the point at which the loss function transitions from the quadratic to linear behavior, where the Huber loss is a squared loss in the interval $[-\kappa, \kappa]$ and an absolute loss otherwise. u is the temporal difference error (i.e. $u = r + \gamma\theta_j(s', a') - \theta_i(s, a)$. Hence, the quantile Huber loss for all N quantiles can be defined with Equation 4.22.

$$\mathcal{L}_{QR}^{\tau} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\rho_{\tau_i}^{\kappa}\left(r + \gamma\theta_j(s', a') - \theta_i(s, a)\right)\right] \tag{4.22}$$

Subsequently, the action that is chosen for the next state is the greedy action with respect to the mean of the next state-action value distribution, as given by Equation 4.23.

$$a' = \arg\max_{a'} = \mathbb{E}_{z\sim Z(x',a')}[z] \tag{4.23}$$

Implicit Quantiles

Dabney et al.'s *IQN* method expands QR-DQN beyond training on a limited set of quantiles to encompass the entire continuous quantile function of the return distribution [111]. In their method, the quantiles τ_i are first sampled from a uniform distribution $U([0,1])$, after which they are input to an *implicit quantile network* (IQN). This results in a continues mapping between probabilities and return distributions.

The IQN is a deterministic parametric function that is trained to reparameterize samples from a base distribution $\tau \sim U([0,1])$ into the corresponding quantile values of a target distribution. This network comprises of several layers with different functions and it can be represented by a tuple (f, Ψ, Φ). Ψ is the layer that encodes the state and Φ approximates the values of N sampled quantiles τ. More specifically, the sampled quantiles τ are fed through a layer known as a cosine-embedding layer, as given by Equation 4.24. Here, w are the weights of the network, b are its biases and C are the embedding dimensions. Moreover, σ is the activation function, which can be ReLU as used by [111], although [116] suggests that the Sigmoid function enhances the stability of the learning process.

$$\phi_j(\tau) = \sigma\left(\sum_{i=0}^{C-1}\cos(\pi i\tau)w_{ij} + b_j\right) \tag{4.24}$$

Finally, f takes in the product of an element-wise Hadamard multiplication (denoted with $\odot$) of state embeddings and the approximated values of τ, i.e. $\Psi \odot \Phi$. The result of these operations is the quantile function Z, given by Equation 4.25. This entire architecture is summarized in Figure 4.4, which assumes an

$(i+1)$-dimensional state space, an j-dimensional action space and the input of $k+1$ quantile samples.

$$Z_\tau(s,a) \approx f\left(\psi(s), \phi(\tau)\right)_a \tag{4.25}$$

Dabney et al. [111] then integrated the aforementioned network with DQN and trained the network using off-policy TD-targets. There are several advantages of IQN compared to QR-DQN. First, it is a generalization that is not contingent on the discrete number of quantiles, but instead relies on the size and training of the network. This characteristic facilitates the tuning of the number of τ samples used per update. Additionally, the formulation provides a simple means of implementing risk-sensitive policies through the incorporation of a distortion mapping to the uniform distribution that the samples are drawn from.

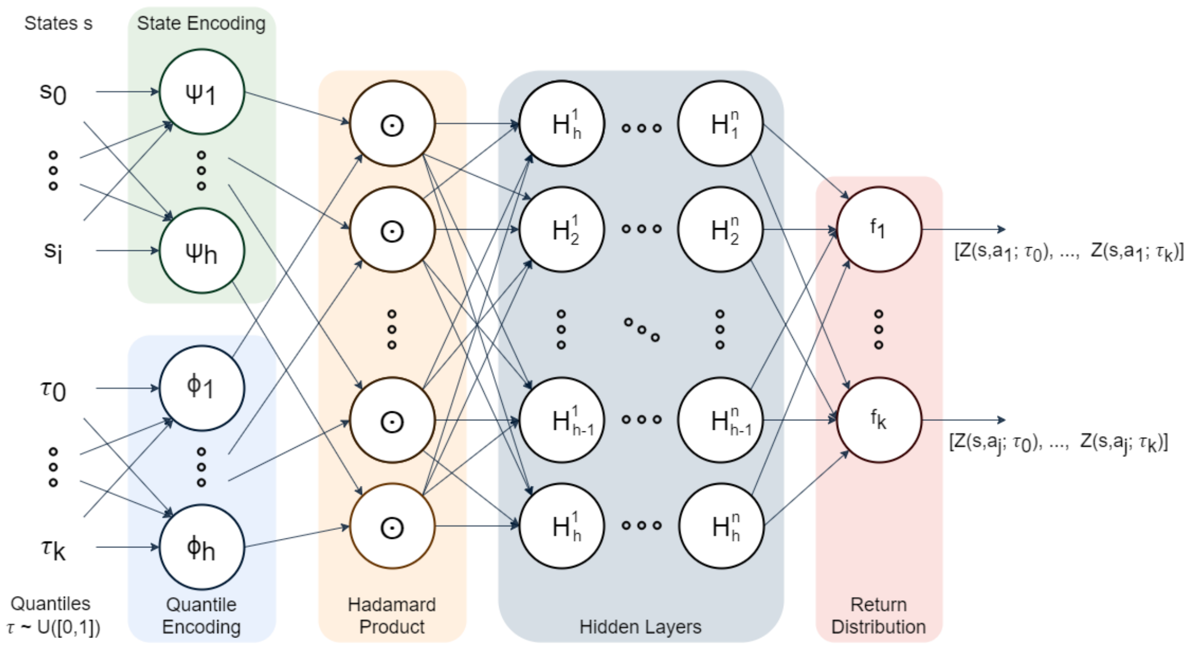


Figure 4.4: The IQN architecture, as described in the work of Dabney et al. [111].

Fully Parameterized Quantile Functions

IQN could approximate the full quantile function only with an infinite amount of network capacity and an infinite number of quantiles, which is impractical. The sampling method in IQN primarily aids in training the implicit quantile network rather than approximating the quantile function. Therefore, it does not guarantee better quantile function approximation than fixed probabilities in QR-DQN. The Fully Parameterized Quantile Function (FQF) [130] addresses this by fully parameterizing the quantile function. In contrast to QR-DQN and IQN, where quantile fractions are either fixed or sampled and only the corresponding quantile values are parameterized, FQF jointly parameterizes both the quantile fractions and their corresponding quantile values. This requires training two ANNs in a joint manner to estimate the distribution of returns:

1. The Fraction Proposal Network (FPN) that outputs a quantile fraction set for each state-action pair.
2. The Quantile Value Network (QVN) that maps probabilities to quantile values.

In general, the FQF estimates N adjustable quantile values for N adjustable quantile fractions to approximate the quantile function defined in Equation 4.26.

$$F_Z^{-1}(p) := \inf\{z \in \mathbb{R} : p \leq F_Z(z)\} \tag{4.26}$$

A weighted combination of N Dirac functions is used to estimate the return distributions, as given by Equation 4.27.

$$Z_{\theta,\tau}(x,a) := \sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i)\delta_{\theta_i(x,a)} \tag{4.27}$$

The FPN takes the state embeddings as the input and outputs N fraction proposals. However, the output proposals need to be arranged in ascending order (i.e. $\tau_{i-1} < \tau_i$) and need to be confined in the interval $[0, 1]$, with $\tau_0 = 0$ and $\tau_N = 1$. Therefore, a *cumulated softmax* layer is applied to sort the output proposals. Specifically, if $q \in \mathbb{R}^N$ is the output of the softmax layer, then $q_i \in (0, 1)$, $i \in [0, N-1]$ and $\sum_{i=0}^{N-1} q_i = 1$. Defining τ_i as $\tau_i = \sum_{j=0}^{i-1} q_j$ for $i \in [0, N]$ then assures that $\tau_i < \tau_j$ for $\forall i < j$ and $\tau_0 = 0$, $\tau_N = 1$.

The FPN is trained to minimize the 1-Wasserstein distance between the approximated and true distribution. The self-adjusting fractions improve the accuracy of the true distribution approximation compared to fixed or sampled fractions. Nevertheless, computing the 1-Wasserstein distance without bias is often infeasible in practice, hence Yang et al. propose to minimize this metric by iteratively applying gradients descent to the weights of FPN (without computing the 1-Wasserstein, see Equation 4.28).

$$\frac{\partial W}{\partial \tau_i} = 2F_Z^{-1}(\tau_i) - F_Z^{-1}(F_Z^{-1}(\hat{\tau}_i)) - F_Z^{-1}(\hat{\tau}_{i-1}) \quad \text{where} \quad \hat{\tau}_i = \frac{\tau_i + \tau_{i+1}}{2} \tag{4.28}$$

Subsequently, the set of quantile fractions is forwarded to the QPN, which is simply the IQN presented earlier. This network is then trained with the quantile Huber loss to map probabilities to quantile values. Hence, this architecture can be represented as IQN with an additional fully-connected multi-layer perceptron (MLP) layer embodying the FPN. The benefit of including the FPN layer in the IQN architecture to optimize the quantile locations is depicted in Figure 10.1 and Figure 4.6.

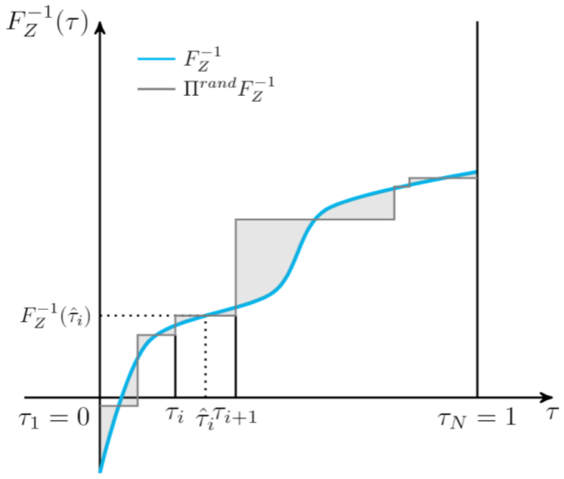


Figure 4.5: Randomly chosen τ with larger 1-Wasserstein error

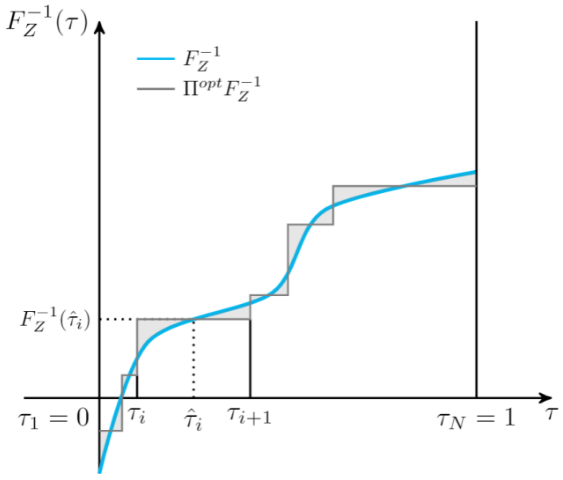


Figure 4.6: Finely-adjusted τ with minimized 1-Wasserstein error.

Summary
As mentioned, DRL methods are defined by the return parametrization and the distance metric. These two aspects are summarized for the discussed methods in Figure 4.7 and Table 4.2, respectively.

Table 4.2: State-of-the-art DRL algorithms and their key characteristics, adopted from [52].

Algorithm	Probability Distribution Representation	Probability Metric
DQN	Expectation (non-distributional)	L1 metric
CDQN	Categorical PDF	KL divergence
QR-DQN	Discrete Quantile Fractions	Wasserstein distance
IQN	Continuous Quantile Fractions (drawn from $\mathcal{U}$([0,1]))	Wasserstein distance
FQF	Continuous Quantile Fractions (sampled by a DNN)	Wasserstein distance

4.3.4. Uncertainty Management in Distributional Reinforcement Learning

Replacing the expected returns with value distribution functions enables designing more sophisticated strategies that take into account the entire distribution and facilitate the use of risk-sensitive policies. C51

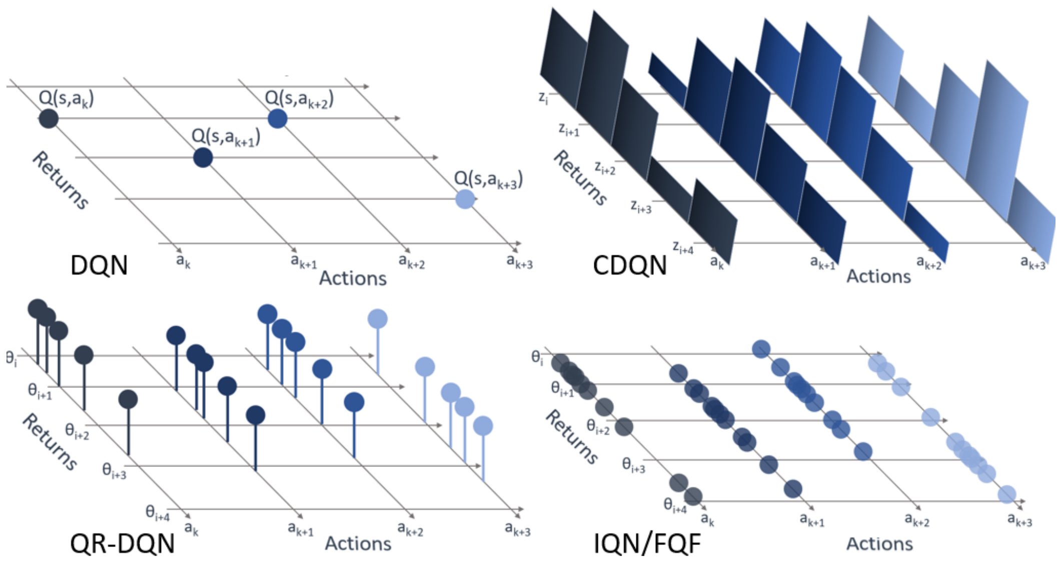


Figure 4.7: Different parametrizations of the return distribution, adapted from [111]

and QR-DQN have shown significant improvements in sample efficiency, learning stability, and overall performance by approximating complete return probability distributions. However, both methods exclusively utilized the distribution's expectation for action selection, thereby neglecting the wealth of information stored within the approximated distribution. The IQN architecture, on the other hand, allows for the application of *risk distortions*, adjusting the level of risk aversion in decision-making by manipulating the distribution from which quantiles are sampled. Specifically, convex risk distortion function will lead to risk-seeking policies, whereas concave functions will lead to risk-averse policies. The policy then follows the expectation of the distribution resulting from applying one of these functions to the sampling distribution. Four examples of risk-distortion functions are presented in Equation 4.29 to Equation 4.32 [111].

$$\text{CVaR}(\eta,\tau) = \eta\tau \qquad (4.29) \qquad\qquad \text{Wang}(\eta,\tau) = \Phi(\Phi^{-1}(\tau)+\eta) \qquad (4.30)$$

$$\text{CPW}(\eta,\tau) = \frac{\tau^{\eta}}{\left(\tau^{\eta}+(1+\tau)^{1/\epsilon}\right)} \qquad (4.31) \qquad\qquad \text{Pow}(\eta,\tau) = \begin{cases} \tau^{1/(1+|\eta|)} & \text{if } \eta \geq 0 \\ 1-(1-\tau)^{1/(1+|\eta|)} & \text{otherwise} \end{cases} \qquad (4.32)$$

The η represents a *risk-tendency parameter* and Φ denotes the standard Normal cumulative distribution function. In *Wang* and *standard power formula* (Pow), the sign of η determines the policy's risk-sensitiveness, where $\eta > 0$ leads to risk-averse behaviour for Pow and risk-seeking for Wang. Alternatively, *Conditional Value-at-Risk* (CVaR) manipulates the risk-sensitivity by changing the sampling distribution from $\tau \sim U([0,1])$ to $\tau \sim U([0,\epsilon])$. Finally, *Cumulative Probability Weighting* (CPW) assigns higher weights to lower quantiles and lower weights to higher quantiles, promoting risk-averse behavior. The choice of the risk distortion and corresponding hyperparameters highly depends on the environment, although risk-averse policies were observed to outperform risk-neutral and risk-seeking settings [111].

In exploring techniques to address uncertainty, Cheng et al. employed the lower Tail Conditional Variance (TCV) of the learned return distribution as an estimate of aleatoric uncertainty, and applied *exponentially weighted average forecasting* (EWAF)[1] to dynamically adjust the risk-tendency of CVaR according to the estimated level of uncertainty [131]. Furthermore, Mavrin et al. [132] propose to further enhance the efficiency of the QR-DQN by implementing two additional components - a decaying schedule to suppress the aleatoric uncertainty and an exploration bonus from upper quantiles of the learnt distribution. Again, this exploration bonus is applied through TCV, which is defined in Equation 4.33. This approach is also referred to as *optimism in the face of uncertainty*.

$$TCV_x(\theta) = Var(\theta - \bar{\theta} | \theta > x) \qquad (4.33)$$

The Method of Mavrin et al. significantly outperformed the original QR-DQN in Atari 2600.

[1]EWAF is a time series analysis technique that calculates a weighted average of past observations, with more recent observations given greater weight than older ones, to predict future values.

4.3.5. Distributional RL with Continuous Actions

The DRL algorithms described so far are value-based methods that derive their policies from the value distribution. Like traditional value-based RL methods, it is challenging to apply these policies to high-dimensional action spaces since an extra optimization problem is required to determine the best action. To address this issue in continuous action spaces, actor-critic architectures can be employed. In this setup, the critic estimates the complete value distribution function using a distributional approach.

For example, the *Distributed Distributional DDPG* (D4PG) algorithm has shown success in estimating rich return distributions and solving complex continuous control tasks [112]. It incorporates a distributional actor-critic framework, with notable improvements in sample efficiency attributed to the distributional representation. Specifically D4PG uses Categorical DQN as a critic.

Alternatively, Distributional SAC (DSAC) was proposed [116], where an IQN critic was implemented in SAC to improve its learning performance. Indeed, a DRL-based flight controller proposed by [28] was built upon the foundation of DSAC, which was motivated by several factors. Firstly, combining a probabilistic view over action (as SAC represent its policy as a distribution) with modelling the distribution of returns has been demonstrated as highly effective in learning high-dimensional continuous control tasks [116]. Furthermore, DSAC demonstrated improved sample efficiency when compared with the original SAC, TD3 and TD4, a distributional version of TD3 [116]. Lastly, DSAC provides a comprehensive framework for incorporating risk-sensitive learning within the distributional soft policy iteration architecture. These arguments render it particularly intriguing for the present study as well.

The DSAC architecture requires the reformulation of the Bellman operator into its distributional soft version, which is provided in Equation 4.34. In line with SAC, α represents the entropy temperature parameter that controls the exploration-exploitation trade-off.

$$\mathcal{T}_{DS}^{\pi} Z_{\pi}(s,a) \overset{\mathcal{D}}{:=} R(s,a) + \gamma \left[Z(s',a') - \alpha \log \pi_w(a'|s') \right] \quad | \quad s' \sim \mathcal{P}(\cdot|s,a), a' \sim \pi(\cdot|s') \tag{4.34}$$

DSAC uses a distributional soft value network $Z_\tau(s,a;\theta)$ as the critic and a stochastic policy network $\pi_w(a|s)$ as the actor. Akin to SAC, DSAC uses double networks parameterized by $\theta_k,\ k=1,2$ to overcome the overestimation. Based on Equation 4.34, the TD error (between two quantile fractions $\hat{\tau}_i$ and $\hat{\tau}_i$) used to update the Z_τ estimates can be redefined as provided in Equation 4.35.

$$\delta_{ij}^{k} = r + \gamma \left[\min_{k=1,2} Z_{\hat{\tau}_i}(s',a';\bar{\theta}_k) - \alpha \log \pi(a'|s';\bar{w}) \right] - Z_{\hat{\tau}_j}(s,a;\theta_k) \tag{4.35}$$

Here, $\hat{\tau}_{i,j} = (\tau_{i,j} + \tau_{i+1,j+1})/2$, $\bar{\theta}_k$ is a parameter vector of the target quantile value distribution network and $\bar{w}$ is a parameter vector of the target policy. Quantile regression is used to train θ by minimizing the weighted pairwise Huber regression loss of different quantile fractions, which results in the critic loss defined in Equation 4.36 [110]. Like in SAC, the target double networks are then softly updated towards the current networks using the Polyak step-size ζ to ensure training stability.

$$\mathcal{L}_Z(\theta) = -\sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) Z_{\tau_i}(s,a;\theta) \tag{4.36}$$

Finally, DSAC provides a unified framework that can handle several risk functionals. Specifically, it uses a *risk measure function* $\Psi : \mathcal{Z} \to \mathbb{R}$, which is then used to define the *risk soft action-value* as $Q^r(s,a) = \Psi\left[Z(s,a)\right]$. This means that the expectation of the (soft) action-value distribution $Z(s,a)$ can be seen as a risk-neutral measure function with $\Psi[\cdot] = \mathbb{E}[\cdot]$, but Ψ can be also replaced by any of the risk distortion functions discussed earlier. The distributional policy loss function accounting for the distorted action-value Q_r can then be defined by Equation 4.37 .

$$\mathcal{L}_\pi(w) = \mathbb{E}\left[Q_\theta^r(s,\tilde{a}_w(s)) - \alpha \log \pi_w(\tilde{a}_w(s)|s)\right] \quad \text{with} \quad Q_\theta^r(s,\tilde{a}_w(s)) = \Psi\left[\min_{i=1,2} Z_{\theta_i}\left(s,\tilde{a}_w(s)\right)\right] \tag{4.37}$$

The described modifications to SAC can be added to Figure 3.3, which results in the DSAC architecture visualized in Figure 4.8. Specifically, the critic was replaced by a distributional critic and a risk measure function Ψ was added, which feeds the policy loss $\mathcal{L}_\pi$ with a risk distorted expectation Q^r.

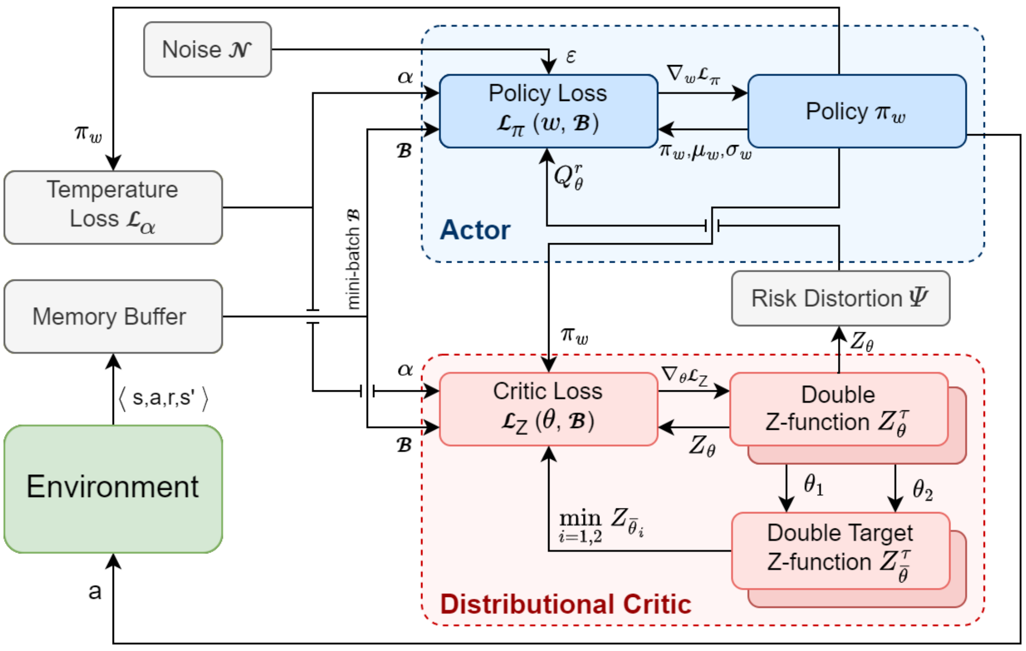

Noise $\mathcal{N}$
ε
α
Policy Loss $\mathcal{L}_\pi(w, \mathcal{B})$
$\nabla_w \mathcal{L}_\pi$
Policy π_w
π_w, μ_w, σ_w
$\mathcal{B}$
Q_θ^r
π_w
Temperature Loss $\mathcal{L}_\alpha$
Actor
mini-batch $\mathcal{B}$
Memory Buffer
Risk Distortion Ψ
Z_θ
π_w
α
Critic Loss $\mathcal{L}_Z(\theta, \mathcal{B})$
$\nabla_\theta \mathcal{L}_Z$
Double Z-function Z_θ^τ
Z_θ
$\langle$ s,a,r,s' $\rangle$
Environment
θ_1
θ_2
$\min_{i=1,2} Z_{\bar\theta_i}$
Double Target Z-function $Z_{\bar\theta}^\tau$
a
Distributional Critic

Figure 4.8: Distributional SAC (DSAC) Architecture.

4.3.6. Applications of Distributional RL for Control

To conclude this section, several application of DRL in control and navigation will be briefly outlined.

The aforementioned theoretical work on DSAC applied to the attitude control of a jet aircraft demonstrated a great potential for the application of DRL in flight control in uncertain environments [28]. This potential was also highlighted by an application-oriented research, where QR-DQN was applied to control superpressure stratospheric balloons in a station-keeping task [133]. The real-time distributional controller was shown to outperform the balloon's previous algorithm and to be robust to the natural diversity in stratospheric winds. In another flight-control application, an Adaptive Risk-Tendency IQN (ART-IQN) is proposed and deployed on a nano quadrocopter UAV, whose task was to learn to navigate in an a-priori unknown and partially-observable cluttered environment. Within the ART-IQN framework, the agent's risk-tendency is dynamically adjusted based on estimated uncertainty, allowing for an adaptive balance between efficiency and safety. A similar approach was also applied outside of the flight control domain, where an IQN-based architecture learned adaptive policies that dynamically regulate their level of conservatism according to the desired level of safety and comfort for passengers of an autonomous vehicle [134]. This decreased uncomfortable safety interference in noisy environments as compared to DQN-based algorithms.

In another research, IQN was integrated into a multi-agent RL (MARL) to control a bus system and resolve the bus bunching problem[2] [135]. The risk-sensitive distributional version of MARL showed improved robustness in handling various extreme events, such as traffic state perturbations, service interruptions, and demand surges, resulting in a more efficient and reliable system.

DRL proved to be effective in multiple robotics applications as well. For example, an algorithm called Universal Distributional Q-learning (UDQL) was used in a stable steering policy for a robot-assisted flexible needle insertion task [136]. Compared with previous methods, UDQL showed increased accuracy and robustness to uncertain needle-tissue interactions, while providing distributional information to clinicians to assist their decisions. Another research proposed a Quantile QT-Opt (Q2-Opt) for real vision-based robotic grasping tasks. Q2-Opt demonstrated superior success rate and sample efficiency, while allowing for the incorporation of risk-sensitive policies. Similarly, a proposed Risk-Conditioned DSAC achieved improved performance and safety in partially observed navigation tasks when deployed on a navigation robot [137]. In another research on robust route planing for mobile robots, the proposed framework uses QR-DQN to learn the environmental stochasticity, while it can use different policies based on user's preferences

[2]Bus bunching refers to the phenomenon where scheduled buses, instead of maintaining regular spacing, end up arriving at a stop or along a route in close succession due to delays or other factors.

on route robustness. Last but not least, QRDQN was applied in autonomous driving in intersection crossing scenarios, which resulted in enhanced safety in environments with inherent uncertainty about other participants' behaviors while avoiding too conservative driving [138].

4.4. Bayesian Reinforcement Learning

This section presents an overview of various BRL algorithms developed to address uncertainty and generalization challenges in RL. This work primarily adopts the BRL taxonomy proposed by Ghavamzadeh et al. [120] as visualized in Figure 4.9, but it recognizes the existence of numerous alternative categorizations. The algorithms are divided into bandit problems, model-based and model-free approaches, and risk-aware extensions of these methods that incorporate uncertainty information in action selection. This categorization does not capture all the possible BRL approaches, which is highlighted by the empty branch.

Multi-Armed Bandit (MAB) problems are a set of slot machines or arms with unknown reward probabilities. The goal is to find an optimal strategy that maximizes cumulative rewards by balancing exploration of different bandits and exploitation of the most promising ones. As these approaches are designed to solve problems in discrete settings with small action and state spaces, they are not relevant for this work. However, Thomson sampling will be discussed briefly to demonstrate the power of Bayesian approaches.

In model-based BRL, the prior information can be expressed over the parameters of the Markov model. In contrast, in model-free BRL, the priors are expressed over the value functions or policies. This section will emphasize model-free methods for similar reasons as discussed in Chapter 3. Finally, risk-sensitive BRL methods are those that complement model-based or model-free BRL methods with some performance criteria that take into account not only the expected return, but also some additional statistics of it.

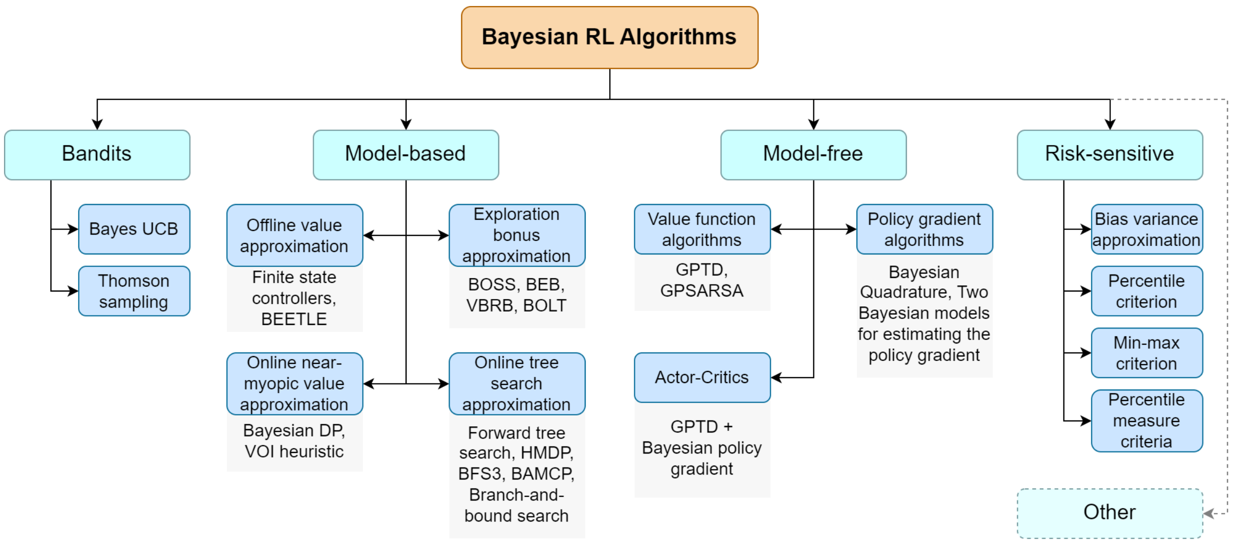

Bayesian RL Algorithms
Bandits
Model-based
Model-free
Risk-sensitive
Bayes UCB
Thomson sampling
Offline value approximation
Finite state controllers, BEETLE
Exploration bonus approximation
BOSS, BEB, VBRB, BOLT
Online near-myopic value approximation
Bayesian DP, VOI heuristic
Online tree search approximation
Forward tree search, HMDP, BFS3, BAMCP, Branch-and-bound search
Value function algorithms
GPTD, GPSARSA
Policy gradient algorithms
Bayesian Quadrature, Two Bayesian models for estimating the policy gradient
Actor-Critics
GPTD + Bayesian policy gradient
Bias variance approximation
Percentile criterion
Min-max criterion
Percentile measure criteria
Other

Figure 4.9: Taxonomy of BRL algorithms, adapted from [120]

4.4.1. Thomson Sampling

Thomson sampling (TS) is one of the most classical BRL algorithms designed to address the exploration-exploitation dilemma for Multi Armed Bandit (MAB) problems using probability matching [139]. Let the probabilities for the outcomes of the actions $P(\cdot|a)$ be parameterized by an unknown vector θ, which will be denoted $P_\theta(\cdot|a)$, and let P_{post} represent the posterior distribution of θ given the observations up to time t. Subsequently, a value of $\hat{\theta}$ is sampled from P_{post} at each time step, which is then used to define a model for selecting the optimal action. This model aligns the probability of choosing an action with the posterior probability of that action being optimal based on the given $\hat{\theta}$. The outcome observation is subsequently used to update the posterior distribution P_{post}.

The exceptional empirical performance of TS has recently garnered significant attention [120]. In fact, TS has been known to achieve optimal and near optimal regret bounds in stochastic bandits [140] and MDPs [141]. This has resulted in its adoption in multiple industrial applications, such as online advertising [120].

However, similar to other Bayesian methods, maintaining the posterior distribution of value-action functions required by TS is generally intractable for tasks with high-dimensional state-action spaces [142]. As a result, TS has predominantly been applied to simple, low-dimensional applications. Various approaches have been explored to address this limitation, such as using variational distributions as an approximation for the posterior. For instance, Aravindan and Lee [142] developed a variational TS approximation for DQNs which uses a DNN whose parameters are perturbed by a learned variational noise distribution.

Furthermore, although TS effectively minimizes exploration costs for identifying the optimal action, it may not be suitable for time-sensitive learning problems. In such cases, prioritizing shorter-term performance and identifying a satisfying action rather than the exact optimum becomes more practical. Algorithms that explore less aggressively can outperform TS in these scenarios [143]. An example of such a situation in flight control settings could involve an unexpected subsystem failure that alters the aircraft's dynamics, where the primary focus would be on rapidly learning to maintain a safe and stable flight rather than optimizing flight performance under the new dynamics.

4.4.2. Model-based Bayesian Reinforcement Learning

Model-based BRL methods maintain a posterior distribution over the environment model parameters. A helpful perspective on this problem is to consider the parameters as unobservable states of the system. This way, the MDP with unknown parameters can be framed as a problem of planning under uncertainty, using the POMDP formulation. In the POMDP setting, the challenge of handling uncertainty with states is addressed through the utilization of *belief states* $b(s)$. For model-based BRL, this can be translated into a belief about the model parameters $b(\theta)$, which results in a so called *belief-MDP* representation [120].

Model-based BRL can be further divided into different approaches. First, *offline value approximation* methods refer to the algorithms that aim to compute an optimal policy a priori for any possible state and posterior. These methods include, for example, the Bayesian Exploration Exploitation Trade-off in LEarning (BEETLE), which uses Bayesian techniques to optimize policy offline and then monitors the belief and action selection online [144]. However, these methods are limited to small simulation domains [120]. Alternatively, *exploration bonus approximation* methods employ the notion of *optimism in the face of uncertainty*, using optimistic models of the MDP to guide action selection and gather rewards or gain information for improving the model. These methods include algorithms such as BOSS (see Section 4.2) or Bayesian Exploration Bonus (BEB), which adds an exploration bonus term in the value function [145].

The *online near-myopic value approximation* and *online tree search* methods update the Bayesian model on an online step-by-step basis. The near-myopic approaches, including Value of (Perfect) Information (VOI) methods [122], prioritize short-term rewards when making decisions. The primary drawback is that the immediate value gained from information may offer only a narrow perspective on the potential information benefits of specific actions. Instead, to choose actions based on a more comprehensive understanding of model uncertainty, an alternative approach is to conduct a tree search within the hyper-state space [120]. Such algorithms incrementally search through a tree structure, with nodes representing different states and priors, and edges denoting the transitions between them [146]. However, considering the emphasis of this work on model-free offline approaches, further exploration of these methods will not be pursued.

4.4.3. Gaussian Processes

Gaussian processes (GPs) are one of the most favored prior choices in model-free BRL (discussed later in Section 4.4.4), mainly due to their flexibility, interpretability, function approximation power and natural uncertainty quantification. Furthermore, they model normal distributions, which are prevalent due to the *central limit theorem* [147]. For these reasons, they have been also applied in many RL tasks (including robotics and control) [148, 149]. Moreover, as a non-parametric approach, GPs belong to *conjugate family* of distributions, which means that the prior and posterior distributions maintain the same functional form. This allows for analytical calculations and closed-form updates, which heavily simplifies the calculation of the integral in Equation 4.5 [120]. The theory presented in this brief introduction is primarily based on the book *Gaussian Processes for Machine Learning* [150] and on the work of J. Görtler et al. [147].

The most common application of GPs is *regression*, where an infinite number of functions that could fit a given set of training points exist. GPs address this issue by assigning a probability distribution to these functions, with the distribution mean representing the most likely fit [150]. Moreover, taking a probabilistic approach permits the incorporation of prediction confidence into the regression outcome. This characteristic makes GPs an elegant solution for predictive modeling for BRL.

The basic building block of the GPs is the Gaussian distribution (also known as *normal* distribution). This can be extended to multivariate Gaussian distribution for high-dimensional cases, which involve more than one random variable. To define the multivariate Gaussian distribution, two essential elements come into play, namely the mean vector μ and the covariance matrix Σ. Each of the components of μ describes the expected value of the corresponding dimension μ_i. The diagonal of Σ captures the variance along each dimension σ_i^2, while the correlation between different random variables are described by the off-diagonal elements σ_{ij}. By definition, the covariance matrix is always positive semi-definite and symmetric [151]. Mathematically, this can be expressed by Equation 4.38. Visually, an example of a multivariate Gaussian distribution for two variables is presented in Figure 4.10.

$$X = \left[X_1, X_2, \ldots, X_n\right]^T \sim \mathcal{N}(\mu, \Sigma) \tag{4.38}$$

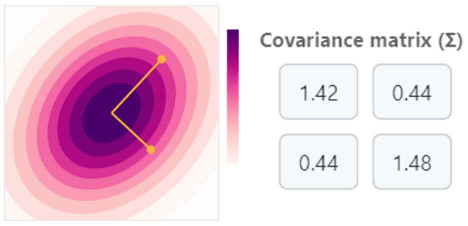


Figure 4.10: Two-variate Gaussian Distribution [147]

As discusses before, GPs are conjugate functions, which means that they are closed under *conditioning* and *marginalization* [147]. This means that if a Gaussian distribution is conditioned or marginalized, the resulting distribution will also be Gaussian, making many problems in ML tractable. To explain the concepts of marginalization and conditioning, consider Equation 4.39.

$$P(X,Y) = \begin{bmatrix} X \\ Y \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}, \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{bmatrix}\right) \tag{4.39}$$

With *marginalization*, it is possible to compute the marginal distribution of the function values at a subset of the input points, without explicitly considering the rest of the input space. This results in a lower-dimensional Gaussian distribution over the function values of interest. For instance, the marginalized probability distribution of random variables X can be determined as X $\sim \mathcal{N}(\mu_X, \Sigma_{XX})$. To marginalize out this probability density, it is necessary to consider all possible outcomes of Y that can jointly lead to the result, as given by Equation 4.40.

$$P_X(x) = \int_y P_{X,Y}(x,y)dy = \int_y P_{X|Y}(x|y)P_Y(y)dy \tag{4.40}$$

Conditioning is a method used to determine the probability of one variable depending on another. This operation is the fundamental aspect of GPs, as it permits Bayesian inference. Mathematically, conditioning is defined by equations Equation 4.41 and Equation 4.42.

$$X|Y \sim \mathcal{N}(\mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(Y - \mu_Y), \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}) \tag{4.41}$$

$$Y|X \sim \mathcal{N}(\mu_Y + \Sigma_{YX}\Sigma_{XX}^{-1}(X - \mu_X), \Sigma_{YY} - \Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XY}) \tag{4.42}$$

Now that the properties of Gaussian distributions were explained, the GPs can be introduced. First, the transition from the continuous viewpoint to the discrete representation of a function is necessary. Rather than finding an implicit function, the aim is to predict function values at specific test points X. Stochastic processes, such as GPs, are fundamentally a set of random variables, with each variable linked to an index i. In this description, this index refers to the i-th dimension of an n-dimensional multivariate distributions. Moreover, the training data will be denoted as Y. The key idea of GPs is to model the underlying distribution of X together with Y as a multivariate normal distribution. This regression problem is treated as *Bayesian inference*, the idea of which is to update the hypothesis as new training data are obtained. As the resulting posterior distribution $P(X|Y)$ is also distributed normally, the goal is to find the μ and Σ of the posterior.

When it comes to μ, the GPs usually start by assuming μ = 0, which simplifies the equations for conditioning [147]. This assumption can be made even if μ is not 0, as the true μ can be added after the prediction step. This rather straightforward process is referred to as *centering*. The tricky part is the covariance matrix Σ, which determines the shape of the distribution and the characteristics of the function. This covariance matrix is modelled with a *covariance function* k, which is also referred to as a *kernel*. The input to a kernel are two points, t, $t' \in \mathbb{R}$, while the output is a scalar value characterizing the similarity between these points, as given by Equation 4.43.

$$k : \mathbb{R}^n \text{ x } \mathbb{R}^n \rightarrow \mathbb{R}, \;\; \Sigma = Cov(X, X') = k(t, t') \tag{4.43}$$

Since the kernel controls the possible shape a fitted function can adopt, it is important to select it in such a way that it produces a matrix that follows the properties of a covariance matrix. Figure 4.11 shows examples of covariance matrices created with some of the most common kernels in GP and their corresponding equations. Many more kernels are used in practice, such as squared exponential, rational quadratic, and others [152]. Moreover, these kernels can be combined together by multiplication or addition to produce more specialized kernels.

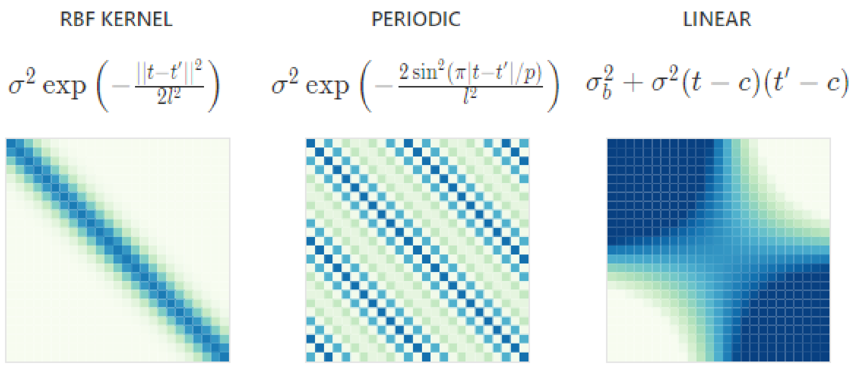

RBF KERNEL
PERIODIC
LINEAR
$\sigma^2 \exp\left(-\frac{||t-t'||^2}{2l^2}\right)$
$\sigma^2 \exp\left(-\frac{2\sin^2(\pi|t-t'|/p)}{l^2}\right)$
$\sigma_b^2 + \sigma^2(t-c)(t'-c)$

Figure 4.11: Common kernels used in Gaussian processes [147]. Dark color signifies high correlation.

Kernels can be categorized into two types: *stationary* and *non-stationary*. Stationary kernels, like the RBF kernel or the periodic kernel, are invariant to translations, and the correlation between two points relies solely on their relative distance. On the other hand, non-stationary kernels, like the linear kernel, do not follow this restriction and are dependent on a fixed location. The kernel's stationary characteristic can be seen in the diagonal bands of its covariance matrix, as seen in Figure 4.11.

To define the *prior* distribution $P(X)$, μ is initially set to 0, as discussed before. Subsequently, the kernel is used to set up the Σ with the dimensions NxN, where N is the number of test points. Once the training data are observed, the *posterior* $P(X|Y)$ can be computed. In the first step, the joint distribution $P(X, Y)$ between the test points X and the training points Y is formed, which generates a multivariate Gaussian distribution with dimensions $|Y|$+$|X|$. Subsequently, *conditioning* is used to find $P(X|Y)$ from $P(X, Y)$. The $P(X|Y)$ distribution forces the set of functions to pass through each training point precisely, which often results in fitting unnecessary complex functions. Additionally, the training points Y have been assumed to be perfect measurements thus far, but this is not a realistic assumption for real-world scenarios where measurement errors or uncertainties are common. To address these issues, an error term $\epsilon \sim \mathcal{N}(0, \psi^2)$ is added to each training point (i.e. Y = $f(X)$ + ϵ). Hence, the joint distribution is modified to Equation 4.44.

$$P(X, Y) = \begin{bmatrix} X \\ Y \end{bmatrix} \sim \mathcal{N}(0, \Sigma) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} + \psi^2 I \end{bmatrix}\right) \tag{4.44}$$

Conditioning also corrects the mean μ' to a non-zero value. Subsequently, the marginalization is used to extract the respective mean function value μ'_i and standard deviation σ'_{ii} = Σ'_{ii} for the i-th test point. The changes in the mean and covariance matrix after the inference are visualized in Figure 4.12 for two different kernels. This figure also stresses out the impact of the kernel choice.

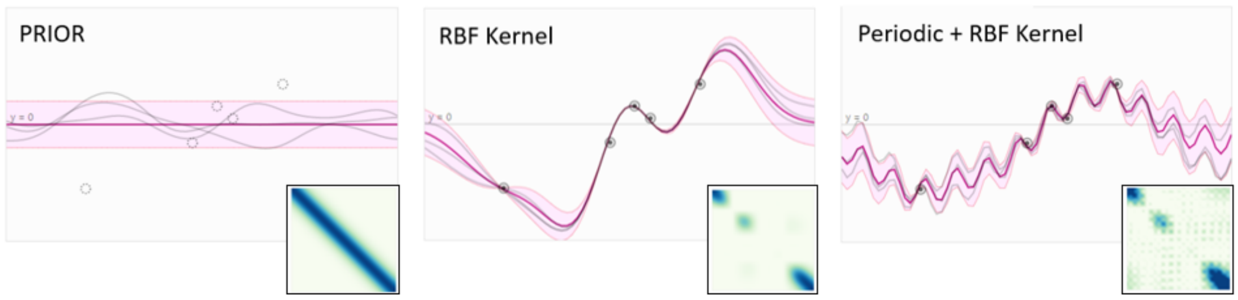


Figure 4.12: Inference for Gaussian processes, adopted from [147]. The squares in the corner of the graphs represent the corresponding covariance matrices.

Finally, the advantages and disadvantages of GPs can be summarized:

Advantages of Gaussian processes

- ✓ *Approximation performance:* As demonstrated by the examples in this section, GPs offer a highly effective means of approximating arbitrary functions.
- ✓ *Uncertainty specification:* GPs capture uncertainty directly. This is not the case for many other function approximators, such as neural networks [100].
- ✓ *Sample efficiency:* GPs are able to learn efficiently from very few observations [153].
- ✓ *Fewer hyperparameters than ANN:* GP typically have fewer hyperparameters to tune compared to ANNs. Only The choice of kernel function and its associated hyperparameters (e.g., length scales l, noise level) determines the behavior of a GP.

Disadvantages of Gaussian processes

- ✗ *Computational time:* Since GPs are non-parametric (although kernels introduce some hyperparameters) they need to consider the entire training dataset whenever making a prediction. As a result, not only must the training data be retained during inference, but the computational expense of making predictions also increases as the number of training examples N grows. In fact, the GPs suffer from *cubic* computational complexity $\mathcal{O}(N^3)$ [154]. This could be problematic for flight control, where quick decisions are required, especially in unexpected events, such as failures. This issue can be party mitigated by sparse GPs techniques such as Fully Independent Training Conditional (FITC) [155], but it comes at a cost of limited expressivity, need of choice of inducing points, difficult hyperparameter tuning and lower approximation performance.
- ✗ *Scalability:* GPs lose efficiency in high-dimensional spaces, i.e. in spaces with high number of features [156]. Unlike ANNs, GPs cannot automatically learn relevant features and they rely on handcrafted kernels to capture dependencies between input variables.
- ✗ *Necessity of a prior:* Incorporating predefined kernels can speed up the learning process, but it can also be a disadvantage if the kernel selection is uncertain. For instance, predicting the structure of the Q-function in a complex flight controller, with intricate flight dynamics, is challenging. As shown in Figure 4.12, the kernel choice greatly affects the function shape. A poor kernel selection can lead to slow convergence or even hinder it. On the other hand, estimators like ANNs offer more adaptability by not relying on prior assumptions, albeit requiring a larger dataset for convergence.
- ✗ *Assumption on a shape of the distribution:* GPs presuppose a Gaussian distribution of uncertainty, which may not align with empirical observations. For example, this assumption does not hold if the outputs are strongly positive or bounded between 2 values.

4.4.4. Model-free Bayesian Reinforcement Learning

Model-free BRL methods typically maintain a posterior distribution over the policy or value function and update it based on observed data, incorporating prior knowledge and updating beliefs through Bayesian inference [120]. The following paragraphs will present a class of value function BRL called GP Temporal Difference (GPTD) and two classes of policy search BRL, including Bayesian Policy Gradient (BPG) and Bayesian Actor-Critic (BAC).

Similar to non-Bayesian methods, value function BRL extracts information about the value function from the noisy samples of the reward signals. However, in addition to providing value estimates through the posterior mean, the Bayesian solution also gives the variance around this mean, which serves as a measure of uncertainty for the value estimates. GPs prove suitable for this problem, which is why they are applied in many model-free Bayesian approaches, such as *GPTD* and *GPSARSA* (which extends GPTD to state-action value functions) [120]. GPSARSA models the return as a random variable $Z(x) = \sum_{t=0}^{\infty} \gamma^t R(x_t)$, expressed as a sum of its mean $Q(x)$ and zero-mean residual $\Delta Q(x)$, with the state-action vector $x \in \mathcal{X} \rightarrow \mathcal{S} \times \mathcal{A}$. Subsequently, the observation process can be described by Equation 4.45.

$$R(x) = Q(x) - \gamma Q(x') + [\Delta Q(x) - \gamma \Delta Q(x')] = Q(x) - \gamma Q(x') + \epsilon(x, x') \tag{4.45}$$

If ϵ is assumed to be IID and Gaussian, it can be modelled as $\epsilon \sim \mathcal{N}(0, \Sigma)$, with $\Sigma = \mathrm{Hdiag}(\sigma_{t+1})\mathrm{H}^T$ (where σ_{t+1} represents a vector of variances for each ϵ_i). Assuming σ to be constant for all ϵ_i simplifies Σ to $\Sigma = \sigma^2 \mathrm{HH}^T$. This results in a complete data model for $R(x)$, given by $R(x) = \mathrm{H}Q(x) + \epsilon$, which can be expanded to Equation 4.46.

$$\underbrace{\begin{bmatrix} R(x_0) \\ R(x_1) \\ \vdots \\ R(x_{t-1}) \end{bmatrix}}_{\text{Observable process}} = \underbrace{\begin{bmatrix} 1 & -\gamma & 0 & \cdots & 0 \\ 0 & 1 & -\gamma & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -\gamma \end{bmatrix}}_{\text{Linear Transformation H}} \underbrace{\begin{bmatrix} Q(x_0) \\ Q(x_1) \\ \vdots \\ Q(x_t) \end{bmatrix}}_{\text{Unknown Function}} + \underbrace{\begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_t \end{bmatrix}}_{\sim \mathcal{N}(0, \Sigma = \sigma^2 \mathrm{HH}^T)} \tag{4.46}$$

Equation 4.46 follows the structure of GP likelihood model, $y = f(x) + \epsilon$, where both equations assume that outputs $R \sim y$ are noisy observations of a latent function $Q \sim f$. Therefore, Equation 4.46 can be directly used for Gaussian regression. GPTD and GPSARSA methods improve upon the data efficiency of TD and SARSA by departing from the contractive nature of Bellman's equation [157]. Nevertheless, they suffer from the GP's disadvantages. For instance, they are computationally infeasible except for small and simple problems [120, 157], hence they require additional online sparsification.

Alternatively, *Bayesian Policy Gradient* (BPG) methods refer to a class of RL algorithms that leverage Bayesian inference and *Bayesian quadrature* (BQ) to estimate the gradients of the policy [120, 158]. These methods aim to optimize the parameters of a policy by iteratively updating them using gradient information obtained through BQ [159]. BQ is a numerical integration technique that uses a probabilistic approach to estimate intractable integrals [160]. Hence, it can be used to approximate numerical integration under a probability distribution using samples drawn from the same distribution. In the context of BPG, BQ can be used to estimate the expected reward of a policy by integrating over the state-action space. This is done by approximating the integrand with GP and then using BQ to compute the integral [158]. Despite their sample efficiency, these methods also suffer from high computational complexity. Some recent works attempt to mitigate this problem, such as *Deep Bayesian Quadrature Policy Gradient* (DBQPG), which promises a computationally efficient high-dimensional generalization of BQ for policy gradient estimation [158]. Furthermore, DBQPR demonstrated more accurate gradient estimation with a significantly lower variance compared to Monte-Carlo methods and showed to be effective on a set of continuous control benchmarks.

Another approach to reduce variance of the policy gradient estimates are *Bayesian Actor-Critic* (BAC) methods, which combine the Actor-Critic architecture with Bayesian inference. The critic in BAC models the action-value function as a GP, which allows the use of Bayes' rule to compute the posterior distribution over action-value functions, conditioned on the observed data [161]. BAC were shown to provide more accurate estimates of the policy gradient than Monte-Carlo methods and BPG [161].

Furthermore, driven by SAC's state-of-the-art performance and the benefits offered by Bayesian approaches, recent research investigates the fusion of these two methods. For instance, an algorithm called PAC-Bayesian Soft Actor-Critic (PACBSAC) provides an alternative for combining Bayesian methods with actor-critic algorithms [162]. PACBSAC approach addresses the training instability in actor-critic algorithms by using a Probably Approximately Correct (PAC) Bayesian bound[3] as the critic training objective. Another research proposed a hierarchical strategy decomposition approach, which separates a complex policy

[3]PAC-Bayes provide an upper bound on the expected difference between the expected error of a Bayesian learning algorithm and its estimated error based on the observed data. This is used to quantify the trade-off between model complexity, data size, and the error in making predictions [163].

into simpler sub-policies and organizes their relationships as Bayesian strategy networks (BSN) [164][4]. The proposed Bayesian soft actor-critic (BSAC) model integrates this approach into the soft actor-critic (SAC) algorithm. Both of these novel methods demonstrate improved sample efficiency on the standard continuous control benchmarks, making them interesting for flight control.

4.4.5. Risk Measures for Bayesian Reinforcement Learning

The methods discussed earlier only optimize the expected return of the policy, but the estimated uncertainty can be also leveraged to minimize risk. The risk in this section refers to performance criteria that consider not only the expected return but also other statistics of it, such as Value-at-Risk (VaR), Conditional VaR (CVaR), etc. Furthermore, work on risk-sensitive BRL aims at addressing risk due to *epistemic* uncertainty, as BRL methods offer a principled framework for evaluating this uncertainty by drawing several MDPs with unknown parameters from the posterior distribution [120].

Percentile criterion, which is mathematically equivalent to VaR, is one of such risk measure. It involves setting a desired risk level ϵ with the goal of ensuring that the policy performs within this risk level or below it with a probability of at least 1-ϵ [166]. Unfortunately, solving such a percentile optimization problem is NP-hard [166]. To improve the tractability, the individual percentiles can be replaced with a *measure* over percentiles [167]. These approaches include *CVaR*, which quantifies the *expected* losses that occur beyond the VaR threshold. Alternatively, a *max-min* criterion can be employed, the goal of which is to maximize the performance under the worst realizable posterior transition probability [120]. However, it's conservative nature may prevent it from exploration of potentially beneficial actions [168].

4.5. Conclusion

This chapter recognized two types of uncertainty - epistemic and aleatoric. Epistemic uncertainty arises from a lack of knowledge, such as an incomplete aircraft dynamics model. This type of uncertainty can be used to enhance the exploration-exploitation trade-off, improving sample efficiency. On the other hand, aleatoric uncertainty originates from stochasticity in the environment, such as turbulence. Modeling aleatoric uncertainty allows RL algorithms to make more informed decisions about risky or uncertain actions, leading to more efficient exploration and exploitation, while improving robustness. This answers **RQ-M-2**.

Next, different RL methods capturing uncertainty were discussed. Count-based methods quantify epistemic uncertainty through state visit counts, MC-dropout and bootstrapping capture epistemic uncertainty by generating multiple stochastic predictions and assessing their variance, DRL directly learns the whole value distributions to represent aleatoric uncertainty, while BRL updates probability distributions using Bayes' rule to infer epistemic uncertainty in variables. This answers **RQ-M-3a**. Furthermore, DRL and BRL were specifically selected for further evaluation to assess their applicability in the context of flight control. Now is the time to assess the suitability of these two methods for the flight control task, ultimately determining which one will be the focus of further practical research. More specifically, DRL will be compared with model-free BRL, as both these approaches can be used to represent uncertainties around Q-functions.

While both DRL and (model-free) BRL can handle continuous Q-functions, DRL focuses on estimating the distribution of Q-values directly, whereas BRL focuses on learning and updating posterior distributions through Bayesian inference. The posterior distribution over the value function or policy in BRL methods naturally captures the full state of knowledge subject to the chosen parametric representation, hence it provides a principled approach to handle epistemic uncertainty. On the other hand, by representing the entire distribution, including its spread and shape, DRL algorithms explicitly account for the inherent randomness and variability in the RL problem. Therefore, DRL captures the aleatoric uncertainty in the limit, but the intermediate distribution models both epistemic and aleatoric uncertainty [132]. It is possible to separate these two kinds of uncertainty and use the epistemic uncertainty improve the exploration characteristics [132]. Finally, given the propensity for randomness to emerge from diverse sources during flight, such as turbulence or sensor noise, it makes sense to model aleatoric uncertainty within an RL flight controller.

The benefits and limitations of both approaches are summarized in a few bullet points below. For Bayesian RL, many of them resemble these presented for GP, as GP are vastly used in model-free Bayesian RL.

Advantages of Bayesian Reinforcement Learning

✓ A principled way for exploration-exploitation trade-off based on the state of knowledge, where the

[4]Bayesian Networks are probabilistic graphical models that represent and reason about uncertain knowledge using a directed acyclic graph and conditional probability distributions [165].

actions are chosen to efficiently reduce the epistemic uncertainty. Together with a proper choice of a prior distribution over the Q-values, this can enhance the sample-efficiency [122, 148].

✓ Assuming a prior on a value function facilitates regularization, as it prevents overfitting to a few data points [161].

Disadvantages of Bayesian Reinforcement Learning

✗ High computational cost, especially with increasing state and action space dimensions, which limits the scalability [169].

✗ The choice of a prior impacts the performance of BRL. Selecting a prior that aligns closely with the true underlying distribution facilitates efficient exploration. However, inaccurate informative or uninformative priors can nullify this advantage and impact the quality of distribution estimates. Furthermore, it can lead to intractable problems, bias the inference, and negatively affect the system as it undergoes variations, such as faults [170, 171, 172]

✗ Despite of its elegance, BRL has not been widely applied in robotics or other autonomous tasks [120] and there is a scarcity of dedicated resources (such as textbooks) that address its theoretical principles [173].

Advantages of Distributional Reinforcement Learning

✓ Significantly higher computational efficiency.

✓ Considerably better scalability, as DRL usually uses function approximation techniques, such as DNN.

✓ Simpler implementation and fewer assumptions, as DRL avoids the specification of the prior.

✓ Suitable for dealing with aleatoric uncertainty, which is prevalent in flight control due to phenomena such as turbulence or noisy sensor measurements. This uncertainty information can then aid in, for example, making risk-sensitive decisions.

✓ DRL has been studied and successfully applied in various continuous control tasks, including flight control, and its theory is well-documented in a dedicated textbook [108] and other literature.

Disadvantages of Distributional Reinforcement Learning

✗ The epistemic uncertainty is not explicitly accounted for, although this uncertainty is useful for improving the sample efficiency by a principled exploration-exploitation trade-off.

✗ The understanding of DRL's empirical effectiveness and the theoretical advantages it holds over expectation-based RL is still limited [114].

Considering the advantages and disadvantages, it can be concluded that model-free BRL, particularly with a carefully selected prior, has the potential to improve the sample efficiency of an RL-based flight controller. However, its scalability limitations make it impractical for utilization as a controller in continuous, fully coupled 6-DOF complex flight dynamics. Additionally, its high computational demands may pose challenges for real-time flight control. Moreover, finding an appropriate prior to accurately represent the value function of the flight controller remains an open question. Deducing a prior over a state value function (V) is believed to be simpler, as the definition of the reward function (for example, as a penalty proportional to the discrepancy between the desired and actual states) could directly provide a mathematical basis for the prior. However, the state-action value function (Q) presents greater challenges due to the inclusion of the action component, which introduces complexities associated with the intricate flight dynamics and control constraints. Consequently, resorting to uninformative priors becomes necessary, potentially diminishing the advantage of improved sample efficiency that BRL offers.

Therefore, DRL appears to be more feasible for flight control. The disadvantage of this approach is that it does not directly capture the epistemic uncertainty, which could be leveraged to improve the sample efficiency. Nevertheless, just modelling the entire reward distribution instead of only its mean was empirically shown to improve the sample efficiency. Although no conclusive theoretical explanations exist, it is hypothesized that this improvements are due to the auxiliary predictions[5] of the ANN and the regularization effect of modeling a distribution, which may reduce variance in predictions and improve optimization behaviour [115, 107]. Nevertheless, the current lack of comprehensive understanding of DLR renders it an appealing avenue for

[5]Auxiliary tasks are additional objectives related to the main problem that an RL agent can follow and observe from the environment in a self-supervised fashion [174, 175]. DRL inherently incorporates these auxiliary tasks by learning the probabilities associated with various returns, as opposed to focusing solely on a single value [107].

further research, presenting an opportunity for valuable scientific contributions.

Finally, the combination of well-documented theoretical foundations and the successful application of DRL in various continuous tasks, including flight control, encourages further exploration of this approach. Furthermore, ample opportunities remain for enhancing this methodology, such as ensuring the monotonicity of probabilistic functions. *Consequently, the remaining focus of this research will be dedicated to advancing Distributional RL aplied to flight control*. This answers **RQ-M-3b**, which completes the answer for **RQ-M-3**.

5 Preliminary Analysis

This chapter explores the design, implementation, and testing of a preliminary DSAC framework. The primary objectives are to gain practical insights into the application of uncertainty-aware RL to simple control problems, verify the theoretical findings presented in the literature review, and evaluate the potential adaptability of the DSAC algorithm for continuous environments, leveraging the insights presented in Section 4.3. The preliminary analysis in this chapter also served as a crucible for developing and evaluating innovative DSAC modifications, targeting enhanced sample efficiency. The detailed results, highlighting potential improvements, are presented in the subsequent sections. Additionally, the chapter aligns with the broader research goal by focusing specifically on assessing the practicality and feasibility of implementing uncertainty-aware RL methods within the realm of flight control, as partly articulated in **RQ-M-3b**.

This chapter is organized as follows. First, a simple continuous control testing environment is outlined in Section 5.1. Then, the algorithms employed in the preliminary analysis are detailed, starting with an introduction of the DSAC algorithm implemented in this research, along with the accompanying pseudocode, in Section 5.2. The novel UM-DSAC algorithm is described in Section 5.3, followed by the introduction of the RUN-DSAC algorithm in Section 5.4. The chapter then proceeds to evaluate the feasibility of implementing DSAC in the context of flight control, detailed in Section 5.5. The insights and findings of this exploration are neatly summarized and concluded in the final section, Section 5.6.

5.1. Testing Environment

The algorithms will be tested in the *LunarLander Continuous V2* environment designed by OpenAI Gym[1]. This control benchmark is a simulated environment that aims to simulate the landing of a lunar lander spacecraft on the surface of the Moon, inspired by the classic Lunar Lander arcade game. The environment was chosen for its ability to furnish a continuous reward signal, transcending the complexity of classic control tasks like the inverted pendulum, double pendulum, and cart-pole system, while remaining less intricate than the MuJoCo tasks. As a result, its level of complexity is adequately high to effectively test the algorithms, while avoiding excessively long computational times.

The goal of the agent in this environment is to learn how to control the three thrusters of a lander in order to successfully land the lander on the lunar surface, taking into account factors such as gravity. The lander and its task are visualized in Figure 5.1, together with coordinate system used to define the lander's state.

The lunar lander operates in a two-dimensional, simplified simulation of lunar gravity. At the start of each experiment, the lunar lander is randomly positioned at the top of the environment map. It starts with a vertical orientation and zero initial speed. The landscape of the lunar terrain is also randomly generated. During each moment of the test, the agent has access to its three thrusters to guide the landing. The objective is to achieve a safe landing on the designated landing pad, which is identified by two yellow flags, and crucially, the lander should have zero speed at landing. Each experiment is terminated after 1000 transitions regardless of the result. Finally, the LunarLander environment is deterministic, which means that the outcomes of actions are completely predictable and do not involve any randomness or uncertainty.

5.1.1. State and Action Space

The state space determines various attributes of the lunar lander. Specifically, the state space is characterized by 8 variables, as summarized by Equation 5.1. These states are derived based on the coordinate

[1]`https://gym.openai.com/envs/LunarLander-v2/` [Visited on 25/06/2023]

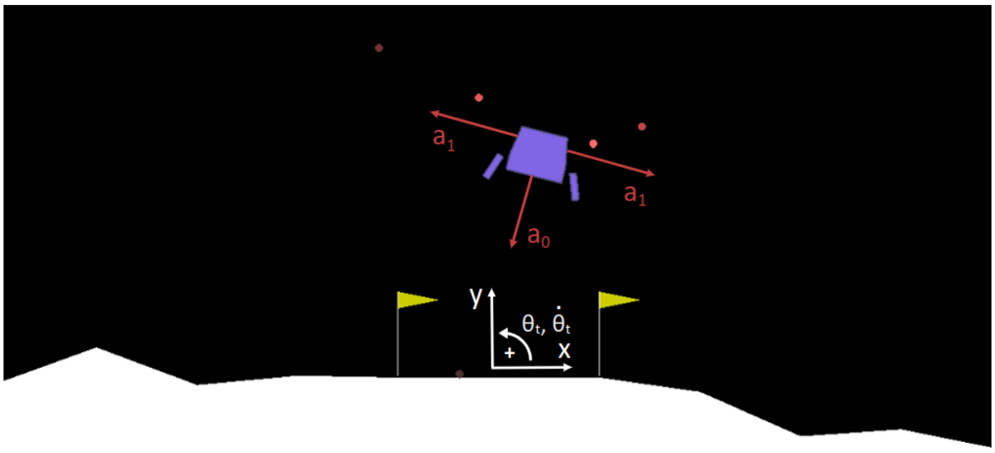


Figure 5.1: The *LunarLander Continuous* environment and its coordinate system.

system presented in Figure 5.1, the origin of which is placed in the middle of the landing pad. Furthermore, while the first 6 states are continuous, the last 2 states, $\text{bool}_{\text{left}}$ and $\text{bool}_{\text{right}}$, are discrete. These Boolean variables indicate whether the lander's legs are in contact with the ground (1) or not (0).

$$\text{state } s \begin{cases} x & \text{x-coordinate of the lander} \\ y & \text{y-coordinate of the lander} \\ \dot{x} & \text{The horizontal velocity} \\ \dot{y} & \text{The vertical velocity} \\ \theta_t & \text{The orientation in space} \\ \dot{\theta}_t & \text{The angular velocity} \\ \text{bool}_{\text{left}} & \text{Left leg touching the ground (Boolean)} \\ \text{bool}_{\text{right}} & \text{Right leg touching the ground (Boolean)} \end{cases} \tag{5.1}$$

When it comes to the actions, the lunar lander can fire either its main engine (at its bottom) or one of its lateral engines. Therefore, the actions can be divided into two alternatives, as given by Equation 5.2. Additionally, it should be emphasized that the main engine is active only when $a_0 > 0$; otherwise, it remains idle. Similarly, the right thruster fires only when $a_1 \leq -0.5$, while the left thruster activates when $a_1 \geq 0.5$. Otherwise, both thrusters are set to idle.

$$\text{action } a \begin{cases} a_0 = T_{\text{main}} \in [-1.0, 1.0] & \text{The thrust value of the main engine} \\ a_1 = T_{\text{lateral}} \in [-1.0, 1.0] & \text{The thrust value of the lateral engines} \end{cases} \tag{5.2}$$

5.1.2. Reward Architecture

The reward structure in the *LunarLander* Gym environment includes several components. Firstly, if the lunar lander moves from the top of the screen to the landing pad with zero speed, it receives a reward ranging from 100 to 140 points. Moving away from the landing pad incurs a penalty equal to the reward that would have been gained by moving the same distance towards the pad. Additionally, there are extra rewards of -100 points for crashing and +100 points for successful landings. Each step that involves using the main thruster results in a penalty of -0.3 reward points due to fuel consumption. Finally, a +10 point reward can be granted for landing with each leg on the ground. Generally, achieving an episode score of 200 or higher is considered a solution[2].

5.2. The DSAC Algorithm

The DSAC algorithm used in this work is summarized in algorithm 1. In words, the algorithm commences with the initialization of the network parameters, the temperature parameter α_T, and the initial (pre-training)

[2]`https://www.gymlibrary.dev/environments/box2d/lunar_lander/` [Visited on 01/07/2023]

$\langle s, a, r, s', d\rangle$ transitions are collected from the environment and stored in a replay buffer $\mathcal{D}$. The algorithms proceeds to run for N_{eps} episodes. Each episode executes $N_{\text{transitions}}$ $\langle s, a, r, s', d\rangle$ transitions using the current policy $\pi_w(\cdot|s)$.

For training the critic networks, two sets of quantile fractions are generated, one for the current and the other for the target estimates of the value distribution. A new action is then sampled from the target policy $\pi_{\bar{w}}(\cdot|s')$, followed by the computation of the temporal difference error δ_{ij}^k for each quantile fraction pair. The critic loss function $J_Z(\theta_k)$ is subsequently evaluated, based on which the critic networks parameters are updated. Target parameters are softly updated with a mixing coefficient ζ.

In the next step, reparametrized action samples $\tilde{a}$ are drawn to calculate the expected return $Q(s, \tilde{a})$, which is then used to calculate the policy loss function $\nabla J_\pi(w)$. This then guides the update of the policy network parameters, including a soft update for the target policy. Finally, at the conclusion of each episode, all the freshly collected trajectories are appended to the replay buffer $\mathcal{D}$, ensuring that learning continues to build on the growing body of experience.

Algorithm 1: DSAC algorithm

```
Initialize replay buffer $\mathcal{D}$
Initialize actor and critic parameters $\theta_1$, $\theta_2$, $w$
Initialize target parameters $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$, $\bar{w} \leftarrow w$
Initialize temperature $\alpha_T$
Collect initial trajectories $\mathcal{T}$ and store in the replay buffer $\mathcal{D}$
for episode = 1 to $N_{eps}$ do
    for transition = 0 to $N_{trans}$ do
        Collect a $\langle s, a, r, s', d\rangle$ transition using $\pi_w(\cdot|s)$
        Sample a batch $\mathcal{B}$ from the replay buffer $\mathcal{D}$
        Generate quantile fractions $\tau_i, i = 0, \dots, N_q$, $\tau_j, j = 0, \dots, N_q$
        Sample $a' \sim \pi_{\bar{w}}(\cdot|s')$
        for i = 0 to $N_q - 1$ do
            $\hat{\tau}_i \leftarrow \frac{\tau_i + \tau_{i+1}}{2}$
            for j = 0 to $N_q - 1$ do
                $\hat{\tau}_j \leftarrow \frac{\tau_j + \tau_{j+1}}{2}$
                $y_i = \min_{k=1,2} Z_{\hat{\tau}_i, \bar{\theta}}(s', a')$
                $\delta_{ij}^k = r + \gamma\,[y_i - \alpha_T \log \pi_{\bar{w}}(a'|s')] - Z_{\hat{\tau}_j, \theta_k}(s, a)$, $k = 1, 2$
            end
        end
        $J_Z(\theta_k) = \frac{1}{N_q} \sum_{i=0}^{N_q - 1} \sum_{j=0}^{N_q - 1} (\tau_{i+1} - \tau_i) \rho_{\hat{\tau}_j}^{\kappa}(\delta_{ij}^k)$, $k = 1, 2$
        Update $\theta_k$ with $\nabla J_Z(\theta_k)$, $k = 1, 2$ (using ADAM)
        Update $\bar{\theta}_k \leftarrow \zeta\theta_k + (1 - \zeta)\bar{\theta}_k$, $k = 1, 2$
        Sample new actions $\tilde{a} \sim \pi_w(\cdot|s)$ with reparametrization trick
        $Q(s, \tilde{a}) = \sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) \min_{k=1,2} Z_{\hat{\tau}_i, \theta_k}(s, \tilde{a})$
        $J_\pi(w) = \log(\pi_w(\tilde{a}|s)) - Q(s, \tilde{a})$
        Update $w$ with $\nabla J_\pi(w)$ (using ADAM)
        Update $\bar{w} \leftarrow \zeta w + (1 - \zeta)\bar{w}$
    end
    Store the collected trajectories $\mathcal{T}$ in the replay buffer $\mathcal{D}$
end
```

5.3. Unconstrained Monotonic DSAC (UM-DSAC)

Four algorithms were tested in the *LunarLander* environment and compared - SAC, DSAC with IQN and randomly generated quantiles, DSAC with IQN and FQF for systematic quantile generation, and UM-DSAC. SAC was chosen as a state-of-the-art reference for comparing against uncertainty-aware RL algorithms, as it represents a well-established approach. In contrast, the uncertainty-aware DSAC was specifically selected due to its remarkable potential for improving the sample efficiency. Specifically, the algorithm that was implemented in this work is presented in algorithm 1. Nevertheless, as Seres [28] observed in his DSAC implementation (based on the IQN architecture, but without FQF), the algorithm often output non-monotonic distributions, especially during early episodes. It is hypothesized that maintaining monotonicity would allow the network to capture the underlying distributions more accurately, enabling more effective learning from the available samples. For this reason, a novel algorithm, employing the Unconstrained Monotonic Neural Network (UMNN, as described in Section 2.3) in the critic architecture, was developed to test this hypothesis and named UM-DSAC.

The distinction between the original DSAC and UM-DSAC lies solely in the architecture of the neural network employed for the critics. Both models utilize the same inputs, comprising states s, actions a, and quantiles τ, and yield identical outputs, specifically the quantile function $F_Z^{-1}(\tau)$. In the case of UM-DSAC, the traditional Quantile Multi-Layer Perceptron (MLP) neural network module is substituted with the Unconstrained Monotonic Neural Network (UMNN) module. The UMNN draws inspiration from the research conducted by Wehenkel and Louppe [51], as well as its subsequent implementation in the DQN algorithm by Théate et al. [52]. Importantly, the underlying learning algorithm, detailed in Section 2.3, remains unchanged.

The main reason for considering the UMNN is to force the monotonicity of the modelled quantile functions. Formally, a UMNN establishes a continuous monotonic quantile function $F_{Z,\theta}^{-1}(\tau|s,a) : \mathbb{R} \to \mathbb{R}$ as follows:

$$F_{Z,\theta}^{-1}(\tau|s,a) := \int_0^\tau g_{\theta_g}(\tau|s,a)\, d\tau + \beta_{\theta_\beta}(s,a) \tag{5.3}$$

where $g_{\theta_g}(\tau|s,a) : \mathbb{R} \to \mathbb{R}$ is a free-form neural network whose output positiveness is ensured through a suitable activation function (such as ReLU or exponential), which is parameterized by θ_g. Furthermore, the scalar offset parameter β is represented by another neural network parametrized by θ_β. Hence, the parameters of the monotonic transformation are $\theta = \theta_g \cup \theta_\beta$. The integration is then performed numerically via Clenshaw-Curtis quadrature.

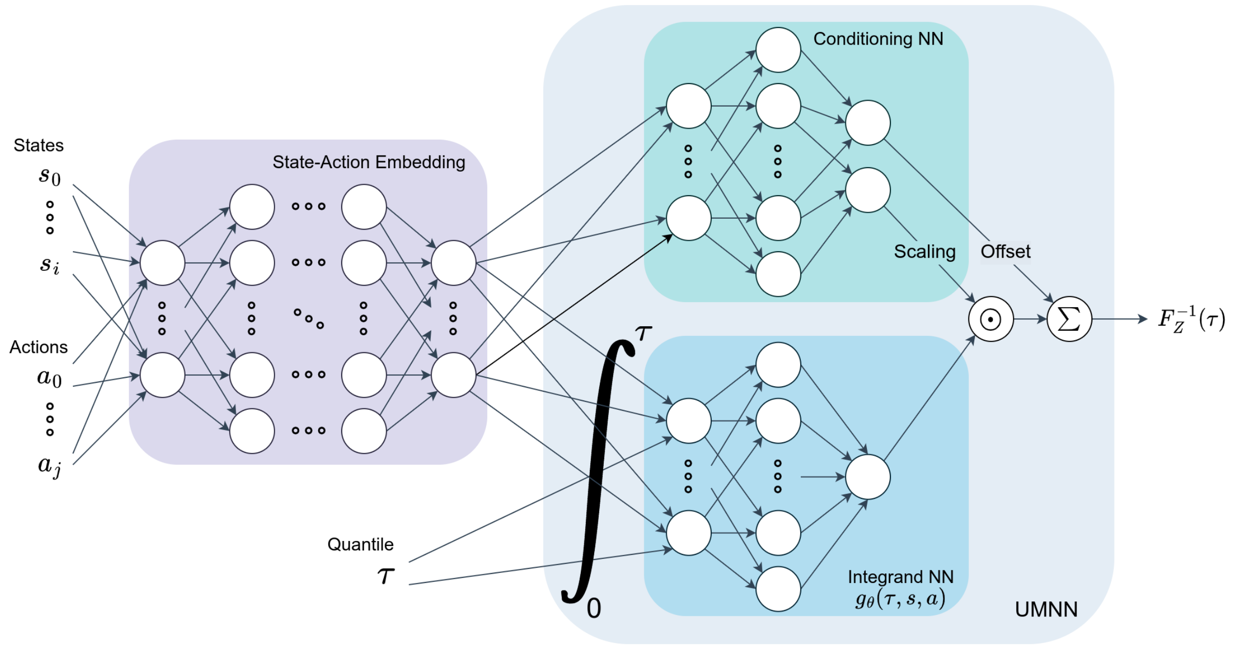


Figure 5.2: Architecture of an UM-DSAC critic network.

The architecture of the UM-DSAC critic network is illustrated in Figure 5.2 and comprises two primary components. The first segment is a Feed Forward Deep Neural Network (DNN) that embeds both the states and actions. The configuration of the hidden layers in this segment is aligned with those used in the Quantile MLP embedding. The subsequent component is the UMNN, which is responsible for ensuring the monotonicity of the output. This UMNN component can be further divided into two parts: the Integrand NN and the Conditioning NN. The Integrand NN takes as input the state-action embedding and a quantile τ, modeling the integrand g_{θ_g}, which is subsequently numerically integrated. Conversely, the Conditioning NN segment of the UMNN adjusts the integration result by supplying both a scaling parameter and the offset parameter β_{θ_β}, contingent on the state-action embedding. This intricate process culminates in the final output of the UMNN, which is the value of the quantile function for a given τ, denoted as $F_Z^{-1}(\tau)$.

5.3.1. Hyperparameters

To ensure a fair comparison, identical hyperparameters were maintained for all the methods, except for those that were specific to each method. The hyperparameters for SAC and both IQN and FQF variants of DSAC were set based on the proposal by Ma et al. [116]. These values are presumed to have already been extensively tuned. Nevertheless, it is worth noting that Ma et al. [116] conducted their evaluations exclusively on *MuJoCo* environments. In these experiments, they utilized identical hyperparameters for all *MuJoCo* environments, with an exception of the temperature parameter. Given that the Hopper environment is the closest analogue to LunarLander in terms of state and action dimensionality, the temperature parameter for the *LunarLander* experiments was inspired by this environment and set at 0.2. This value aligns with what Ma et al. [116] employed for the majority of the MuJoCo environments they tested. Table 5.1 summarizes the hyperparameters used for SAC, DSAC and UM-DSAC.

Table 5.1: Hyperparameters for SAC, DSAC and UM-DSAC.

SAC		**DSAC (IQN/FQF)** ***Extra parameters***	
Batch size	256	Num. of quantile fractions (N_q)	32
Num. steps per epizode	1000	Quantile fraction embedding size	64
Min steps before training	10000	Huber regression threshold	1
Num. of epochs	300	FQF network hidden layers	2
Actor - hidden layers	2	FQF network hidden layer sizes	[128, 128]
Actor - hidden layer sizes	[256, 256]	FQF network output activation	SoftMax
Critic - hidden layers	2	**UM-DSAC** ***Extra parameters***	
Critic - hidden layer sizes	[256, 256]	Embedding size	64
Optimizer	Adam	UMNN - hidden layers	1
Actor networks learning rate	0.003	UMNN - hidden layer size	[128]
Critic networks learning	0.003	Number of integration steps	40
Discount factor	0.99		
Target smoothing	0.005		
Temperature parameter	0.2		
Target update period	1		
Replay buffer size	10^6		
Non-linear Activation	ReLU		

5.3.2. Results

Several versions of DSAC were implemented and subjected to training within the *LunarLander* environment. The comparative analysis focused on the following variants:

- SAC: The standard Soft Actor-Critic method,
- DSAC with Implicit Quantile Networks (IQN): Uses IQN to parameterize the quantile function,
- DSAC with Fully parameterized Quantile Function (FQF): Employs FQF to enhance the approximation of the distribution,
- UM-DSAC: Uses UMNN to parametrize the quantile function and enforce its monotonicity.

The learning performance of these methods is depicted in Figure 5.3. This figure illustrates the mean returns of 10 independent runs (smoothed using a window of length 10) for each variant, with the shaded region surrounding each curve representing the corresponding standard deviation.

It is evident from Figure 5.3 that all distributional methods surpass the standard SAC method in both sample efficiency and consistency, displaying less variance. To probe the underlying causes of SAC's inferior performance and larger variance, individual runs were examined and presented in Figure 5.4. From this analysis, DSAC runs can be categorized into two distinct groups: one group that learns at a relatively rapid pace, reaching returns over 200 (considered a solution) within 300 episodes (or about $3 \cdot 10^5$ steps), and a second group that learns substantially more slowly, failing to achieve positive returns within the same

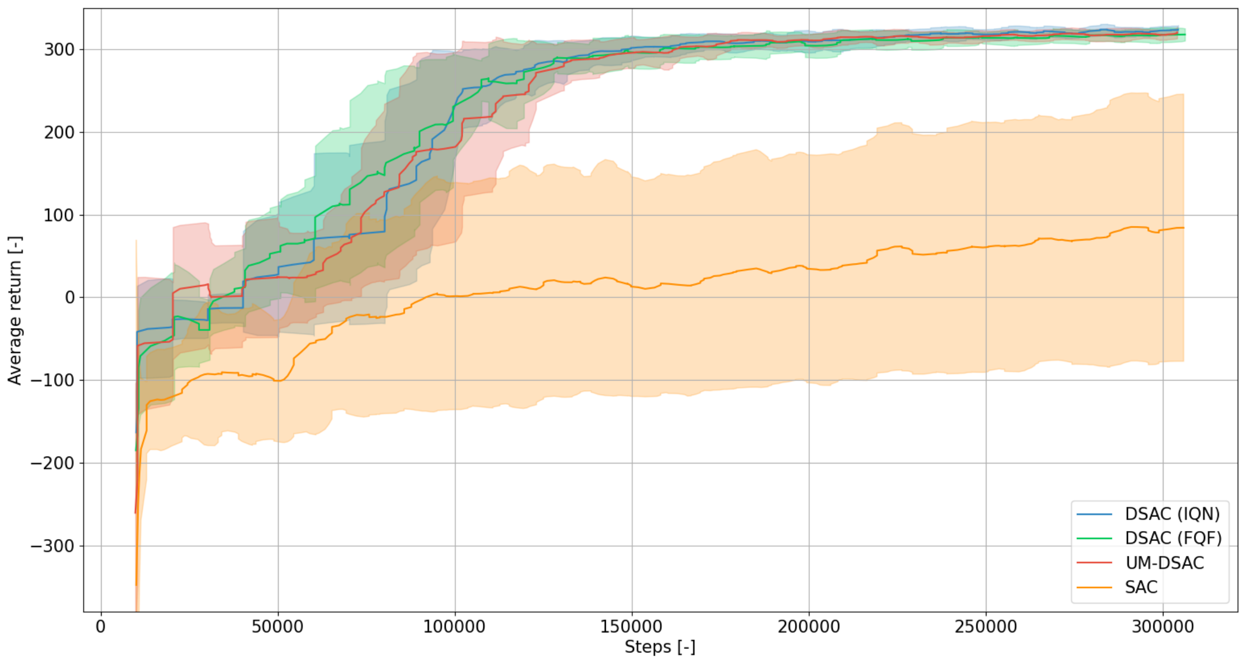


Figure 5.3: Comparison of the UM-DSAC learning curve with SAC and other DSAC variants: training performance in the *LunarLander* environment.

learning frame. Despite the accelerated learning of the faster group, it still falls behind the performance achieved by the distributional methods. The inconsistency in SAC's learning could potentially be mitigated through further hyperparameter tuning, although the existing settings are expected to have been finely tuned by Ma et al. [116]. However, these hyperparameters were not specifically tailored for the *LunarLander* environment, leaving room for potential performance gains through targeted tuning. Nevertheless, these results are consistent with prior research on distributional RL, supporting the findings that modeling reward distributions can enhance sample efficiency and diminish *chattering*, a phenomenon that might otherwise obstruct policy convergence [107, 111]. Furthermore, the observed findings align with the results presented by Ma et al. [116]. In their work, they too identified a notably enhanced performance of DSAC over the traditional SAC method for most of the environments, as illustrated in Figure 5.5.

In the analysis of the distributional methods, it is observed that they generally converge after similar numbers of steps. Although the FQF variant appears to speed up convergence initially, the IQN variant catches up in the later stages, resulting in both methods achieving convergence in equivalent number of episodes. The novel UM-DSAC algorithm, characterized by enforcing monotonicity from the onset of learning, shows potential for some sample efficiency gains during the initial learning phases. However, as learning progresses, it seems to be surpassed by both the IQN and FQF variants, eventually converging slightly later than these two methods. Consequently, these findings suggest that the integration of DSAC with UMNN does not confer any discernible advantage in terms of sample efficiency, thus refuting the original hypothesis.

Furthermore, doubts arise regarding the accuracy of the quantile function $F_Z^{-1}(\tau)$ approximations learned by UM-DSAC. Figure 5.6 illustrates a quantile function learned by the DSAC algorithm, juxtaposed with an example from UM-DSAC, presented in Figure 5.7. The DSAC-learned quantile function exhibits a resemblance to an ”S-shape,” characteristic of a normal distribution of Q-values for a given action-state tuple. Yet, the lack of strict monotonicity is evident, though the divergence from monotonic behavior is relatively minor, and interestingly, the monotonicity was observed to improve as the episode count increases. Conversely, Figure 5.7 reveals that the UM-DSAC algorithm consistently approximates quantile functions as linear lines, achieving monotonicity but at the potential cost of realism. This pattern persisted across various architectural modifications of UM-DSAC and different hyperparameter combinations.

A linear quantile function implies a uniform distribution. In other words, for a specific action in a designated state, all returns within an output range are equally probable. However, upon closer examination of various samples of the modeled quantile function, it was observed that the range of quantile function values was relatively narrow, becoming even more constricted as the number of episodes increased. Furthermore,

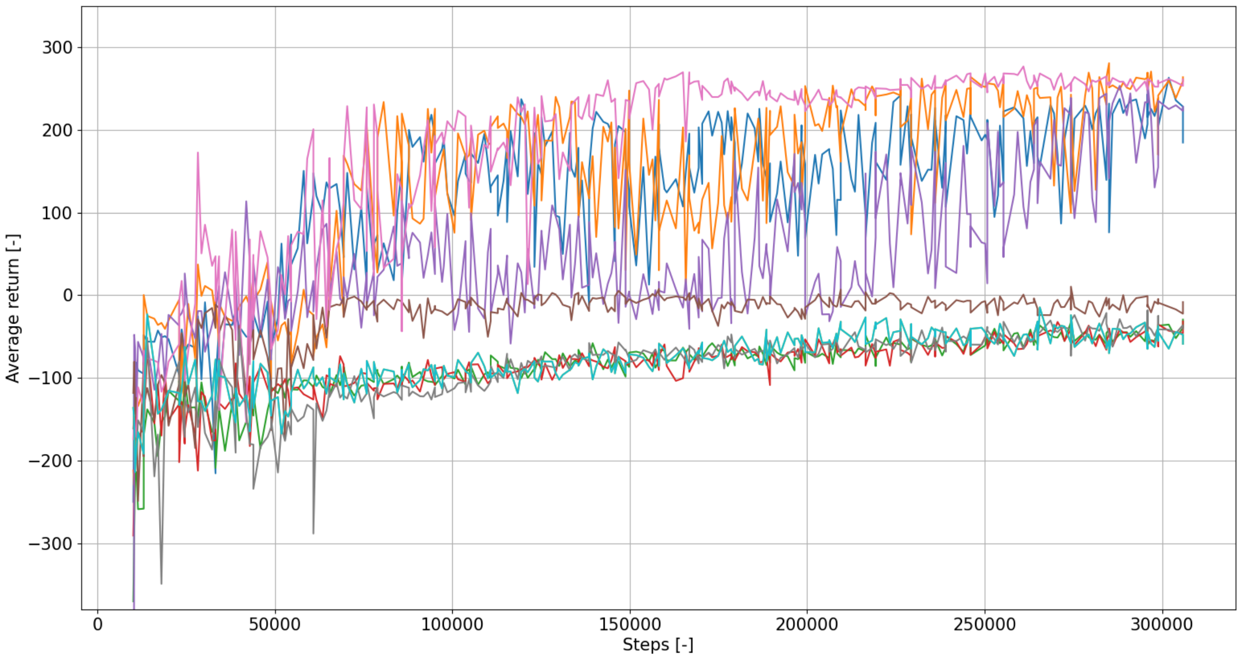


Figure 5.4: Learning curves for SAC applied to the *LunarLander* environment.

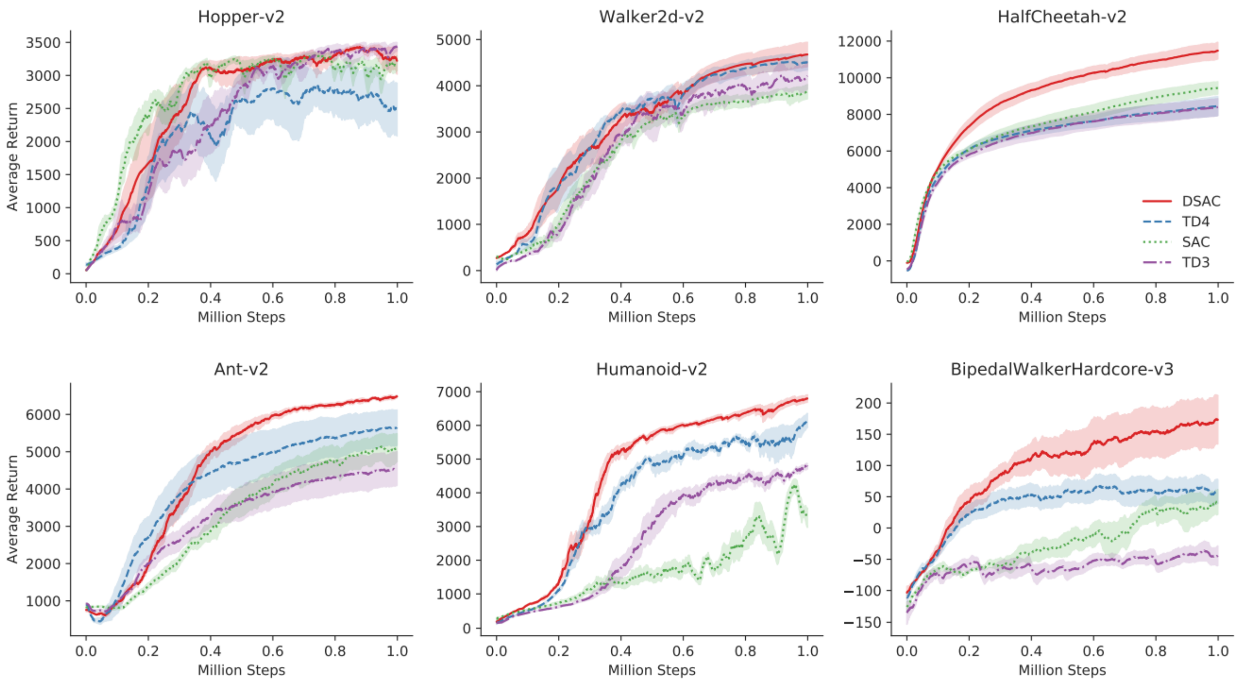


Figure 5.5: Comparison of DSAC learning curves with SAC and other state-of-the-art RL methods in various benchmarks, adopted from [116].

when applied to an LTI short period aircraft control environment, a deterministic environment simpler than the *LunarLander* and detailed later in Section 5.5,the linear quantile function's range shrank further, occasionally approximating a constant line. Such a pattern signifies a degenerate distribution focused at a singular value. Consequently, the observed linear line for the *LunarLander* environment is hypothesized to stem from the deterministic nature of this environment.

Another hypothesis was deduced after observing the results of Théate et al. [52], who combined the DQN algorithm with the UMNN. The left graph Figure 5.8 shows the comparison between the quantile functions learnt by the UMDQN algorithm for each of the actions in a specific state in the stochastic grid world

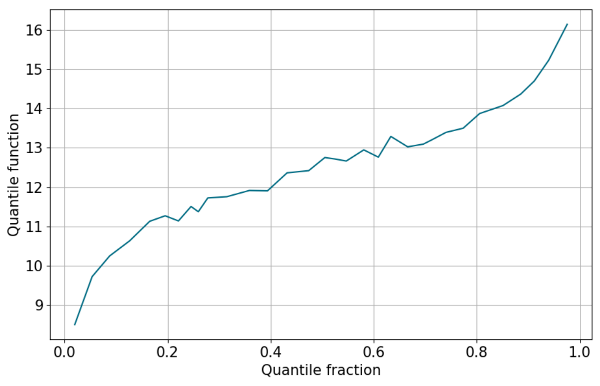


Figure 5.6: Example of a quantile function $F_Z^{-1}(\tau)$ learnt by DSAC (IQN) after 5 episodes.

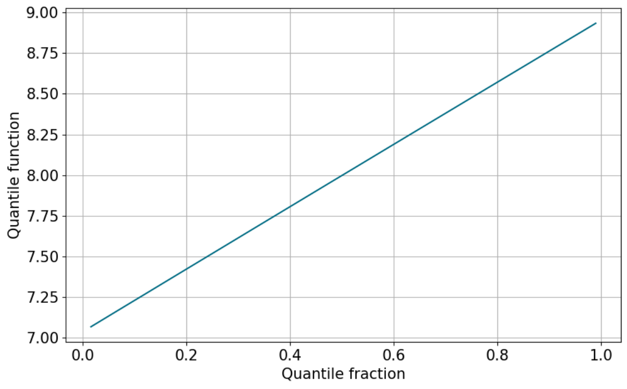


Figure 5.7: Example of a quantile function $F_Z^{-1}(\tau)$ learnt by UM-DSAC after 50 episodes.

environment with the true underlying quantile functions. The stochastic grid world is visualized in the right part of this figure, which consists of a 7 × 7 grid world with a prize and a trap at a certain location, while both transition and reward functions are set stochastic. It can be seen that even that the learnt quantile functions are perfectly monotonic, their accuracy is rather poor. Although these functions retain a slight "S-shape", they clearly lose the multimodality of the distribution, which can be seen mainly on the blue line and they become more linear.

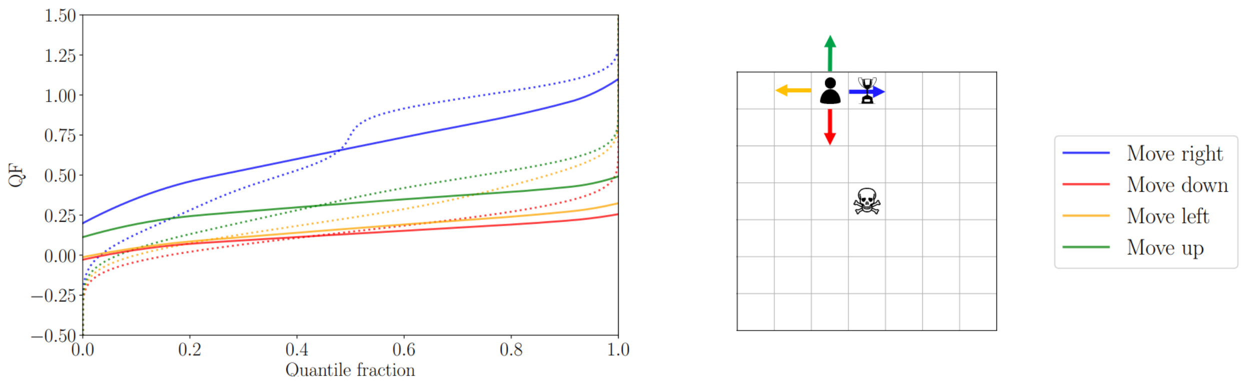


Figure 5.8: Comparison between the quantile functions learned by the UMDQN algorithm (solid lines) and the true quantile function associated with an optimal policy (dotted lines), as estimated via Monte Carlo simulation for a specific state in the stochastic grid world environment (on the right), adopted from [52].

Théate et al. [52] sought to unravel the inconsistency in the learned quantile functions by attempting to mathematically validate loss functions derived from various probability metrics. They successfully derived proofs for both the KL divergence D_{KL} (utilized in modeling a PDF) and the Cramér distance D_C (used in modeling a CDF). However, a rigorous proof for the loss function based on the Wasserstein distance $\mathcal{L}_W$ (detailed in Equation 5.4) remained elusive. Instead, a series of experiments were conducted which showed that Equation 5.4 is likely an approximation of $F^{-1}_{\mathcal{T}^\pi, Z^\pi}$. While this approximation yields a random variable with accurate expectation, it may result in inconsistent higher-order moments. It is worth noting, however, that Bellemare et al. [107] established that the distributional Bellman operator $\mathcal{T}^\pi$ acts as a contraction in terms of variance.

$$\mathcal{L}_W(\mathcal{T}^\pi Z^\pi(s,a), Z^\pi(s,a)) = W_p(F^{-1}_{\mathcal{T}^\pi Z^\pi}(\tau|s,a), F^{-1}_{Z^\pi}(\tau|s,a)) \simeq W_p\left(\mathop{\mathbb{E}}_{s',r}\left[r + \gamma F^{-1}_{Z^\pi}(\tau|s', \pi(s'))\right], F^{-1}_{Z^\pi}(\tau|s,a)\right) \tag{5.4}$$

This finding presents a nuanced situation. If the goal is to accurately discern the probability distribution of the random return for risk-aware policies, vital in safety-sensitive applications like aircraft control, this approximation could present challenges. However, if the objective is merely to optimize policies to maximize the expectation of the random return, this approximation might not pose any hindrance, as the learned distribution retains the correct first-order moment.

Therefore, it is posited that the expectation of the random return is accurately captured in Figure 5.7, though the higher-order moments of the distribution may be imprecise. This conclusion underscores the need for careful consideration of the particular requirements and constraints of a given application when selecting the appropriate probability metric and loss function.

Nonetheless, the results obtained in this study bear a resemblance to those achieved by Théate et al. [52] using their UMDQN algorithm. As illustrated in Figure 5.9, UMDQN's learning of quantile functions with a loss based on the Wasserstein distance (represented by a green line in the graphs) leads to slightly enhanced sample efficiency in the early stages of learning. However, as the number of episodes progresses, its performance declines, and other RL methods surpass this algorithm in terms of sample efficiency. This observation lends support to the theory that enforcing monotonicity can bolster learning stability in initial phases but does not necessarily accelerate convergence.

Despite these findings, the UM-DSAC continues to present valuable insights. As demonstrated, its learning performance was comparable to the state-of-the-art DSAC methods in the *LunarLander* environment. In contrast, the UMDQN-W algorithm struggled to learn in this environment, as evidenced in Figure 5.9. This discrepancy suggests that UM-DSAC might still hold potential for further research, representing a promising avenue of exploration.

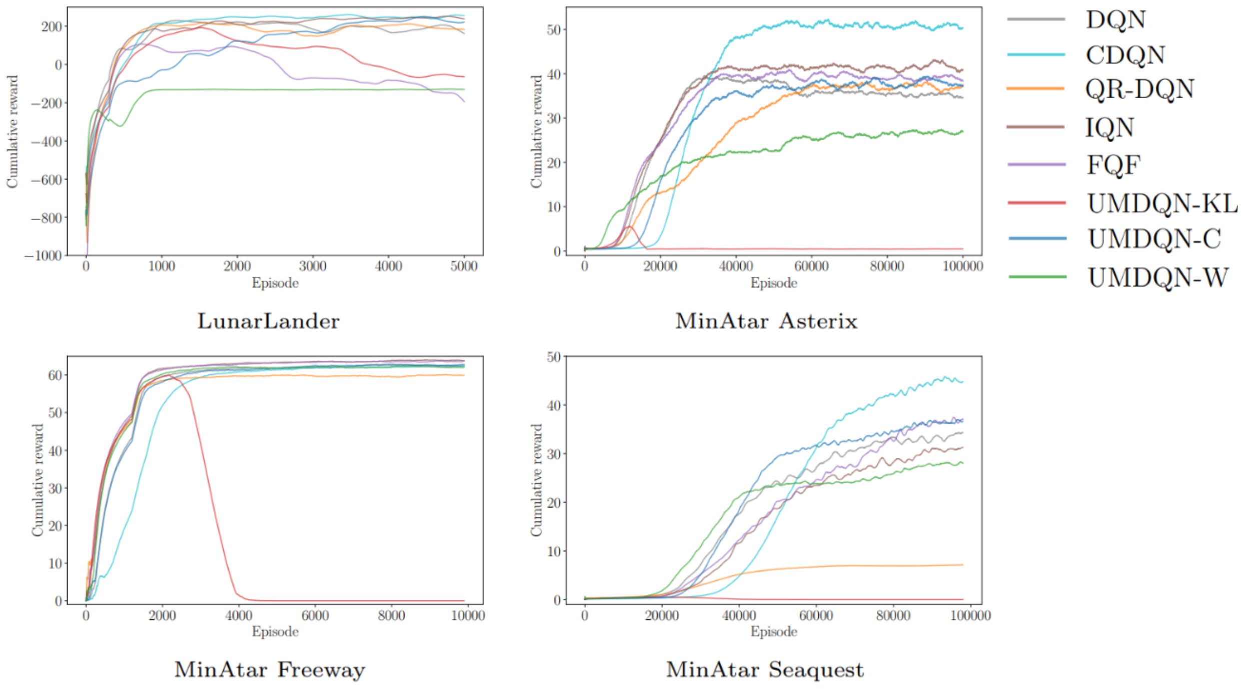


Figure 5.9: Comparison of UMDQN learning curves with other RL methods in various benchmarks, adapted from [116].

Finally, the evaluation of the UM-DSAC algorithm revealed that its time complexity is significantly higher than its counterparts. A comparison of computational time between SAC, DSAC with IQN, DSAC with FQF, and UM-DSAC is provided in Figure 5.10. In the *LunarLander* environment, the SAC algorithm required an average of $7.2s$ per episode. However, when transitioning from learning the expectation of returns to modeling the entire returns distribution, a considerable increase in computational time was observed. Specifically, DSAC's average computation time was triple that of SAC, amounting to $21.3s$, while incorporating FQF to learn the optimal set of quantiles further increased the time to $37.1s$ per episode, more than five times SAC's average. The most substantial increase was seen with UM-DSAC, which required $233.8s$ per episode on average, a staggering 32.5 times more than SAC. Such computational intensity renders UM-DSAC impractical for controlling more intricate systems, such as aircraft. The computational time is expected to rise exponentially with the increase in the number of dimensions, reflecting the challenges associated with the *curse of dimensionality*. This increased demand in computational resources for UM-DSAC can be attributed to the underlying UMNN architecture, which necessitates the solution of an integral for each of the two critic networks.

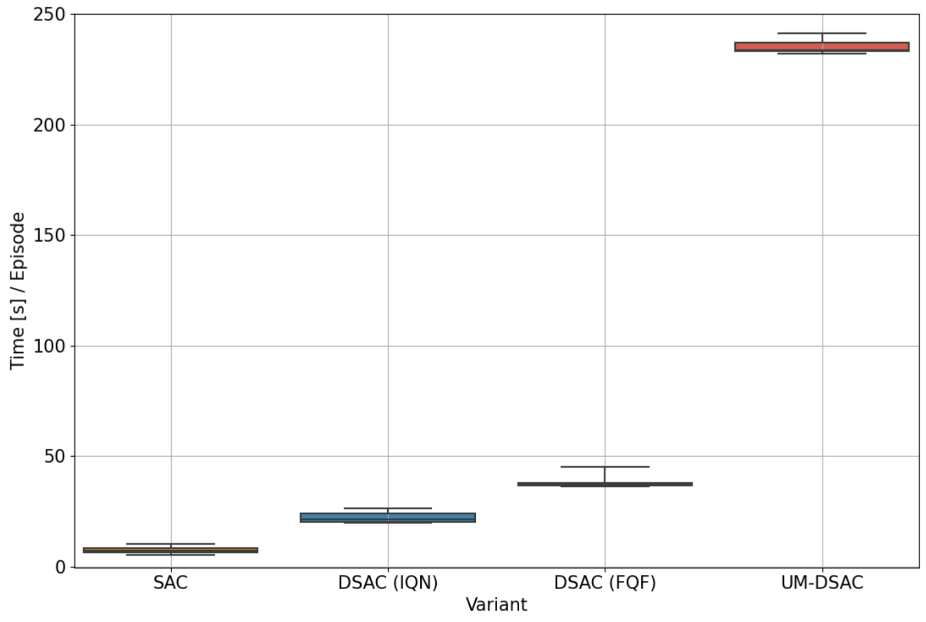


Figure 5.10: Computational complexity comparison of SAC and different DSAC variants.

Several additional smaller experiments were carried out to explore ways to ensure the monotonicity of the quantile function. One such experiment involved penalizing non-monotonicity in the loss function. Regrettably, this approach had a significant negative impact on sample efficiency and gave rise to sub-optimal policies. An alternative method to enforce monotonicity involves constraining weight and activation functions. This approach, however, places limitations on the expressiveness of the transformations that can be represented. For instance, imposing constraints on the signs of the weights is incompatible with widely-used non-saturated activation functions, as it restricts the approximation to convex functions [176].

In a more experimental vein, an attempt was made to artificially enforce monotonicity by sorting the outputs of the critic neural networks in ascending order. While it was anticipated that such a non-differentiable operation might disrupt learning, due to alteration of the gradients used in backpropagation and the potential for confusing training dynamics, the experiment surprisingly revealed that this method had virtually no impact on the learning process. Eventually, it was concluded that the quantile functions learned by DSAC already demonstrated acceptable levels of monotonicity from the early stages of learning, as corroborated by Figure 5.6. Ultimately, it became clear that additional efforts to enforce monotonicity would likely result in negligible improvements to sample efficiency, and potentially at the cost of substantial computational time. Consequently, the focus shifted toward seeking alternative strategies to enhance the sample efficiency of DSAC, rather than pursuing further efforts to ensure monotonicity.

5.4. Returns Uncertainty-Navigated DSAC (RUN-DSAC)

Due to the observed limitations of UM-DSAC in terms of sample efficiency and computational complexity (which would be further pronounced when dealing with more complex systems like aircraft controllers), an alternative approach was pursued to enhance the sample efficiency of DSAC. The fundamental premise of UM-DSAC aimed to enhance learning stability by refining the accuracy and enforcing monotonicity of quantile function estimations, especially during the initial learning stages. In contrast, the proposed method introduced in this study, named *Returns Uncertainty-Navigated DSAC (RUN-DSAC)*, alters the decision-making process of DSAC's actor to achieve superior sample efficiency. Specifically, the core concept behind RUN-DSAC revolves around leveraging the additional information derived from modeling the distribution of returns to enable more informed decision-making. This approach draws inspiration from the work of Liu, van Kampen, and de Croon [131], where they utilized DSAC's learned quantile functions to estimate the right truncated variance (RTV) and subsequently used this metric to modify the quantile function fed into the policy, thereby enabling adaptive control of risk tendencies. In contrast, RUN-DSAC, introduced in this study for improved sample efficiency, uses the full returns distribution variance for a comprehensive understanding of returns variability, unlike RTV, which mainly focuses on negative returns and uncertainty in unfavorable outcomes.

The guiding principle of RUN-DSAC capitalizes on the information embedded in the variance of a distribution to inform the policy's decisions. Notably, variance is an intuitive metric capturing the uncertainty, fluctuations, fairness, and robustness of a decision. Furthermore, Bellemare et al. [107] highlighted that the distributional Bellman operator $\mathcal{T}_D^{\pi}$, contracts in terms of second moment of the discounted returns, hence the convergence in the value distribution space is indicative of a good estimation of variance. Therefore, although reservations are raised in Section 5.3 concerning the accuracy of variance and higher-order moment estimations when using a Wasserstein distance-based loss function, the continued exploration of variance as a metric for quantifying uncertainty in returns within the RUN-DSAC framework is deemed justified.

To include the variance information in the the learning, the objective was changed from optimizing the mean, $\Psi(Z) = \mathbb{E}[Z]$, to optimizing a trade-off between mean and standard deviation, $\Psi(Z) = \mathbb{E}[Z] + \mu\sqrt{\mathbb{V}[Z]}$. Specifically, the Q-function is augmented according to Equation 5.5.

$$Q_V(s,a) = Q(s,a) + \mu\sigma_Q \tag{5.5}$$

The standard deviation σ_Q is approximated with Equation 5.6, where $Q(s,a)$ is the expectation.

$$\sigma_Q = \sqrt{\sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) \left[Z_{\hat{\tau}_i,\theta}(s,a) - Q(s,a)\right]^2} \tag{5.6}$$

μ is a coefficient that balances between the mean and the standard deviation in the objective.

When μ is positive, the algorithm is inclined to favor actions that possess both a high expected return and a significant standard deviation. This propensity translates into a preference for more explorative or potentially riskier decisions, as it accentuates the importance of uncertainty. This approach might prove beneficial in scenarios where thorough exploration of the state space is vital for uncovering optimal solutions.

Conversely, a negative μ directs the policy towards more conservative or safer choices, effectively penalizing uncertainty. Such a stance might foster more stable and predictable behavior, a trait that could be highly valued in environments where safety is paramount. However, it's worth recognizing that an excessive focus on reducing variance could stifle exploration, potentially leading to a convergence on suboptimal solutions. Therefore, the selection of μ must be undertaken with a careful consideration of the specific characteristics of the environment and the objectives of the learning process.

5.4.1. Results

RUN-DSAC was assessed using two distinct configurations, corresponding to positive μ and negative μ. Specifically, μ was designated as 1 in what may be referred to as the "risky" scenario and -1 in the "conservative" scenario, with a linear diminishment to 0 across the first 100 episodes. This configuration arises from an assumption that greater variance in returns is anticipated during the initial stages of learning when the agent is relatively unfamiliar with its environment, compared to the more predictable returns observed in later phases.

The corresponding learning curves of these RUN-DSAC strategies are depicted and juxtaposed in Figure 5.11 where each line represents the average of 10 runs, and the shaded regions denote the corresponding standard deviations. From this comparison, it is evident that incentivizing the agent to explore state-action combinations characterized by higher variance in returns facilitates a discernible enhancement in sample efficiency throughout the learning process. This, in turn, enables the policy to converge more rapidly, showcasing the potential effectiveness of this approach in promoting exploration without compromising the integrity of the learning process. In contrast, penalizing variance, thereby inducing more conservative policies, manifests as diminished sample efficiency and slower convergence.

In the context of the *LunarLander* environment, an emphasis on higher variance appears to foster more explorative behavior, enabling the agent to uncover potentially superior paths and strategies. This explorative emphasis may contribute to the discovery of more optimal policies, as evidenced by the accelerated convergence observed. However, in a field that demands stringent safety considerations, such as flight control, the priorities might shift towards achieving stable and predictable behaviors. In these contexts, the encouragement of higher variance and associated exploration might introduce unwarranted risks. Consequently, the optimal balance between exploration (represented by variance) and exploitation (represented by mean) may differ across scenarios, and prioritizing state-action tuples with low variance might prove to be more beneficial for learning performance. Thus, it becomes vital to meticulously tune and schedule the parameter μ in alignment with the specific requirements and characteristics of a given environment.

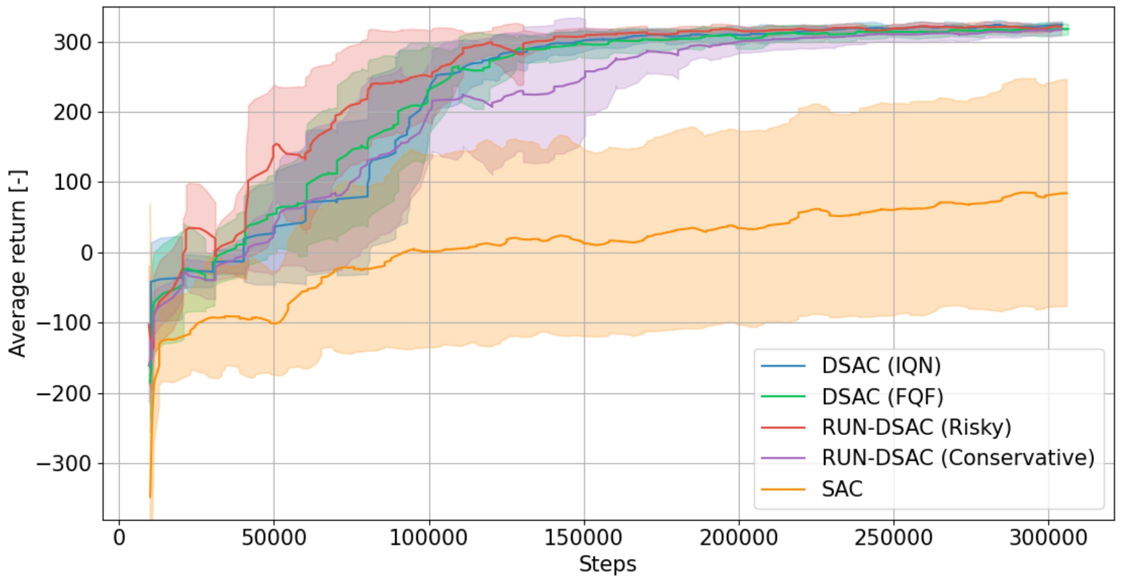


Figure 5.11: Comparison of the RUN-DSAC learning curve with SAC and other DSAC variants: training performance in the *LunarLander* environment.

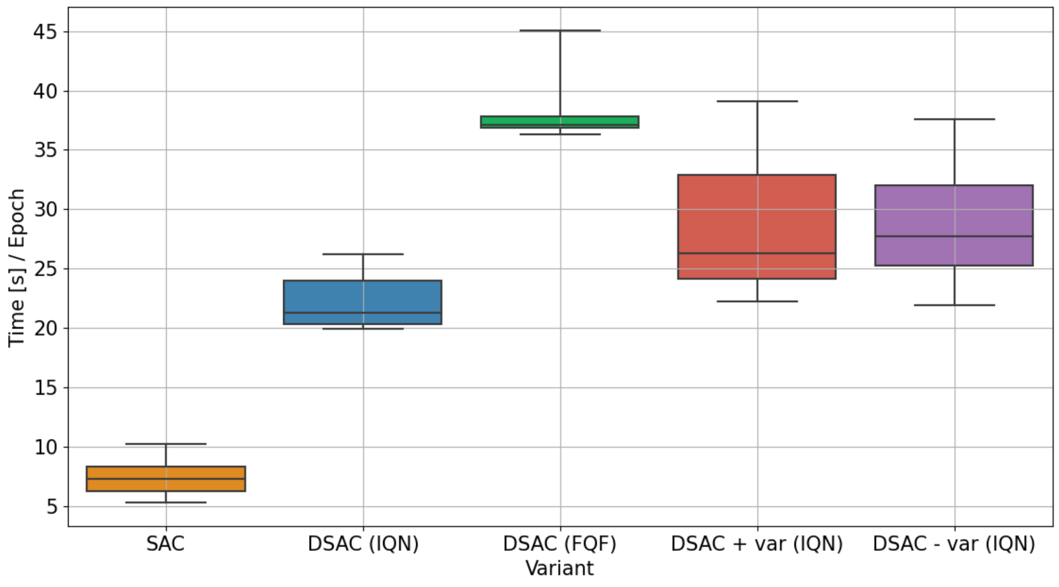


Figure 5.12: Computational complexity comparison of RUN-DSAC with SAC and other DSAC variants.

The time complexity of RUN-DSAC in comparison with SAC and other DSAC variants is detailed in Figure 5.12. Specifically, the version of RUN-DSAC that encourages variance in state-action tuples required an average of $26.3s$ per episode, while the version that penalizes variance took $27.6s$ on average. These durations correspond to an increase of 23.5% and 29.6%, respectively, compared to the computational time required by the IQN variant of DSAC. Furthermore, RUN-DSAC is still more efficient than the FQF version of DSAC.

5.4.2. Conclusions

The experimental results detailed in Section 5.3 and Section 5.4 substantiate the theoretical insights presented in Section 4.3, establishing that learning the entire distribution of returns, rather than merely their expectation, indeed yields an enhancement in sample efficiency over previous state-of-the-art methods such as SAC. These findings are particularly promising for this research's primary objective: enhancing the sample efficiency of RL methods utilized in flight control through uncertainty-aware approaches.

Initially, the DSAC method demonstrated substantial advancements in both sample efficiency and learning stability compared to SAC. Inspired by these improvements, two separate modifications of DSAC were explored to further augment its sample efficiency. In the first approach, Unconstrained Monotonic Neural Networks (UMNNs) were implemented in place of Quantile MLP neural networks within the critics. This aimed to enforce monotonicity in the learned quantile function, thereby hypothesizing an improvement in early learning phases. Despite observing some initial enhancements in sample efficiency, the UM-DSAC converged slightly later than the IQN and FQF variants of DSAC. Additionally, its excessive computational complexity rendered it unsuitable for application in more intricate systems such as flight control.

In light of these limitations, a second approach, named RUN-DSAC, was formulated and examined. Unlike UM-DSAC, which targeted the critic component of DSAC, RUN-DSAC focuses on the actor part, leveraging the learned distributions to modify the policy. This modification either rewards or penalizes (depending on hyperparameter selection) the variance in returns derived from the modeled quantile functions. The implementation that rewarded high-variance yielded marked improvements in sample efficiency within the Lunar Lander environment, all while incurring only a marginal increase in computational cost. Thus, RUN-DSAC's uncertainty-aware strategy emerges as a viable and promising avenue for augmenting the sample efficiency of RL-based flight controllers. This method's potential applicability and effectiveness will be further scrutinized and validated on a full-scale aircraft dynamics model in the subsequent phases of this research.

The next part of this preliminary analysis will concentrate on assessing the feasibility of implementing DSAC in flight control.

5.5. Aircraft Control

The main objective of this research project is to evaluate the uncertainty-aware RL controller using the validated simulation model of the Cessna Citation II PH-LAB research aircraft. Therefore, a portion of the preliminary analysis was dedicated to verifying the feasibility of implementing the proposed DSAC algorithm on a simpler aircraft model. For this purpose, a simplified linear time-invariant (LTI) model of the Cessna Ce500, inspired by Mulder et al. [177], was developed. Specifically, the state space model was designed to approximate the short period motion of Ce500, with the aircraft trimmed at an airspeed of 59.9 m/s. This implementation of the controller on the simplified aircraft dynamics serves as an introductory step towards integrating RL into an aircraft controller, which paves the way for future implementations of the algorithm on a more complex model of the Cessna Citation II.

Short period is one of the primary modes of oscillation an aircraft can exhibit. The short period motion primarily involves the aircraft's pitch rotation and is characterized by a rapid pitching motion around the lateral axis. During the short period motion, the aircraft experiences a short-duration cycle of nose-up and nose-down oscillation. To model this motion, a simple state space model was established. This model contains two states, namely the angle of attack α [rad] and the pitch rate q [rad/s]. These states are controlled by a single continuous action input, the elevator surface deflection δ_e [rad]. These states and action are visualized in Figure 5.13.

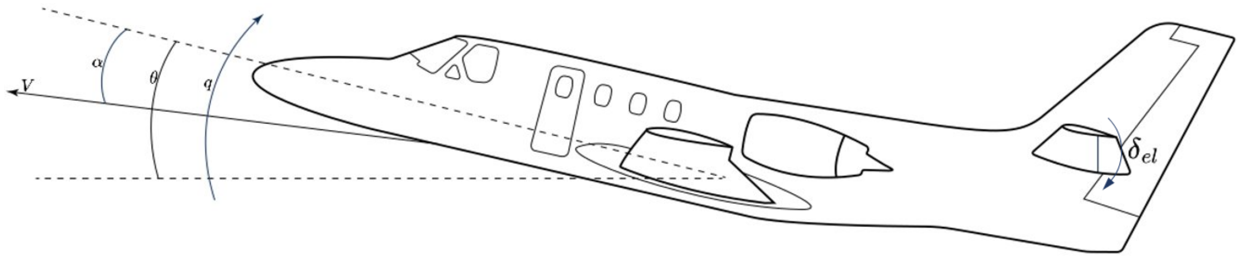


Figure 5.13: States and actions used in the Cessna 500 short period LTI model.

The state matrix A and the input matrix B are shown in Equation 5.7, with the parameters outlined in Table 5.2. It is assumed that all the states are perfectly observable, hence $C = I$ and $D = 0$

$$A = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{Z_\alpha}}{2\mu_c - C_{Z_{\dot{\alpha}}}} & \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V}{\bar{c}} \frac{C_{M_\alpha} + C_{Z_\alpha} \frac{C_{M_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} & \frac{V}{\bar{c}} \frac{C_{M_q} + C_{M_{\dot{\alpha}}} \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix}, \quad B = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{Z_{\delta_e}}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V}{\bar{c}} \frac{C_{M_{\delta_e}} + \frac{C_{M_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix} \tag{5.7}$$

Table 5.2: LTI short period model parameters for Cessna 500

Geometry & Trim Conditions		Stability & Control Derivatives			
$\bar{c}$	2.022 m	C_{Z_α}	-5.1600	C_{m_a}	-0.4300
μ_c	102.7	$C_{Z_{\dot{\alpha}}}$	-1.4300	$C_{m_{\dot{a}}}$	-3.700
K_y^2	0.98	C_{Z_q}	-3.8600	C_{m_q}	-7.0400
V	59.9 m/s	$C_{Z_{\delta_e}}$	-0.6238	$C_{m_{\delta_e}}$	-1.5530

5.5.1. DSAC Flight Control

The controller aims to follow the α_{ref} reference angle of attack signal. The base DSAC controller for the Cessna Ce500 LTI short period model is depicted in the block diagram of Figure 5.14. The agent takes two input signals - the absolute error between reference α_{ref} and actual α, and reward. It then determines the next elevator deflection δ_e within a specified range.

Regarding the reward function, it is negatively proporsional to the absolute α tracking error, as outlined in Equation 5.8. Additionally, the reward is scaled with $\frac{\Delta t}{T_{tot}}$, where Δt is the simulation sampling time and T_{tot} is the total duration of the episode. This term ensures the reward remains consistent across different time settings. This reward is then multiplied by a tracking error penalty weight K_e. Scaling the reward function is crucial in SAC-based algorithms since it is related to the policy's temperature and it affects the stochasticity of the policy, impacting the exploration-exploitation trade-off [71]. Improper scaling leads to subpar learning outcomes: small scales result in uniform policies unable to exploit rewards, while large scales lead to determinism and suboptimal minima. The reward scale is a key hyperparameter requiring tuning, and its task-specific nature underscores its importance in SAC's effectiveness.

$$R(s,a) = -\frac{\Delta t}{T_{tot}} K_e |\alpha_{ref} - \alpha| \tag{5.8}$$

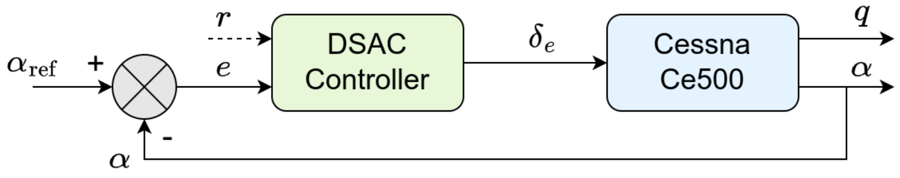


Figure 5.14: DSAC controller architecture for the α tracking in the Cessna Ce500 LTI model.

To train the agent for the Cessna Ce500 control task, a random training signal is generated at the beginning of each episode. This signal comprises a sequence of random steps with amplitudes drawn from a uniform distribution $U([-A_s, A_s])$ [deg]. Each step block has a width of w_s [s] and is slightly offset in timing to prevent agents from exploiting any periodicity. Figure 5.15 illustrates an example of the randomized step sequence.

The controllers, once trained, are evaluated using a 3-2-1-1 pulse reference signal, depicted in Figure 5.16. This specific signal was selected due to its standardized nature, facilitating straightforward comparisons of the agent's performance with other controllers. Additionally, its frequency sweep offers a comprehensive range of frequencies within a single compact test maneuver. This characteristic is advantageous for identifying system dynamics across different frequency bands.

The implementation of the DSAC agent for elevator control, along with its training and evaluation, necessitates additional settings and hyperparameters. These are concisely presented in Table 5.3, including the specific values employed in this preliminary analysis. The DSAC hyperparameters mirror those employed for the *LunarLander* environment, detailed in Table 5.1. However, the architecture of the critic and actor networks was changed to three hidden layers with 64 units each.

The results of training a DSAC agent on the LTI short period Cessna Ce500 model are displayed in Figure 5.17. In the top-left graph, the learning curve is depicted. The bold line illustrates the average performance across 10 trained agents, while the shaded regions indicate the standard deviation. The learning plateaued in just 13 episodes, which can be explained by the simplicity of the problem.

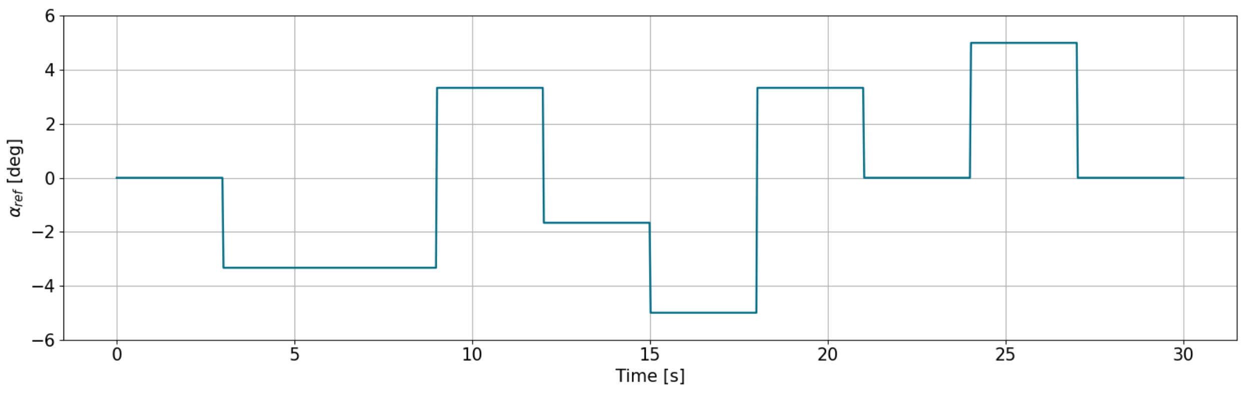


Figure 5.15: Sample of a randomized step-sequence α reference signal used in training the agent.

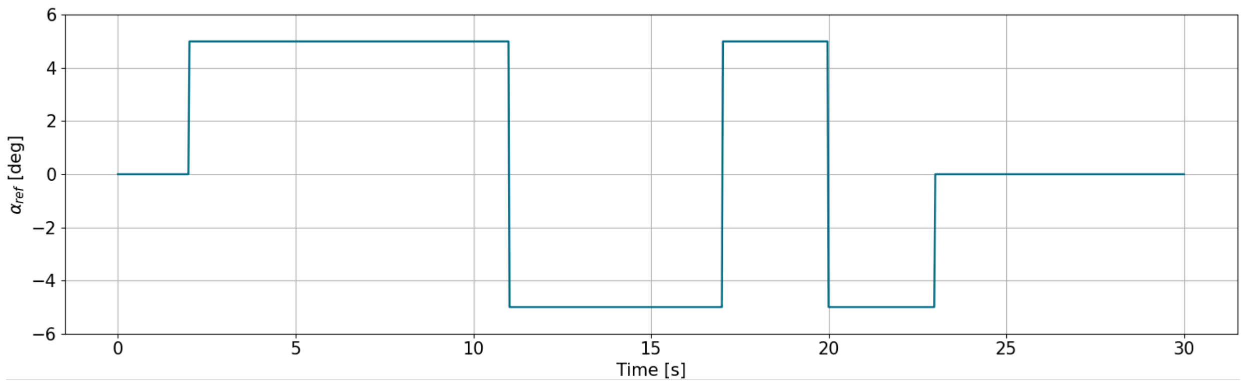


Figure 5.16: A 3-2-1-1 test signal.

The graph at the bottom-left illustrates the mean of the five best performing policies, complemented by its standard deviation. The policy is intuitive: for a positive error, where the actual value of α falls below α_{ref}, the agent opts for a negative action (the elevator deflects upwards) to align the aircraft closer to the reference. For no error, the mean policy brings the elevator towards zero deflection. Furthermore, the policy's behavior resembles a linear controller within the error range of $-5°$ to $5°$. Here, a larger absolute error corresponds to a more pronounced elevator deflection. Beyond this range, the policy consistently reaches a peak negative deviation of $-20°$ for positive errors and approximately $15°$ for negative ones.

In the top-right graph, the agent's α tracking performance is illustrated, which is derived from a policy that achieves the highest average return in the last 10 training episodes. While the trained agent adeptly follows the reference α_{ref}, the aircraft's response exhibits noticeable oscillations. Such oscillatory behavior is undesirable. It not only subjects the aircraft's mechanical components to undue stress but also compromises passenger comfort.

An oscillatory response often indicates that the control system is over-compensating for errors, which also leads to unnecessary consumption of energy. This makes it imperative to examine the agent's actions, as illustrated in the bottom-right plot of Figure 5.17. It's evident that the agent maximally deflects the elevator at the onset of a step. However, this causes an overshoot of the reference. To counteract, the agent responds with a similarly forceful action in the reverse direction, and this cycle continues. This behaviour characterizes a ”bang-bang” control, marked by the elevator's sharp and rapid oscillations. Such frequent switching between extreme states can lead to increased wear and tear on mechanical components of the elevator and is also energy-draining. Furthermore, in some instances, the elevator changes the deflection from $10°$ to $-20°$ within the span of $0.01s$, which is the simulation time step. However, such abrupt changes in the elevator state might be feasible in reality. This behavior arises because there were no imposed constraints upon the elevator rate $\dot{\delta}_e$.

To enhance the smoothness of the actions learned by the DSAC agent and bolster their practical applicability, two distinct methods were explored and compared - Conditioning for Action Policy Smoothness (CAPS) and elevator rate $\dot{\delta}_e$.

Table 5.3: Additional training and evaluation settings and hyperparameters for the DSAC agent in the Short Period LTI Cessna Ce500 environment.

Parameter	**Notation**	**Value**	**Unit**
Temporal settings			
Episode span	T_{tot}	30.0	[s]
Sampling time step	Δt	0.01	[s]
Reward architecture			
Tracking error weight	K_e	1000	[-]
Tracking error weight	K_e	1000	[-]
Action bounds			
Elevator deflection interval	I_{δ_e}	[-20.0, 20.0]	[deg]
Training signal settings			
Max. block amplitude	A_a	5.0	[deg]
Block width	w_s	3.0±0.02	[s]
Num. of block levels	N_{block}	7	[-]
Evaluation signal settings			
Start time	t_{start}	2.0	[s]
Block amplitude	$A_{s,ev}$	5	[deg]
”1” block width	$w_{s,ev}$	3	[s]

5.5.2. DSAC Flight Control with CAPS

CAPS presents an effective regularization technique on action policies [178]. CAPS was previously successfully used in a RL-based flight controllers in the works of Teirlinck and van Kampen [81] and Seres [28]. It consistently enhances the smoothness of the state-to-action mappings in neural network controllers, effectively eliminating high-frequency components from the control signal. Fundamentally, two smoothness loss factors are integrated into the policy loss function:

1. **Temporal Smoothness**: This ensures consistency between subsequent actions, making actions at time t resemble those at time $t+1$:
$$L_T = ||\pi(s_t) - \pi(s_{t+1})||_2 \tag{5.9}$$
where $||.||_2$ denotes the L_2 norm.
2. **Spatial Smoothness**: This ensures that actions are consistent for closely related states, mapping similar states to similar actions:
$$L_S = ||\pi(s) - \pi(\bar{s})||_2 \tag{5.10}$$
where $\bar{s} \sim N(s, \sigma_{\text{CAPS}})$ is drawn from a normal distribution $\mathcal{N}(s, \theta)$ centered around s and σ_{CAPS} is usually based on expected measurement noise.

These CAPS terms are subsequently incorporated into the policy loss function as weighted losses, as presented in Equation 5.11.
$$J_\pi^{CAPS} = J_\pi - \lambda_T L_T - \lambda_S L_S \tag{5.11}$$

The CAPS regularization technique offers the benefit of fostering smoother control policies without adding new states to the learning process, maintaining the problem’s inherent complexity. However, the inclusion of two additional terms in the policy loss function could potentially disrupt the policy optimization, diminishing its efficacy. Additionally, CAPS necessitates the fine-tuning of extra hyperparameters. Finally, while CAPS enhances the practical application of RL-trained controllers, it lacks explicit constraints on the elevator rate, meaning that the real-world applicability of controllers trained with this architecture is not guaranteed.

The additional hyperparameters that were used for the CAPS regularization in this experiment are summarized in Table 5.4. The results of training a DSAC agent on the LTI short period Cessna Ce500 model are displayed in Figure 5.19.

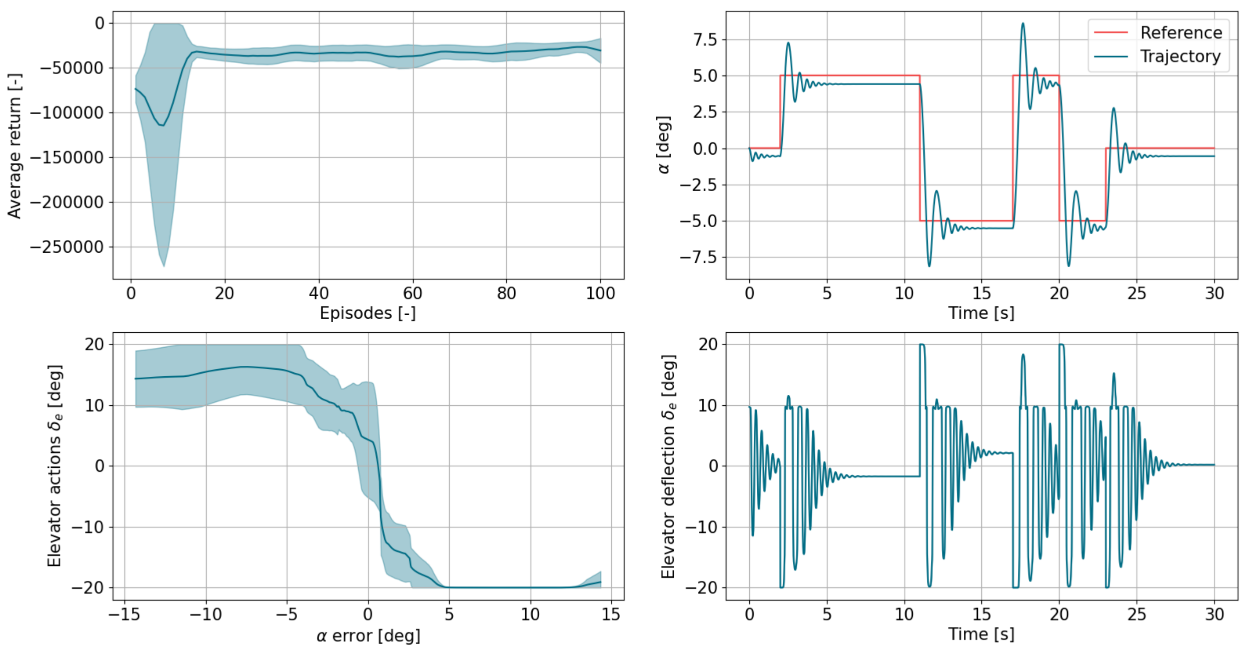


Figure 5.17: Learning and tracking performance of DSAC implemented on the LTI short period Cessna Ce500 model.

Table 5.4: CAPS hyperparameters.

Parameter	**Notation**	**Value**	**Unit**
Temporal smoothness weight	λ_T	300	[-]
Spatial smoothness weight	λ_S	300	[-]
State perturbation standard deviation	σ_{CAPS}	0.05	[-]

In Figure 5.19, the top-left graph demonstrates the policy's still convergence within 13 episodes. The influence of CAPS is prominently evident in the other three graphs. First, the bottom-right graph illustrates that, as expected, the policy with CAPS closely mirrors the shape of its counterpart without CAPS, but with a noticeably smoother profile. This enhanced smoothness can be attributed to the spatial smoothness component, L_S, which ensures actions are consistent for similar states.

Shifting focus to the top-right graph, it highlights a significant enhancement in tracking performance. While the aircraft response retains its oscillatory nature, the frequency of these oscillations is markedly reduced from the baseline scenario. Impressively, the DSAC agent appears to more closely align the aircraft response with the reference α_{ref} trajectory.

The bottom-right graph offers clarity on the dampened oscillations in the response. Observable are the less pronounced oscillations in the elevator deflections, which approach their maximum deflection limits only at the onsets of the reference steps. This not only optimizes power usage but also extends the lifespan of the elevator actuators by reducing wear. However, it's evident that there are instances where the elevator exhibits abrupt deflections, which could pose challenges in practical applications. Consequently, additional tuning of the temporal smoothness parameter is required for optimal performance.

5.5.3. DSAC Incremental Flight Control

This work investigated an alternative smoothing technique that centers on regulating the elevator deflection rate rather than its ultimate deflection. Consequently, the DSAC controller outputs $\dot{\delta_e}$ instead of δ_e. The subsequent action is derived using Equation 5.12, where δ_e denotes the current elevator deflection, δ_e' represents the next deflection, and Δt signifies a simulation time-step.

$$\delta_e' = \delta_e + \dot{\delta}_e \cdot \Delta t \tag{5.12}$$

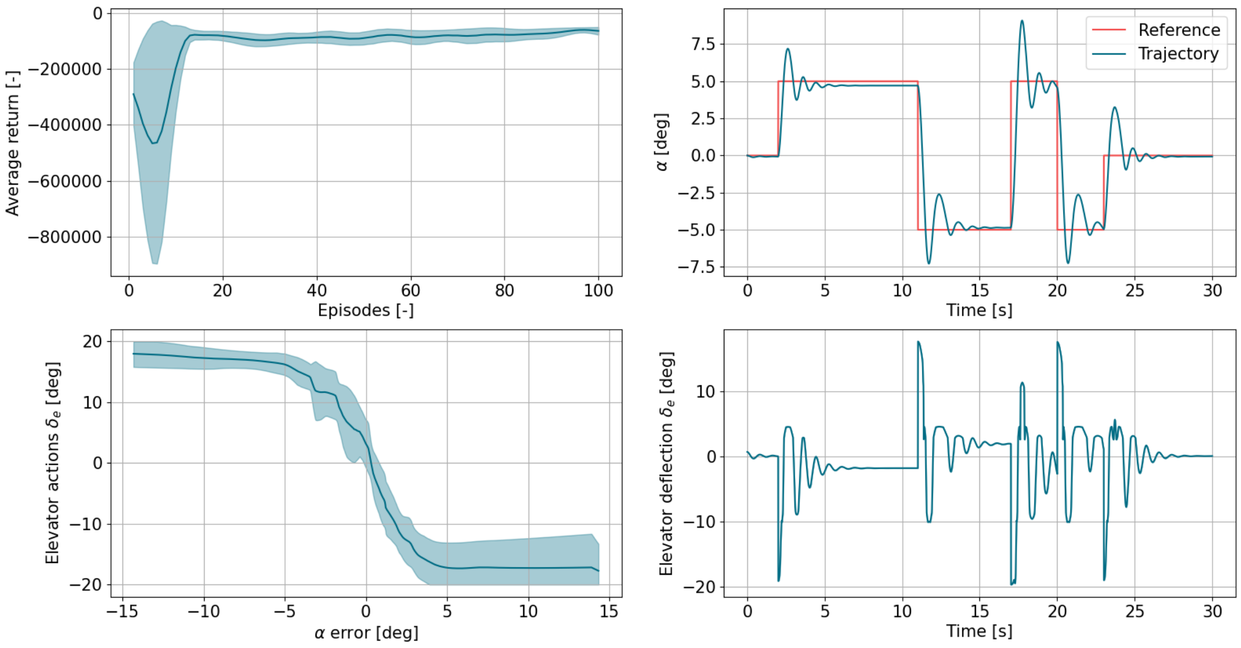


Figure 5.18: Learning and tracking performance of DSAC with CAPS implemented on the LTI short period Cessna Ce500 model.

To accommodate this approach, the controller's architecture was adjusted, as illustrated in Figure 5.19. It now integrates the current elevator deflection as an added input, alongside the error and the reward. The output is defined as $\Delta\delta_e = \dot{\delta}_e \cdot \Delta t$, which, when combined with the existing elevator deflection, yields the new deflection. This approach, where an elevation deflection increment is appended to the prior deflection, is termed as *incremental* in this study.

This revised approach inherently guarantees the feasibility of the controller's actions by limiting the elevator deflection rate. Such a design allows for more nuanced control over the system. Instead of jumping to a target deflection, the controller can make smaller adjustments, which might be more appropriate for situations that require a gentle response. Incremental action can also bolster the control system's stability, especially in situations where the desired setpoint changes frequently or suddenly. This reduces the risk of overshooting the target or inducing oscillations. Additionally, the predictability of the controller's actions is enhanced with this method due to smoother transitions, streamlining monitoring and troubleshooting processes. Yet, this sophistication comes with its own set of challenges: the introduction of an additional input amplifies the algorithm's complexity, which may translate to higher computational needs and slower convergence.

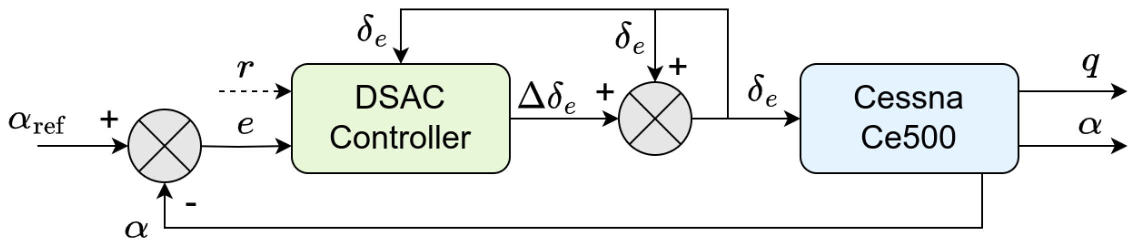


Figure 5.19: DSAC controller for the α tracking in the Cessna Ce500 LTI model (incremental version).

The results of training a DSAC agent with incremental elevator control on the LTI short period Cessna Ce500 model are displayed in Figure 5.20. The top-left graph indicates that the policy now converges within 21 episodes. The policy is represented as a heat map in the bottom-left graph, reflecting the agent's decision-making based on two inputs. This visualization averages the five best-performing policies. Note that the current action pertains to the elevator deflection rate $\dot{\delta}_e$ rather than the elevator deflection δ_e. The policy's logic is evident: for example, given an initial elevator deflection of $0.0°$, a positive tracking error corresponds to a negative elevator deflection rate $\dot{\delta}_e$ (upward motion) and vice versa. Furthermore, the boundary, where the elevator deflection rate transitions from positive to negative, varies with the initial elevator deflection. For instance, when the α error is positive but is accompanied by a pre-existing negative elevator deflection, contributing to error reduction, the α error must be larger to induce a negative elevator rate. If not, the elevator deflection rate $\dot{\delta}_e$ will remain positive even with a positive α error, serving to dampen the aircraft's response.

The damping effect is evident in the top-right graph of Figure 5.20, displaying the tracking performance of the top-performing policy. The DSAC controller effectively mitigates the oscillations in the aircraft's response. However, this benefit came with a noticeable cost in α tracking performance. Compared to the CAPS technique, the aircraft's response to steps is slower, and it struggles to meet the target α_{ref} signal, especially when α_{ref} is negative.

The bottom-right graph showcases the elevator deflections δ_e as the aircraft responds to the 3-2-1-1 reference α_{ref} signal. This representation exhibits significantly fewer oscillations compared to the other two controller versions. This can contribute to smoother flight dynamics and potentially increased safety and comfort for the aircraft and its occupants. Additionally, the elevator deflections peak at just $5.3°$, markedly below the maximum deflection of $20.0°$ observed with the other two methods. This implies reduced mechanical stress on the elevator and its actuating components, potentially leading to longer component lifetimes and reduced maintenance needs.

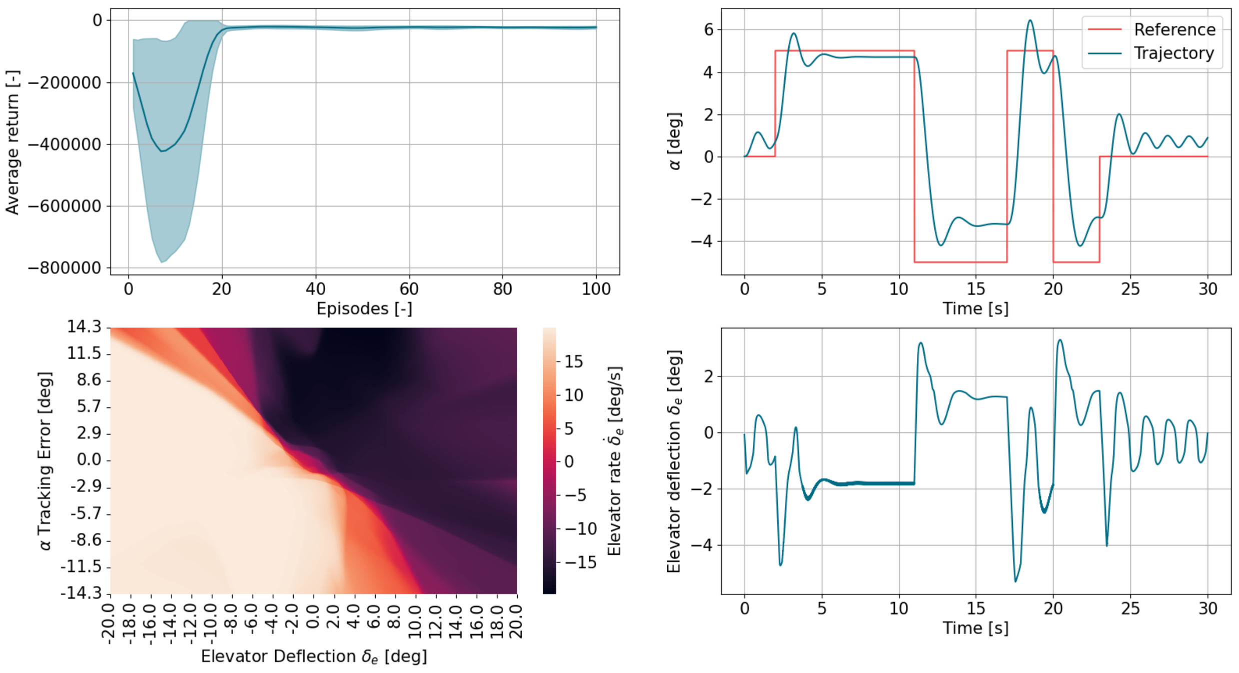


Figure 5.20: Learning and tracking performance of DSAC with incremental elevator control implemented on the LTI short period Cessna Ce500 model.

A comparative evaluation of the three methodologies is presented in Figure 5.21, highlighting the learning curves of each approach. Observations reveal that both the CAPS and incremental techniques manifest greater variance during the learning phase relative to the baseline method. This variance, however, tends to recede upon policy convergence. Notably, a marginally elevated variance persists in the trained policy of the CAPS technique relative to the baseline and incremental methods. A clearer visualization of this observation is provided in Figure 5.22, which presents the distribution of the final scores across the examined variants.

In terms of convergence dynamics, the CAPS method parallels the original controller's trajectory, underscoring a consistent level of problem complexity across both methods. Conversely, the incremental controller demands an extended span of 8 episodes to attain convergence. This observation is attributed to the augmented problem complexity introduced by the incorporation of an extra input.

From a performance standpoint, the CAPS variant shows a modestly reduced final average return when set against the other two methodologies, which can be also seen in Figure 5.22. This outcome is anticipated given that the integration of two supplementary parameters in the CAPS framework can potentially perturb the native policy structure. Consequently, the resultant optimized policy may not achieve the performance levels inherent to its non-augmented counterpart. Pivoting to the incremental approach, it exhibits a marginally superior final average performance relative to the original method, which can be also observed in Figure 5.22. This performance elevation is plausibly derived from the access to an enriched state information set, facilitating the agent's capacity for more nuanced decision-making.

Incorporating an additional state invariably introduces greater complexity, manifesting as increased computational time per episode. As evidenced in Figure 5.23, the incremental controller requires, on average, four times the duration per episode compared to the CAPS variant. The escalation in time complexity, resulting from the inclusion of an additional input, is likely to intensify as the problem's dimensionality increases, which is a consequence of the *curse of dimensionality*.

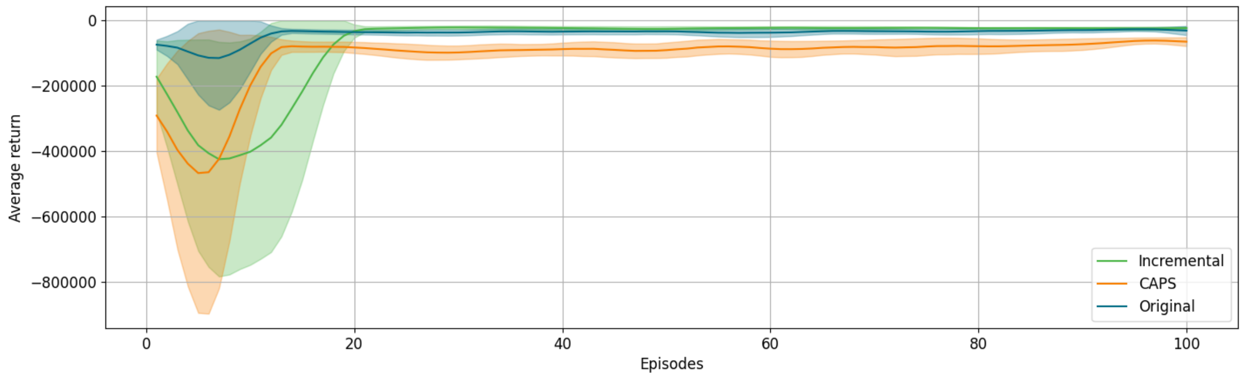


Figure 5.21: Comparison of the learning curves of different DSAC flight controller variants implemented on the LTI short period Cessna Ce500 model.

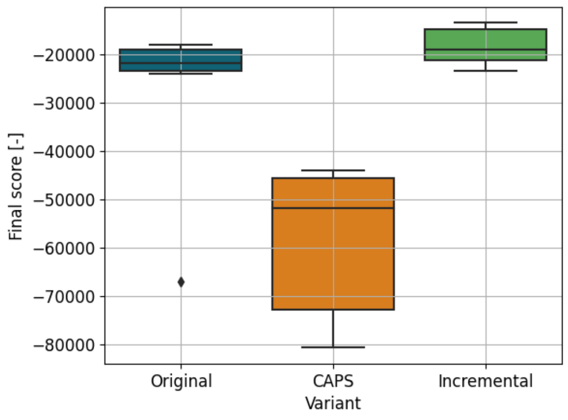


Figure 5.22: Final score distribution for different DSAC variants.

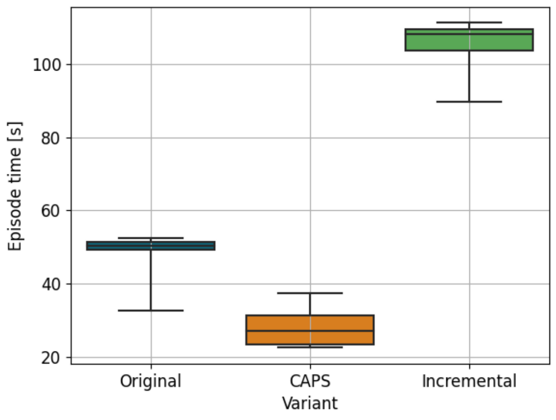


Figure 5.23: Episode time distribution for different DSAC variants.

Table 5.5 shows the normalized Mean Absolute Error (nMAE) for the tracking performance of the 3-2-1-1 reference α_{ref} signal for each of the studied controller versions. The nMAE is calculated with Equation 5.13.

$$\text{nMAE} = \frac{1}{N} \frac{\sum_{i=1}^{N} |\alpha_{\text{ref},i} - \alpha_i|}{(\alpha_{\text{ref, max}} - \alpha_{\text{ref, min}})} \tag{5.13}$$

In Table 5.5, it's evident that the DSAC integrated with the CAPS smoothing technique enhances the tracking performance relative to the original version. Conversely, the incremental control leads to a diminished tracking performance with the test signal, a trend also discernible in the top-right graph of Figure 5.19. Intriguingly, this contrasts with the findings in Figure 5.22, where the incremental variant demonstrated superior performance post-learning. One plausible explanation for this inconsistency is that the incremental model, due to its increased complexity, may have overfitted to the training data, whereas the simpler original or CAPS designs might be more robust in this regard.

Table 5.5: nMAE for α tracking performance using a 3-2-1-1 testing signal for different controller variants.

	Original	CAPS	Incremental
nMAE	10.07%	8.28%	15.78%

5.5.4. Conclusions

In summary, both the CAPS and incremental control techniques demonstrated enhanced smoothness in elevator deflections and the corresponding aircraft response. The CAPS method further exhibited superior tracking performance on the test signal without adding to problem complexity. Nonetheless, it still falls short in ensuring the practical applicability of the learned policy as it does not explicitly cap the elevator deflection rate $\dot{\delta}_e$, and it necessitates tuning for two supplementary hyperparameters. Conversely, incremental control offers a rigorous limit on the elevator deflection rate $\dot{\delta}_e$, guaranteeing refined control without the demand for extra hyperparameter tuning. This advantage, however, is offset by the escalation in problem complexity arising from the inclusion of an extra input, a factor that evidently impairs sample efficiency. The significance of this drawback is anticipated to amplify with a surge in input dimensionality (e.g., in full aircraft control). This escalation is attributed to the *curse of dimensionality*, wherein the computational space expands exponentially with each added dimension. Moreover, despite achieving well-dampened oscillations and minimal elevator deflections δ_e with the incremental controller, there was a discernible decline in tracking performance on the test signal. Given the principal aim of this research, to enhance the sample efficiency of RL-based control algorithms, the CAPS method was chosen for continued exploration in this study.

5.6. Conclusion

The outcomes and insights derived from the conducted experiments within the Preliminary Analysis harmonize closely with the theoretical foundations laid out in the literature review. Significantly, the results illuminated that a shift towards capturing the entire distribution of returns, as opposed to focusing solely on their mean values, yields substantial enhancements in both the efficacy of sample utilization and the stability of learning. This advancement notably surpasses the capabilities of established state-of-the-art methodologies like SAC. Consequently, DSAC emerges as a highly promising avenue for uncertainty-aware approaches, offering potential to elevate the sample efficiency of RL-based flight control systems.

Furthermore, novel variants of the DSAC framework were developed and evaluated to push the boundaries of sample efficiency even further. First, an amalgamation of DSAC with the UMNN developed by Wehenkel and Louppe [51] was explored. This architectural fusion demonstrated comparable learning performance to state-of-the-art DSAC variants, while demonstrating a modest increase in sample efficiency during early learning stages. However, this augmentation did not yield accelerated policy convergence in the later phases, and the computational complexity associated with this approach impeded its applicability to intricate systems such as flight control. Moreover, while enforcing monotonicity through UMNN showed promise initially, the observed loss of monotonicity in the learned quantile functions raised concerns about their accuracy. Interestingly, the original DSAC algorithm exhibited reasonable monotonicity in the quantile functions from early stages of learning, indicating that enforcing additional monotonicity may not yield substantial learning performance improvements.

Conversely, the introduction of the RUN-DSAC strategy, which anchors policy guidance in variance considerations, showcased encouraging results in terms of sample efficiency and computational resource utilization. This approach underscored the latent potential of harnessing uncertainty as a tool for refining RL-based flight controllers. Consequently, the forthcoming research phase will entail the implementation, training, and testing of the RUN-DSAC variant within the ambit of full-scale aircraft control scenarios.

Subsequently, the study delved into the feasibility of implementing the DSAC methodology in flight control, specifically focusing on the angle of attack α tracking task using an LTI model of the Cessna Ce500 aircraft's short period dynamics. The successful training and adept tracking of the reference α_{ref} signal underscore the viability of DSAC's implementation and training in flight control contexts. This chapter not only demonstrates the feasibility of implementing and effectively training this RL algorithm within flight control systems, but also highlights its potential to enhance the sample efficiency of prevailing state-of-the-art methods previously employed in flight control. These findings lend robust support to DSAC's capacity to contribute to the solution of research question **RQ-M-3**. However, it was noted that the controller exhibited pronounced oscillatory actions, yielding oscillatory responses. In response, two smoothing techniques were explored: Conditioning for Action Policy Smoothness (CAPS) and incremental control. Both techniques succeeded in enhancing the smoothness of elevator deflection actions. Notably, while incremental control imposed an upper bound on elevator deflection rate, CAPS facilitated policy smoothness through the introduction of supplementary terms in the policy loss function. Despite the lack of an explicit deflection rate constraint, the CAPS approach was selected for continued exploration due to its non-increase in problem complexity, thereby preserving sample efficiency.

The forthcoming stages of this research endeavor will capitalize on these pivotal findings, delving deeper into the prospects of implementing DSAC within full-scale flight control scenarios, particularly focusing on the dynamics model of the Cessna Citation II aircraft.

Part III

Additional Results

6 In-Flight Failure Robustness and Generalization Evaluation

A comparative analysis was described in Part I, juxtaposing the performance robustness of SAC, DSAC, and both *Conservative* and *Risky* variants of RUN-DSAC against system faults, unforeseen conditions, and external disturbances. This chapter provides further detail on the time-response data corresponding to the same analysis. To facilitate direct comparison, time histories of the traced variables were plotted using identical reference signals. These supplemental results serve to extend the initial findings in Part I, thereby enriching the understanding of the distinctions among the control policies generated by the agents.

As indicated in Table 6.1, the robustness analysis was divided into two distinct categories: Category F, which simulates a range of in-flight fault conditions, and Category G, which evaluates the generalization capability of RL-based flight controllers when facing flight scenarios they were not trained for.

Table 6.1: Comparative analysis of nMAE and standard deviation across thirteen scenarios for SAC, DSAC, and RUN-DSAC algorithms. (R) stands for *Risky*, while (C) stands for *Conservative*. Bold values represent nMAE significantly different from DSAC ($p < 0.05$).

Identifier	Scenario	SAC	DSAC	RUN-DSAC (R)	RUN-DSAC (C)
N0	Nominal	**23.53±4.40**	5.95±1.15	**9.36±2.48**	**3.95±0.42**
F1	Ice Accretion on Wings	**23.77±4.52**	7.68±1.82	**11.40±1.89**	**6.03±0.38**
F2	CG Shifted Aft	**25.49±3.66**	7.81±1.16	8.72±2.71	**5.32±0.96**
F3	CG Shifted Forward	**26.19±5.91**	6.34±1.00	**9.72±2.41**	**4.53±0.64**
F4	Saturated Aileron	**25.78±7.91**	7.67±2.53	**10.45±2.86**	**5.69±0.61**
F5	Saturated Elevator	**26.05±4.75**	9.69±1.46	10.41±3.44	**6.34±0.72**
F6	Damaged Elevator	**27.72±4.64**	10.64±1.81	**13.97±3.71**	**9.03±1.12**
F7	Immobilized Rudder	**40.06±6.32**	34.40±2.97	35.66±1.85	33.31±1.52
G1	Wind Gust	**24.05±5.10**	6.15±1.15	**9.67±2.67**	**4.07±0.46**
G2	Noisy Signal	**26.07±3.55**	6.10±1.15	**9.33±2.32**	**4.17±0.86**
G3	High Dynamic Pressure	**29.91±4.15**	7.38±3.64	9.51±3.28	**4.06±0.85**
G4	Low Dynamic Pressure	**31.18±4.32**	9.23±1.41	10.81±2.26	**7.46±0.50**
G5	Stall	**54.21±14.04**	14.57±3.14	18.87±7.83	**9.76±1.70**

N0: Nominal Condition

To set the baseline for comparison, the RL agents are first subjected to a test scenario consistent with their training trim conditions, specifically at an altitude h of $2000\ m$ and a true airspeed V_{tas} of $90\ m/s$. The trajectory responses for this scenario are depicted in a series of figures: Figure 6.1 for the SAC agent, Figure 6.2 for the DSAC agent, Figure 6.3 for the *Risky* RUN-DSAC, and Figure 6.4 for the *Conservative* RUN-DSAC.

In the analysis of the SAC agent's response (Figure 6.1), the agent exhibits limited efficacy in tracking the three reference trajectories, as indicated by significant deviations and high variability in signal responses, which can be also concluded from Table 6.1. Such subpar performance aligns with the SAC learning curve, as discussed in Part I, characterized by lower average returns and considerably higher variance within 250 training episodes compared to distributional agents. This pattern can be attributed to the poor convergence properties of SAC, as highlighted in prior research by Dally and van Kampen [179], which reported a mere 26% success rate for SAC in converging on attitude control tasks. Although the SAC trajectories shown by Dally and van Kampen are smoother and show improved tracking, it is important to note that they represent the responses of the selected top converged agents, contrasting with the unfiltered training approach used in this study. Other strategies to enhance the SAC's performance may involve enlarging the action space (see Section 8.2) or extending the number of training frames.

The DSAC agent demonstrates a significant improvement over the SAC, as evidenced in its superior tracking performance and smoother actions. The *Conservative* RUN-DSAC agent further elevates this performance, exhibiting the most precise adherence to the reference signals and the least variability among all evaluated agents, as proved in Table 6.1. In contrast, the *Risky* RUN-DSAC demonstrates a decline in tracking performance and increased state variance relative to the other distributional agents. This outcome aligns with theoretical expectations, as the *Risky* RUN-DSAC agent is designed to prioritize uncertainty exploration during the learning process. Consequently, this propensity for exploration manifests as oscillatory actuator commands, evidenced by increased variance in actuator deflections. This, in turn, induces greater noise in angular velocities, predominantly in the pitch rate q, compared to its *Conservative* counterpart.

The state time-traces of the distributional agents, which are easier to interpret due to less noise than the SAC, also reveal a coupling between pitch and roll. This coupling becomes apparent, for example, at around $t = 10s$, when the roll is initiated, leading to a minor degradation in pitch tracking. Additionally, the coupling between roll and yaw dynamics is clearly observable, as reflected in the yaw rate graphs of the distributional agents, which mimic the shape of the roll reference trajectory. The agents appear to have learnt these couplings, as reflected in the correlation observed in the actuator commands. This is the most notable for the *Conservative* RUN-DSAC agent, which deflects the elevator to trim the aircraft at the beginning of the response (visible as a sharp negative spike in the elevator deflection), which also seems to trigger sharp deflections in the aileron and rudder, despite zero reference commands for roll and yaw. These adjustments induce transient roll and yaw rates that the agent then quickly neutralizes, returning the control surfaces to their neutral positions.

In yaw control, all distributional agents effectively regulate sideslip angles, maintaining minimal deviations, where even the SAC agent manages to keep the sideslip angle within approximately [-3°, 3°] on average, albeit with notable variance. Remarkably, the *Conservative* RUN-DSAC consistently maintains the mean yaw angle within a narrow range of [-1°, 1°], indicating superior yaw stability. Conversely, the *Risky* RUN-DSAC shows a slight challenge in counteracting slight negative yaw deviations.

F1: Ice Accretion on Wings

Ice accretion on aircraft wing surfaces constitutes a dynamic and complex fault condition, characterized by its gradual evolution, which imposes challenges on its timely detection and monitoring. This aerodynamic perturbation typically results in an amplification in the skin friction drag coefficient and a downward shift in the lift coefficient curve. For the purpose of modeling, the fault induced by ice accretion is quantified as a 0.06 increase in the drag coefficient (corresponding to a conservative increase in the skin friction drag of up to $\sim 200\%$) and a 30% reduction in the lift curve. The trajectory responses for the icing scenario are depicted in a series of figures: Figure 6.5 for the SAC agent, Figure 6.6 for the DSAC agent, Figure 6.7 for the *Risky* RUN-DSAC, and Figure 6.8 for the *Conservative* RUN-DSAC.

The icing scenario results in tracking degradation, as evidenced by Table 6.1. The most pronounced deterioration is observed in pitch tracking under positive reference angles, presumably due to diminished lift. This is further exacerbated by an increased drag coefficient, as the aircraft must overcome additional resistance to maintain pitch during maneuver, which can lead to greater deviations from the desired pitch attitude. Notably, this effect is more discernible among distributional agents, whereas the SAC agent exhibits noisier responses that nonetheless show similar tracking deterioration. The drag increment is also mirrored by a reduced aircraft velocity during maneuvers compared to the nominal condition.

Increased tracking challenges at positive pitch angles necessitate increased compensatory efforts from the agents, evidenced by the elevator's increased negative deflection when compared to the nominal case.

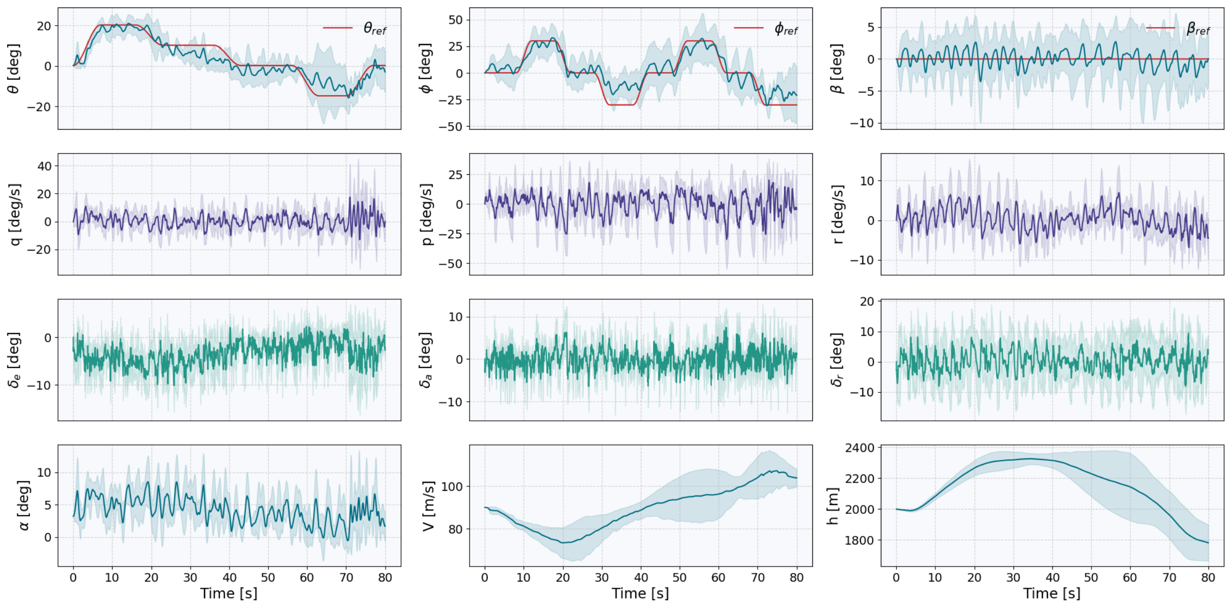


Figure 6.1: States time-traces of the SAC policy in the **nominal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

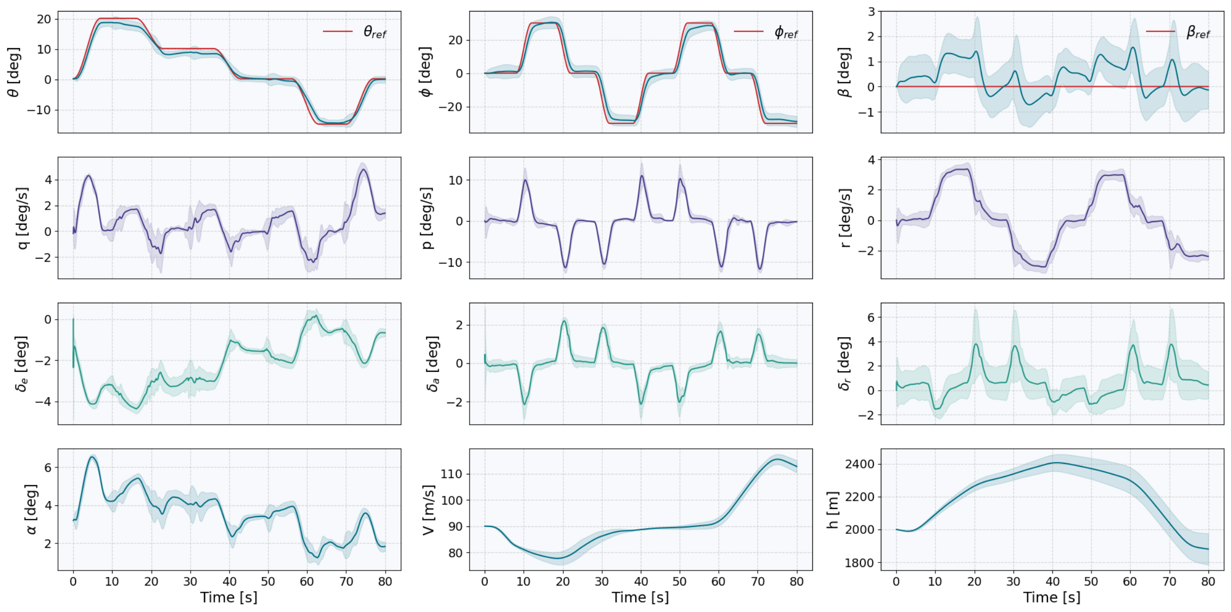


Figure 6.2: States time-traces of the DSAC policy in the **nominal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

Furthermore, oscillatory state response patterns emerge at $t \approx 10s$ and persist when the pitch reference is positive. This behavior is most acute in the DSAC and the *Conservative* RUN-DSAC agents, which exhibit a 29.08% and a 52.8% increase in mean nMAE tracking degradation, respectively (contrasting with the 21.8% for *Risky* RUN-DSAC and only 1.0% for SAC). These oscillations, accentuated in angular rates and control surface deflections, correlate with high angles of attack. Due to the 30% reduction in the maximum lift coefficient, it is hypothesized that these high angles of attack could result in flight near the stall boundary, which is characterized by a decreased aerodynamic efficiency and a non-linear lift response, which could induce the oscillations and for which the controller was not trained.

Increasing oscillations are observed particularly in the pitch rate trajectory of the *Conservative* RUN-DSAC when the pitch reference is positive. This indicates a potential instability issue. While the *Conservative*

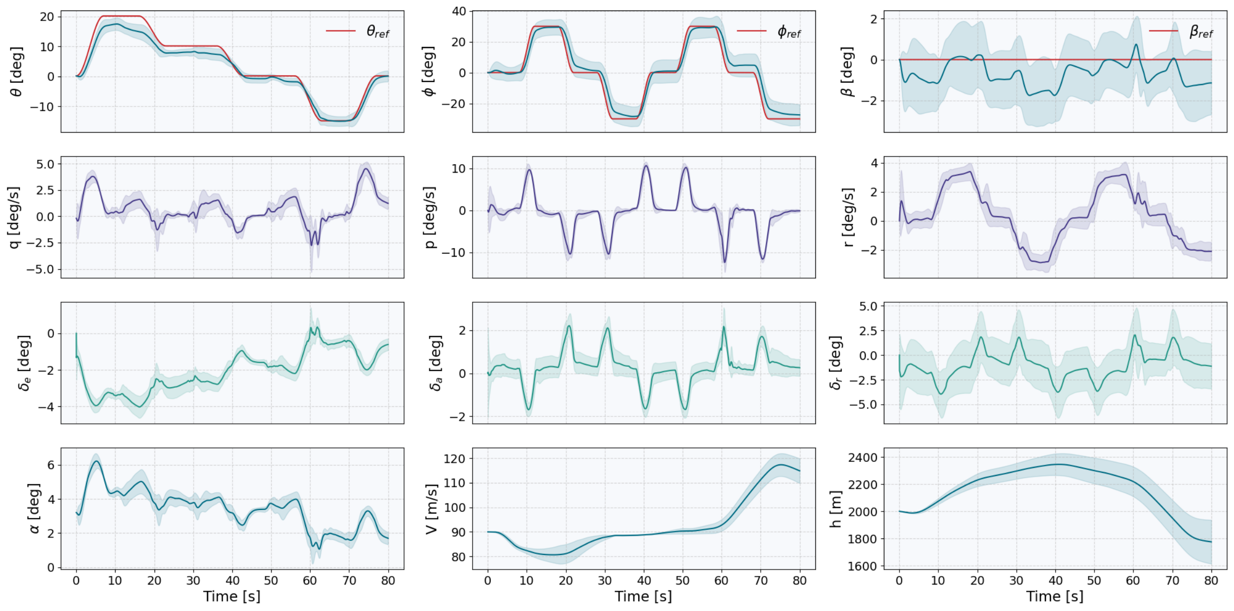


Figure 6.3: States time-traces of the *Risky* RUN-DSAC policy in the **nominal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

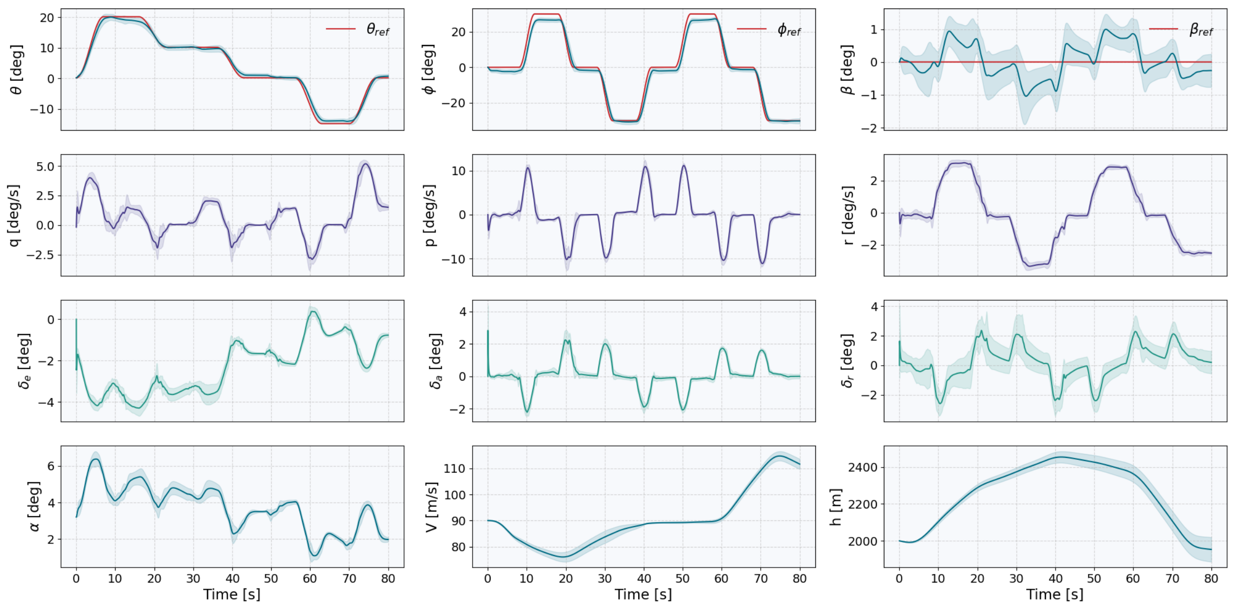


Figure 6.4: States time-traces of the *Conservative* RUN-DSAC policy in the **nominal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

RUN-DSAC's elevator was engaged to compensate for the oscillations, it's deflections seem to escalate over time, which suggest an overcompensation that exacerbates rather than mitigates the instability. This disturbance is then propagated into the lateral dynamics due to the longitudinal-lateral coupling, manifesting as oscillations in roll and yaw. Such response is undesirable, as it heightens structural stress and may compromise safety and comfort. In contrast, the DSAC exhibits less pronounced oscillations and these are minimally evident in the *Risky* RUN-DSAC approach. Hence, prioritizing predictability, the *Conservative* RUN-DSAC algorithm may inherently bias towards stable policies within its training environment. However, this approach risks overfitting, as it potentially underrepresents the breadth of varied and unpredictable outcomes essential for robust performance in diverse, unencountered scenarios. Consequently, this could result in suboptimal responses.

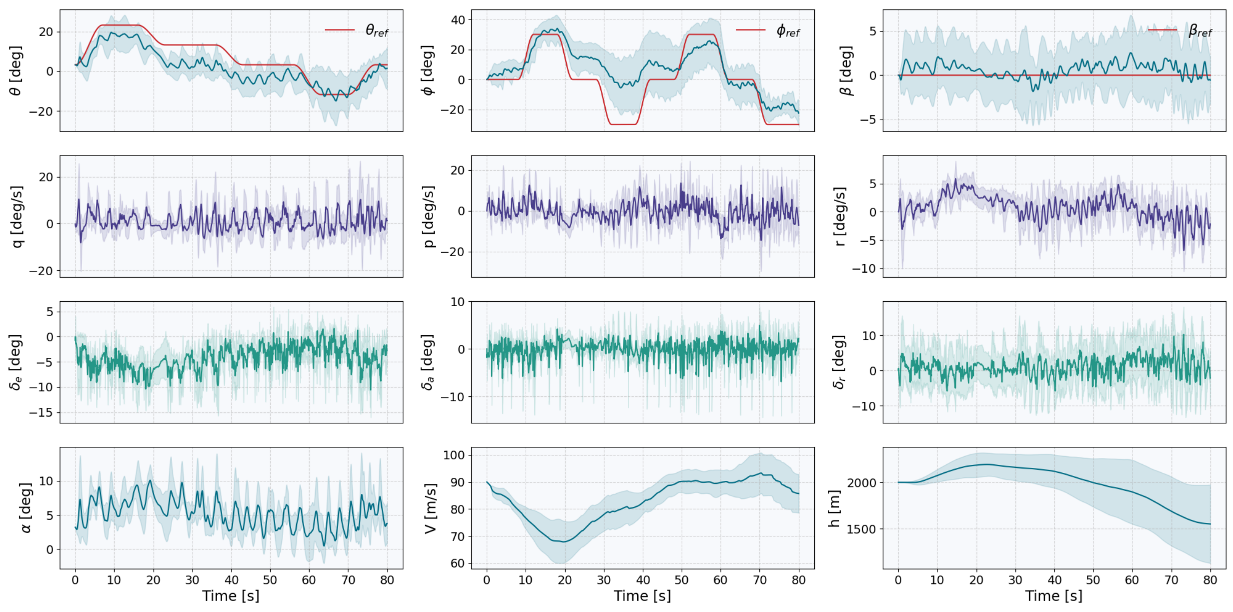


Figure 6.5: States time-traces of the SAC policy in the **ice accretion** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

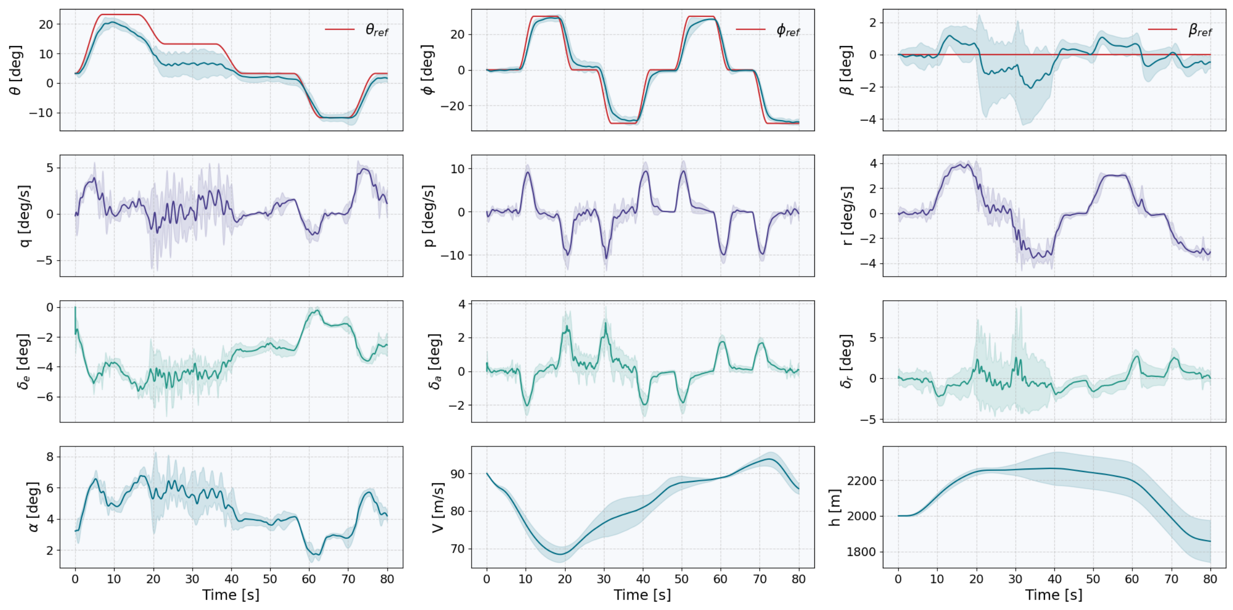


Figure 6.6: States time-traces of the DSAC policy in the **ice accretion** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

F2: Center of Gravity Shifted Aft

Translocating the center of gravity (CG) aft by $0.25\ m$ (equivalent to relocating a ∼300kg payload from the front to the rear of the passenger cabin in a Cessna Citation II) brings the aircraft closer to the margin of its static stability. While such altered dynamics can still be managed through feedback control systems, it is prone to instability during pitch-up maneuvers. The trajectory responses for the aft-shifted CG scenario are

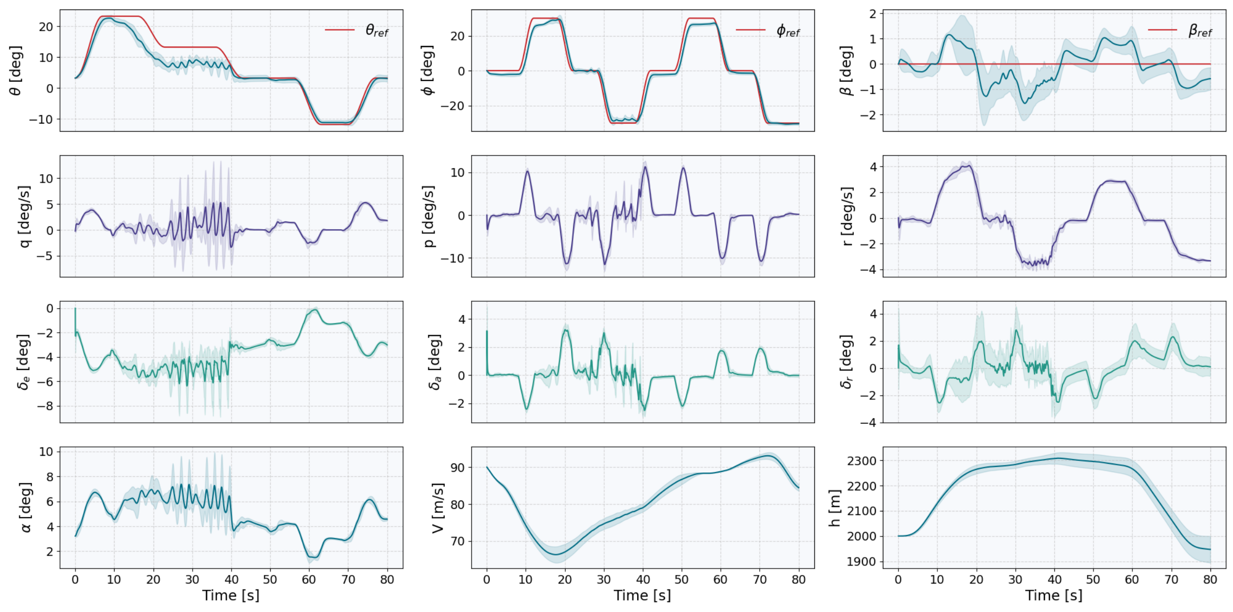


Figure 6.7: States time-traces of the *Risky* RUN-DSAC policy in the **ice accretion** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

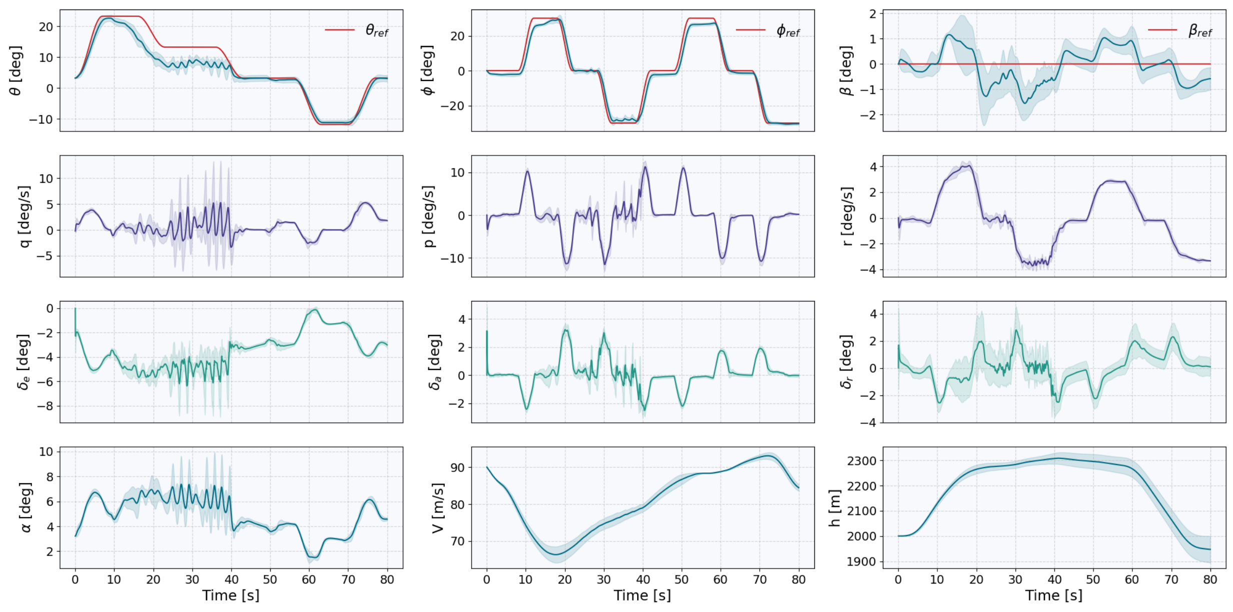


Figure 6.8: States time-traces of the *Conservative* RUN-DSAC policy in the **ice accretion** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

portrayed in a series of figures: Figure 6.9 for the SAC agent, Figure 6.10 for the DSAC agent, Figure 6.11 for the *Risky* RUN-DSAC, and Figure 6.12 for the *Conservative* RUN-DSAC.

The performance of the *Conservative* RUN-DSAC was the most impacted in terms of the relative increase in normalized nMAE with respect to the nominal condition, rising by 34.68%. Despite this, it continued to exhibit superior tracking performance compared to all the other agents. The nMAE for SAC rose by 8.32%, and for DSAC, it escalated by 31.36%. Notably, the *Risky* RUN-DSAC demonstrated an unexpected improvement, reducing its nMAE by 6.84%.

Upon the aftward shift of the CG, distributional agents (particularly DSAC and *Conservative* RUN-DSAC) exhibit a minor overshoot in pitch angle over the reference, resulting also in a greater altitude gain compared to the nominal response. This phenomenon can be attributed to the aircraft's reduced static stability margin, as the proximity of the CG to the aerodynamic center increases the aircraft's sensitivity to pitch inputs and external disturbances. Intriguingly, this condition appears advantageous for the pitch tracking capabilities of the *Risky* RUN-DSAC agent, which previously faced challenges in achieving high pitch angles under nominal conditions, but it adversely affects the pitch tracking performance of other agents. This explains the decrease in nMAE for the *Risky* RUN-DSAC agent in this scenario.

With the CG moved aft, the aircraft naturally tends to pitch upwards, which requires less upward elevator deflection. Instead, this alteration in dynamics prompts the agents to adopt a more positive elevator deflection in response to longitudinal moments induced by the shift, as opposed to their response under nominal conditions, to reduce the nose-up moments due to the elevator deflection. Meanwhile, the performance in sideslip tracking remains similar to the nominal scenario, underscoring the minimal impact of CG shift on directional stability and control, despite the reduction in tail arm length.

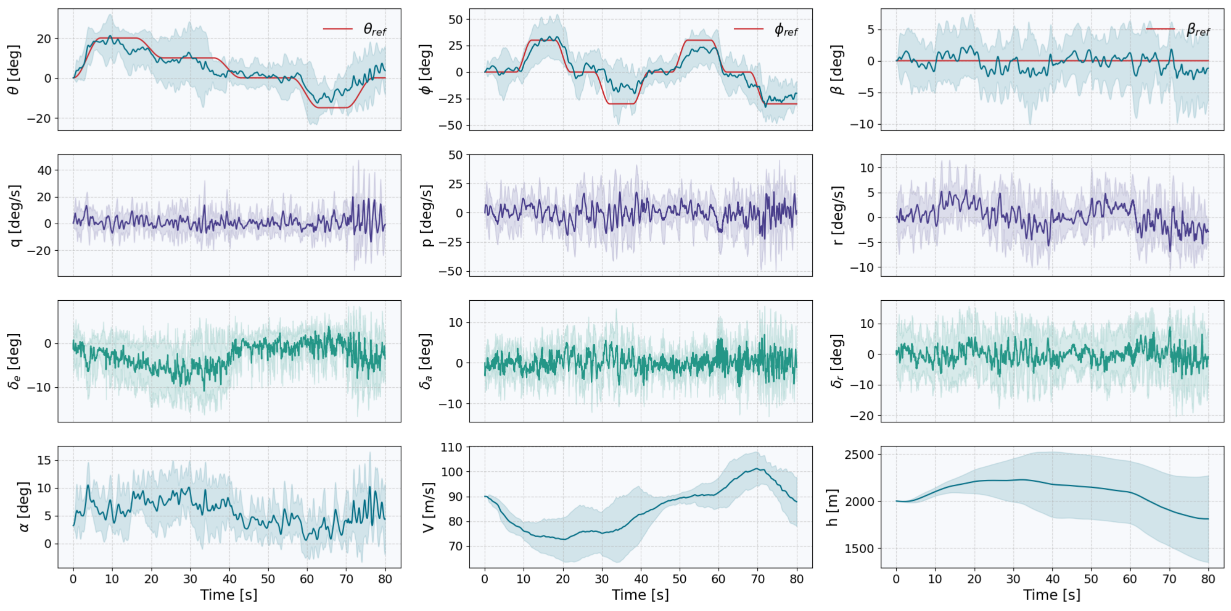


Figure 6.9: States time-traces of the SAC policy in the **aft-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

F3: Center of Gravity Shifted Forward

Conducting tests with a forward displacement of the CG by $0.25\ m$ situates the aircraft at a point more distant from its static stability margin. This test serves to assess the RL-agents' adaptability to alterations in aircraft dynamics, specifically increased static stability and diminished maneuverability, for which the agent was not originally trained. The trajectory responses for the forward-shifted CG scenario are shown in a series of figures: Figure 6.13 for the SAC agent, Figure 6.14 for the DSAC agent, Figure 6.15 for the *Risky* RUN-DSAC, and Figure 6.16 for the *Conservative* RUN-DSAC.

In comparison to the nominal scenario, the nMAE for SAC and DSAC worsened by 11.30% and 6.55%, respectively. The *Risky* RUN-DSAC exhibited the least degradation in performance, with a mere 3.85% increase in nMAE, a finding corroborated by the minimal disparities observed in its response patterns. Conversely, the *Conservative* RUN-DSAC experienced the most significant performance decline, with its nMAE escalating by 14.68%.

In the forward-shifted CG scenario, all agents encountered difficulties in achieving higher positive pitch angles, which was the most pronounced in the performance of the *Conservative* RUN-DSAC in comparison to its nominal performance. This challenge stems from the enhanced static stability of the aircraft, which reduces its responsiveness to pitch control inputs. Given that the agents were trained under nominal conditions, they may not have learned the necessary control adjustments for the increased stability and

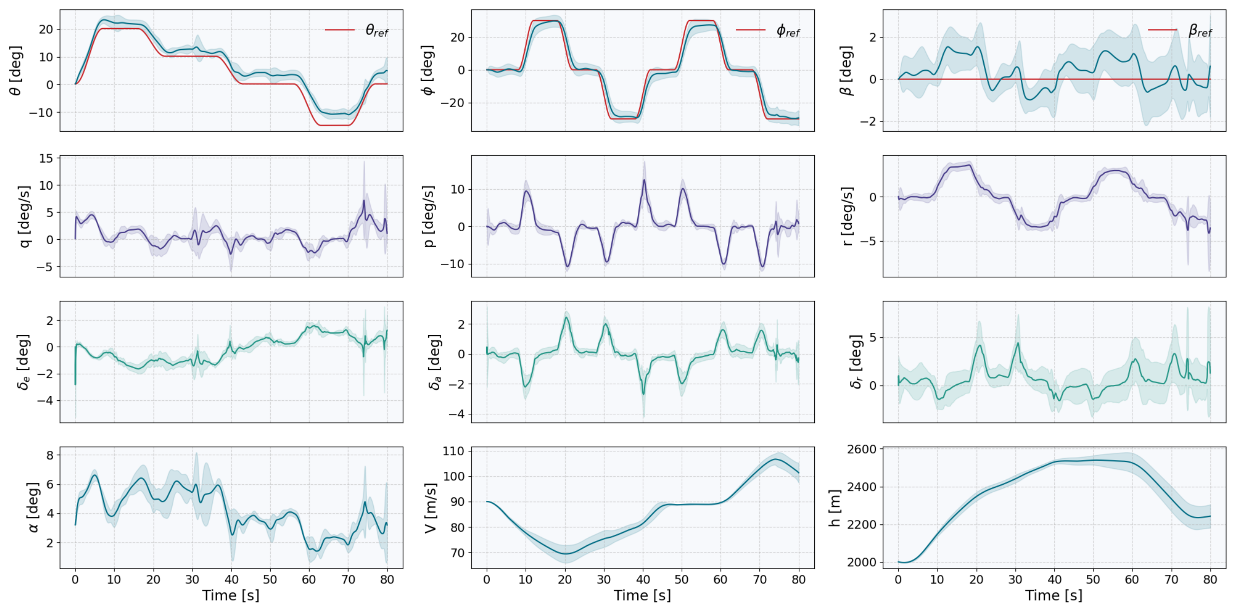


Figure 6.10: States time-traces of the DSAC policy in the **aft-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

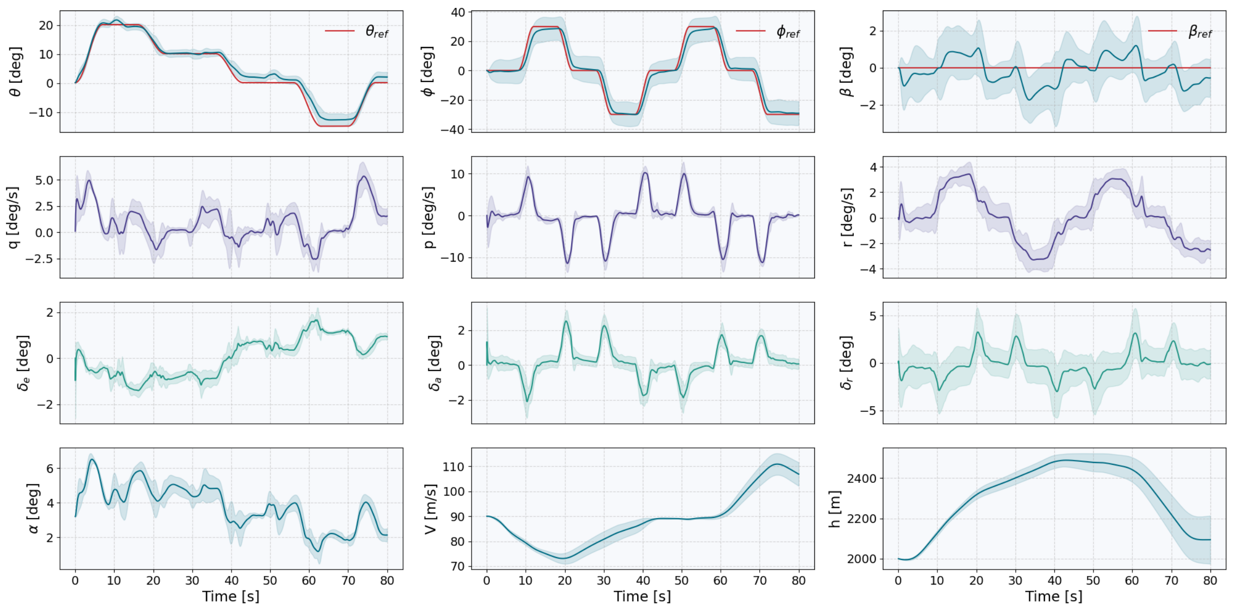


Figure 6.11: States time-traces of the *Risky* RUN-DSAC policy in the **aft-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

decreased maneuverability associated with the forward CG shift. Moreover, the *Conservative* RUN-DSAC exhibits an increased variance in its pitch angle response for high reference pitch angles, where this variance also extends to velocity trajectories. In contrast, yaw control appears relatively unaffected for the agents, despite the forward CG shift influencing this axis as well, primarily due to the elongation of the tail arm (which increases the directional stability).

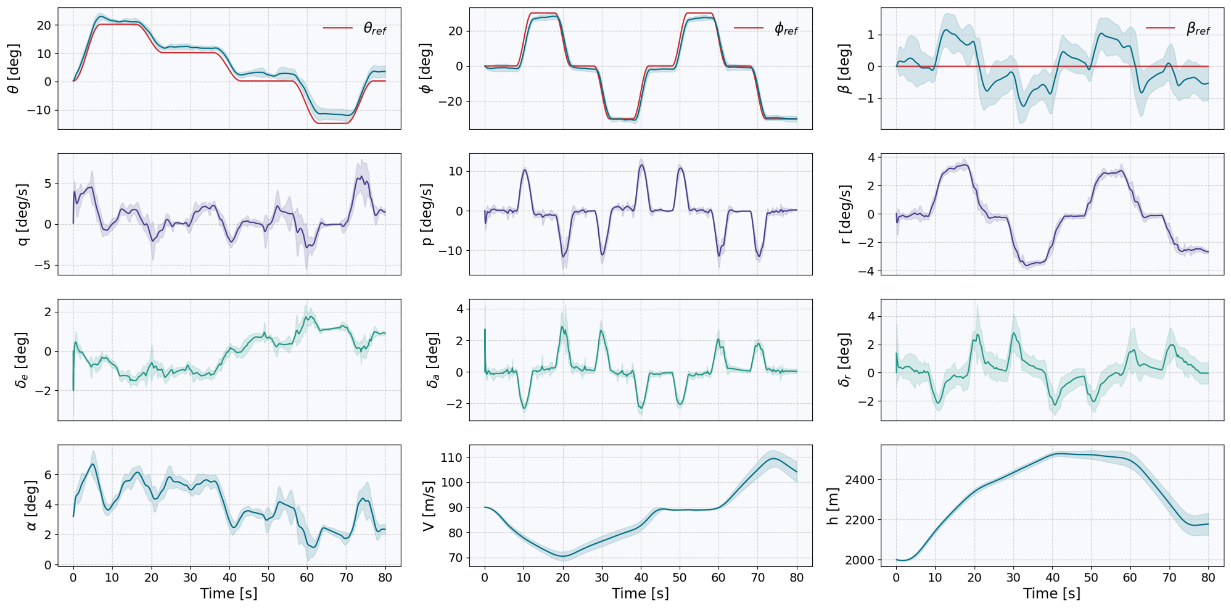


Figure 6.12: States time-traces of the *Conservative* RUN-DSAC policy in the **aft-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

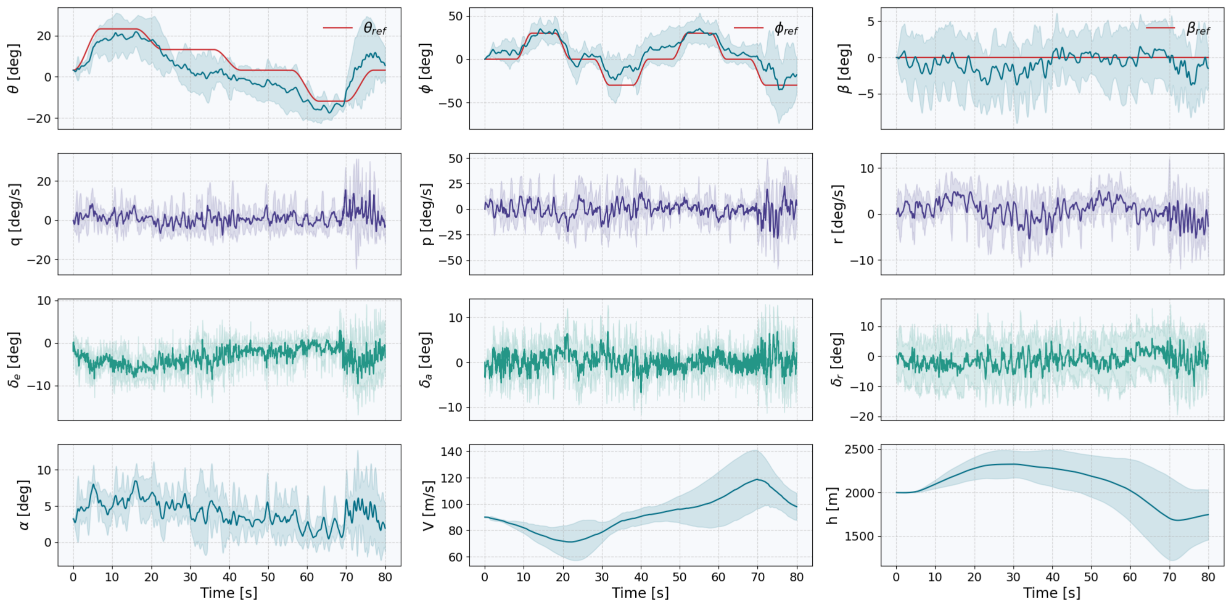


Figure 6.13: States time-traces of the SAC policy in the **forward-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

F4: Saturated Aileron

In the **F4** scenario, the aileron deflection is restricted and clipped at $\pm 1^\circ$. Testing this case serves to emulate real-world situations where the control surface may experience saturation due to mechanical failures. Furthermore, the imposed restriction on the aileron deflection challenges the RL-agent's ability to maintain lateral stability and control under constrained conditions. The trajectory responses for the saturated aileron scenario are shown in a series of figures: Figure 6.17 for the SAC agent, Figure 6.18 for the DSAC agent, Figure 6.19 for the *Risky* RUN-DSAC, and Figure 6.20 for the *Conservative* RUN-DSAC. The interrupted yellow lines in these graphs represent the aileron saturation limits.

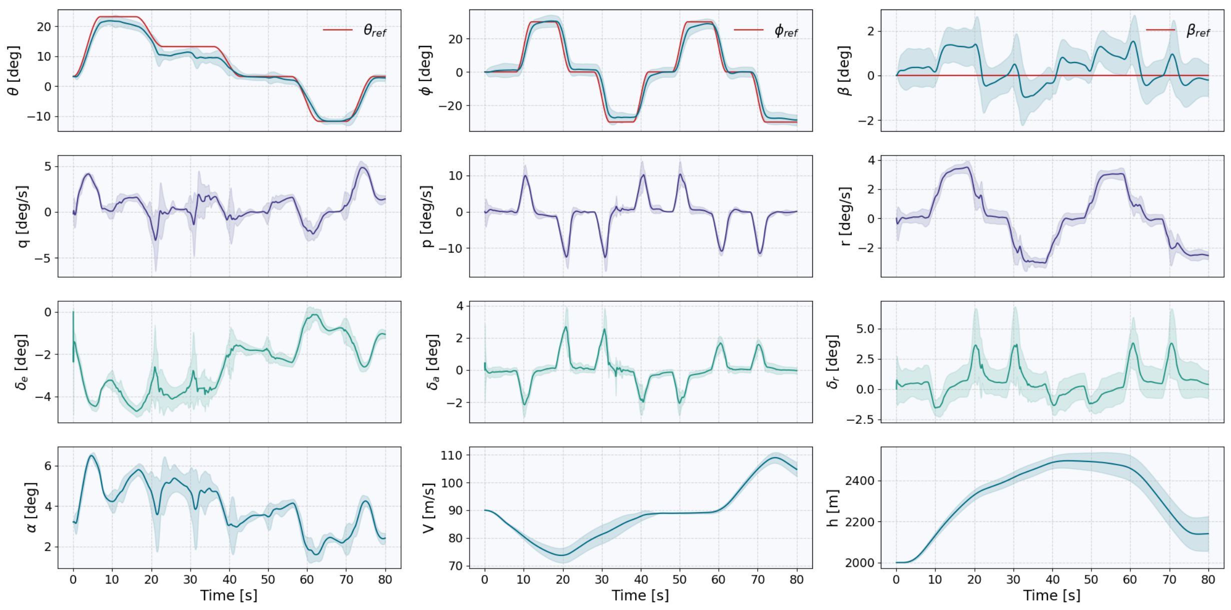


Figure 6.14: States time-traces of the DSAC policy in the **forward-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

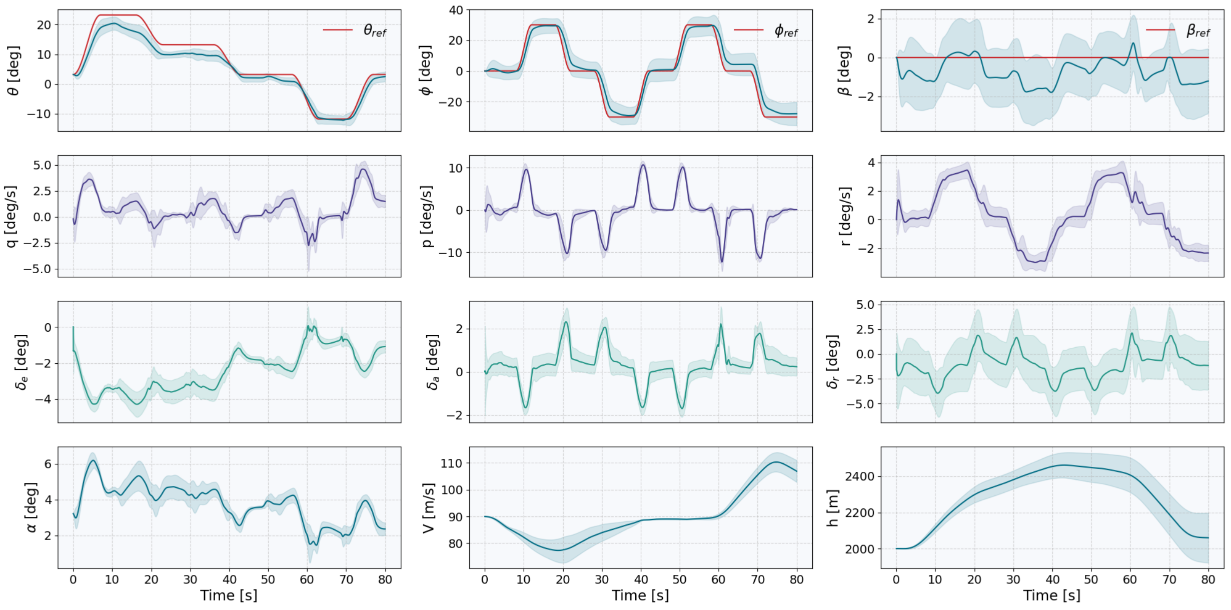


Figure 6.15: States time-traces of the *Risky* RUN-DSAC policy in the **forward-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

Relative to the nominal scenario, the nMAE for SAC and DSAC increased by 9.56% and 28.91%, respectively. Among the RUN-DSAC variants, the *Risky* RUN-DSAC variant exhibited an 11.65% increase in nMAE, while the *Conservative* RUN-DSAC variant encountered the most pronounced performance degradation, with its nMAE escalating by 44.05%.

During the training phase, expansive actuating limits enable the control policies to leverage these bounds for aggressive control actions. However, these learned behaviors become ineffective when aileron movements are subsequently capped at $\pm 1°$, leading to a significant impairment in roll angle tracking, particularly for SAC. The impact is also evident among the distributional agents, with responses becoming less accurate

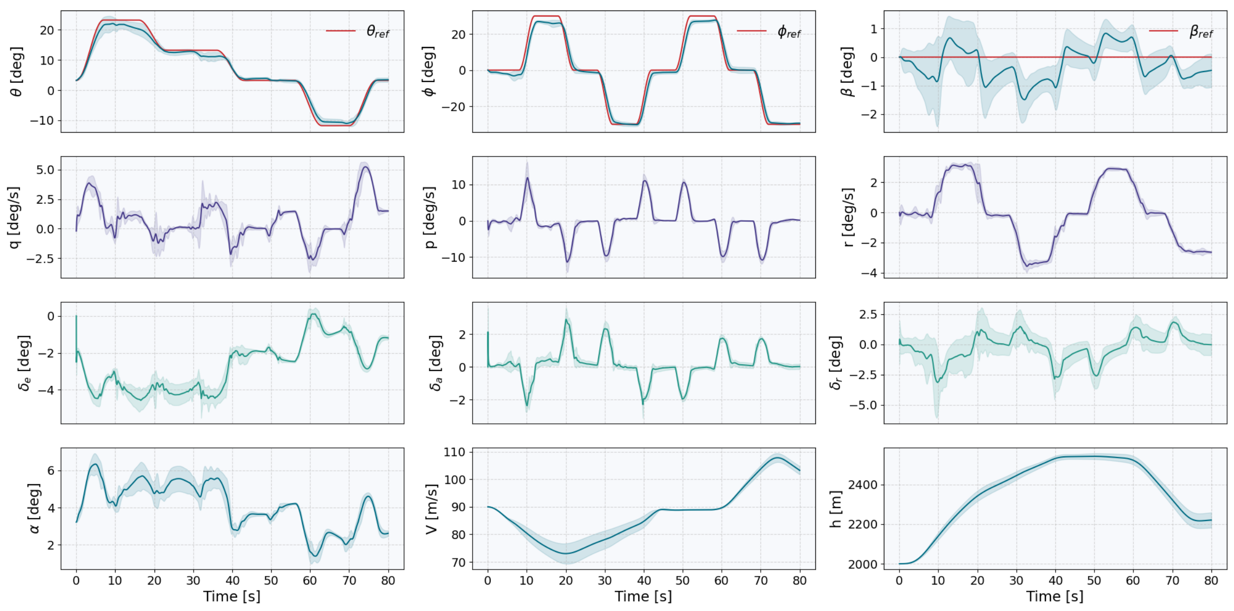


Figure 6.16: States time-traces of the *Conservative* RUN-DSAC policy in the **forward-shifted CG** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

and exhibiting increased lag compared to nominal conditions. Unaware of the fault, the distributional agents are attempting to counteract by pushing the aileron deflections further, reaching approximately $\pm 5°$, contrasted with the $\pm 2°$ in nominal conditions. Consequently, the observed roll rates are diminished, exhibiting a $\sim$20% decrease for *Risky* RUN-DSAC, $\sim$30% for *Conservative* RUN-DSAC, and up to $\sim$50% for DSAC.

Besides the roll tracking, degradation in sideslip control is also observed for the agents. This may be attributed to the agent's internal model, which has learned to correlate increased aileron deflection with compensatory rudder inputs to counteract adverse yaw. Consequently, rudder deflections intensify due to the agent's learned behavior of increasing rudder input proportionally with aileron deflection, ultimately impairing sideslip control.

Additionally, the imposed aileron constraints lead to notable oscillations in both elevator and rudder deflections and in the pitch and roll rates upon aileron actuation, which are particularly evident in the *Conservative* RUN-DSAC's response. These oscillations are presumably due to an overcompensation feedback loop: the agent, not accounting for the aileron limitation, erroneously predicts aileron-induced coupled response in pitch and yaw, and preemptively adjusts the elevator and rudder to counteract these expected effects. However, the predicted aerodynamic responses fail to manifest due to the constrained actuator, leading to unnecessary counteractive control surface movements. Consequently, this instigates a feedback loop of corrective actions by the elevator and rudder, culminating in the observed oscillations.

F5: Saturated Elevator

The **F5** fault scenario models the occurrence of elevator saturation, specifically restricting its deflection to a range of $\pm 2.5°$. This scenario models another form of operational limitation, which is affecting the aircraft's longitudinal control authority. Similar to the **F4** condition, this constraint may arise due to mechanical faults in real-world flight scenarios. The purpose of evaluating this restrictive case is twofold: it tests the RL-agent's proficiency in managing pitch control under limitations and assesses how such constraints could impact the overall flight stability and performance. The response trajectories for the saturated elevator scenario are depicted across several figures: Figure 6.21 illustrates the SAC agent's performance, Figure 6.22 details the DSAC agent's response, Figure 6.23 presents the *Risky* RUN-DSAC's behavior, and Figure 6.24 portrays the *Conservative* RUN-DSAC's response. The dashed yellow lines in these graphs denote the imposed elevator saturation boundaries.

Relative to the nominal scenario, the normalized Mean Absolute Error (nMAE) increased by 10.71% for

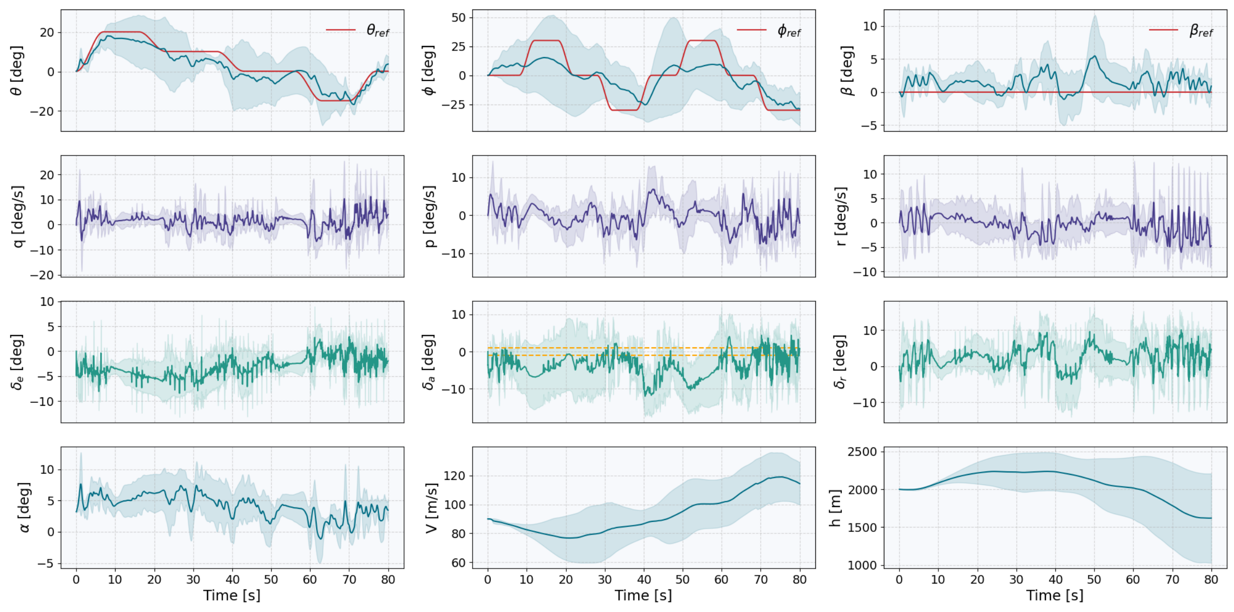


Figure 6.17: States time-traces of the SAC policy in the **saturated aileron** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

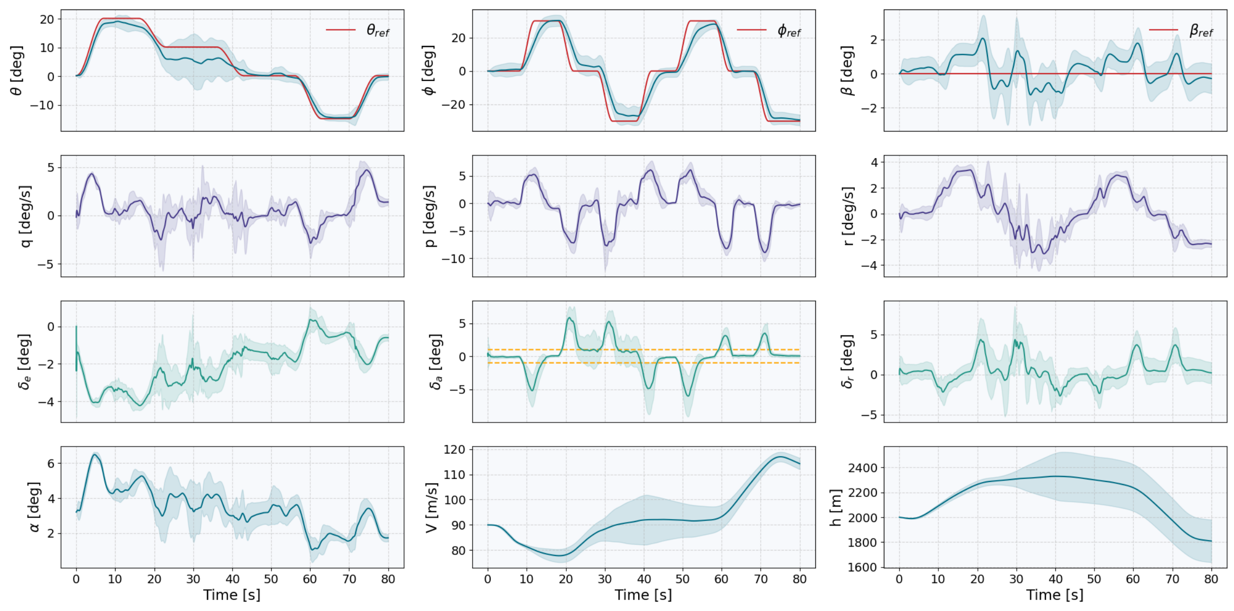


Figure 6.18: States time-traces of the DSAC policy in the **saturated aileron** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

SAC, 62.86% for DSAC, 11.22% for *Risky* RUN-DSAC, and 60.51% for *Conservative* RUN-DSAC.

In the constrained elevator scenario, agents exhibit an inability to achieve mainly high positive pitch angles due to the imposed restrictions, which is consistent with the asymmetric impact of the saturation limits - the trim elevator deflection at $\delta_{e,\text{trim}} = -1.43°$ suggests that negative commands are affected more, thereby impeding steep pitch-up maneuvers. This constraint logically leads to a reduction in maximum pitch rate. To address this limitation, the agents, which were trained under unrestricted conditions and are unaware of the fault, double the negative elevator deflection. This excessive deflection triggers subtle perturbations in both roll and yaw due to the learnt inter-axis coupling, similar to the saturated aileron scenario. These are

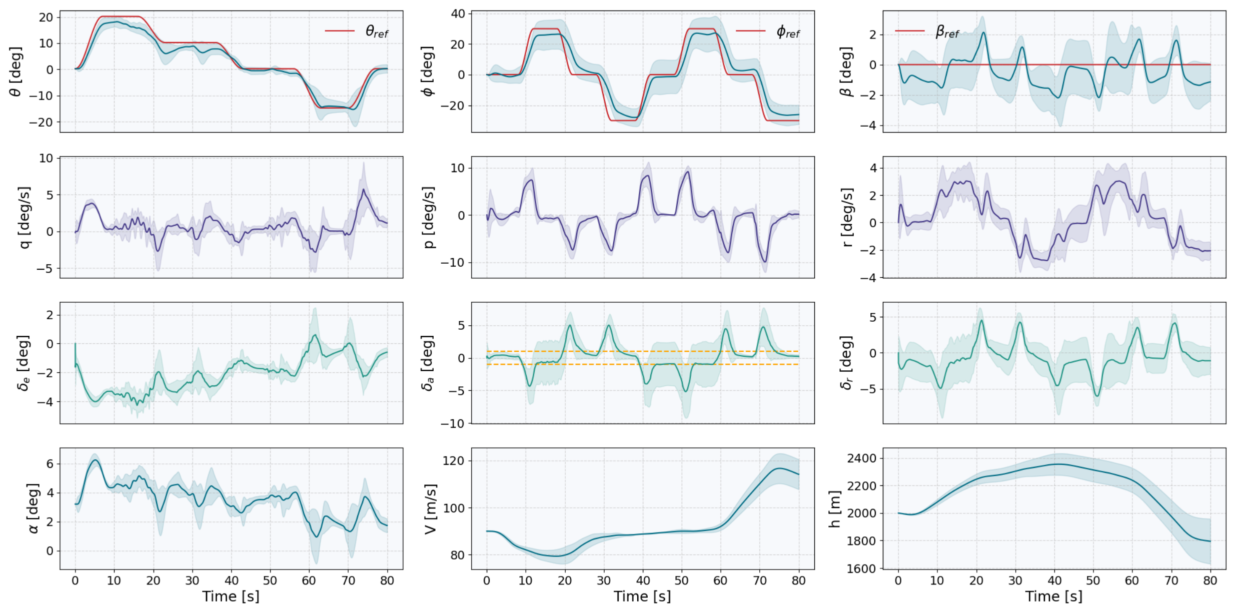


Figure 6.19: States time-traces of the *Risky* RUN-DSAC policy in the **saturated aileron** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

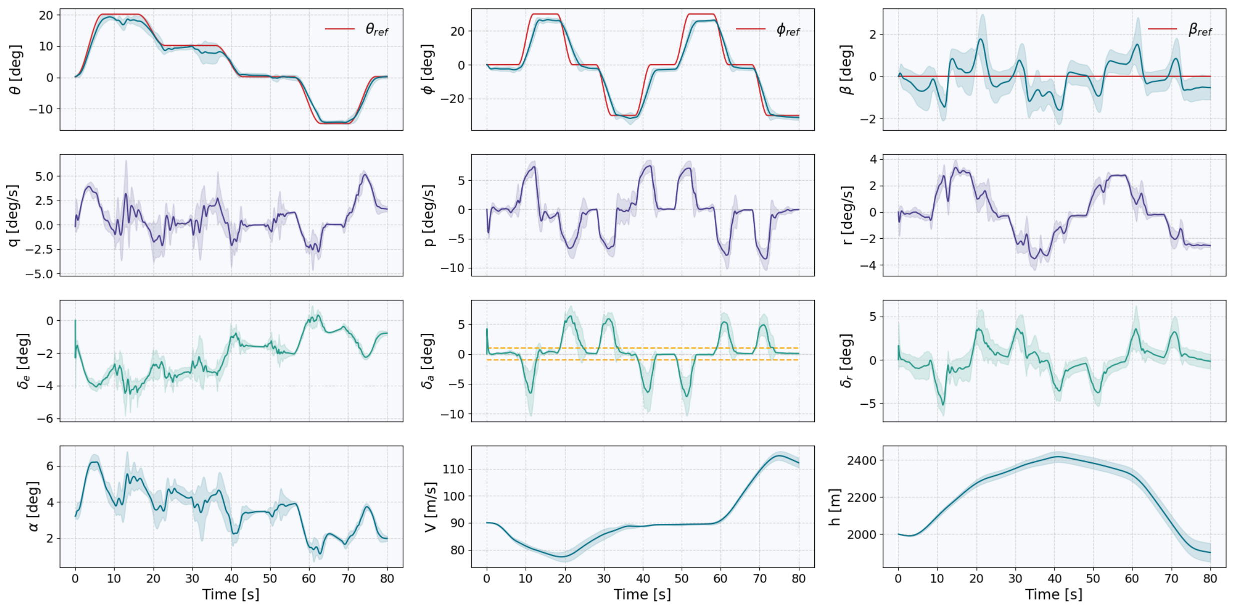


Figure 6.20: States time-traces of the *Conservative* RUN-DSAC policy in the **saturated aileron** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

mainly visible in the DSAC and *Conservative* RUN-DSAC case.

Interestingly, the *Conservative* RUN-DSAC's failure to reach high positive angles of attack appears to inadvertently benefit its roll tracking capability, as evidenced by the successful attainment of the first positive roll reference step, albeit with heightened variability. This phenomenon is attributable to heightened dynamic pressure due to the aircraft's maintained airspeed of $\sim 90m/s$, resulting from its restricted pitch-up capability. Finally, despite the increased variance during periods demanding positive pitch, the *Conservative* RUN-DSAC maintains the most accurate tracking performance with the lowest response variance among all agents evaluated.

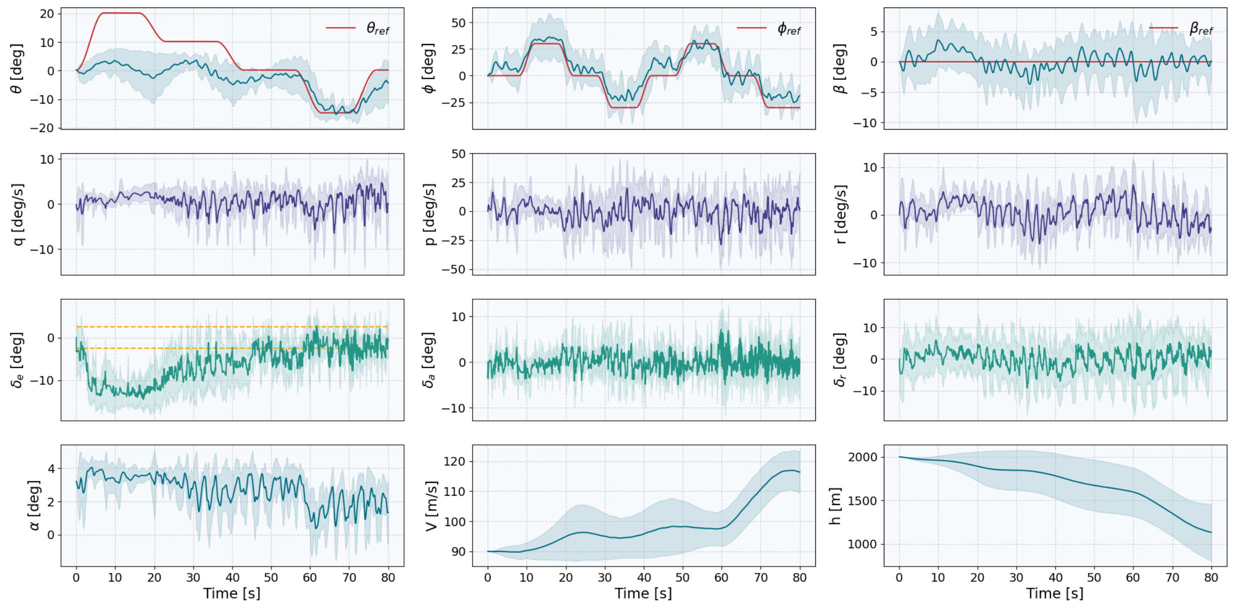


Figure 6.21: States time-traces of the SAC policy in the **saturated elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

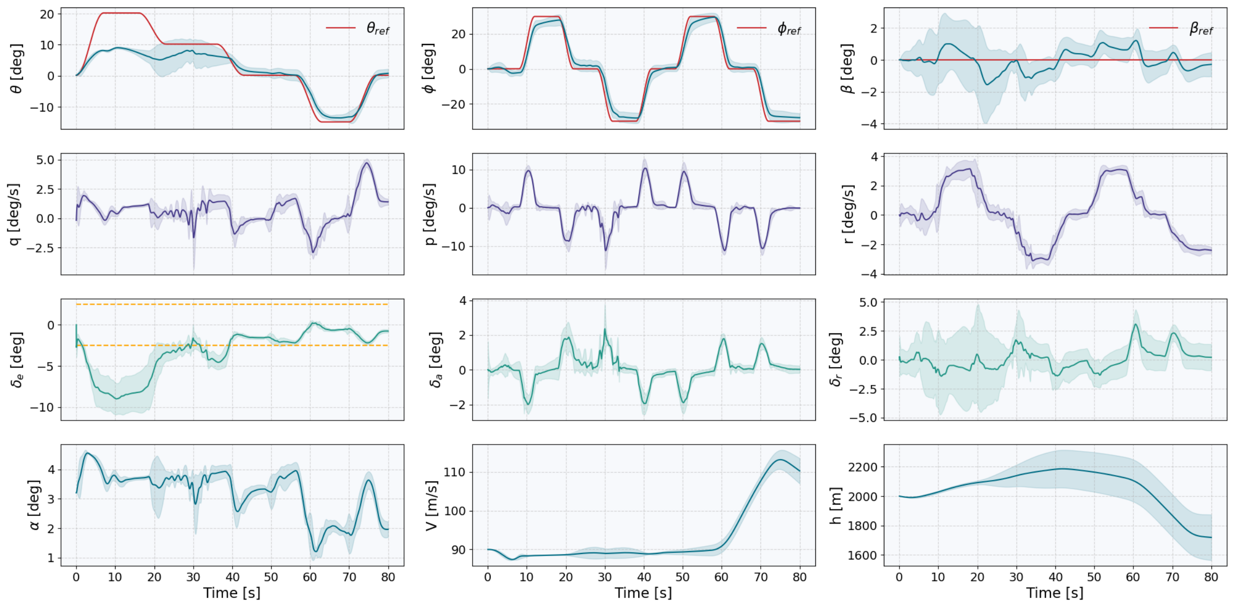


Figure 6.22: States time-traces of the DSAC policy in the **saturated elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

F6: Damaged Elevator

A compromised integrity of the horizontal stabilizer, which could arise from incidents such as tail strikes upon take-off or in-flight structure failure, leads to an acute deterioration in the effectiveness of longitudinal control systems. Such a fault results in a precipitous diminution of the aircraft's capacity to generate pitch moments. This is simulated by applying a gain factor of 0.3 to the aerodynamic coefficients associated with elevator functionality, thereby quantifying the compromised operational efficacy in the longitudinal control domain. The flight path responses in the damaged elevator scenario are illustrated in a series of charts: Figure 6.25 shows the SAC agent's performance, Figure 6.26 captures the DSAC agent's behavior, Figure 6.27 reveals

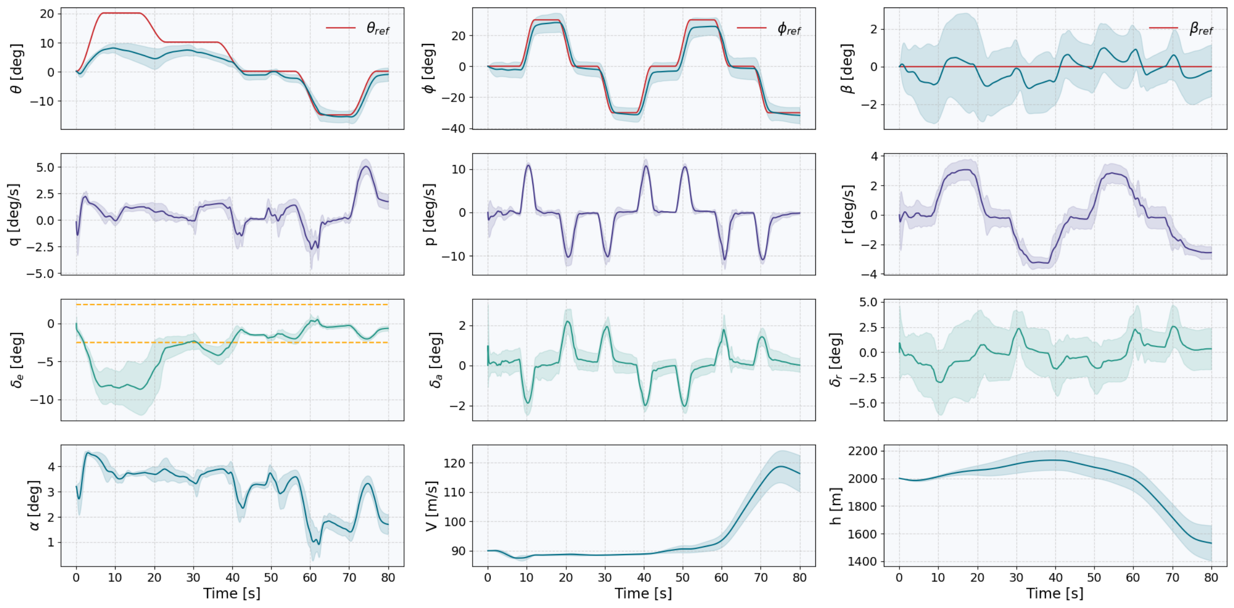


Figure 6.23: States time-traces of the *Risky* RUN-DSAC policy in the **saturated elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

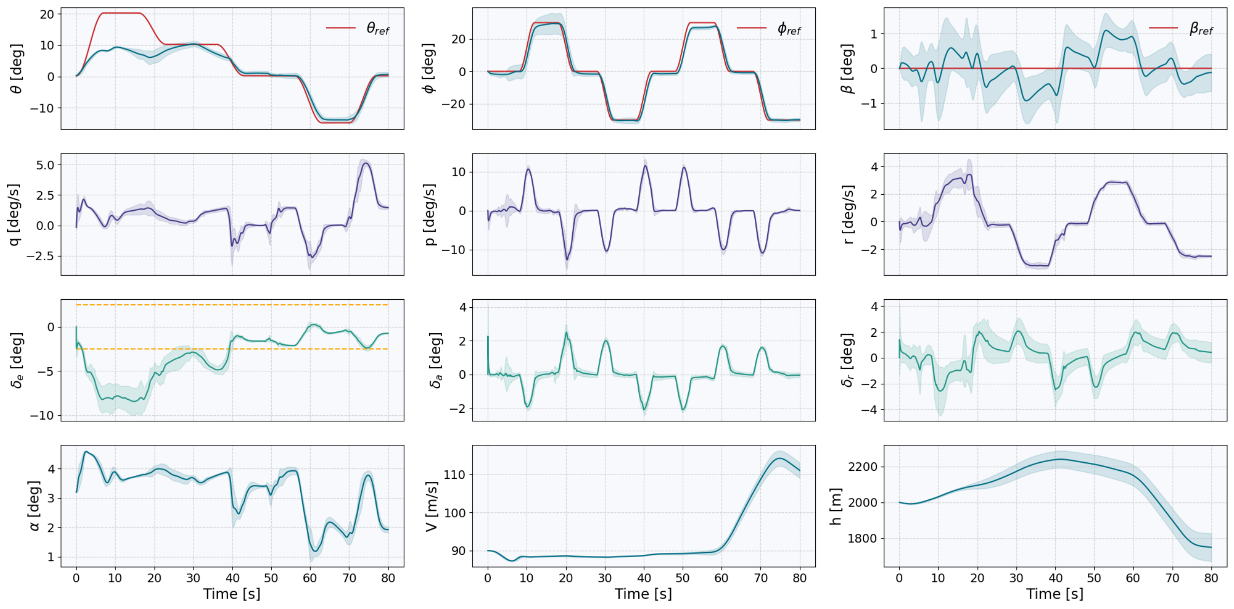


Figure 6.24: States time-traces of the *Conservative* RUN-DSAC policy in the **saturated elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

the *Risky* RUN-DSAC's actions, and Figure 6.28 depicts the *Conservative* RUN-DSAC's maneuvers.

In comparison with the nominal scenario, the relative increase in the nMAE was observed at 17.81% for SAC, 78.82% for DSAC, and 49.25% for *Risky* RUN-DSAC. The *Conservative* RUN-DSAC encountered the most significant deterioration, with a surge of 128.61% in nMAE.

The reduced effectiveness of longitudinal control markedly compromises the aircraft's response in a manner similar to what is observed in the elevator saturation scenario. The agents are particularly challenged in achieving positive pitch angles due to the established trim settings, although tracking of negative pitch angles is also more adversely affected than in the saturated elevator scenario. This is due to the fact that even smaller elevator deflections are also affected, leading to reduced pitch rate and increased time delays in response.

Again, the agents, which are unaware of the fault, attempt to counteract the deficiency by commanding greater elevator deflections. This response exacerbates the nonlinear coupling effects, adversely influencing the roll and sideslip behavior, especially when the aircraft is required to maintain positive pitch angles with high negative elevator inputs. These disturbances are even more pronounced than in the saturated elevator case, yet their manifestation varies among the agents. For the distributional agents, DSAC and *Conservative* RUN-DSAC exhibit the most significant perturbations, whereas the responses from *Risky* RUN-DSAC remain relatively stable and disturbances are barely noticeable. The pronounced disturbances in DSAC and *Conservative* RUN-DSAC hint at their heightened sensitivity to changes in control surface effectiveness, potentially due to a propensity for overfitting within the training envelope. In contrast, *Risky* RUN-DSAC's minimal response disturbances align with previous observations, suggesting its lower likelihood of overfitting and a possible intrinsic robustness to unexpected flight dynamics despite lower tracking performance.

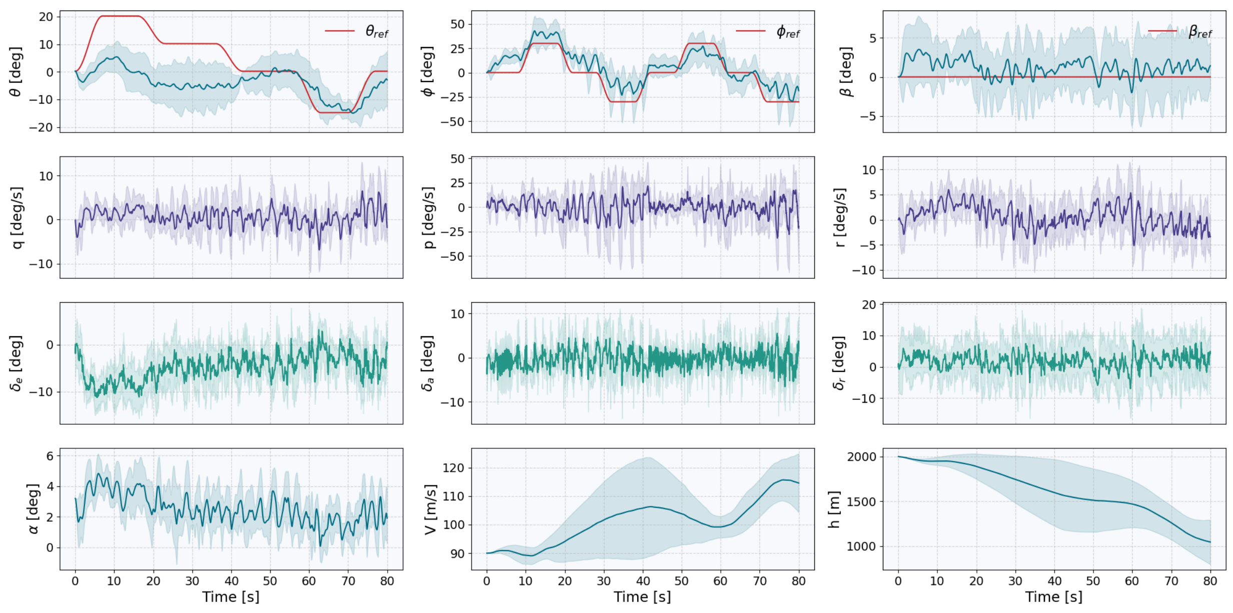


Figure 6.25: States time-traces of the SAC policy in the **damaged elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The aileron saturation limits are depicted as dashed yellow lines.

F7: Immobilized Rudder

The **F7** fault case models a scenario, wherein the rudder remains immobilized at a fixed deflection of $15°$. Such a condition could arise from various sources, such as actuator mechanical failures, control system malfunctions, or damage due to external factors like bird strikes or in-flight icing. This failure mode presents a unique set of challenges as it severely impacts the aircraft's directional control, especially during critical flight phases like take-off and landing. Furthermore, it imposes asymmetrical aerodynamic loads that can lead to yaw instability, potentially resulting in adverse roll effects. A sequence of figures demonstrates the trajectory responses in the jammed rudder scenario: Figure 6.29 displays the performance of the SAC agent, Figure 6.30 delineates the DSAC agent's conduct, Figure 6.31 exhibits the actions of the *Risky* RUN-DSAC, and Figure 6.32 visualizes the *Conservative* RUN-DSAC's strategies. The interrupted yellow lines in these graphs represent the actual rudder deflection.

The case of a locked rudder posed a formidable challenge for the agents, as evidenced by a significant increase in their tracking nMAE. Specifically, the nMAE increased by 70.25% for SAC, 478.15% for DSAC, 280.98% for the *Risky* RUN-DSAC variant, and 743.29% for the *Conservative* RUN-DSAC variant, which even reached the nMAE of DSAC.

The agents, unable to nullify the sideslip due to the jammed rudder, are observed to maintain a steady sideslip angle of $\sim$10°. As anticipated, they attempt to deflect the rudder in the opposite direction to the one

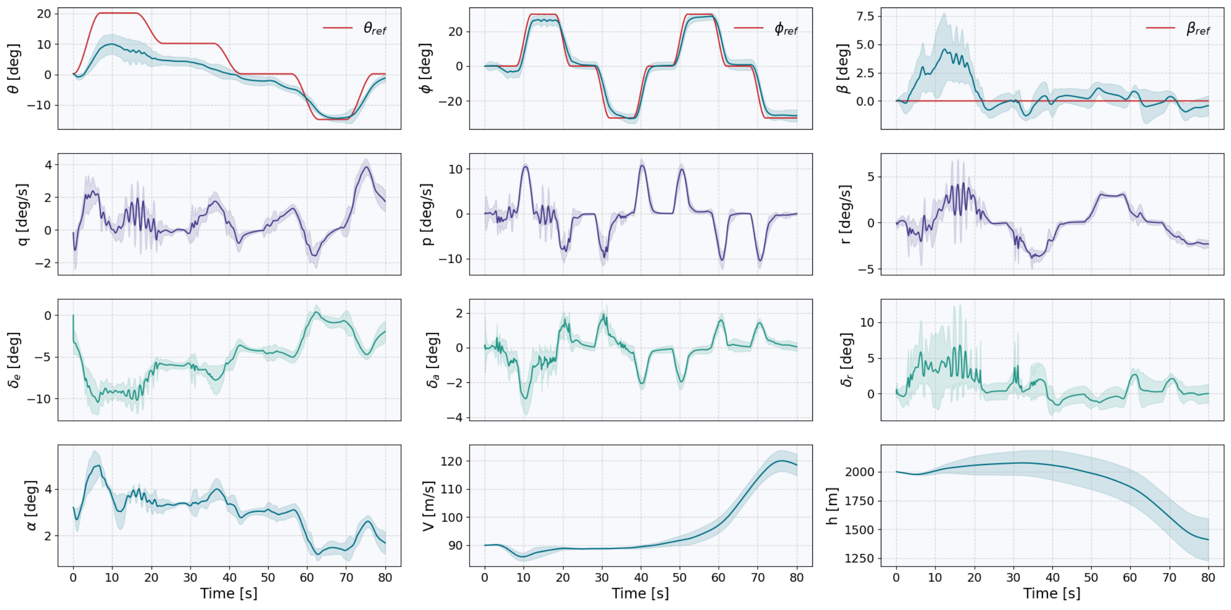


Figure 6.26: States time-traces of the DSAC policy in the **damaged elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

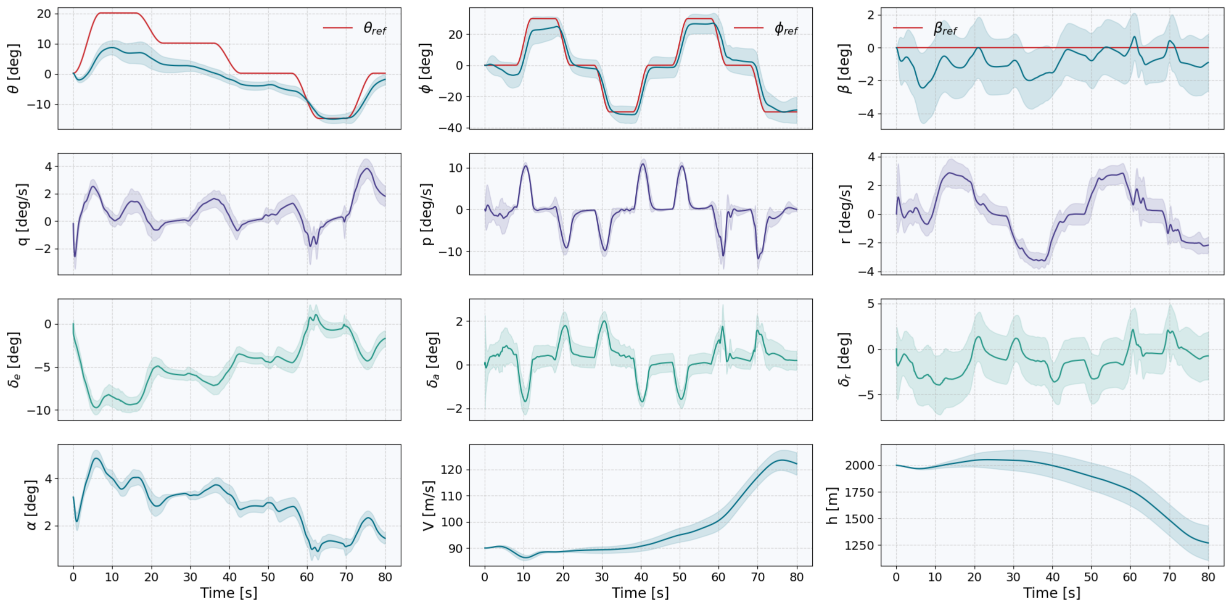


Figure 6.27: States time-traces of the *Risky* RUN-DSAC policy in the **damaged elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

in which it is stuck, but with no effect. Nevertheless, the control signal sent to the rudder triggers aileron deflection guided by the roll-yaw coupling learnt by the agents. Specifically, the induced yaw rate causes the aircraft to roll negatively (i.e. to the left in the direction of flight) due to the dihedral effect, which is promptly compensated by the negative aileron response. The leftward roll alters the lift vector, creating a force opposing the sideslip and aiding in realigning the aircraft's longitudinal axis with the relative wind. Therefore, this response effectively stabilizes the sideslip and halts further rolling, potentially preventing a spiral dive, yet it results in the introduction of a negative roll angle offset. Due to the coupling in pitch-roll axes, this roll offset impacts pitch tracking. To compensate, the agents increase elevator deflection beyond

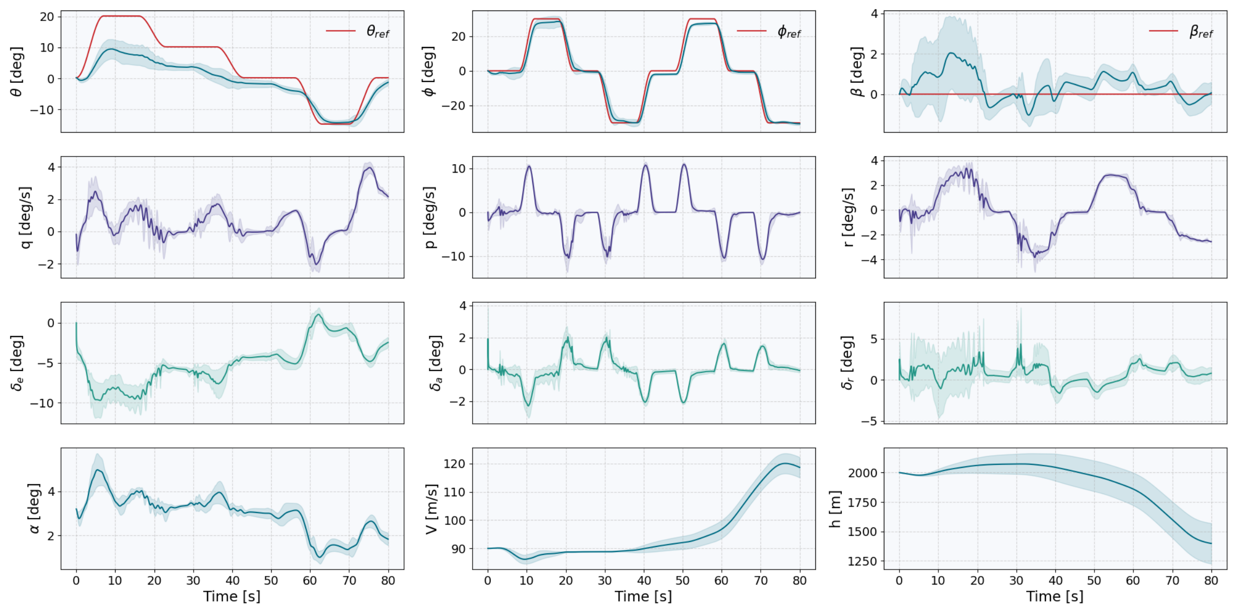


Figure 6.28: States time-traces of the *Conservative* RUN-DSAC policy in the **damaged elevator** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

angles observed in the nominal case, aiming to restore balance.

With the immobilized rudder changing the flight dynamics, a significant increase in oscillatory behavior is predominantly exhibited in the *Conservative* RUN-DSAC's pitch and roll tracking responses. This is in contrast to the minimal disturbances exhibited by the *Risky* RUN-DSAC, with the DSAC demonstrating intermediate behavior. Such a pattern aligns with previous observations in other failure scenarios where control surface functionality was compromised, hence the hypothesis for the pronounced oscillations in the *Conservative* RUN-DSAC policies is attributed to a tendency towards overfitting. This overfitting results in policies that, although highly efficient under training conditions, become less practical when a control channel is completely obstructed and the control authority on that axis is nullified.

G1: Wind Gust

In this scenario, the aircraft is subjected to a vertical wind gust with a velocity of $15\ ft/s$ for a duration of $3\ s$. Such atmospheric disturbances can be induced by meteorological phenomena like microbursts, thermal updrafts, or frontal activity, affecting the aircraft's pitch attitude and vertical speed momentarily. This evaluates the RL's adaptability and resilience in maintaining pitch control under sudden aerodynamic disturbances. The response of flight paths to wind gusts is detailed through a sequence of charts: Figure 6.33 illustrates the SAC agent's performance, Figure 6.34 demonstrates the DSAC agent's response, Figure 6.35 displays the performance of the *Risky* RUN-DSAC, and Figure 6.36 portrays the maneuvers of the *Conservative* RUN-DSAC.

The wind gust exerts minimal influence on the tracking performance of the agents, as evident in Table 6.1. Specifically, the nMAE of SAC experienced a marginal deterioration of only 2.21%, DSAC declined by 3.36%, *Risky* RUN-DSAC by 3.31%, and *Conservative* RUN-DSAC by 3.04%.

The wind gut was programmed to occur between $t = 4s$ and $t = 7s$ because during this time frame, it is anticipated to exert the most substantial influence on the aircraft's performance, as the aircraft already flies at high angles of attack due to high positive pitch reference. For the distributional agents, this gust results in the exacerbated angles of attack peaking at $\sim 9°$, whereas for the SAC, it reaches up to $\sim 12°$.

In response to the increased angle of attack, the *Conservative* RUN-DSAC agent reduces the magnitude of negative elevator deflection in an attempt to pitch the aircraft down, which also leads to oscillations throughout the gust duration. A similar but less intense elevator response is observed in the DSAC. Contrarily, the *Risky* RUN-DSAC seems to disregard the wind gust, maintaining elevator deflection as in

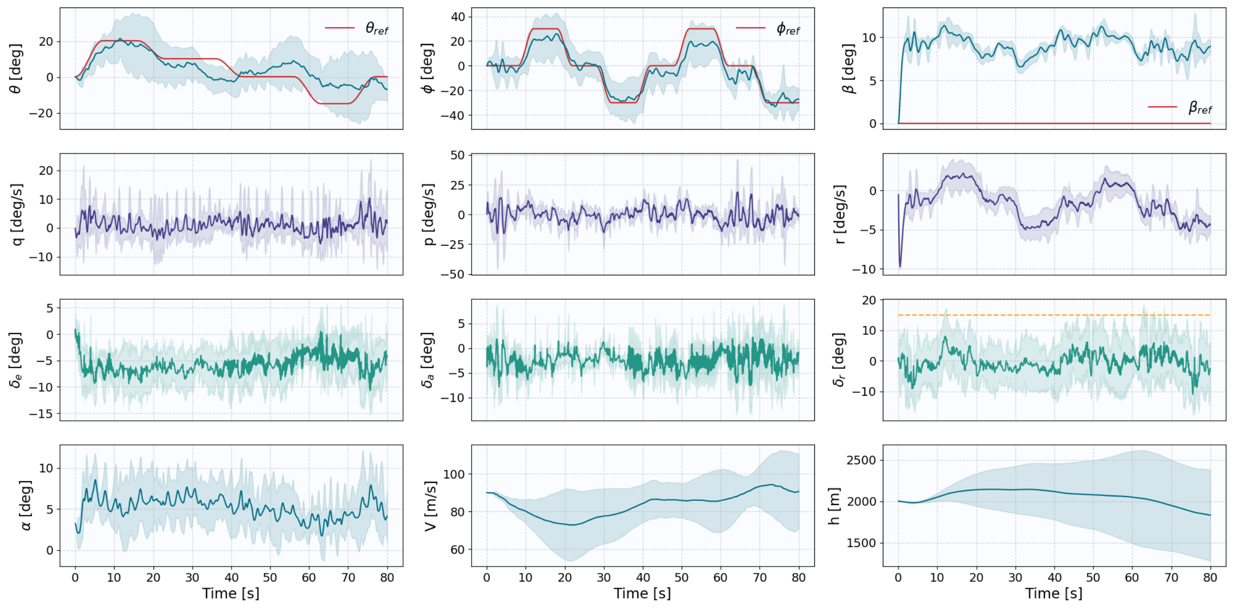


Figure 6.29: States time-traces of the SAC policy in the **immobilized rudder** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The actual rudder deflection is represented by a dashed yellow line.

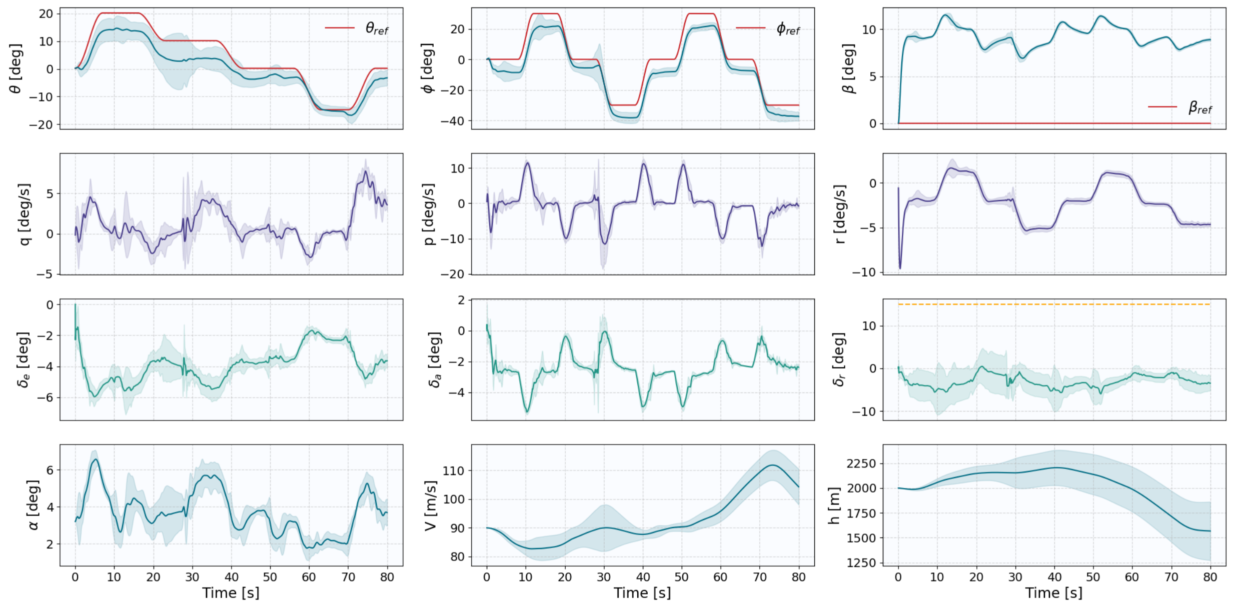


Figure 6.30: States time-traces of the DSAC policy in the **immobilized rudder** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The actual rudder deflection is represented by a dashed yellow line.

nominal conditions. This approach could stem from the *Risky* RUN-DSAC's broader training experience with varied angles of attack (see the analysis of the response to the *G5: Stall* scenario), enhancing its robustness against such conditions. Alternatively, it could have learned to assign a lower importance to the angle of attack in its decision-making, unlike the *Conservative* RUN-DSAC and DSAC.

The wind gust adversely affects pitch angle tracking, most notably in the DSAC and *Conservative* RUN-DSAC, while it also mildly disrupts the yaw and roll axes due to coupling. After the wind gust dissipates, the agents promptly revert to their nominal performance. Nevertheless, contrary to expectations of a pitch angle overshoot due to increased angles of attack, an undershoot is observed. This can be attributed to the

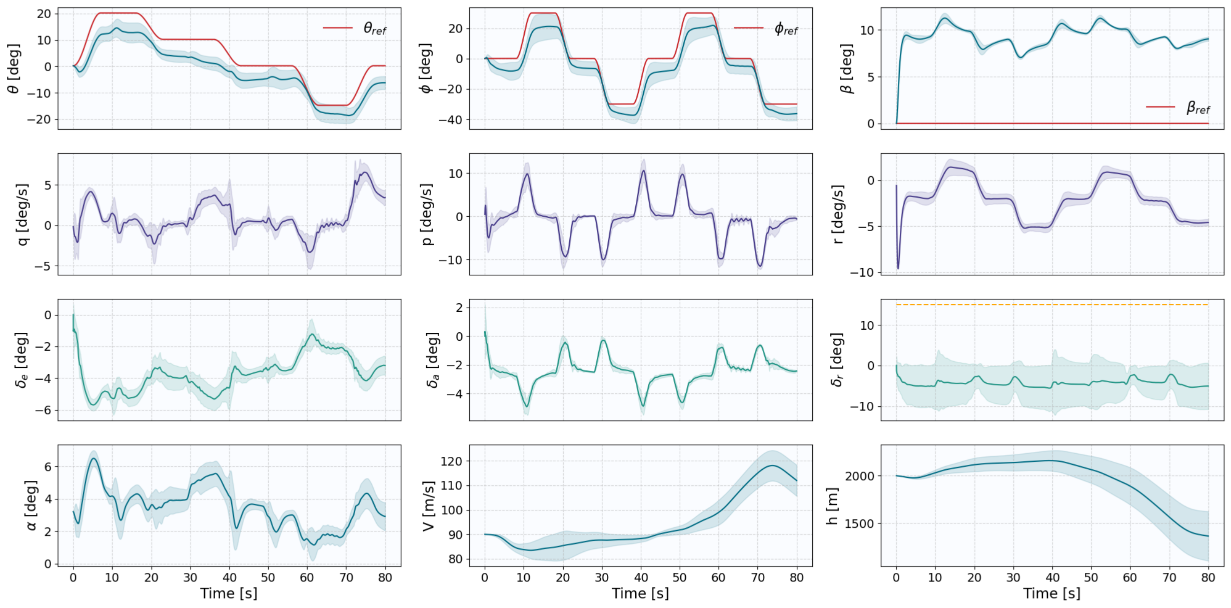


Figure 6.31: States time-traces of the *Risky* RUN-DSAC policy in the **immobilized rudder** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The actual rudder deflection is represented by a dashed yellow line.

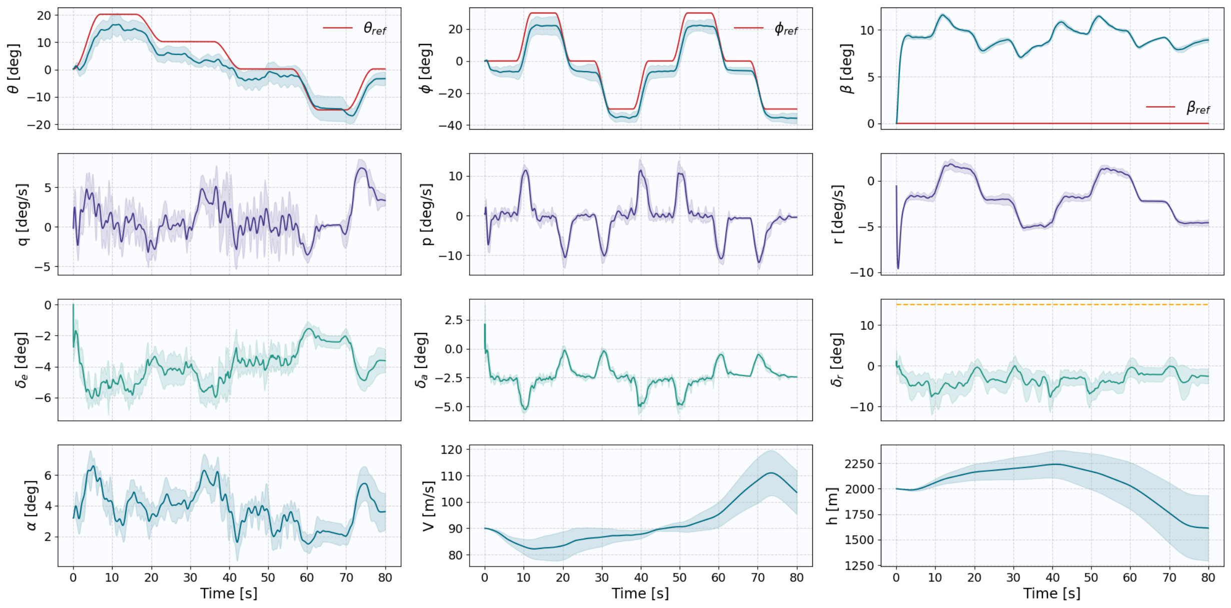


Figure 6.32: States time-traces of the *Conservative* RUN-DSAC policy in the **immobilized rudder** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The actual rudder deflection is represented by a dashed yellow line.

modeling simplifications for the wind gust scenario. The vertical gust was modelled by solely augmenting the angle of attack by an increment equivalent to the $15\ ft/s$ of vertical wind for the given airspeed. As a result, this simplification does not directly incorporate changes in other impacted factors, such as aerodynamic forces. Therefore, it can be posited that this scenario rather assesses the robustness to the faults in the angle of attack sensor. The undershoot in the angle of attack is thus attributable to the elevator deflection's response to the augmented angle of attack, anticipating an overshoot that does not materialize. While this simplified analysis sufficed for evaluating the aircraft's response to the wind gust in this context, it is recommended to develop a higher fidelity aerodynamic simulation to procure a more representative response of the agents to the wind gust.

Due to the limitations inherent in the wind gust model, the agents also exhibited contrasting responses under varying wind gust initiation conditions. For instance, Figure 6.37 illustrates the response of the *Conservative* RUN-DSAC to a 3-second vertical gust initiated at $t = 45s$, where both theta and roll angle references are set to zero. Surprisingly, the agent reacts to the wind gust by deflecting the aileron negatively, thereby further increasing the angle of attack and exacerbating the wind gust's impact. In fact, this counterintuitive response aids the agent in mitigating the negative offset in pitch angle tracking. A similar behavior was previously observed in a SAC flight controller in a prior study [27]. A plausible hypothesis suggests that the agent has learnt to associate the specific scenario of an abrupt increase in the angles of attack, a concurrent negative deviation in pitch angle tracking, and an initially low pitch rate with the necessity for a pitch-up maneuver to follow elevated pitch angles. This could explain its distinct response to the scenario in which the gust is initiated at $t = 4s$, where the angles of attack and the pitch rate inputs are already high.

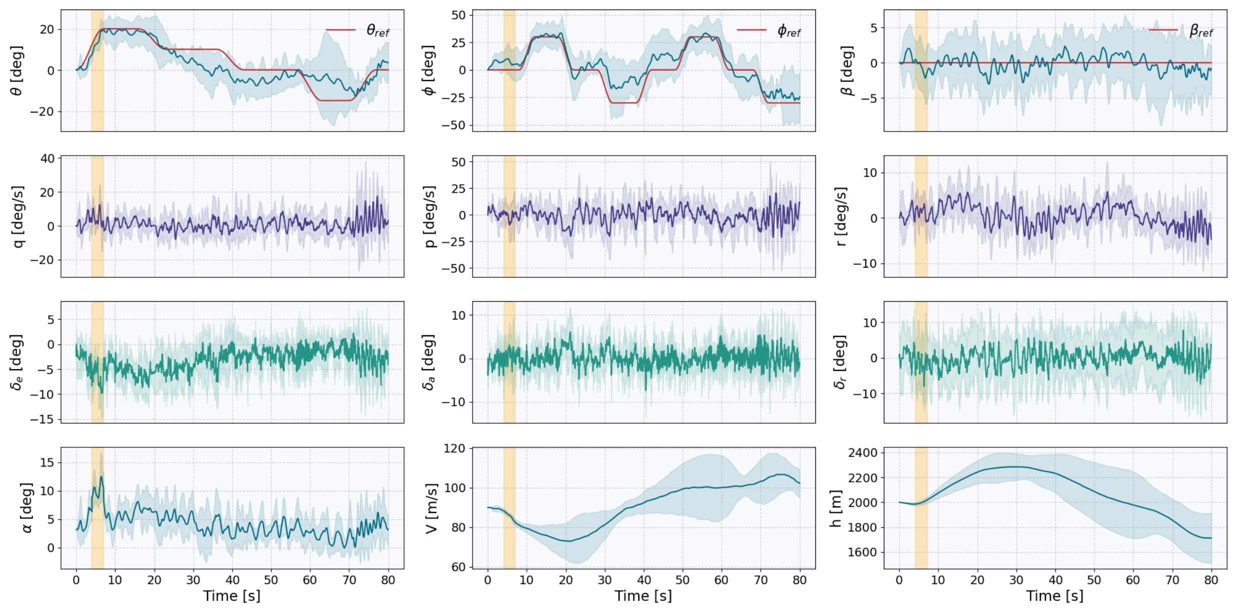


Figure 6.33: States time-traces of the SAC policy in the **vertical wind gust** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The temporal interval of wind gust occurrence is highlighted in yellow for clarity.

G2: Noisy Signal

The inclusion of the **G2** scenario serves multiple objectives. First, it assesses the robustness of RL-agents in handling sensor noise, which can introduce non-linearities and uncertainties into the control loop. Second, it enables a more realistic transition from simulation-based RL training to real-world applications by mimicking true sensor behaviors. This represents a crucial step in bridging the gap between RL simulations and real-world avionic systems, thereby ensuring the controllers' practical applicability and efficacy. In this setup, the control systems are subjected to Gaussian noise with an inherent bias, modeled based on empirical in-flight sensor measurements, as given in Table 6.2. Noise inherent in the measurements of control surface deflections is disregarded, as these observations are not utilized by the agents in determining actions. A series of figures illustrates trajectory responses in a noisy signal scenario: Figure 6.38 presents the SAC agent's performance, Figure 6.39 shows the DSAC agent's response, Figure Figure 6.40 details the *Risky* RUN-DSAC's actions, and Figure Figure 6.41 outlines the *Conservative* RUN-DSAC's strategies.

Incorporating biased noise marginally affects tracking performance, as evidenced in Table 6.1. Here, the nMAE only shows modest deterioration: 8.57% for SAC, 2.52% for DSAC, and 5.57% for *Conservative* RUN-DSAC. Intriguingly, it decreases by 0.32% for *Risky* RUN-DSAC. Comparing the obtained trajectories with nominal condition responses confirms this trend, where all controllers maintain tracking errors comparable to those in the nominal scenario. Despite a reduction in action smoothness leading to increased noise in angular rates, the impact remains limited as none of the trained policies amplify observation noise.

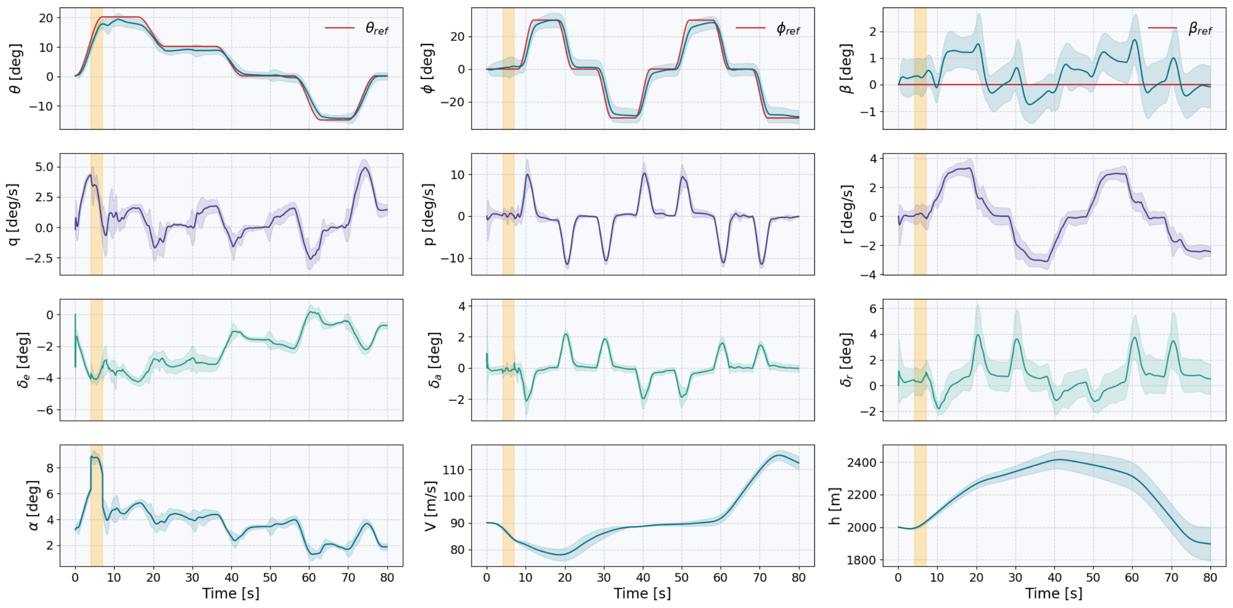


Figure 6.34: States time-traces of the DSAC policy in the **vertical wind gust** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The temporal interval of wind gust occurrence is highlighted in yellow for clarity.

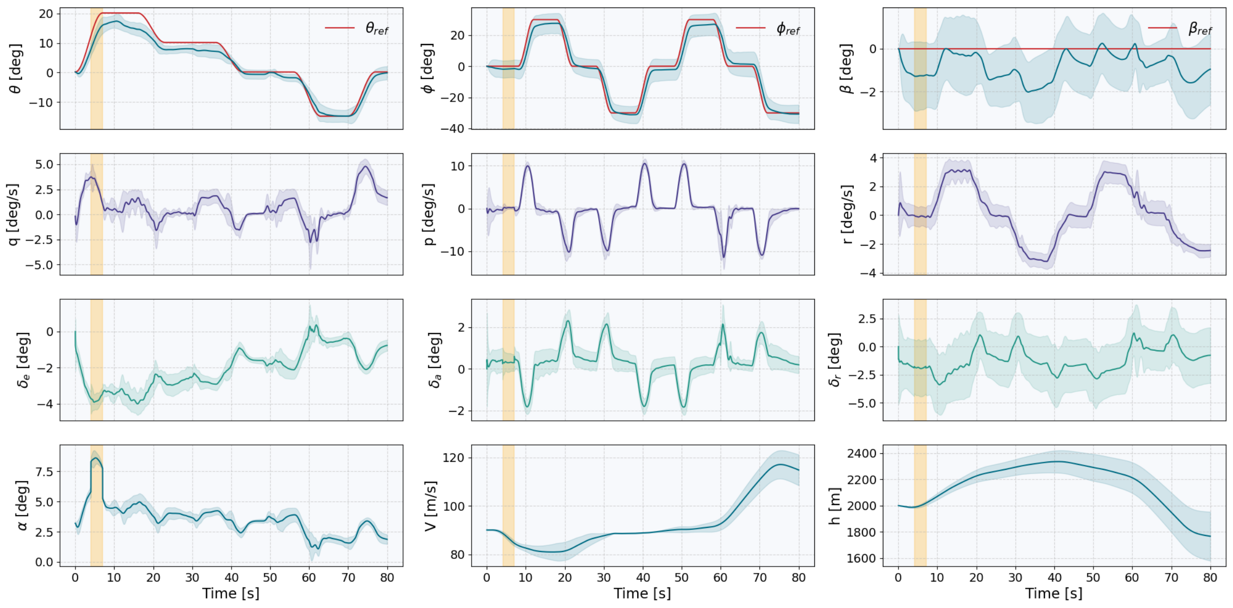


Figure 6.35: States time-traces of the *Risky* RUN-DSAC policy in the **vertical wind gust** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The temporal interval of wind gust occurrence is highlighted in yellow for clarity.

G3: High Dynamic Pressure

In the **G3** scenario, the aircraft operates under a trim condition characterized by a high dynamic pressure setting, with an altitude $H = 2,000\,m$ and true airspeed $V_{tas} = 150\,m/s$, as opposed to the $V_{tas} = 90\,m/s$, on which the agents were trained. This case evaluates the RL-agents' capability to adapt to varying aerodynamic forces, moments and actuator effectiveness, which are intrinsically linked to changes in dynamic pressure. Consequently, it serves to assess and compare the robustness and adaptability of different RL-based control algorithms when exposed to previously unencountered flight regimes. A series of illustrations showcases

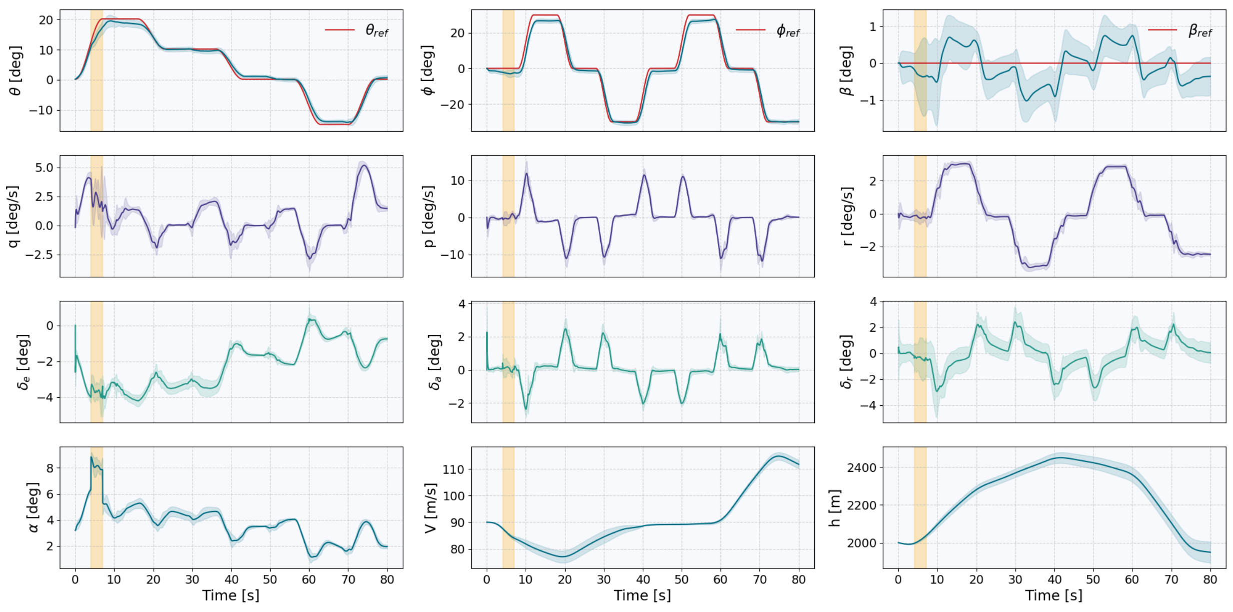


Figure 6.36: States time-traces of the *Conservative* RUN-DSAC policy in the **vertical wind gust** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The temporal interval of wind gust occurrence is highlighted in yellow for clarity.

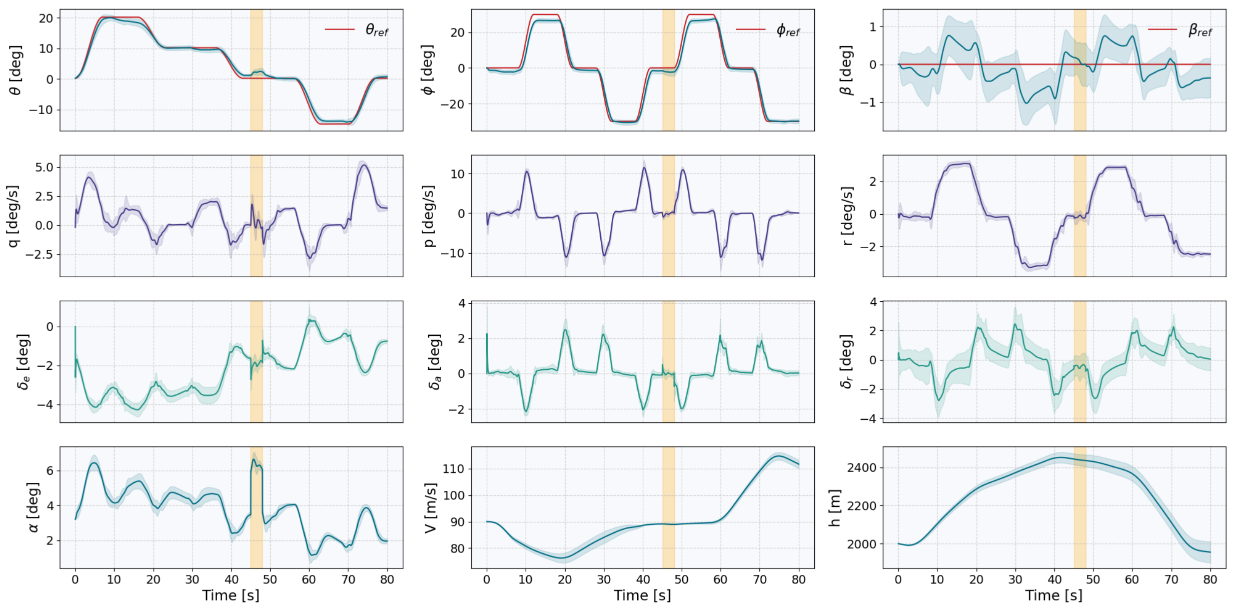


Figure 6.37: States time-traces of the *Conservative* RUN-DSAC policy in the **vertical wind gust** scenario, when the gust is initiated during level flight: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation. The temporal interval of wind gust occurrence is highlighted in yellow for clarity.

the trajectory responses during the increased dynamic pressure scenario: Figure 6.42 illustrates the SAC agent's performance, Figure 6.43 outlines the behavior of the DSAC agent, Figure 6.44 displays the actions of the *Risky* RUN-DSAC, and Figure 6.45 provides trajectories of the *Conservative* RUN-DSAC.

In increased velocity conditions, the nMAE rose by 27.11% for SAC and 24.03% for DSAC compared to nominal conditions. Interestingly, agents that take uncertainty in the Q-values into account during learning appear to be less susceptible to this condition. Specifically, the nMAE increased by a mere 1.60% for *Risky*

Table 6.2: Sensor models characterized by Gaussian noise distribution based on in-flight data from the PH-LAB aircraft measurements [180].

Signal	Unit	Noise (σ^2)	Bias
p, q, r	rad/s	4.0×10^{-7}	3.0×10^{-5}
θ, ϕ	rad	1.0×10^{-9}	4.0×10^{-3}
α	rad	4.0×10^{-10}	-

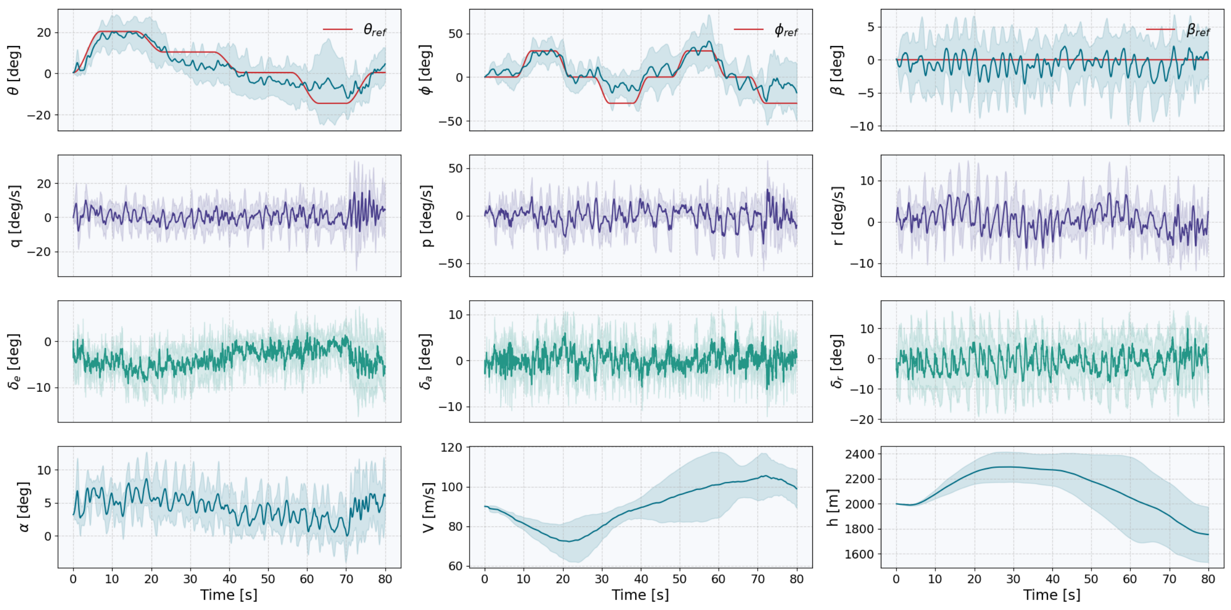


Figure 6.38: States time-traces of the SAC policy in the **noisy signal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

RUN-DSAC and 2.78% for *Conservative* RUN-DSAC.

The heightened dynamic pressure increases the efficiency of the control surfaces and results in enhanced aerodynamic damping. With higher efficiency, the required deflections for maneuvers are reduced. This change in the flight dynamics is recognized and adapted to by distributional agents, as evidenced by their trajectories. The increased dynamic pressure also directly translates into a higher lift force, elevating the aircraft to greater altitudes during pitch-up maneuvers than in nominal conditions, as evidenced in the responses of all the agents.

The SAC policy, noted for its aggressive and oscillatory nature, becomes increasingly suboptimal under conditions of high dynamic pressure. The pronounced effectiveness of the actuators under such conditions means that the inherent oscillations in the SAC's control inputs are amplified, which severely undermines the tracking performance. For DSAC and *Conservative* RUN-DSAC, the increased dynamic pressure introduces a positive offset in pitch angle tracking, particularly evident at zero and negative reference angles where velocity, and consequently dynamic pressure, peaks. While this heightened dynamic pressure impairs DSAC's roll angle tracking performance, *Conservative* RUN-DSAC exhibits an improved roll tracking performance compared to nominal conditions. Conversely, *Risky* RUN-DSAC displays an enhanced pitch tracking ability, successfully following even high pitch angle references. However, it shows some offset for negative pitch references. Its roll reference tracking remains comparable to nominal conditions, albeit with an increase in variance.

Oscillations are present across the responses of all agents, occurring primarily at high velocities. These are more pronounced in *Risky* RUN-DSAC compared to both its *Conservative* counterpart and DSAC. These oscillations are attributed to the increased sensitivity of agents trained at lower dynamic pressures, leading to overcorrections and oscillations when control inputs are derived from these lower-speed training

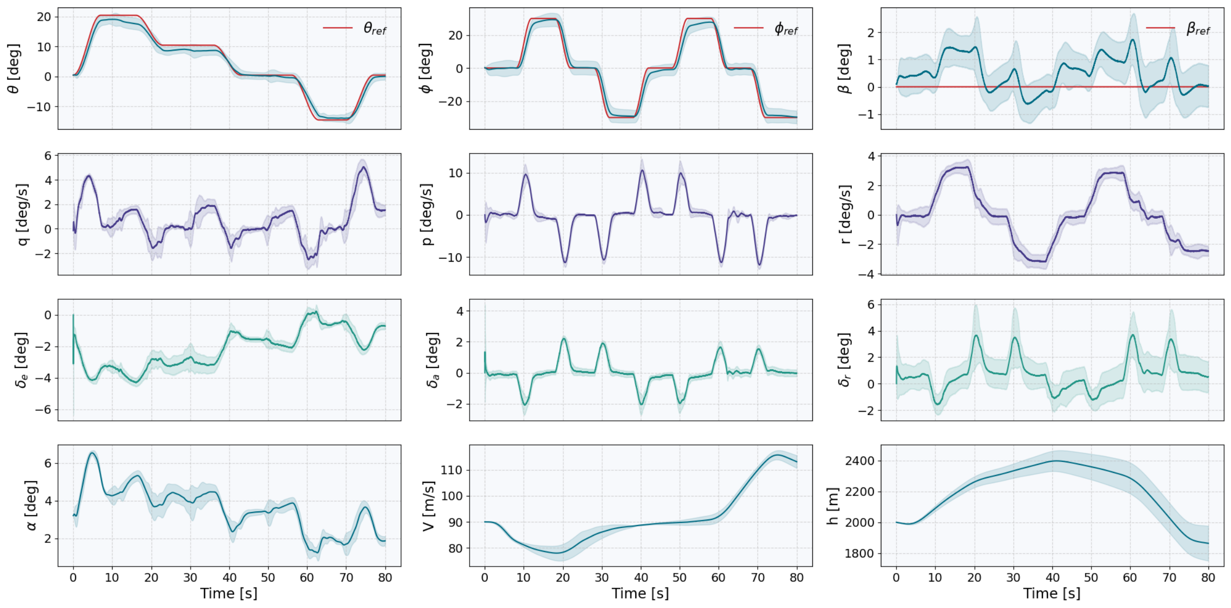


Figure 6.39: States time-traces of the DSAC policy in the **noisy signal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

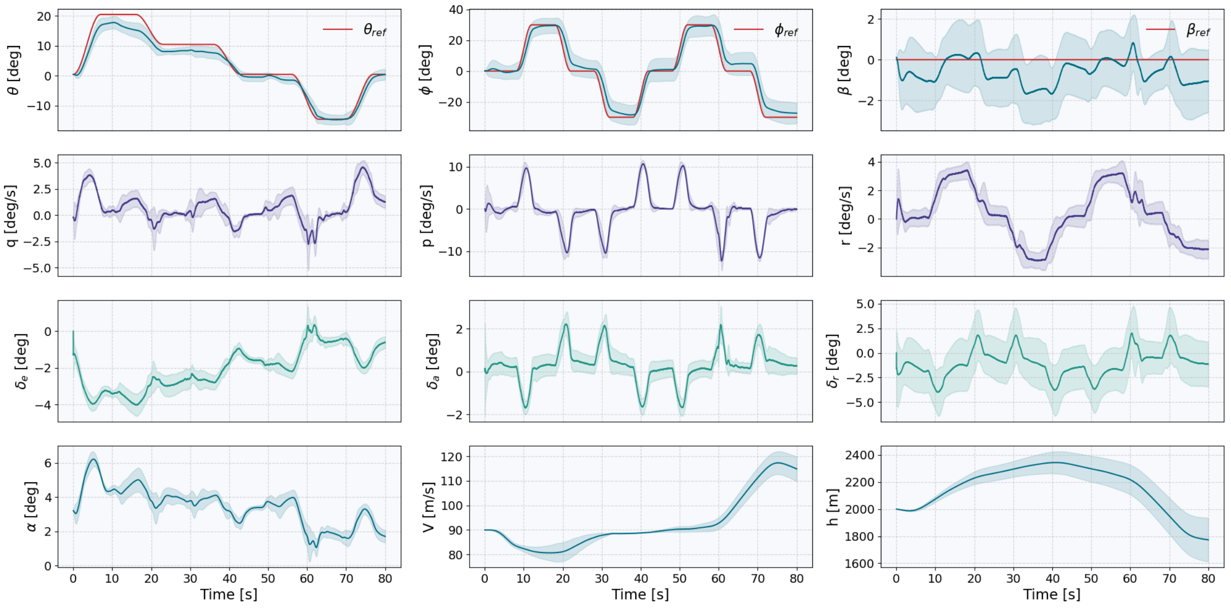


Figure 6.40: States time-traces of the *Risky* RUN-DSAC policy in the **noisy signal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

conditions.

The differences between the distributional agents can be explained through linking the behaviour of the agent's actor neural network to a gain parameter in classical control systems. In control theory, gain dictates a system's response to an error signal: high gain leads to aggressive responses, potentially causing instability or oscillations, while low gain results in more conservative responses. The DSAC appears to have learned a control policy with lower gain, resulting in minimal oscillations but also reduced tracking performance and increased policy variance. On the other hand, the *Conservative* RUN-DSAC exhibits a policy with higher gain, resulting in improved tracking performance compared to DSAC and reduced

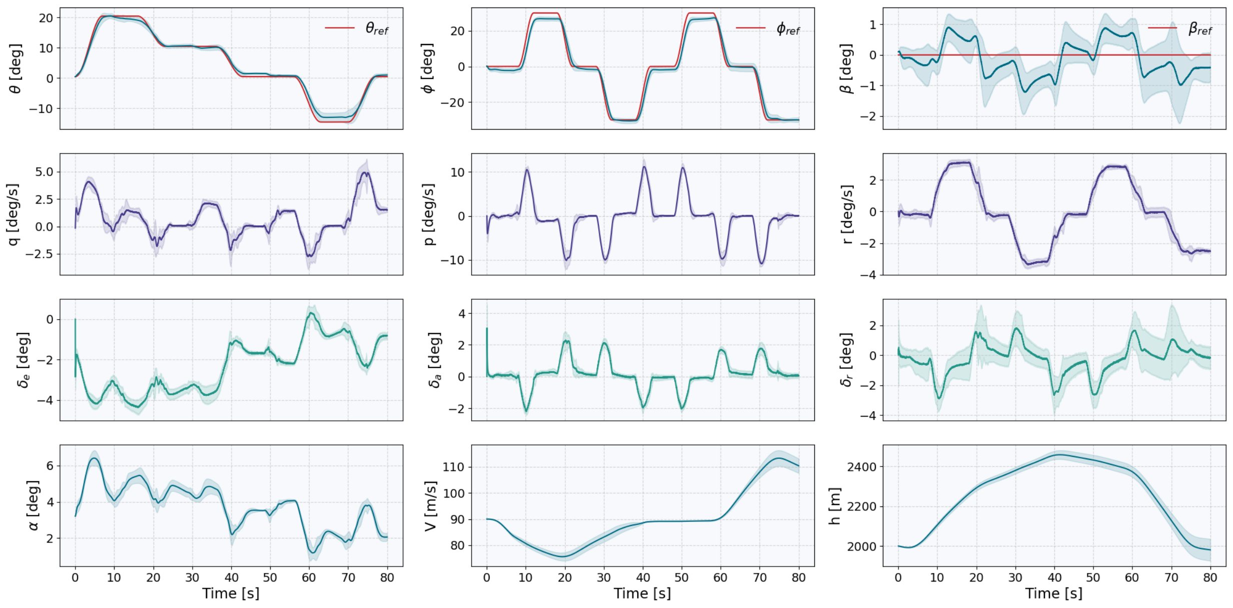


Figure 6.41: States time-traces of the *Conservative* RUN-DSAC policy in the **noisy signal** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

policy variance, but it also exhibits more pronounced oscillations under higher dynamic pressure conditions. In contrast, the *Risky* RUN-DSAC demonstrates a policy with excessively high gain, leading to the most significant oscillations, while it does not outperform the *Conservative* variant in tracking performance.

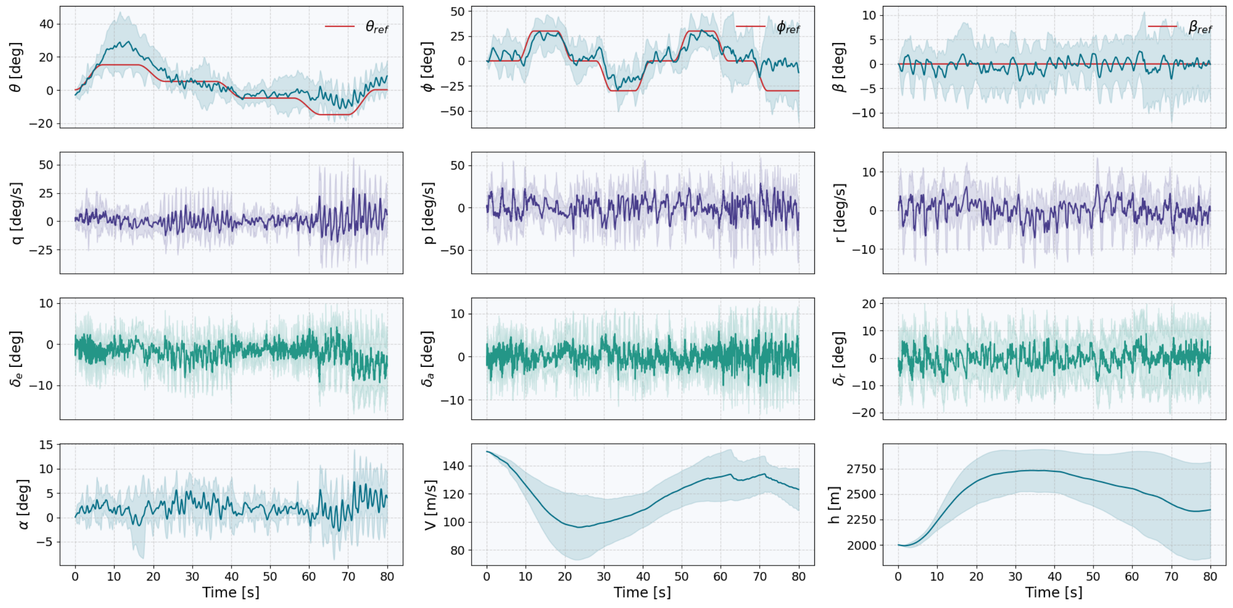


Figure 6.42: States time-traces of the SAC policy in the **high dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

G4: Low Dynamic Pressure

In the **G4** case, the aircraft is subjected to a trim condition indicative of a low dynamic pressure environment, specifically at an altitude $H = 10,000\,m$ and a true airspeed $V_{tas} = 90\,m/s$. Much like its high dynamic

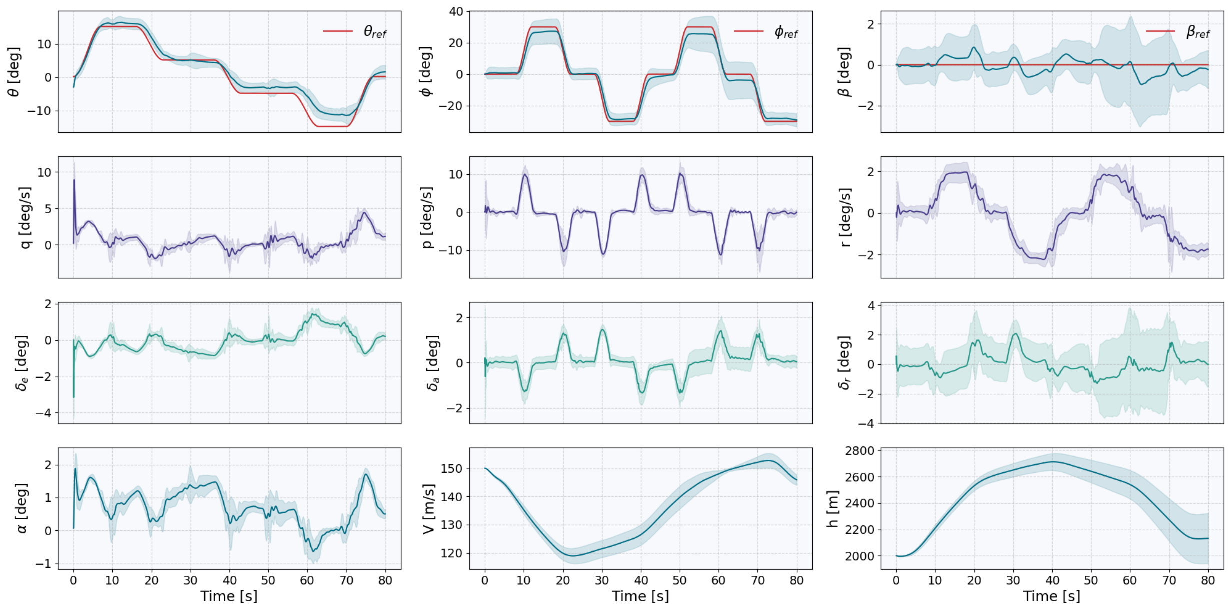


Figure 6.43: States time-traces of the DSAC policy in the **high dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

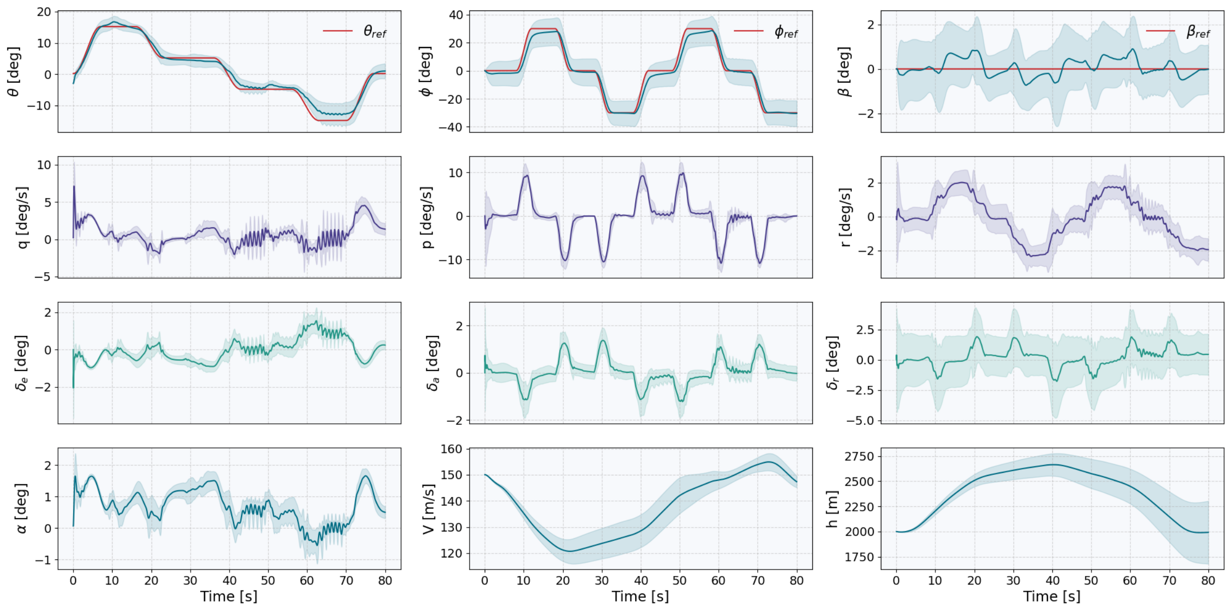


Figure 6.44: States time-traces of the *Risky* RUN-DSAC policy in the **high dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

pressure counterpart, this scenario examines the adaptability of the RL-agents in a regime where aerodynamic forces and control surface effectiveness are markedly different due to reduced dynamic pressure. The trajectory reactions in response to decreased dynamic pressure are illustrated through a set of visuals: Figure 6.46 represents the SAC agent, Figure 6.47 pertains to the DSAC agent, Figure 6.48 displays the outcomes of the *Risky* RUN-DSAC, and Figure 6.49 showcases the results of the *Conservative* RUN-DSAC.

Under conditions of increased altitude, the nMAE rose by 32.51% for SAC, 55.12% for DSAC, and 88.86% for the *Conservative* RUN-DSAC, with respect to the nominal conditions. In contrast, the *Risky* RUN-DSAC was affected the least, with its nMAE rising by only 15.49%.

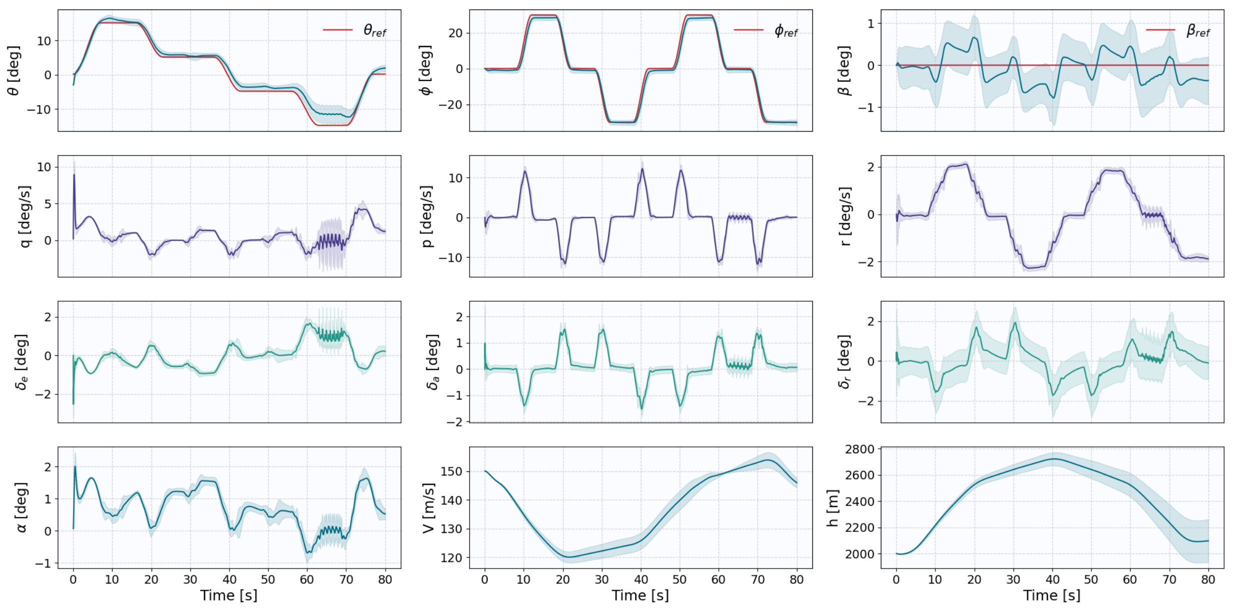


Figure 6.45: States time-traces of the *Conservative* RUN-DSAC policy in the **high dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

When aircraft operate under low dynamic pressure, the responses of the control systems become lethargic, leading to increased average tracking errors and increased standard deviations. This is due to the reduced effectiveness of all three actuating channels, which makes it difficult for the controllers to react swiftly and accurately to changes in flight conditions.

Examining the pitch axis response, there is a noticeable initial rapid pitch-down maneuver by the aircraft to align its pitch with the reference angle. This action is necessitated due to the aircraft's high initial pitch angle, a result of the low-pressure trim setting, in contrast to the reference pitch angle that starts near zero. However, this rapid adjustment results in overcompensation, triggering the feedback loop to make corrective actions, which leads to induced oscillations. These oscillations are further aggravated due to the flight dynamics at low dynamic pressures, as the aircraft is required to fly at high angles of attack to generate adequate lift, positioning it near the stall boundary. Flying at such high angles of attack is characterized by decreased aerodynamic efficiency and a non-linear lift response, which could induce oscillations and for which the controllers were not trained. Moreover, the lower aerodynamic damping present at these reduced pressures contributes to the persistence of these oscillations. Moreover, the proximity to stalling conditions hinders the agents' ability to maintain high pitch angles, even when attempting to compensate by deflecting the elevator further than in the nominal case.

Due to coupling, these pitch oscillations translate into the roll and yaw axes. However, the roll angle and sideslip angle tracking performance remains relatively consistent with the nominal scenario for all distributional agents, although the roll tracking performance deteriorates for SAC.

Contrasting with the high dynamic pressure conditions, the most oscillations are observed with the original DSAC, followed by the *Conservative* RUN-DSAC, and then the *Risky* RUN-DSAC. This pattern aligns with previous observations in near-stall conditions, such as during ice accretion scenarios, and is consistent with the established ”gain theory” from high dynamic pressure scenario. The *Risky* RUN-DSAC, with its higher gain policy, shows more responsiveness to control inputs, which offsets the lower effectiveness of the control surfaces under low dynamic pressure. Conversely, the original DSAC, with its lower gain, shows less responsiveness with larger and potentially delayed corrections, leading to under-compensation for errors and consequently more pronounced oscillations. Furthermore, the difference in oscillatory behavior between the *Conservative* and *Risky* RUN-DSAC can be explained by their respective training environments, where the *Conservative* RUN-DSAC, despite its bias towards more stable policies, may risk overfitting, as it might not fully represent the breadth of varied and unpredictable outcomes essential for robust performance. This could lead to more oscillatory responses in practice. Furthermore, the difference in oscillatory behavior between *Conservative* and *Risky* RUN-DSAC can be attributed to their training biases, as discussed for scenarios like ice accretion. In short, the *Conservative* RUN-DSAC, inclined towards stable policies, may overfit within its specific training regime, failing to capture the full spectrum of complex, unpredictable flight conditions critical for comprehensive performance.

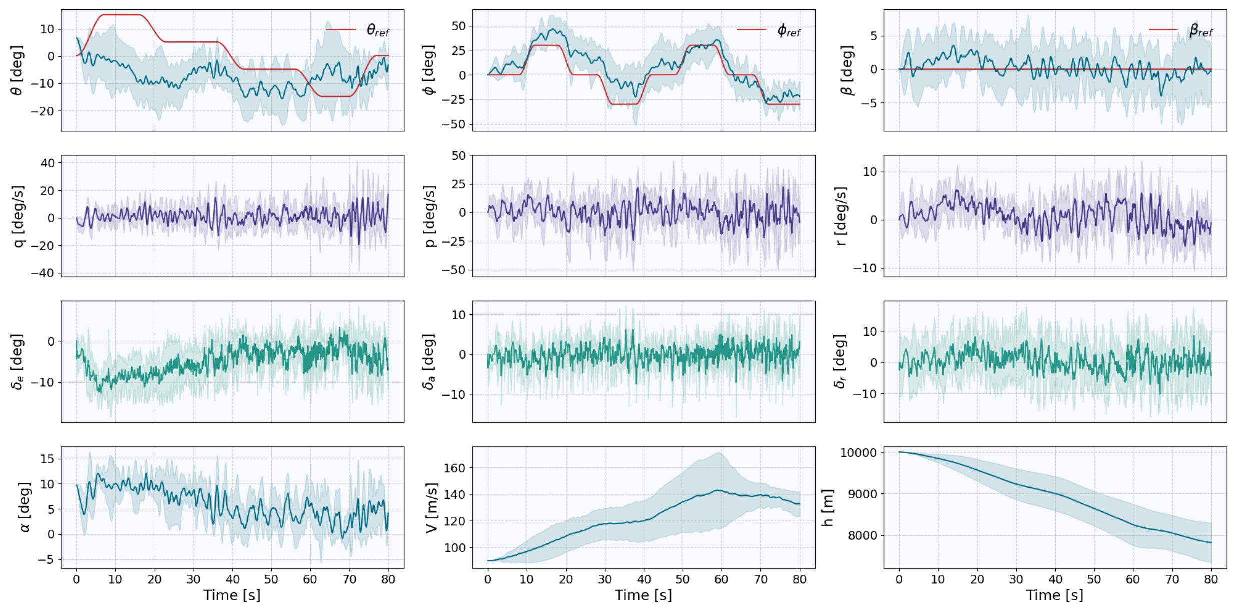


Figure 6.46: States time-traces of the SAC policy in the **low dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

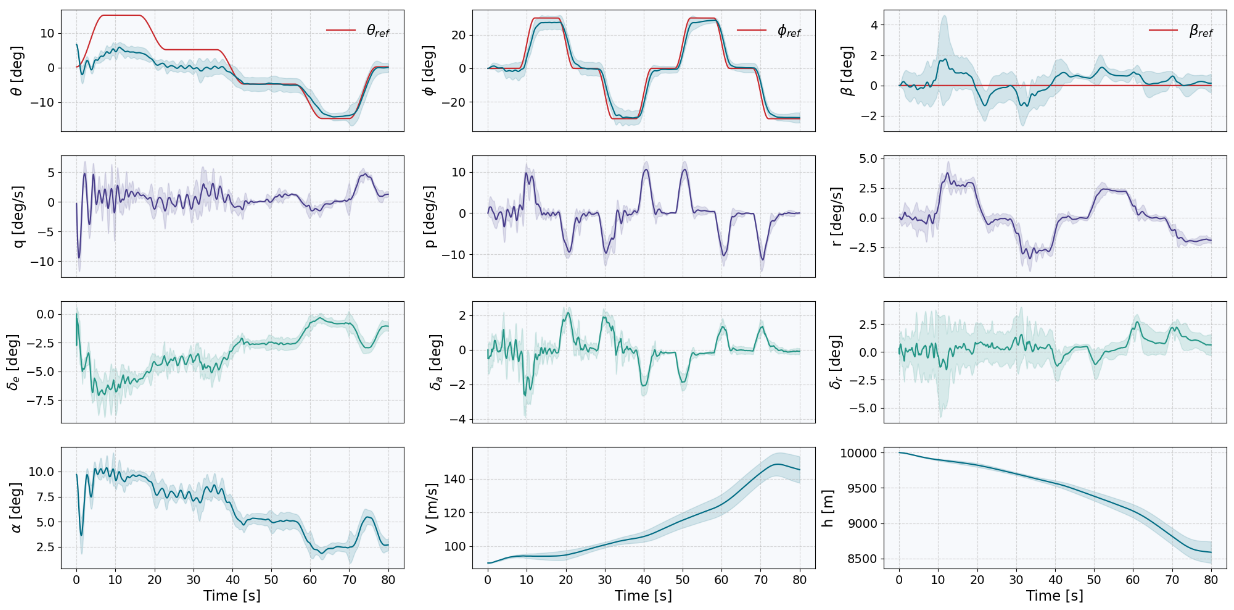


Figure 6.47: States time-traces of the DSAC policy in the **low dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

G5: Stall

The stall scenario in this analysis is modelled by a high pitch-up maneuver that brings the aircraft into near-stall conditions. This scenario is of particular interest, as the uncertainty associated with near-stall conditions is notably high due to limited prior exploration and unobservable dynamics dependent on airspeed

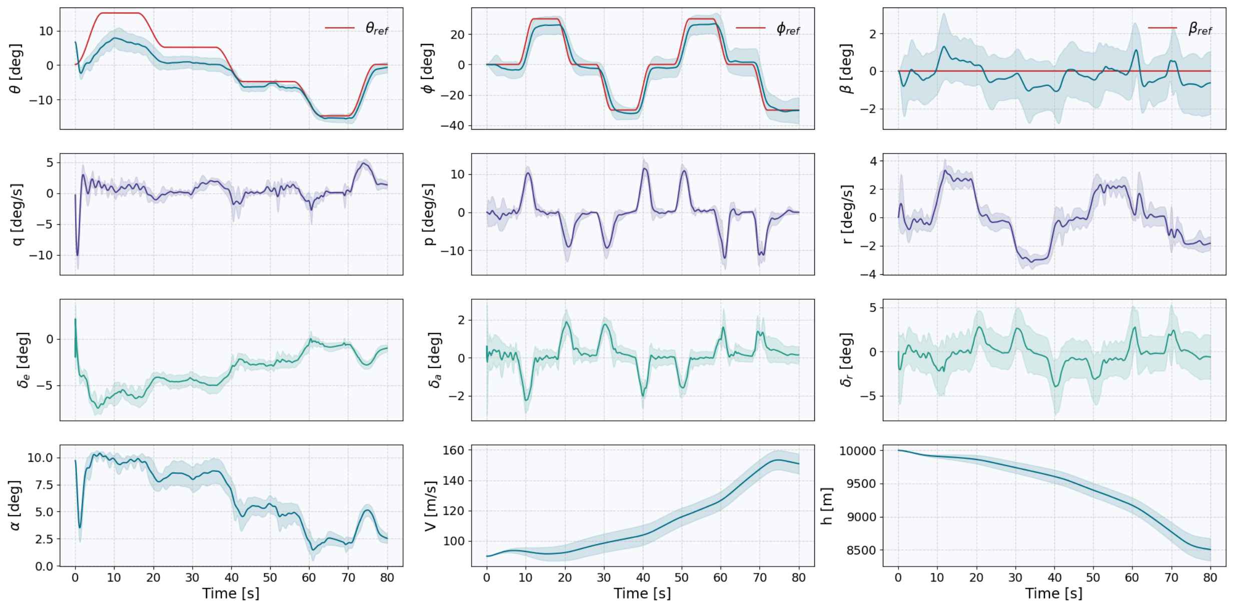


Figure 6.48: States time-traces of the *Risky* RUN-DSAC policy in the **low dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

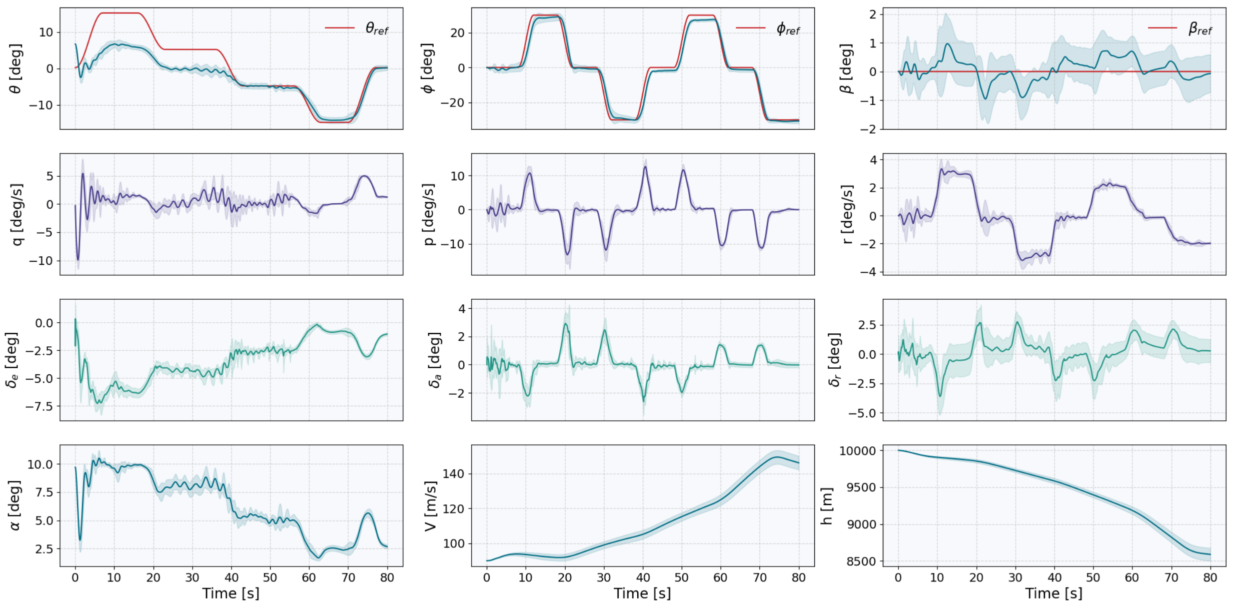


Figure 6.49: States time-traces of the *Conservative* RUN-DSAC policy in the **low dynamic pressure** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

and altitude, such as airflow separation. Secondly, these conditions hold direct implications for flight risk, as identified by the International Civil Aviation Organization (ICAO) [181], given that such flight states are considered hazardous and can precipitate loss-of-control (LOC) events. The responses to the near-stall scenario are depicted through a series of figures: Figure 6.50 is indicative of the SAC agent's behavior, Figure 6.51 relates to the DSAC agent, Figure 6.52 presents the performance of the *Risky* RUN-DSAC, and Figure 6.53 demonstrates the outcomes associated with the *Conservative* RUN-DSAC. The nMAE relative percentage changes for this scenario are not compared to the nominal case, due to the differing reference trajectories between the scenarios.

During the execution of the maneuver, the aircraft is subjected to a high pitch-angle reference that is sustained throughout the evaluation time. This significantly influences the aircraft's aerodynamic behavior, leading to a reduction in airspeed and an increase in altitude. At $t = 40s$, the aircraft is required to achieve a pitch angle of $40°$. Such a high pitch angle reference poses a significant challenge for the agents, as attempting to reach this pitch angle without inducing aerodynamic instability is not feasible for the agents, hence stall-induced oscillations occur.

The comparison of the responses exhibited by different distributional RL agents reveals varying degrees of oscillation. The original DSAC controller demonstrates the most pronounced oscillations, followed by the *Conservative* RUN-DSAC. In contrast, the *Risky* RUN-DSAC shows the least oscillations in its average response. This pattern of behavior aligns with observations from other scenarios involving high angles of attack, such as cases with low dynamic pressure or ice accretion on the wings. Therefore, the observed behavior can be interpreted through the analogy of the learnt control policy and gain, or in the context of the Conservative DSAC's tendency to overfit, as has been established in these high-angle-of-attack scenarios. Finally, the stall-induced instabilities extend beyond the pitch axis due to coupling, notably affecting the roll and yaw axes of the aircraft.

The *Conservative* RUN-DSAC's training strategy emphasizes predictability, potentially leading to overfitting in training scenarios, while the *Risky* RUN-DSAC adopts a more exploratory approach by favoring uncertain Q-values. This likely results in the *Risky* RUN-DSAC encountering a wider array of states and actions, albeit with a trade-off in tracking performance and increased policy variance for training scenarios. However, this exploratory nature may enhance robustness in unforeseen scenarios and mitigate oscillations, as previously discussed. In the given stall scenario, where the angle of attack, one of the observations available for the RL agents, reaches unusually high values, the exploratory tendency of *Risky* RUN-DSAC could explain its superior performance in dampening oscillations compared to DSAC and *Conservative* RUN-DSAC. To substantiate this hypothesis, a Gaussian kernel smoothed density estimation of the angle of attack distributions experienced by each agent during a single training run is illustrated in Figure 6.54. Despite the symmetric distribution of training reference pitch angles around zero, the distributions naturally lean towards positive angles, which are essential for lift generation. This analysis also reveals a broader spectrum of angle of attack experiences for the *Risky* RUN-DSAC, encompassing even extreme high and low angles. In contrast, the *Conservative* RUN-DSAC exhibits the most constrained distribution of experiences, encountering extreme values less frequently than even the DSAC. This constrained exposure potentially underpins its susceptibility to overfitting and may account for the observed oscillations in unforeseen scenarios.

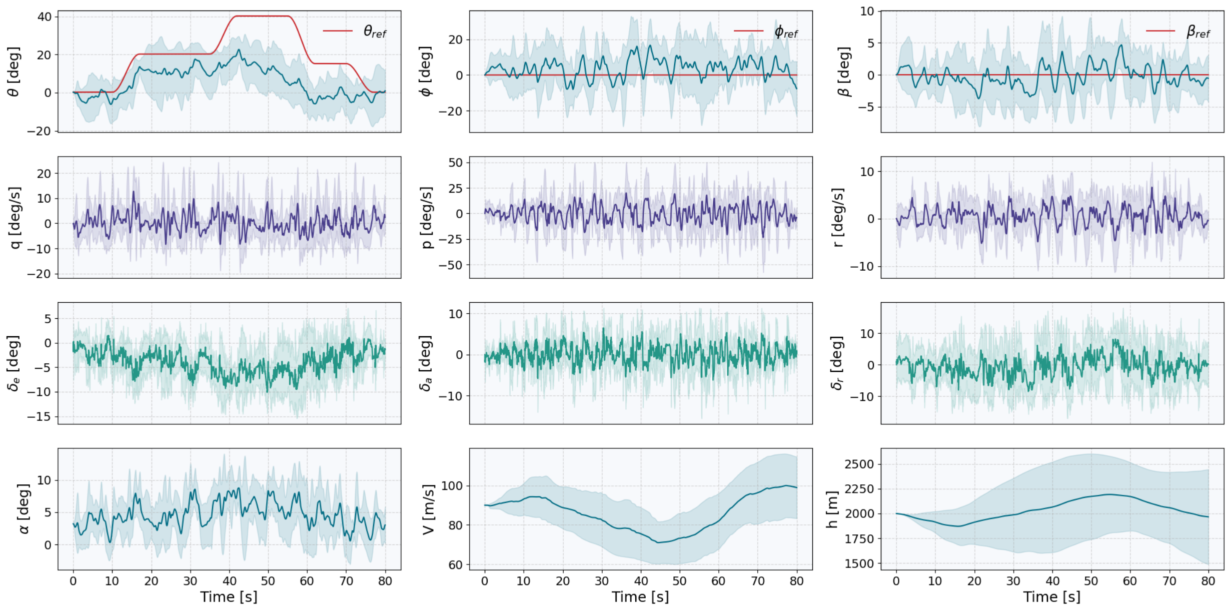


Figure 6.50: States time-traces of the SAC policy in the **stall** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

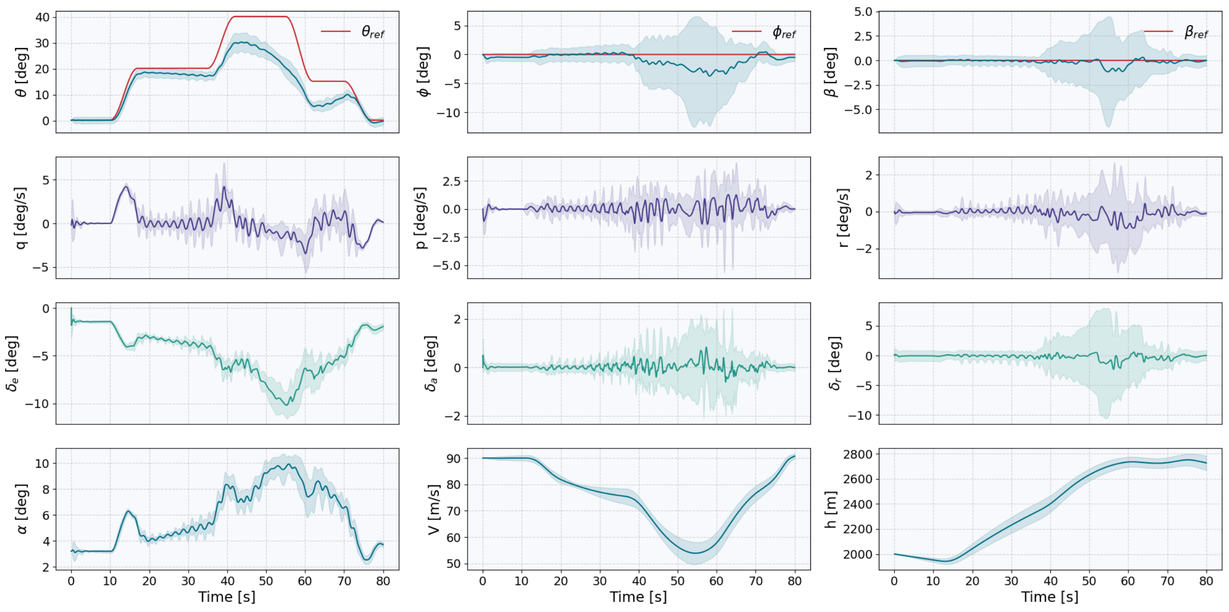


Figure 6.51: States time-traces of the DSAC policy in the **stall** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

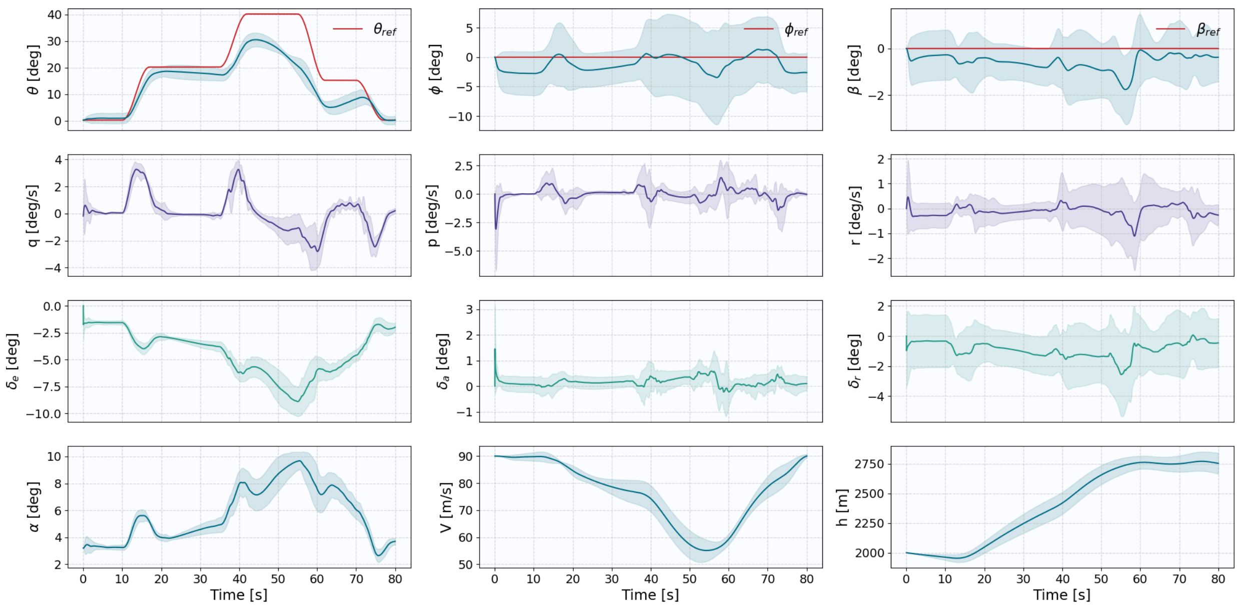


Figure 6.52: States time-traces of the *Risky* RUN-DSAC policy in the **stall** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

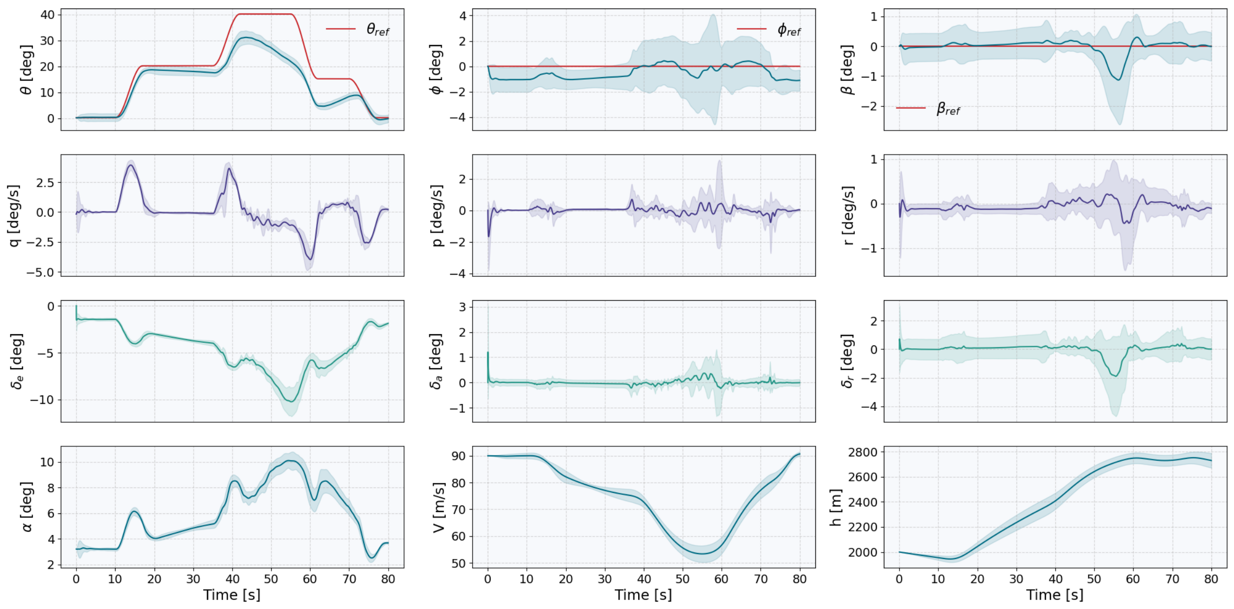


Figure 6.53: States time-traces of the *Conservative* RUN-DSAC policy in the **stall** scenario: solid lines show the average response of 10 agents, red lines denote reference signals, and shaded areas represent standard deviation.

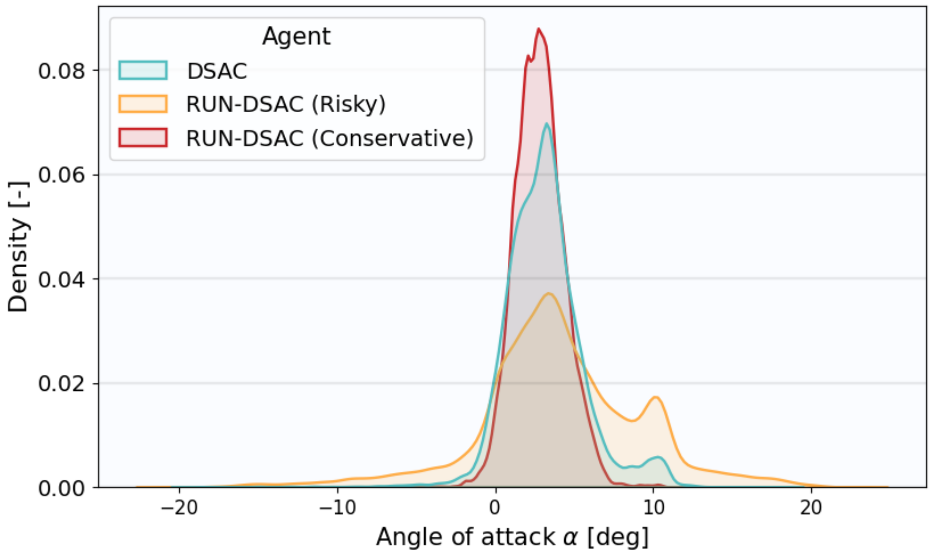


Figure 6.54: Distribution of angle of attack (α) experienced by various distributional agents during a single training run, reflecting the range of α values encountered by each agent.

7 Hyperparameter Optimization

The hyperparameter optimization was conducted using a Gaussian Process-based search algorithm, as implemented in the Weights&Biases[1] platform. The objective function for this optimization task was the average of the episodic returns from the final five subsequent episodes, calculated after completing 200 episodes of learning. To obtain a statistically robust metric, each agent's episodic return was evaluated using an ensemble approach, wherein the return was averaged across 10 independent evaluations initiated from an identical random seed. This mitigates the effects of stochasticity and ensures a more reliable estimation of the performance metric being optimized.

A sweep of 100 distinct hyperparameter configurations was conducted. The hyperparameters selected for tuning — minibatch size, DNN architecture, discount factor, replay buffer size, actor learning rate, and critic learning rate — were identified as potentially having the most significant impact on model performance. The range of values chosen for these hyperparameters was guided by empirical findings from existing literature on SAC and DSAC algorithms [27, 116, 28].

The interplay between these hyperparameters and the evaluation metric is visualized through a parallel coordinates chart, as displayed in Figure 7.1. In this representation, each axis corresponds to a specific hyperparameter, and each line signifies an individual trial. The color gradient scale, positioned on the right side of the chart, provides a color-coded assessment of the model performance. Specifically, brighter yellow hues indicate higher returns, which are considered favorable outcomes, whereas darker blue shades denote lower returns, signaling suboptimal performance. The line coloration is correspondingly mapped to the evaluation metric, offering an immediate visual cue for the efficacy of each hyperparameter set.

Figure 7.1 shows that most of the hyperparameter resulted in desirable outcomes, thereby corroborating the relative robustness of the DSAC algorithm with respect to hyperparameter sensitivity, as previously evidenced in the literature [28]. However, a few combinations failed to learn sufficiently good policy within the 200 episodes, resulting in very low returns.

The limitation of Figure 7.1 is that it does not provide an indication of the impact of each of the hyperparameters on the selected metric. Hence, Figure 7.2 was included, which shows which of the parameters were the best predictors of, and highly correlated to desirable values of the selected metric[2]. Here, *Correlation* is the linear correlation between the hyperparameter and the Average Returns. So positive correlation means that when the hyperparameter has a higher value, the metric also has higher values (visualized by green color, where a longer color bar means higher correlation) and vice versa (visualized with red color).

Although correlation offers insights into hyperparameter relevance, it fails to encapsulate second-order interactions between inputs. Moreover, comparing inputs with vastly disparate ranges, such as those examined for learning rates, can lead to complex and less interpretable results. Therefore, an *importance* metric was also evaluated. This metric is obtained using a *Random Forest* model trained with hyperparameters as features and the target Average Returns metric as the output. Within each decision tree of the Random Forest, *impurity* is quantified using the Mean Squared Error (MSE). Specifically, the algorithm evaluates each potential node split by calculating the weighted sum of the MSEs for the prospective child nodes, with weights proportional to the sample sizes. The optimal split minimizes this weighted sum. This reduction in impurity, attributed to each hyperparameter, is then aggregated and averaged across the ensemble of

[1]Weights&Biases documentation: https://docs.wandb.ai/ [Visited 01/09/2023]

[2]Weights&Biases documentation: https://docs.wandb.ai/ [Visited 01/09/2023]

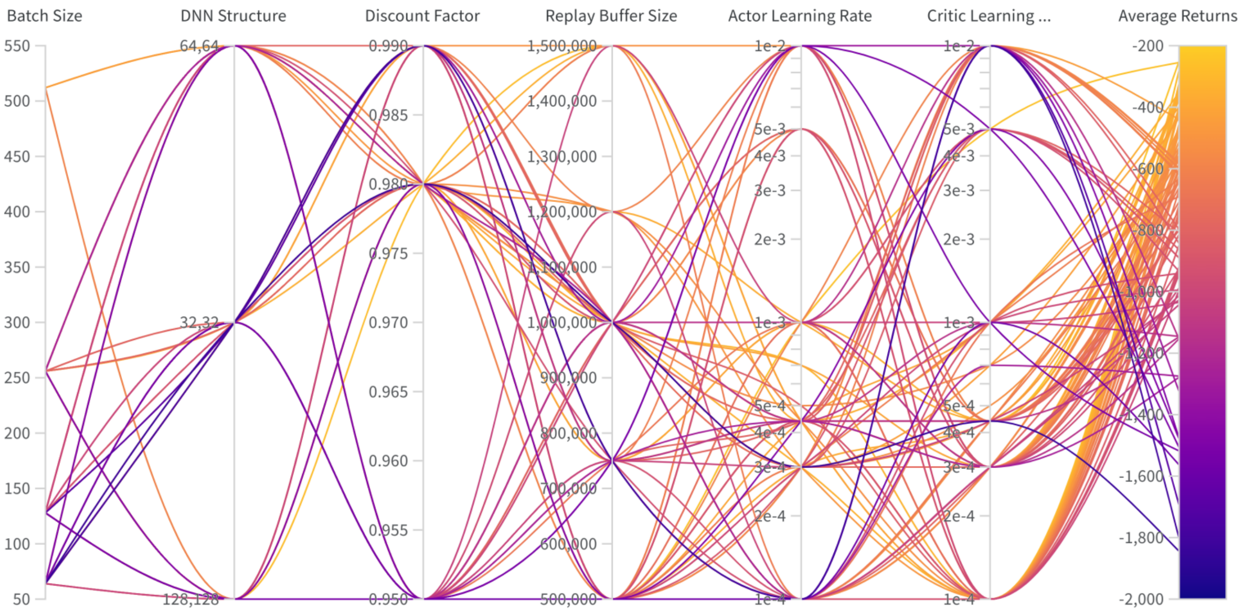
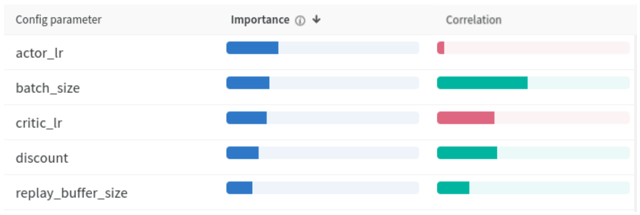


Figure 7.1: Parallel coordinates chart for a sweep of selected DSAC hyperparameters. Each configuration was trained for 200 episodes.

trees to generate a *feature importance* score for each hyperparameter. These scores are subsequently normalized to sum to one, providing a relative ranking of hyperparameter importance.

As depicted in Figure 7.2, none of the examined hyperparameters exhibit a marked difference in importance when compared with each other. Notably, the actor learning rate is ranked as the most influential, despite having a minimal and negative correlation with Average Returns. This paradoxical finding can be attributed to the wide range of values explored for the actor learning rate: both exceedingly low and high values detrimentally impact performance, thereby skewing the correlation metric. In contrast, the replay buffer size was identified as the least influential hyperparameter based on the same evaluation metric.

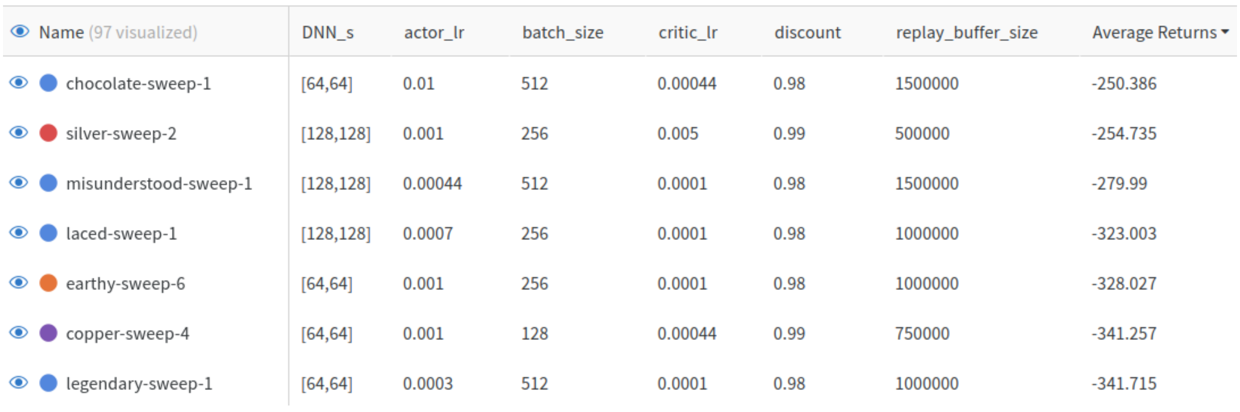
Two other parameters were examined but are not included in Figure 7.2. Firstly, Runtime was anticipatedly ranked as the most important variable. Nevertheless, this variable was excluded from further comparative analysis, as its behavior is contingent upon the values of the other hyperparameters under study. Secondly, the DNN structure is also omitted from Figure 7.2. This is because the DNN's structural complexity was represented as a tuple rather than a scalar value during hyperparameter optimization. As a result, the *Weights and Biase*s tool was unable to compute its correlation and importance in the same manner as it did for the other hyperparameters.



Config parameter	Importance ⓘ ↓	Correlation
actor_lr		
batch_size		
critic_lr		
discount		
replay_buffer_size		

Figure 7.2: Correlation and importance of the optimized hyperparameters.

In the conducted experiments, seven out of 100 trials achieved an Average Returns score exceeding -350. The hyperparameter configurations corresponding to these successful runs are documented in Figure 7.3. These configurations served as a basis for hyperparameter selection. It is crucial to clarify that a comprehensive search over all possible hyperparameter combinations was not conducted, owing to computational inefficiency. Rather, a stochastic sampling approach was employed to explore the hyperparameter space. Given this methodological limitation, it is conceivable that untested combinations of hyperparameters could yield even better performance. Consequently, the optimal hyperparameter settings were not directly adopted from the best-performing trial. Instead, the selections were made through a multi-criteria analysis, incorporating insights from the top seven configurations displayed in Figure 7.3, the importance and correlation metrics from Figure 7.2, as well as findings from prior research on SAC and DSAC.



Name (97 visualized)	DNN_s	actor_lr	batch_size	critic_lr	discount	replay_buffer_size	Average Returns
🔵 chocolate-sweep-1	[64,64]	0.01	512	0.00044	0.98	1500000	-250.386
🔴 silver-sweep-2	[128,128]	0.001	256	0.005	0.99	500000	-254.735
🔵 misunderstood-sweep-1	[128,128]	0.00044	512	0.0001	0.98	1500000	-279.99
🔵 laced-sweep-1	[128,128]	0.0007	256	0.0001	0.98	1000000	-323.003
🟠 earthy-sweep-6	[64,64]	0.001	256	0.0001	0.98	1000000	-328.027
🟣 copper-sweep-4	[64,64]	0.001	128	0.00044	0.99	750000	-341.257
🔵 legendary-sweep-1	[64,64]	0.0003	512	0.0001	0.98	1000000	-341.715

Figure 7.3: The best seven runs obtained during the hyperparameters tuning and the corresponding hyperparameter settings.

The following hyperparameter choices were made:

- **DNN Structure = [64, 64]:** Selected as it was used in the best-performing run and in 4 out of the top 7 runs. Furthermore, it aligns with previous works on SAC and DSAC for aircraft attitude tracking [27, 28]. Also, this structure brings additional advantages in comparison to a larger [128, 128] network, as it is faster to train, requires less computational resources, and is less susceptible to overfitting.
- **Actor Learning Rate = 10^{-3}:** Selected because 3 out of the top 7 runs used this value, and it is close to the value used by the previous work on DSAC of $4.4 \cdot 10^{-4}$ [28].
- **Batch Size = 256:** Preferred as 3 out of the top 7 runs used this value. Furthermore, this batch size was used in previous work on SAC and DSAC [27, 28]. Moreover, smaller batch sizes like 256 consume less GPU memory compared to larger sizes like 512.
- **Critic Learning Rate = $4.4 \cdot 10^{-4}$:** Set as it is the top-performing value and it is in line with previous work on DSAC [28]. Furthermore, it falls within the same range as the second considered value, 0.0001, which appears in 4 instances among the 7 top performing runs.
- **Discount Value = 0.98:** Chosen as it appeared 5 times in the top 7 runs, reflecting its effectiveness.
- **Replay Buffer Size = 10^6:** A replay buffer size was evaluated as the least important in Figure 7.2, which is also reflected in Figure 7.3, where the replay buffer sizes of the top performing agents are inconsistent. Hence, a value of 10^6 was selected, similar to previous work on DSAC [28].

The remaining DSAC hyperparameters were based on the preliminary analysis (see Chapter 5) and previous literature [116, 111, 28]. Furthermore, the RUN-DSAC algorithm, developed in this study, has one more hyperparameter that requires tuning - the number of episodes during which the uncertainty modulation factor μ linearly decays from an initial value to 0, N_μ. N_μ was optimized for two cases, for an initial μ of 1 and an initial μ of -1, with the goal of maximizing the average returns after 250 episodes. For both cases, N_μ of 125 was observed to be the best performing, which corresponds to 1/2 of the total number of training episodes.

Finally, the hyperparameters for the SAC algorithm were systematically optimized utilizing an identical approach to that deployed for DSAC, with consideration of the previous works on SAC applied to flight attitude control [179, 28]. In fact, the same values of hyperparameters were observed to be best-performing as for DSAC, which facilitates the comparison of the two agents. The optimized hyperparameters for SAC, DSAC and RUN-DSAC variants are summarized in Table 7.1.

Table 7.1: Optimized hyperparameters for the SAC, DSAC and RUN-DSAC agents

Hyperparameter	Notation	Value
Shared parameters		
Hidden layers structure	$\bar{h}$	64x64
Actor learning rate		10^{-3}
Critic learning rate		$4.4\cdot 10^{-4}$
Discount factor	γ	0.98
Batch size	$\|\mathcal{B}\|$	256
Replay buffer size	$\|\mathcal{D}\|$	10^{6}
RUN-DSAC parameters		
Initial uncertainty modulation factor*	μ_{init}	-1 or 1
Uncertainty modulation decay horizon*	N_{μ}	125

8 Sensitivity of the RL-based controllers to Environment Design

In the comparative evaluation of the RL-based flight controllers, it was empirically determined that the SAC algorithm manifests heightened sensitivity to variations in both agent-level and aircraft control environmental parameters. This observation is consistent with prior research findings [179, 28]. Additionally, this research indicated that SAC's learning curve exhibits slower convergence rates relative to its DSAC and RUN-DSAC counterparts. These findings inspired subsequent investigation into the environmental variables affecting the learning performance of RL-based flight controllers, particularly SAC. The focus of this analysis is to gauge the impact of reward function constraints, as detailed in Section 8.1, and action space sensitivity, elaborated upon in Section 8.2.

The present research also identifies that the risk-affinity parameters of the new RUN-DSAC algorithm produced contrasting effects on learning outcomes between the *LunarLander* and the attitude flight control environments. Hence, to delve deeper into the sensitivity of the results to the task complexity, Section 8.3 examines the impact of the flight attitude tracking task simplification on the performance of both the *Risky* and *Conservative* RUN-DSAC variants.

8.1. Constrained vs. Unconstrained Reward Functions

To investigate the sensitivity of the SAC algorithm to various environmental parameters, a key focus was the characterization of the reward function. This study aimed to elucidate the impact of constraint severity on learning efficacy by analyzing two distinct reward functions:

- **A stringent rewards function** that terminates the episode and imposes a significant penalty if the agent exceeds specified attitude ($|\theta| \leq 60°$, $|\phi| \leq 75°$) and altitude thresholds ($h \geq 100m$);
- **A lenient reward function** that solely penalizes the agent for altitude violations ($h \geq 100m$) without any constraints on attitude boundaries.

One compelling reason for implementing stringent constraints coupled with high-penalty rewards is that they can act as a strong guiding mechanism, effectively channeling the agent towards desired behaviors by making deviations highly costly [182]. However, it's important to consider the potential downside: such constraints can introduce discontinuities in the reward function. These discontinuities create sharp gradients in the loss landscape [183], which can in turn destabilize the learning process by leading to function approximation errors, local minima traps, or even divergence.

The empirical results obtained for the SAC RL-based flight controller applied in the attitude tracking task, illustrated in Figure 8.1, indicate that the application of stricter reward functions leads to more rapid convergence rates. This suggests that the constrained learning environment effectively guides the agent towards the achievement of predefined objectives. The observed increase in variance of learning outcomes can be attributed to the substantial penalties imposed upon the crossing of attitude boundaries, thereby introducing additional variability in returns obtained in the learning process.

8.2. Action Space Sensitivity Analysis

While the original action space, given in Equation 8.1 [180], facilitated fast convergence for the DSAC and RUN-DSAC flight controllers, the same could not initially be stated for SAC. As a result, the action space

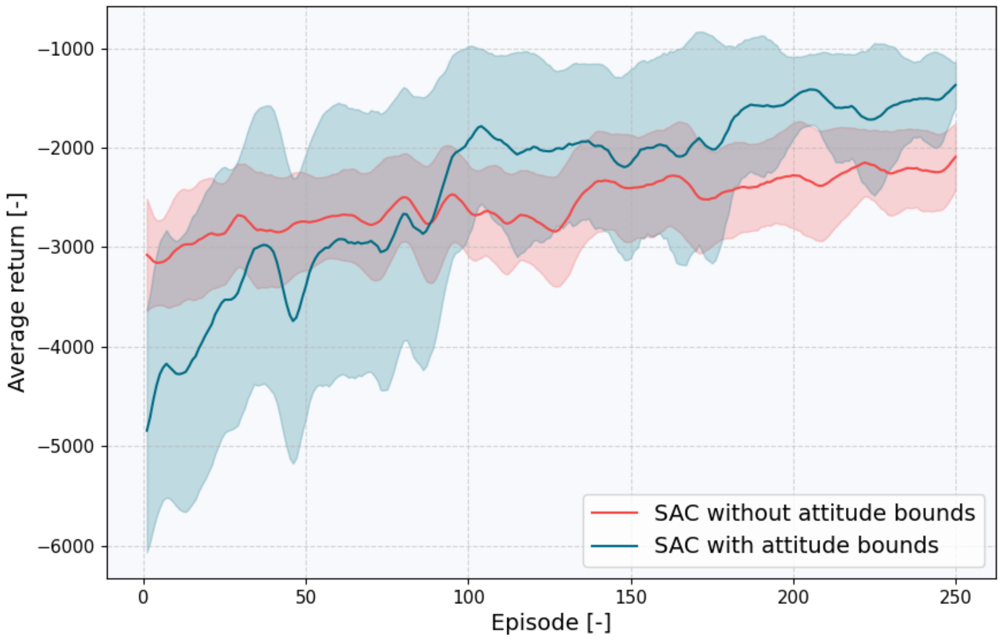


Figure 8.1: SAC learning curves for stringent and lenient reward functions.

was progressively expanded in a stepwise manner to identify conditions under which SAC demonstrated better convergence characteristics.

$$\mathcal{A} = \underbrace{[-17°, 15°]}_{\delta_e} \times \underbrace{[-19°, 15°]}_{\delta_a} \times \underbrace{[-22°, 22°]}_{\delta_r} \in \mathbb{R}^3 \tag{8.1}$$

Upon enlarging the action space to $\mathcal{A} = [-20°, 20°] \times [-20°, 20°] \times [-22°, 22°]$, a bifurcation in learning behavior of SAC was observed, as seen in Figure 8.2, which shows SAC learning curves of 10 independent runs for the aircraft attitude control task. In a minority of the trials (3 out of 10), SAC achieved rapid learning, significantly outperforming the runs with more constrained action-space in terms of both sample efficiency and final average returns. Conversely, in the majority of cases (7 out of 10), SAC exhibited deficient learning, resulting in subpar performance metrics compared to its more constrained counterpart. The observed divergence in SAC's learning outcomes upon increasing a certain hyperparameter is not unique to this algorithm. For instance, previous research has shown that an increase in learning rate in an IDHP flight controller improves convergence speed at the expense of reduced convergence success ratio [24].

Several hypotheses could be posited to explain the divergent behaviors observed when expanding the action space for SAC. One plausible explanation is the *curse of dimensionality*, where an increase in the action space dimensions exacerbates the complexity of the optimization landscape that the algorithm needs to navigate. In this context, SAC's variability in performance may be indicative of its sensitivity to initial conditions and stochastic elements within the learning process. Another potential reason lies in the balance between exploration and exploitation. A larger action space allows for more exploration, which could potentially lead to faster learning if the agent stumbles upon a particularly effective strategy. However, excessive exploration can also make the learning process more unstable, as evidenced by the 7 out of 10 runs where SAC failed to learn effectively. Lastly, the algorithm's internal hyperparameters and learning rates, which were not adjusted to account for the expanded action space, could also contribute to the observed inconsistency.

Nevertheless, these findings underscore the robustness of DSAC variants in comparison to SAC, as DSAC consistently demonstrated reliable learning even within more restrictive action spaces. Additionally, when the action space was enlarged to $\mathcal{A} = [-20°, 20°] \times [-20°, 20°] \times [-22°, 22°]$, DSAC continued to demonstrate stable convergence across all training runs, corroborated by 10 independent learning curves in Figure 8.3. This suggests that, unlike SAC, whose performance is sensitive to action space dimensions, DSAC offers a more reliable alternative for achieving stable learning outcomes.

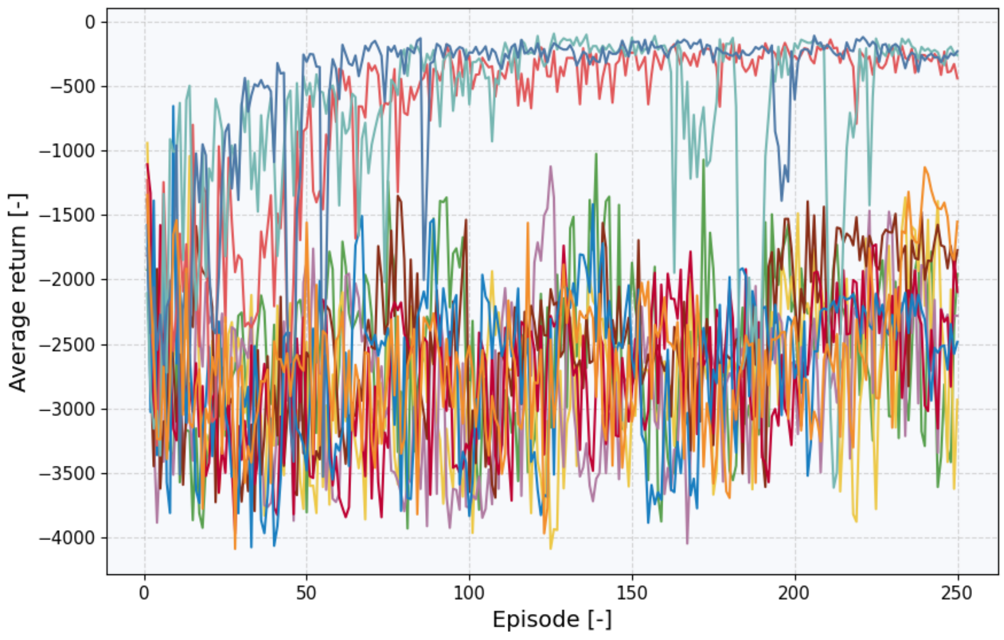


Figure 8.2: SAC learning curves with expanded action space.

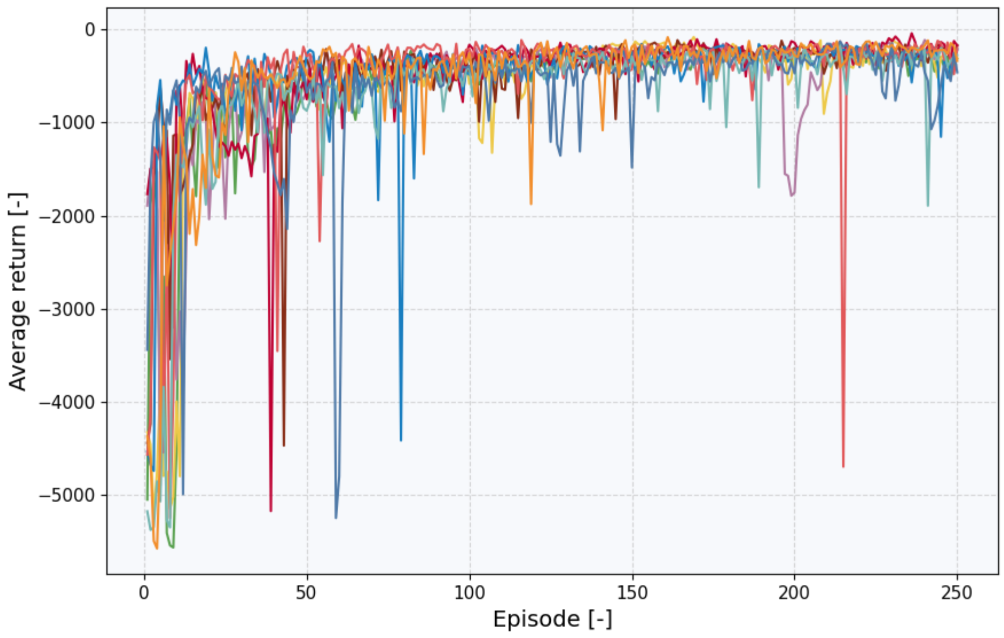


Figure 8.3: DSAC learning curves with expanded action space.

A rigorous sensitivity analysis could offer further insight into SAC's performance variance across different action space dimensions. Potential methodologies include high-resolution mapping of action space configurations against performance metrics like convergence rates and convergence success ratios. However, such detailed analyses are beyond this study's scope, which prioritizes comparative evaluation with DSAC variants rather than SAC optimization.

8.3. Task Complexity Sensitivity Analysis

As demonstrated in Part I, the *Conservative* RUN-DSAC variant enhances sample efficiency and controller robustness to unexpected scenarios. Conversely, the *Risky* variant, characterized by its preference for taking riskier actions, compromises sample efficiency through frequent violations of the predefined flight envelope, thus incurring substantial penalties. Nevertheless, Section 5.4.1 illustrates an improved sample efficiency for the *Risky* variant within the *LunarLander* environment. This difference can be interpreted by considering the unique characteristics and reward structures inherent to each environment.

In the *LunarLander* environment, the *Risky* RUN-DSAC approach seems to align well with the reward system. The environment incentivizes bold maneuvers that lead to a successful landing while penalizing unnecessary fuel usage and crashes. By favoring actions with higher uncertainty, the *Risky* variant encourages exploration, which can be particularly beneficial in discovering efficient and score-maximizing trajectories. The significant positive rewards offered for successful landings present an opportunity for the algorithm to leverage risk for potential high returns. Furthermore, the undesired behaviour is not punished as harshly as in the attitude control case, which makes it ”safer” for the agent to explore.

Conversely, the flight attitude control task for the Cessna Citation II aircraft demands high precision to maintain a specific trajectory and to ensure adherence to strict safety constraints. In this scenario, the *Conservative* RUN-DSAC variant, which subtracts the standard deviation from the Q-value, promotes actions with lower variability. This conservative strategy is beneficial in a context where deviation from the desired trajectory may result in violations of operational constraints or lead to catastrophic outcomes. The reward function, which imposes high penalties for boundary violations and altitude deviations, necessitates a careful approach that minimizes risk. The *Conservative* algorithm thus likely converges towards a policy that prioritizes safety and reliability over potential performance gains, which in the context of aircraft dynamics, is essential for successful task completion.

The observed performance differences of the RUN-DSAC variants underline the significance of algorithmic alignment with task objectives and constraints. This consideration has led to an exploration of the algorithm’s behavior under conditions of reduced complexity. By training the algorithm on simplified reference signals that reduce the likelihood of the aircraft crossing the predefined flight envelope, it is theorized that the performance of the *Risky* RUN-DSAC variant may be enhanced.

The original training task for the attitude flight control is designed as a smoothed step sequence featuring 15 evenly distributed levels within the given maximum amplitude range, as detailed in Part I. This task was simplified by lowering the number of levels to 5 and reducing the maximum amplitude of the pitch and roll angle reference signals to 5°, significantly less than the original settings of 25° and 45°, respectively. Analysis of the average learning curves from 10 independent runs for each agent, depicted in Figure 8.4, alongside a quantitative evaluation using three performance metrics detailed in Part I and presented in Table 8.1, indicates that the *Conservative* RUN-DSAC maintains superior performance over the *Risky* variant, even improving across all metrics compared to the original training settings. Furthermore, the comparison of the simplified task learning curves in Figure 8.4 with those from the original task in Part I reveals that, counterintuitively, the *Risky* RUN-DSAC’s performance degrades in the simplified scenario, which can be also observed in the performance metrics. It appears that the *Risky* RUN-DSAC’s inclination towards exploration over exploitation becomes a drawback, as the narrower range of acceptable actions necessitates precision over novelty.

Table 8.1: Assessment of agent learning efficiency using three metrics shows mean scores with standard deviations and percent difference from the respective learning curves trained with the original training tasks.

			M1 (Returns)	M2 (# Episodes)	M3 (Area)
Max. reference amplitude: 5°	RUN-DSAC (Risky)	Value	-1835.96±844.76	56.22±27.96	$39.72{\cdot}10^4 \pm 14.36{\cdot}10^4$
		Difference	-20.1%	+48.7%	+10.3%
	RUN-DSAC (Conservative)	Value	-306.96±163.13	7.56±6.82	$10.30{\cdot}10^4 \pm 5.60{\cdot}10^4$
		Difference	+23.0%	-16.9%	-20.2%
Max. reference amplitude: 2°	RUN-DSAC (Risky)	Value	-1296.53±584.84	68.67±24.47	$41.53{\cdot}10^4 \pm 15.50{\cdot}10^4$
		Difference	+15.2%	+81.7%	+14.7%
	RUN-DSAC (Conservative)	Value	-498.48±450.27	10.00±9.20	$13.52{\cdot}10^4 \pm 8.82{\cdot}10^4$
		Difference	-25.1%	+9.9%	+4.8%

However, the data from Figure 8.4 and Table 8.1 show increased variance in the learning curves for the *Conservative* variant compared to the original training settings. In contrast, the *Risky* variant generally

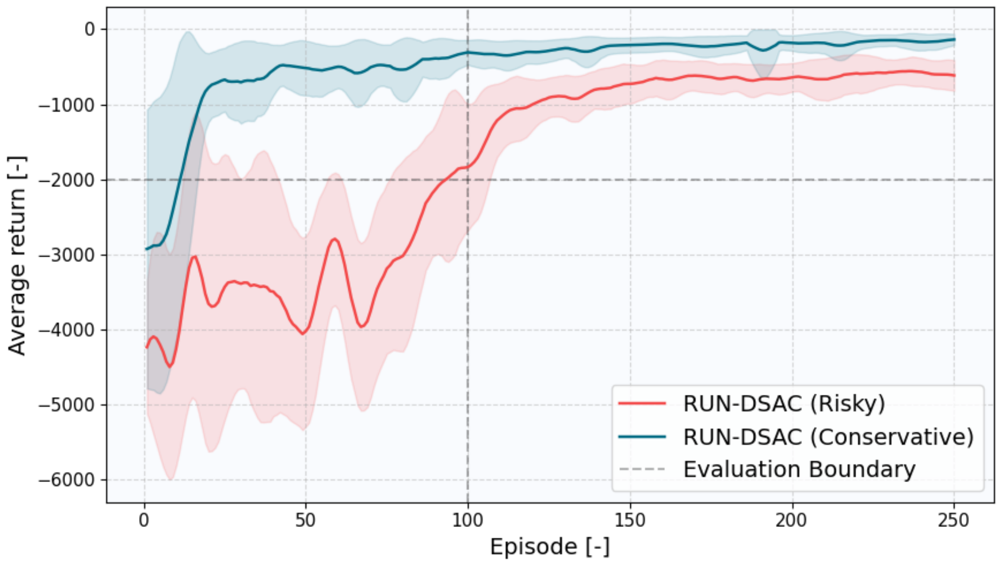


Figure 8.4: Average learning curves for training on reference signals with a maximum amplitude of 5°. The bold line represents the mean of 10 independent trials for each agent, with the shaded area indicating the standard deviation.

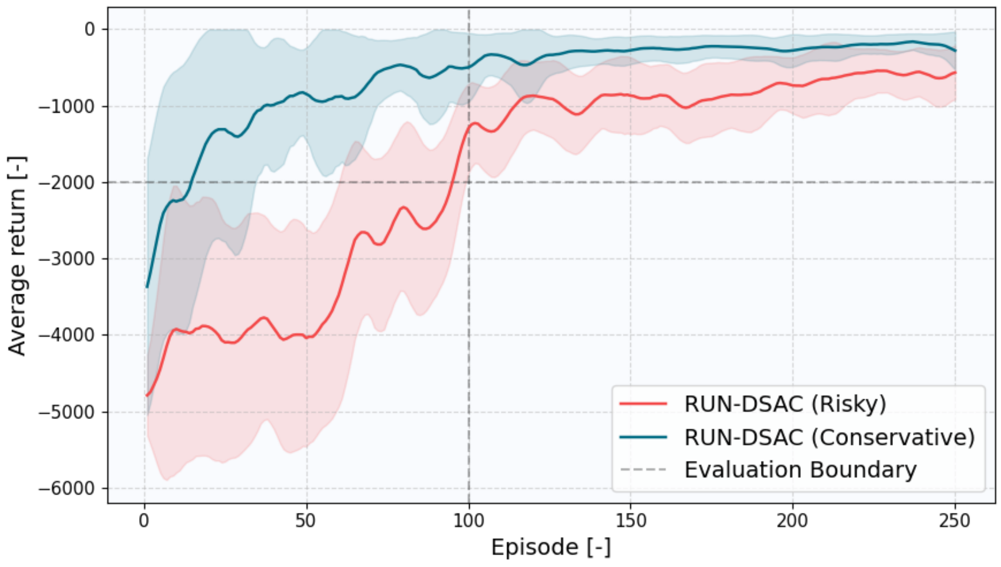


Figure 8.5: Average learning curves for training on reference signals with a maximum amplitude of 2°. The bold line represents the mean of 10 independent trials for each agent, with the shaded area indicating the standard deviation.

displays reduced variance in performance metrics, with the exception of a minor increase in the area-under-the-curve metric (M3). The *Conservative* variant's increased variance may stem from the the diminished amplitude range, which renders the reference signals more homogenous, thereby complicating the agent's ability to differentiate between them. Tracking subtle movements demands precise control, and the uniformity of actions, coupled with their limited magnitude, may impede the agent's learning process. Furthermore, the confluence of similar state-actions leading to comparable returns is hypothesized to be further obscured by introducing the uncertainty information into the Q-value estimations. On the other hand, the observed reduction in variance for the *Risky* RUN-DSAC likely results from the restricted scope for taking varied high-risk actions, as the tighter task constraints confine the range of viable maneuvers. This constrained operational space inherently curtails the possibility of engaging in the broad spectrum of actions that previously led to high variability, thereby yielding a more uniform and stable performance across different learning trials.

To test these hypotheses, an additional set of 10 independent runs for each RUN-DSAC variant was executed, further reducing the maximum amplitude of θ_{ref} and ϕ_{ref} to 2°. As evidenced in Figure 8.5 and confirmed by Table 8.1, this adjustment led to an increased variance in the learning curves of the *Conservative* variant, which also demonstrated diminished learning performance relative to the original learning curves. Meanwhile, the *Risky* variant's variance decreased in the initial learning phase, though not evidently in Figure 8.5. Notably, as learning approached convergence, the variance for the *Risky* version appeared elevated compared to the 5° scenario, potentially accounting for the slight increase in variance of the area-under-the-curve metric (M3).

9 Architectural Complexity and Computational Performance

This research has demonstrated how different variants of the DSAC algorithm can extend the capabilities of the SAC in flight control by estimating a complete return distribution. However, these advancements lead to increased computational demands, arising from modeling return distributions that requires additional network parameters, and the computation of pairwise TD-errors among randomly generated quantile fractions. This chapter delves into the intricacies of this trade-off, specifically examining the impact of incorporating distributional reinforcement learning on the critic architecture complexity in Section 9.1 and the training time required per episode in Section 9.2.

9.1. Critic Network Coplexity

The DSAC and RUN-DSAC agents utilize the same *tanh* Gaussian actor network as the SAC agent, with increased complexity manifested solely in their critic networks. Both DSAC and RUN-DSAC utilize an identical critic structure realized as an IQN that processes state-action pairs and a set of N uniformly sampled quantiles $\tau \sim U([0,1])$, outputing an inverse cumulative distribution function of potential outcomes ($\mathcal{S} \times \mathcal{A} \times \mathbb{R}^N \rightarrow \mathbb{R}^N$). The complexity enhancement arises from the IQN's expanded neural architecture, necessitating an increased quantity of trainable weights and biases attributable to:

- **Quantile Embedding:** An embedding of the size of $\Phi = 64$ (as inspired by previous research [111]) is employed for quantile fractions, increasing parameter count. This embedding layer maps the quantile fractions to a higher-dimensional space, which is subsequently used in the Hadamard product with the embedding of the states and actions.
- **Input-Output Expansion:** Unlike traditional critics that output a singular expected return, the distributional critics of DSAC and RUN-DSAC estimate a complete return distribution, thereby expanding both the input and output dimensions to accommodate N quantile estimations for each state-action input.
- **Layer Normalization:** Both models use layer normalization, but the distributional critics have more instances of it due to the additional layers related to quantile embeddings. Each normalization layer also adds a small number of parameters for scaling and shifting the normalized output.

Accounting for these enhancements and based on the specified critic network hyperparameters (refer to Part I), the distributional agents exhibit an 82.7% increase in parameter count over SAC, as detailed in Table 9.1. This increment underscores the computational trade-offs inherent in the pursuit of a more granular policy evaluation.

Table 9.1: Number of trainable parameters in critic models used in this research with percentage difference relative to the SAC critic.

	SAC	DSAC/RUN-DSAC	% Difference
Number of trainable parameters	5185	9473	+82.7

9.2. Computational Time

The increase in parameters generally corresponds to an increase in computational complexity. More parameters mean that there are more weights to be updated during back-propagation, which increases the computational load. This leads to longer training times, increased memory consumption, and potentially slower convergence per iteration. Empirical data detailed in Table 9.2 demonstrate the average training duration per episode across various agents. Within a flight attitude control scenario (referenced in Part I), training encompassed 250 episodes for 10 distinct runs per agent, with these ample sample sizes lending statistical significance to the results. Notably, the episodes were concluded prematurely if the agent deviated from the prescribed flight envelope (Section 8.1), influencing episode duration. All of the computation were performed on HP ZBook Power G9 Mobile Workstation using the 12^{th} Generation Intel Core i7-12700H × 20 CPU and the NVIDIA RTX A1000 GPU.

Table 9.2 indicates that SAC necessitates 26.5% less computational time compared to DSAC. In the context of this study, the enhanced sample efficiency, consistent learning, and reduced policy variance offered by DSAC outweigh its higher computational demand. However, it is important to note that this trade-off may not be universally applicable in all reinforcement learning scenarios, particularly in applications with different performance and computational constraints.

RUN-DSAC mirrors DSAC in computational performance, sharing the same DNN architecture for the critic and actor, with the only distinction being the computation of return distribution variance for policy updates. This assertion aligns with measured average training times, which reveal a negligible disparity of less than 10% of the *Risky* and *Conservative* RUN-DSAC variants compared to DSAC. Intriguingly, the *Risky* RUN-DSAC variant exhibited a slight decrease in training time relative to DSAC, which can be explained by the reward function's design which predisposes the *Risky* RUN-DSAC towards riskier actions that more frequently culminate in early episode termination. Conversely, the *Conservative* variant's longer average training time is attributable to its less frequent violation of flight constraints compared to DSAC.

Table 9.2: Comparison of average training time per episode among various agents, with (R) denoting *Risky* and (C) indicating *Conservative* strategies. The uncertainty is expressed as standard deviation of the measurements and the % difference represents the comparison of mean values with respect to the DSAC agent, with the significance of differences assessed by corresponding p-values.

	SAC	DSAC	RUN-DSAC (R)	RUN-DSAC (C)
Average time / Episode [s]	11.79$\pm$5.04	16.05$\pm$0.89	15.54$\pm$1.70	17.64$\pm$1.10
% Difference (compared to DSAC)	-26.5	-	-3.2	+9.9
p-value	$\approx$ 0.	-	$\approx$ 0.	$\approx$ 0.

10 Verification and Validation

In the intricate field of RL used within simulations, verification and validation stand as critical pillars for ensuring scientific rigor. In this work, *verification* refers to a process of ensuring that a computational model or simulation is implemented correctly and accurately represents the conceptual model or mathematical equations it is intended to solve. In other words, verification seeks to answer the question: *"Is the model implemented correctly?"*. This involves checking the code for bugs, assessing numerical accuracy, and confirming that the algorithms perform as intended. On the other hand, *validation* refers to a process of confirming that the model or simulation adequately represents the real-world system or phenomena it is designed to simulate or predict. The primary question here is: *"Is the right model being solved?"*. Validation usually involves comparing model predictions with experimental data or observations from the real world to assess how well the model captures the behavior of the system in question. Furthermore, this involves the examination of the underlying assumptions and limitations to delineate the model's range of validity.

The aim of this chapter is to articulate the rigorous quality control measures applied to both the implementation of RL methods and the experiments conducted. Concurrently, it aims to provide a nuanced analysis that assesses both the validity and the limitations intrinsic to the research project. A comprehensive overview of the verification procedures for the implemented RL algorithms and the simulation setup can be found in Section 10.1, while the validation steps are detailed in Section 10.2. Furthermore, repeatability and reproducibility are fundamental principles of scientific methods, which help ensure the integrity of the research by allowing other researchers to independently verify and validate the findings. Consequently, the measures employed to ensure the reproducibility and repeatability of this research work are the subject of discussion in Section 10.3.

10.1. Verification

The paramountcy of verification in the realm of RL and autonomous flight control cannot be overstated. This section aims to provide a meticulous verification process for the SAC and DSAC algorithms implemented in this study, alongside the simulation models employed.

10.1.1. Reinforcement Learning Algorithms

The SAC and DSAC RL algorithms employed in this study, forming the basis for the development of RUN-DSAC, are grounded in an extensive foundation of deep reinforcement learning research spanning multiple years. These algorithms comprise a multitude of distinct, modular tools, methodologies, and constituents. Noteworthy among these are the utilization of double critic networks, the incorporation of experience replay buffers, the employment of DNNs, and the utilization of stochastic gradient optimizers. In order to verify the correct implementation of the SAC and DSAC algorithms, they were subjected to testing in simpler environments to ascertain whether they could achieve the performance levels reported in the existing scientific literature.

Initially, the algorithms were evaluated and juxtaposed within a *LunarLander* toy environment, as expounded in Section 5.1. This specific environment was chosen because it offers a continuous state-action space and continuous reward signals, making it highly suitable for the smooth integration of the RL frameworks into the flight control task. Moreover, its level of complexity is sufficiently high to effectively serve as a rigorous testing ground for the algorithms, all the while mitigating the issue of excessive computational times. The SAC and DSAC algorithms both demonstrated proficiency in mastering the continuous control task at hand, thereby affirming the accurate implementation of these algorithms. Furthermore, the empirical findings corroborated the enhanced learning performance attributed to DSAC when compared to SAC, as discerned in the literature study. However, it is worth noting that the performance of SAC has exhibited a tendency to yield inconsistent results, thereby confirming the findings documented in prior research [179].

The feasibility of the implementation of the proposed DSAC algorithms was further tested by using them to control a simple LTI model of Cessna 500 aircraft. Additionally, this controlled environment served the purpose of verifying multiple facets: firstly, the efficacy of augmenting the observation vector with reference signals; secondly, the effectiveness of the CAPS method in mitigating the pronounced oscillations in control actions; and thirdly, the implementation accuracy of standard reward signals commonly employed in automatic control. Moreover, the precision in the implementation of individual modules within the RL algorithms was tested by verifying that alterations in the hyperparameters relevant to these modules yielded anticipated changes in agent performance. For example, the verification revealed that augmenting the learning rate for both the critic and policy modules resulted in accelerated initial learning. However, setting the learning rates too high induced instability in the learning process.

Moreover, unit tests were designed to rigorously evaluate the correctness of both the RL algorithms and the underlying environments they were employed in. Unit tests serve as a crucial quality assurance mechanism by systematically examining specific components and functions within the code. They verify whether individual units of code, such as functions or methods, behave as expected under various conditions. In this study, these unit tests were implemented with the `unittest` Python framework. An illustrative example of such unit testing applied to the Cessna Citation II RL environment is presented below:

```python
class TestCitationEnv(unittest.TestCase):

    def setUp(self):
        torch.cuda.empty_cache()
        parser = argparse.ArgumentParser(description='Distributional Soft Actor Critic',
                                         conflict_handler='resolve')

        # Add command-line arguments to the parser
        parser.add_argument('--config', type=str, default="../configs/Attitude/config_sac.yaml")
        parser.add_argument('--gpu', type=int, default=0, help="using cpu with -1")
        parser.add_argument('--seed', type=int, default=0)

        # Parse the command-line arguments
        args = parser.parse_args()

        # Open the configuration file specified in the '--config' argument for reading
        with open(args.config, 'r', encoding="utf-8") as f:
            # Load the YAML contents of the file into a Python dictionary
            variant = yaml.load(f, Loader=yaml.FullLoader)

        # Initialize the environment with a specific configuration and mode
        self.env = phlabenv.CitationEnv(configuration=variant, mode="nominal")

    def test_action_space(self):
        # Check if the action space is of the correct type (Box)
        self.assertTrue(isinstance(self.env.action_space, gym.spaces.Box))

    def test_observation_space(self):
        # Check if the observation space is of the correct type (Box)
        self.assertTrue(isinstance(self.env.observation_space, gym.spaces.Box))

    def test_reset(self):
        # Check if the reset function returns an initial observation
        obs = self.env.reset()
        self.assertTrue(isinstance(obs, np.ndarray))
        self.assertTrue(np.allclose(obs[:6], np.zeros(6)))

    def test_step(self):
        # Reset the environment
        self.env.reset()

        # Check if the step function returns the expected values
        action = np.array([0.1, 0.1, 0.1])
        obs, reward, done, info = self.env.step(action)

        # Check the types of returned values
        self.assertTrue(isinstance(obs, np.ndarray))
        self.assertTrue(isinstance(reward, float))
        self.assertTrue(isinstance(done, bool))
        self.assertTrue(isinstance(info, dict))

    def tearDown(self):
        # Close the environment
        self.env.close()
```

Finally, as noted earlier, the learning performance of the SAC algorithm was observed to be inconsistent in the *LunarLander* environment. This high variance in the results was also pronounced when the algorithm was employed for attitude control of an aircraft simulated using a DASMAT model. To verify that the observed performance fluctuations were intrinsic to the SAC algorithm rather than resultant from any implementation errors, additional experiments were conducted in the *Pendulum* environment — a more elementary benchmark as compared to *LunarLander*. The *Pendulum* `gym` environment, depicted in Figure 10.2, is modeled as a frictionless pendulum pivoted at a fixed point, with the free end subject to controllable torque. Initialized at a stochastic position, the objective is to exert appropriate torque to elevate the pendulum to a vertically upright position, aligning its center of gravity directly above the pivot. The state space comprises a three-dimensional observation vector that captures the 2D Cartesian coordinates of the pendulum's free end, as well as its angular velocity. The action space is unidimensional, representing the magnitude of applied torque.

Subsequent analysis of the learning curves, averaged over 7 independent trials as shown in Figure 10.2, reveals that convergence was consistently achieved within six episodes, accompanied by minimal variance. These findings lend credence to the correctness of the SAC implementation, reinforcing the notion that the algorithm's inconsistent performance in more complex settings is, in fact, an inherent attribute rather than a manifestation of implementation inaccuracies.

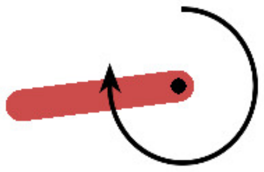


Figure 10.1: Pendulum environment

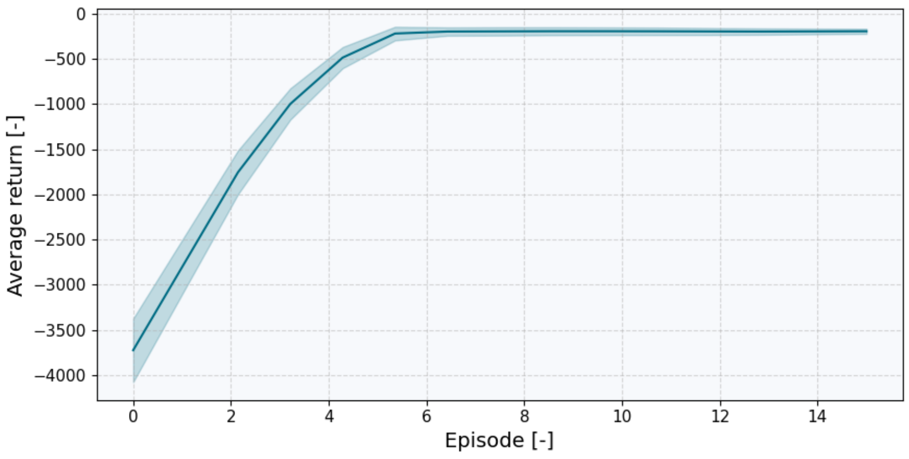


Figure 10.2: Learning curve of SAC trained in the *Pendulum environment*.

10.1.2. Cessna Citation II Simulation Model

This research seeks to evaluate the potential applicability of the proposed RL algorithms to real aircraft control. Due to the considerable safety risks and financial implications of real-world aircraft testing, a thoroughly validated simulation model serves as the foundational platform for this research. Specifically, Delft University of Technology Aircraft Simulation Model and Analysis Tool (DASMAT) model is utilized for scrutinizing the performance of RL-based flight controllers [184]. While originally designed for high-fidelity, 6 DOF simulations of the Cessna Citation I (or Cessna Citation 500), DASMAT has been extended to accommodate the Cessna Citation II (or Cessna Citation 550), which is the subject of the current research.

However, the DASMAT model is constructed in MATLAB Simulink, whereas the RL controllers in this study are implemented using Python. To bridge this gap, the Simulink model was translated into C code and compiled as a shared object executable for various flight configurations. To ensure the accuracy of this conversion, the aircraft responses generated by the Simulink model was compared with the responses obtained from the shared object executable, both of which were excited using identical control inputs. An example of this comparison, carried out under nominal conditions ($V = 90m/s$, $h = 2000m$, no in-flight failures), is presented in Figure 10.3. The aircraft dynamics are activated through a rectangular pulse in both the elevator and aileron controls. If the model compilation was executed correctly, the aircraft responses would be expected to match, which is confirmed by the presented findings.

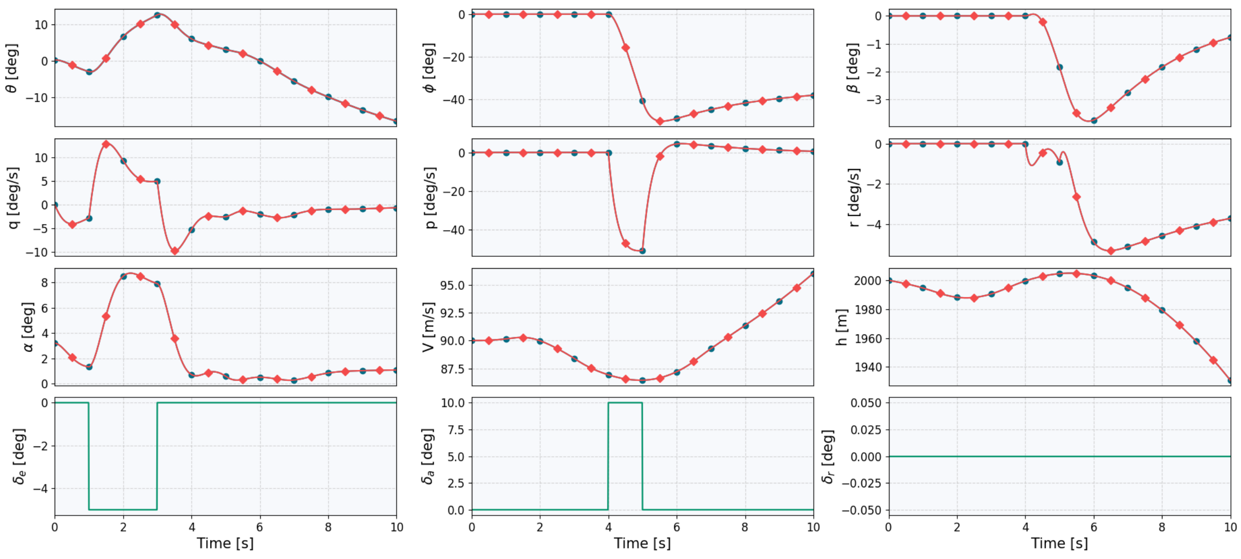


Figure 10.3: Comparison of Step Response: DASMAT Simulink Model vs. Compiled Shared Object Executable. Blue line with circle markers represents the Simulink model, while red line with diamond markers represents the compiled executable. Control inputs, depicted by green lines, were used to excite the states and are consistent across both models.

To further ensure the fidelity of the compiled simulation model, it can be rigorously assessed by comparing the observed outcomes with theoretical expectations grounded in fundamental principles of aircraft dynamics. In the body-fixed coordinate system, a negative elevator deflection should naturally result in a positive pitch rate, culminating in a pitch-up maneuver, which is observable at $t = 1s$. This maneuver is also expected to result in an elevated angle of attack α. Correspondingly, the increased angle of attack amplifies lift-induced drag, leading to an initial reduction in airspeed. Similarly, a positive aileron deflection yields a negative roll rate, as expected, manifesting as a negative bank angle around $t = 4s$. A near-zero roll rate is also indicative of a stable roll angle, a condition met from $t = 8s$ onward. Lastly, the expected coupling between rolling and yawing motions is observable, as initiating negative rolling motion through aileron input leads to a negative yaw rate and sideslip angle β.

For additional model verification, the aircraft dynamics response obtained under nominal conditions in this study was cross-referenced with the verification conducted by Dally [179]. Dally also used the DASMAT model, trimmed and compiled it at identical nominal conditions, and carried out the verification with the same control inputs. Therefore, a congruent aircraft response between this research and Dally's would substantiate the accuracy of the model compilation in this study. The comparative verification from Dally is illustrated in Figure 10.4. Indeed, the profiles and magnitudes of the relevant state variables are in alignment.

10.2. Validation

The indispensability of validation parallels that of verification in the fields of RL and autonomous flight control. This section serves a dual purpose: establishing the credibility of the DASMAT aerodynamic model and scrutinizing the foundational assumptions and limitations underlying this research.

10.2.1. Cessna Citation II Simulation Model

As outlined in Section 10.1.2, the DASMAT aerodynamic model serves as a computational model for the real-world dynamics of a Cessna Citation II aircraft. To validate the model's fidelity, all the longitudinal and lateral force and moment coefficients of the simulated aircraft were compared with those derived from PH-LAB's Cessna Citation II flight data. The relative Root Mean Square Errors (RRMSE) within the nominal, pre-stall flight envelope are documented in Table 10.1 [185]. The majority of the RRMSE values for the

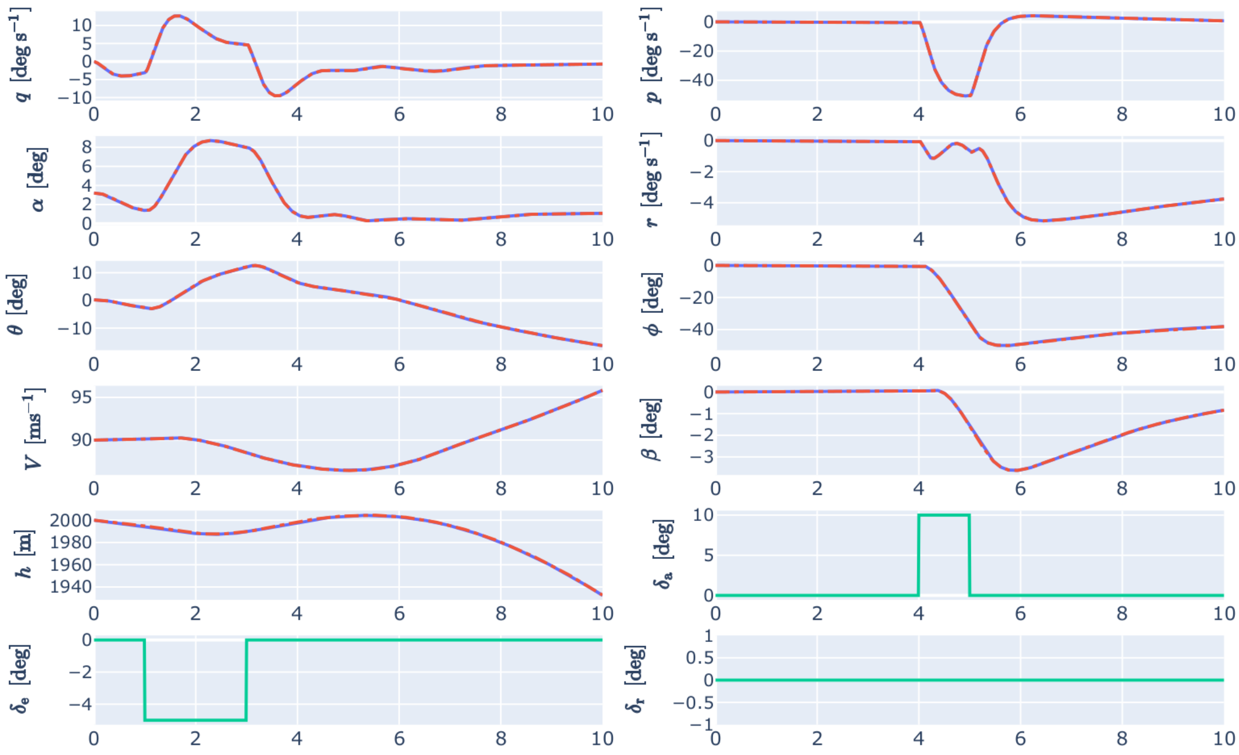


Figure 10.4: Compilation verification conducted by Dally [179]. The DASMAT Simulink model is depicted by dashed green lines, the compiled shared object executable by solid blue lines, and the common control inputs by solid green lines.

coefficients are below 8.80%, peaking at 12.65% for the pitching moment coefficient, C_m. These values are deemed sufficiently accurate to represent real-world aircraft dynamics, thus validating the aerodynamic model for the pre-stall flight envelope.

Table 10.1: Relative Root Mean Square Error (RRMSE) of force and moment coefficients for the DASMAT model in comparison to PH-LAB Cessna Citation II flight measurements [185].

	C_X	C_Y	C_Z	C_l	C_m	C_n
RRMSE [%]	8.79	7.34	7.97	8.65	12.65	8.50

10.2.2. Assumptions and Limitations

While the outcomes presented in this study exhibit promise regarding the deployment of proposed sample-efficient RL-based controllers aboard a real aircraft, it is imperative to acknowledge the presence of numerous assumptions and simplifications that have been incorporated, thereby constraining the scope and generalizability of the findings.

In this research, the DASMAT Simulink model forms the foundation for the evaluation of the RL-based flight controllers. Notably, this model incorporates an inner-loop PID yaw damper and a thrust controller, which simplifies the RL task significantly. Specifically, the RL agents are exclusively trained to control the flight control surface actuators, devoid of any involvement in thrust control, flaps actuators, or trim tabs actuators. Moreover, these control surface actuators are modelled by simple first-order lag dynamics with angle saturation. Furthermore, this model assumes no signal delay in the transmission between agent commands and actuator responses. It is also worth noting that the examination of simulation and controller frequencies, both of which are consistently set at 100 Hz, is not a part of this study.

The flight dynamics model relies on several other assumptions for its practicality. As the maneuvers used for both training and evaluation are short in duration, the Earth is assumed flat, hence no Earth's curvature is considered in the flight dynamics model. Moreover, the Earth is assumed to be non-rotating, therefore no Coriolis acceleration effects and such are considered. The aircraft is also assumed to have constant mass during the maneuvers and it is assumed to be a rigid body. Therefore, no flexibility or deformation of aircraft components are considered, although these are common in real operations and can influence the aerodynamic characteristics of an aircraft and effectiveness of its control surfaces.

Furthermore, although this research has examined the robustness of the proposed RL algorithms in several non-nominal or faulty conditions, several other sources of stochastic processes and uncertainties remain unaddressed, demanding comprehensive investigation prior to the practical deployment of the algorithm. For instance, while the effect of wind gusts were investigated, a more rigorous analysis of atmospheric influences on the performance of the proposed RL-based controllers is warranted. This could involve simulating and evaluating the impact of continuous atmospheric disturbances, possibly incorporating models like the *Dryden wind turbulence model*. In general, a more extensive exploration of the standard flight envelope is imperative to align with established safety standards and ensure the reliability and safety of the proposed RL-based controllers in real-world scenarios. Lastly, the research has omitted an analysis of model uncertainty between the simulated environment and the reality, which is essential for bridging the theoretical framework with practical implementation.

10.3. Repeatability and Reproducibility

Repeatability and reproducibility are cornerstones of scientific research and are particularly crucial in disciplines like RL that inherently involve stochastic processes. Ensuring that experiments are both repeatable and reproducible guarantees the integrity and validity of the findings. Repeatability allows other researchers, or even the same researchers at different times, to obtain similar results under identical conditions, thereby affirming the stability and reliability of the algorithm or methodology in question. Reproducibility, on the other hand, allows for the verification of results under varying conditions or even when using alternative methods, serving as a ”litmus test” for the generalizability of the findings. Several practices were implemented in this research to achieve the repeatability and reproducibility of the results.

To ensure both repeatability and reproducibility of the experiments, pseudo-random number generators were employed in both the stochastic agent and the simulation environment, and their states were initialized using predetermined seeds. Specifying a particular seed constrains the algorithmic behavior to remain invariant across individual runs. This determinism encompasses various stochastic components of the RL framework, including the initialization of neural network parameters (weights and biases), the stochastic gradient descent optimization steps, the sampling of experience tuples from the replay buffer, the stochastic action selection mechanism, and the randomized generation of reference training signals, among other factors. As empirically verified in Figure 10.5, the application of a consistent pseudo-random seed results in congruent learning curves across distinct experimental runs. It should be noted that this level of reproducibility is contingent upon meeting specific software and hardware prerequisites, including but not limited to, library versions, operating system specifications, and computational hardware configurations. To ameliorate the impact of these factors, a dependency file is included within the code package, listing all the requisite software libraries along with their corresponding version numbers.

Furthermore, all the hyperparameters, training, and evaluation configuration settings are stored in a `JSON` file for each experiment run. These configurations can be loaded automatically in the code. In addition, detailed logs are kept in a machine readable `CSV` file. Specifically, a unique identifier (based on the date and time corresponding to the initialization of the run) is generated for each of the experimental runs. Subsequently, a comprehensive set of variables is logged post each episode. These variables comprise returns, rewards, loss indices for both critic and policy architectures, along with predicted and realized critic function evaluations, temperature parameter α_T readings, selected actions, number of steps taken during exploration and evaluation. Moreover, the total computational time and time expended on distinct computational operations, ranging from data sampling and storage to training and evaluation, are also captured. Logging these parameters in detail establishes concrete baseline that offers a point of reference for future endeavors aimed at result replication.

The research also subscribes to an ethos of transparency by sharing the entire codebase publicly via a `Git` repository. In addition, `Git` is used as a version control system to keep track of all changes made to the code, including the addition of new features, bug fixes, and other algorithm modifications. In addition to these measures, code readability and subsequent reproducibility are further bolstered by the inclusion of inline comments that explain the purpose, logic, and algorithmic underpinnings of critical code segments.

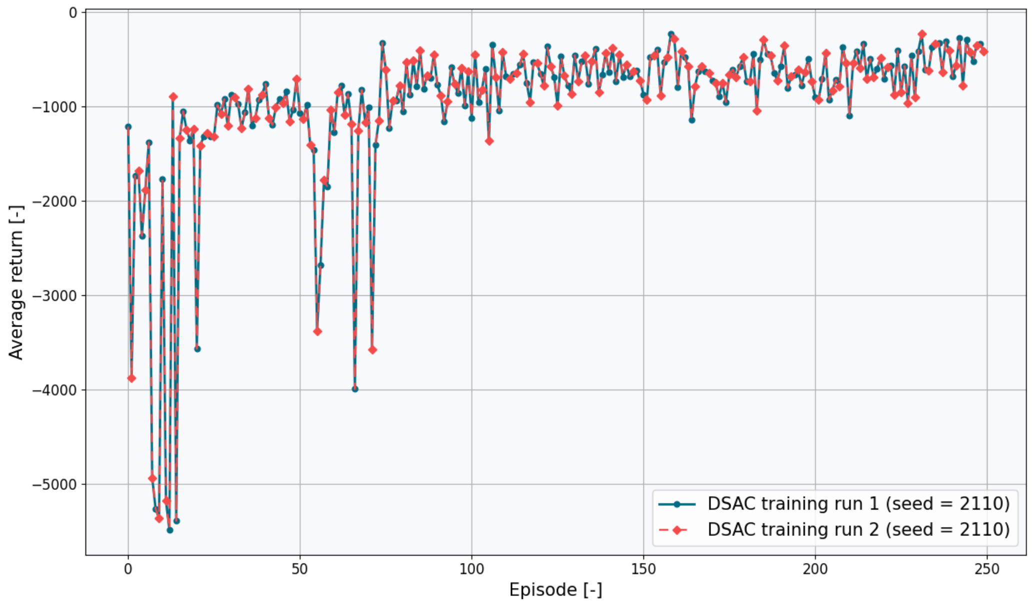


Figure 10.5: Comparison of learning curves from two independent experimental runs initialized with the same pseudo-random seed, demonstrating repeatability in algorithmic performance.

Part IV

Closure

11 Conclusion

While traditional automatic flight control systems are effective under standard conditions, they falter in the face of unanticipated scenarios due to their reliance on predefined models. Furthermore, developing and validating high-fidelity models is time-consuming and expensive, especially for complex aerospace system configurations. To address these limitations, there is a growing need for model-free controller design approaches that streamline development and reduce reliance on extensive model identification. This research proposed using Reinforcement Learning (RL) to create intelligent and adaptable flight control systems that can operate in partially observable and uncertain environments without strict reliance on accurate models.

Recent advancements in Deep RL have enabled the application of RL to complex flight control scenarios using deep neural networks as function approximators, but also introduced new challenges, such as sample efficiency and safety. Previously, methods like Incremental Dual-Heuristic Programming, Hierarchical RL, Soft Actor-Critic, or Shielded RL have been proposed to approach these challenges, each with their limitations and benefits. Nevertheless, the majority of these studies overlooked the incorporation of uncertainty quantification in the learning process, a factor that could potentially enhance both the efficiency and safety dimensions of the learning algorithms. Therefore, this research focuses on the exploration of Uncertainty-Aware RL algorithms applied to flight control with the goal of improving sample efficiency.

11.1. Closing Remarks

In the relentlessly advancing landscape of aerospace flight systems, the demand for model-free flight control systems that excel in efficiency, safety, and robustness under diverse flight conditions and potential in-flight faults is paramount. Contemporary advancements in Deep Reinforcement Learning algorithms have demonstrated remarkable proficiency in tackling tasks once considered intractable, offering a viable solution for modern flight controllers. These algorithms are grounded in the principles of direct interaction with the environment to learn model-free, sophisticated, near-optimal control strategies, thereby bypassing the limitations associated with traditional, model-based methods.

Furthermore, recent research has brought to light the extensive applicability of Distributional RL in diverse domains, particularly in robotics and control systems, including flight controllers. Unlike conventional RL methods that estimate only the expected value of returns, Distributional RL seeks to learn the entire probability distribution of returns. Building upon this concept, this thesis introduces the Returns Uncertainty-Navigated Distributional Soft-Actor Critic (RUN-DSAC) algorithm, which contributes a new perspective in the field of model-free autonomous flight control systems. By harnessing the power of Distributional RL and integrating uncertainty quantification into the RL decision-making process, RUN-DSAC addresses critical challenges in flight control - namely, sample efficiency, adaptability, and safety. In a comprehensive assessment involving a high-dimensional flight control task, RUN-DSAC is rigorously tested on a high-fidelity, non-linear, fully-coupled aerodynamics model of a small fixed-wing jet aircraft. Furthermore, its performance is benchmarked against the current state-of-the-art SAC and DSAC algorithms.

The RUN-DSAC algorithm, particularly its *Conservative* variant, has demonstrated significant improvements in learning efficiency and accuracy in tracking flight trajectories. The *Conservative* RUN-DSAC's emphasis on predictability and stability, by prioritizing actions with minimal uncertainty, is particularly advantageous in the safety-critical realm of aviation. Given these characteristics, the *Conservative* RUN-DSAC algorithm could potentially be highly suitable for online control systems, where real-time and safe adaptability is essential. This approach could be particularly beneficial in scenarios such as in-flight emergency handling or adaptive response to sudden environmental changes, where traditional model-based methods may fall short. In contrast, the *Risky* RUN-DSAC variant, which encourages exploratory behavior by favoring high-uncertainty actions, shows increased resilience to disturbances in control smoothness induced by unseen dynamics, albeit with compromises in tracking precision and increased variability among policies.

This thesis advances the domain of sample-efficient and fault-tolerant autonomous controllers, particularly in safety-critical systems like aerospace. Furthermore, it contributes to bridging the simulation-to-reality gap of intelligent flight controllers based on artificial intelligence by enhancing robustness to uncertainty, improving learning efficiency, and ensuring fault tolerance, achieved with minimal added human-domain knowledge and model dependence.

11.2. Research Questions

The present research aimed to enhance sample efficiency of state-of-the-art RL flight controllers in reference-tracking tasks by integrating uncertainty information into the learning process. Consequently, it has addressed a subset of the research questions proposed in Chapter 1.

Research Questions - Methodology

RQ-M-1 What is the state-of-the-art for RL in flight control?

It was concluded that the most suitable algorithms prioritize the use of an actor to directly parameterize the policy, allowing them to handle continuous action spaces. DDPG, TD3, and SAC are not only considered state-of-the-art in the broader field of Deep RL but have also shown promise in application-oriented research for fixed-wing and multi-rotor flight. This was confirmed by previous research attempts. Furthermore, these algorithms can be used in various combinations and modifications to improve on a certain characteristic. For example, they can be put into a cascaded architectures as done in Hierarchical RL to enhance the sample efficiency, they can be complemented with a safety filter as done in SHERPA to improve their safety, and similar. The most related previous work to this research is the application of Distributional SAC (DSAC) on a flight controller, which learns the entire return distributions instead of their mean values. DSAC was shown to significantly improve learning consistency of SAC. Furthermore, ADP methods such as IDHP were reviewed, which excel in online learning and adaptive optimal control. However, they face challenges when it comes to generalization power and sample efficiency in achieving precise flight control with high DOF. Nevertheless, previous research attempted to combine their online benefits with offline RL in a hybrid SAC-IDHP architecture to bridge the simulation gap.

Research Questions - Methodology

RQ-M-2 What are the sources of uncertainty in RL and how can they be leveraged?

Two distinct forms of uncertainty have been identified - epistemic and aleatoric. Epistemic uncertainty arises due to limited knowledge or incomplete information, such as an incomplete aircraft dynamics model. This type of uncertainty can be leveraged to improve the trade-off between exploration and exploitation, thereby enhancing sample efficiency. By incorporating epistemic uncertainty into RL algorithms, agents can make more informed decisions and explore different options more effectively. On the other hand, aleatoric uncertainty emerges from the inherent stochasticity in the environment, such as turbulence. Modeling and accounting for aleatoric uncertainty enables RL algorithms to make more thoughtful choices regarding actions that carry higher risk or involve greater uncertainty. This allows the algorithms to strike a better balance between exploration and exploitation, leading to potentially more efficient and safe learning while also improving the robustness of the learned policies.

Research Questions - Methodology

RQ-M-3 What is the state-of-the-art in uncertainty-aware RL methods?

RQ-M-3a How do these methods model the uncertainty information?

RQ-M-3b Which of these methods is the most suitable for flight control in terms of implementation feasibility, uncertainty prediction accuracy, and sample efficiency?

Different RL methods were discussed for capturing uncertainty, including Count-based methods, Monte Carlo Dropout (MC-Dropout), Bootstrapping, Distributional RL and Bayesian RL. Count-based methods quantify epistemic uncertainty through state visit counts. MC-dropout and bootstrapping generate multiple stochastic predictions to assess variance and capture epistemic uncertainty. Distributional RL (DRL) directly learns value distributions for representing aleatoric uncertainty. Finally, Bayesian RL (BRL) updates probability distributions using Bayes' rule to infer epistemic uncertainty. Moreover, DRL and model-free BRL were chosen for further evaluation in the context of flight control, as they have shown promise in improving the sample efficiency by accurately modelling the whole uncertainty distribution of the value function.

Model-free BRL shows potential in improving the sample efficiency in RL-based flight control, but its scalability limitations and computational demands make it impractical for complex flight dynamics in real-time control. Furthermore, finding an appropriate prior to accurately represent the value function remains an open question, especially for the Q-function. In contrast, the computationally less expensive and more easily scalable DRL has shown improved sample efficiency by modeling the entire reward distribution without needing a prior, although it does not directly capture epistemic uncertainty. Instead, it's learning performance and robustness are improved by modelling the aleatoric uncertainty, which is prevalent in flight control due to phenomena such as turbulence or noisy sensor measurements. Its well-documented theoretical foundations and successful application in various continuous tasks, including flight control, made it interesting for further exploration and development. Thus, DRL was concluded to be the most suitable for an Uncertainty-Aware RL-based flight controller developed in this study.

The theoretical enhancements in DSAC's sample efficiency over state-of-the-art methods like SAC were substantiated through verification within the Preliminary Analysis, utilizing the *LunarLander* environment as a platform for assessment. Furthermore, the study revealed the potential to extract variance information from the learnt quantile functions, enabling more informed policy decisions and ultimately enhancing sample efficiency. This crucial insight underscores the viability of uncertainty-aware approaches in refining RL-based flight control strategies.

Moreover, the feasibility of DSAC's implementation in flight control was substantiated through successful verification in a reference angle of attack α tracking task using an LTI short period Cessna Ce500 dynamics model. This significant milestone underscores DSAC's adaptability to real-world flight control scenarios. Furthermore, the effectiveness of the CAPS technique in curbing oscillations within controller actions was well-demonstrated. Notably, the study showcased CAPS' ability to achieve this without introducing undue complexity to the algorithms, making it a promising avenue for enhancing controller stability while maintaining sample efficiency.

Research Questions - Implementation

RQ-I-1 How can the proposed uncertainty-aware RL controller be implemented in a Cessna Citation II simulation model?

RQ-I-1a At which control level should the proposed RL controller be implemented?

RQ-I-1b Which simplifications can be implemented in the simulation model, while maintaining its relevant characteristics?

To advance the development of fault-tolerant and fully autonomous flight control systems, a controller capable of operating with the highest level of control authority is necessary. This entails the implementation of full 6-DOF flight control. However, the increased complexity associated with 6-DOF control, primarily due to the *curse of dimensionality*, often necessitates a hierarchical structure of RL-agents, as highlighted in prior research. This study specifically focuses on the safety-critical inner-loop attitude control to isolate and evaluate the RUN-DSAC's effects on sample efficiency, safety, and robustness when applied to flight control, without the confounding variables introduced by a multi-agent control system. This answers **RQ-I-1a**.

In addressing **RQ-I-1b**, this research utilizes the DASMAT model, an advanced simulation framework that replicates non-linear 6-DOF dynamics of fixed-wing aircraft. This high-fidelity model ensures realistic and applicable flight dynamics for both learning and evaluation phases. For the training process, the model was trimmed to maintain steady, straight, symmetric flight at an altitude of $h = 2000\ m$ and a true airspeed $V = 90\ m/s$. Nevertheless, the simulations were limited to in-cruise, clean configuration scenarios with minimal atmospheric disturbances, positioning this study as an initial proof-of-concept for the RUN-DSAC controller. The research also assumes ideal sensors, though biased sensor noise is modelled for the evaluation of RL agents, and the actuators are modeled using low-pass filter dynamics with fixed deflection saturation limits. Furthermore, aspects such as measurement and transport delays were not within the scope of this study and warrant further investigation. Additionally, external disturbances were simulated using basic angle of attack α step disturbances and modifications in controller outputs.

Research Questions - Implementation

RQ-I-2 How can in-flight faults, sensor noise, and atmospheric disturbances be appropriately implemented in the simplified model?

In this research, twelve scenarios involving in-flight faults, sensor noise, and atmospheric disturbances are implemented in the simulation model to evaluate the robustness and fault-tolerance of the flight control agents. In-flight faults are simulated in the DASMAT model by altering aircraft characteristics and control surface behaviors. This involves adjustments to the model's parameters and outputs to represent diverse scenarios, including ice accretion on wings, shifts in the aircraft's center of gravity, and imposed limitations on control surface movements.

Additionally, sensor noise is implemented using Gaussian noise models based on actual in-flight aircraft data. For atmospheric disturbances, the study includes simulations of wind gusts and variations in dynamic pressure to evaluate how well the agents could adapt to conditions they were not trained for and to sudden environmental changes. The wind gusts are modeled as step disturbances impacting the angle of attack α, providing a straightforward yet effective method to test the aircraft's aerodynamic response. Meanwhile, dynamic pressure changes are emulated by adjusting the aircraft's trim settings to different flight conditions, thereby assessing the agents' performance under varying aerodynamic loads.

Research Questions - Evaluation

RQ-E-1 How can the performance of the proposed uncertainty-aware RL controller be evaluated with a Cessna Citation II simulation model?

- **RQ-E-1a** Which metrics can be used to compare the reference-tracking accuracy of flight controllers?
- **RQ-E-1b** Which reference-tracking signals should be selected to evaluate the controller?
- **RQ-E-1c** Which specific fault conditions should be considered during the testing phase?

In response to **RQ-E-1a**, this research adopts the normalized Mean Absolute Error (nMAE) for comparing the reference-tracking accuracy of the RUN-DSAC controller against state-of-the-art RL agents. The nMAE metric is employed for its effectiveness in quantifying tracking errors, normalizing these errors relative to the peak amplitudes of the reference signals. In cases where the reference signal is zero, the error normalization is conducted against a standard amplitude of 5°, as inspired by prior research.

Addressing **RQ-E-1b**, the evaluation of the controller's performance involves executing a series of strongly-coupled maneuvers. These maneuvers include climbing and descending high-bank turns. The agents are tested using $80s$ reference signals, featuring predefined cosine-smoothed step sequences in pitch and roll maneuvers, with the sideslip angle reference set consistently to zero. These reference-tracking signals were uniform across all scenarios for consistent comparison, except in the Stall scenario, where the pitch reference is purposefully set to unattainable angle to induces a stall, and the roll reference is set to zero.

This study selects a range of fault conditions based on their relevance and implementation feasibility. These include ice accretion on wings, center of gravity shifts both aft and forward, actuator saturation for ailerons and elevators, reduced elevator effectiveness due to tail damage, and a scenario with an immobilized rudder. This answers **RQ-E-1c**

Research Questions - Evaluation

RQ-E-2 How does the proposed uncertainty-aware RL architecture perform in reference-tracking flight tasks in terms of sample efficiency and accuracy?

RUN-DSAC, particularly its *Conservative* variant, shows significant enhancements in the learning efficiency and stability, surpassing the performance benchmarks set by existing state-of-the-art SAC and DSAC algorithms. This superiority is quantitatively evidenced through higher averaged converged returns and a notable reduction in the variance of learning curves, indicative of superior accuracy and consistency in reference-tracking, respectively. The algorithm's enhanced performance is rigorously validated using a set of five learning performance metrics, where statistical analysis reveals significant differences in four of these metrics. In contrast, the *Risky* RUN-DSAC variant shows slower convergence and higher initial variance in its learning curves compared to DSAC, attributed to its exploratory nature and the reward function design. Nevertheless, it eventually attains stable and crash-free flight behavior.

Research Questions - Evaluation

RQ-E-3 How robust is the selected uncertainty-aware RL flight controller to unforeseen system faults, sensor noise and atmospheric disturbances?

The *Conservative* RUN-DSAC variant consistently outperforms other evaluated agents, including the state-of-the-art SAC and DSAC, in terms of tracking accuracy and stability across the range of challenging simulated scenarios, including various in-flight faults, atmospheric disturbances, and sensor noise. This variant shows notably lower tracking errors and reduced performance variability, thus enhancing the efficiency and safety of flight control. However, it also displays increased sensitivity to changes in aircraft dynamics, particularly in scenarios affecting control surface functionality, leading to marked disturbances in aircraft flight states. On the other hand, the *Risky* RUN-DSAC variant, while showing higher tracking errors and state variability, demonstrates a greater resilience to oscillations induced by altered conditions.

12 Recommendations

This chapter provides a brief overview of the primary recommendations for the future continuation of this research project.

Online Flight Control
This research exclusively focused on offline training of flight control policies. In light of the promising attributes observed in *Conservative* RUN-DSAC, particularly its efficient learning curve and predictive policy behavior, it is recommended to explore its application in online flight control settings. Future research should assess the algorithm's capability to estimate return uncertainty online and examine the impact of incorporating this uncertainty in learning on the agent's real-time adaptability and operational effectiveness. This investigation should focus on its ability to dynamically adjust to changing flight conditions and in-flight emergencies, its robustness to simulation model imperfections, and the balance between safety and exploration. A comparison with offline training results would further elucidate the benefits and limitations of this approach in practical, real-world applications.

6-DOF Flight Control
This study, focusing solely on the safety-critical attitude control and using only a limited flight envelope to train the agents, has identified potential in applying the RUN-DSAC algorithm for the full 6-DOF flight control. The *Conservative* RUN-DSAC's improvements in learning efficiency provide a solid foundation for addressing the curse of dimensionality associated with 6-DOF control. Previous literature has demonstrated the feasibility of training 6-DOF controllers using a hierarchical approach. Hence, a 6-DOF control of aircraft appears achievable through a hierarchy of RUN-DSAC agents, each specializing on specific flight control domains (attitude, altitude, position, airspeed, etc.) and integrating uncertainty assessments within their respective control levels, complemented by exhaustive exploration of the flight envelope.

Flight Test Validation
Demonstrating improved robustness to varying flight conditions, plant and sensor dynamics, and atmospheric perturbations, RUN-DSAC advances RL's viability for real-system flight attitude control. However, the transition from simulations to real-world applications necessitates rigorous flight trials to confirm the algorithms efficacy and dependability. Future endeavors should focus on bridging the simulation-to-reality gap through techniques like domain randomization and robustness training, including accurate models of actuator dynamics, signal transport delays, high-fidelity Dryden gust models, and others. This is especially crucial given *Conservative* RUN-DSAC's propensity for overfitting. Initial trials on scaled-down aircraft models could also pave the way towards the certification of opaque DNN-based autonomous flight systems on real aircraft.

Hyperparameters Tuning
While critical DNN hyperparameters were extensively tuned, others, like the number of quantiles and policy regularization parameters, were selected based on existing literature. Notably, the newly introduced Initial Uncertainty Modulation Factor μ_{init} and Uncertainty Modulation Decay Horizon N_μ in RUN-DSAC were configured to merely contrast the *Risky* and *Conservative* approaches, without rigorous optimization. Optimizing these parameters could potentially improve RUN-DSAC's performance. Additionally, setting the μ_{init} to a high positive value initially for enhanced exploration, and decaying it to a negative value (instead of 0) for increased conservatism with experience, may provide a trade-off between the necessary exploration for robust performance and certainty required for predictability of policies.

Alternative for Uncertainty Estimation
The effectiveness of integrating uncertainty quantification into RL-based flight controllers is highlighted by the RUN-DSAC model, which evaluates the variance in returns by considering state visitation, exploration, and inherent environmental uncertainty. This inspires exploring other approaches for uncertainty estimation in flight control. Although DSAC was preferred in this research, *Bayesian* RL remains an intriguing alternative. Bayesian methods inherently account for uncertainty by treating system parameters as probabilistic entities, continuously updating their distributions based on incoming data. This framework not only provides a detailed understanding of epistemic uncertainty but also enhances the exploration-exploitation balance in learning. Therefore, adopting Bayesian inference in flight control systems could also lead to improved sample efficiency and uncertainty-informed safety.

Exploiting Symmetry for Enhanced Sample Efficiency
The current research has demonstrated the significant enhancement in learning efficiency achieved through incorporating uncertainty information into the actor-critic framework. This enhancement is pivotal in addressing the curse of dimensionality and facilitating prompt adaptation to dynamic shifts in online learning. It is thus proposed to further augment sample efficiency, for example, by exploring and exploiting inherent symmetrical properties in flight control dynamics, such as those seen in roll and yaw maneuvers.

Robustness to Varying Reference Signals
In this study, the RL-based attitude flight controllers were primarily trained and evaluated using cosine-smoothed step sequence signals. To comprehensively assess RUN-DSAC's robustness and adaptability compared to alternatives, future research should evaluate these controllers with various reference signals distinct from training. Suggested signal types for assessment could include, but are not limited to, sinusoids of varying frequencies, triangular signals, and other waveforms.

Extensions to Other Domains
This research, demonstrating RUN-DSAC's effectiveness in flight control, suggests its applicability extends to other domains contending with uncertainty, such as multi-agent systems in advanced Air Traffic Management (ATM). *Conservative* RUN-DSAC's predictability and safety-centric, uncertainty-based approach could be instrumental in managing the complex and uncertain dynamics of multiple aircraft in shared airspace. This method might enable more efficient airspace utilization by safely reducing aircraft separation standards, thereby increasing overall airspace capacity.

References

[1] Brian L. Stevens et al. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. en. John Wiley & Sons, Oct. 2015.

[2] Eugene A. Morelli et al. *Aircraft System Identification: Theory and Practice*. English. 1st edition. Reston, VA: American Institute of Aeronautics & Astronautics, Sept. 2006.

[3] Justus Benad. *The Flying V - A new Aircraft Configuration for Commercial Passenger Transport*. Nov. 2015. doi: `10.25967/370094`.

[4] Alberto Garcia et al. *Aerodynamic Model Identification of the Flying V from Sub-Scale Flight Test Data*. Jan. 2022. doi: `10.2514/6.2022-0713`.

[5] *Wingcopter 198 – Wingcopter*. de-DE. url: `https://wingcopter.com/wingcopter-198` (visited on 05/28/2023).

[6] *Lilium Jet - The First Electric VTOL (eVTOL) Jet - Lilium*. en. url: `https://lilium.com/jet` (visited on 05/28/2023).

[7] Takaaki Matsumoto et al. *A Hovering Control Strategy for a Tail-Sitter VTOL UAV that Increases Stability Against Large Disturbance*. June 2010. doi: `10.1109/ROBOT.2010.5509183`.

[8] Terrence A. Weisshaar. "Morphing Aircraft Systems: Historical Perspectives and Future Challenges". In: *Journal of Aircraft* 50.2 (Mar. 2013), pp. 337–353. doi: `10.2514/1.C031456`. url: `https://arc.aiaa.org/doi/10.2514/1.C031456` (visited on 05/28/2023).

[9] Tigran Mkhoyan et al. "Morphing wing design using integrated and distributed trailing edge morphing". In: *Smart Materials and Structures* 31 (Nov. 2022). doi: `10.1088/1361-665X/aca18b`.

[10] G.C.H.E. De Croon et al. *The DelFly*. en. Dordrecht: Springer Netherlands, 2016. doi: `10.1007/978-94-017-9208-0`. url: `http://link.springer.com/10.1007/978-94-017-9208-0` (visited on 05/28/2023).

[11] *Flapper Drones - Bioinspired flyig robots*. en-US. Publication Title: Flapper Drones. url: `https://flapper-drones.com/wp/` (visited on 05/28/2023).

[12] *Final Report: Accident to Airbus A330-203 registered F-GZCP, Air France AF 447 Rio de Janeiro - Paris, 1st June 2009 \textbar AAIU.ie*. url: `http://www.aaiu.ie/node/687` (visited on 05/30/2023).

[13] P.P. Khargonekar et al. "Robust stabilization of uncertain linear systems: quadratic stabilizability and H/sup infinity / control theory". In: *IEEE Transactions on Automatic Control* 35.3 (Mar. 1990), pp. 356–361. doi: `10.1109/9.50357`.

[14] Dongyan Chen. "Reliable H∞ control for uncertain affine nonlinear systems". In: *Proceedings of the 29th Chinese Control Conference*. July 2010, pp. 1933–1938.

[15] S. Sieberling et al. "Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction". In: *Journal of Guidance, Control, and Dynamics* 33.6 (Nov. 2010), pp. 1732–1742. doi: `10.2514/1.49978`. url: `https://arc.aiaa.org/doi/10.2514/1.49978` (visited on 05/28/2023).

[16] Ryan James Caverly et al. "Nonlinear Dynamic Inversion of a Flexible Aircraft". en. In: *IFAC-PapersOnLine*. 20th IFAC Symposium on Automatic Control in AerospaceACA 2016 49.17 (Jan. 2016), pp. 338–342. doi: `10.1016/j.ifacol.2016.09.058`. url: `https://www.sciencedirect.com/science/article/pii/S2405896316315300` (visited on 05/28/2023).

[17] Yu Li et al. "Angular acceleration estimation-based incremental nonlinear dynamic inversion for robust flight control". en. In: *Control Engineering Practice* 117 (Dec. 2021), p. 104938. doi: `10.1016/j.conengprac.2021.104938`. url: `https://www.sciencedirect.com/science/article/pii/S096706612100215X` (visited on 05/28/2023).

[18] Paul Acquatella et al. *Incremental backstepping for robust nonlinear flight control*. Apr. 2013.

[19] Ewoud J. J. Smeur et al. “Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles”. In: *Journal of Guidance, Control, and Dynamics* 39.3 (Mar. 2016), pp. 450–461. doi: `10.2514/1.G001490`. url: `https://arc.aiaa.org/doi/10.2514/1.G001490` (visited on 05/28/2023).

[20] 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 (Feb. 2021), pp. 116–130. doi: `10.1109/TRO.2020.3010626`.

[21] Rafael A. Cordeiro et al. “Robustness of Incremental Backstepping Flight Controllers: The Boeing 747 Case Study”. In: *IEEE Transactions on Aerospace and Electronic Systems* 57.5 (Oct. 2021), pp. 3492–3505. doi: `10.1109/TAES.2021.3082663`.

[22] Richard S. Sutton et al. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[23] Stefan Heyer. “Reinforcement Learning for Flight Control: Learning to Fly the PH-LAB”. en. In: (2019). url: `https://repository.tudelft.nl/islandora/object/uuid%3Adc63cae7-4289-47c7-889e-253f7abd7c72` (visited on 05/20/2023).

[24] Jun Lee. “Longitudinal Flight Control by Reinforcement Learning: Online Adaptive Critic Design Approach to Altitude Control”. en. In: (2019). url: `https://repository.tudelft.nl/islandora/object/uuid%3Ac1201f27-964c-4257-ad65-89224bef94a1` (visited on 05/20/2023).

[25] Rick Feith. “Safe Reinforcement Learning in Flight Control: Introduction to Safe Incremental Dual Heuristic Programming”. en. In: (2020). url: `https://repository.tudelft.nl/islandora/object/uuid%3A07f53daa-b236-4bb9-b010-bc6654383744` (visited on 05/29/2023).

[26] Giulia Gatti. “Shielded reinforcement learning for flight control”. en. In: (2023). url: `https://repository.tudelft.nl/islandora/object/uuid%3A1c4f10ef-c279-4969-8943-41c46e6b2d8f` (visited on 05/30/2023).

[27] Killian Dally et al. “Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control”. In: *AIAA SCITECH 2022 Forum*. Jan. 2022. doi: `10.2514/6.2022-2078`. url: `http://arxiv.org/abs/2202.09262` (visited on 05/15/2023).

[28] Peter Seres. “Distributional Reinforcement Learning for Flight Control: A risk-sensitive approach to aircraft attitude control using Distributional RL”. en. In: (2022). url: `https://repository.tudelft.nl/islandora/object/uuid%3A6cd3efd1-b755-4b04-8b9b-93f9dabb6108` (visited on 05/30/2023).

[29] Burrhus Frederic Skinner. *The Behavior of Organisms: An Experimental Analysis*. en. D. Appleton-Century Company, incorporated, 1938.

[30] Alan Turing. “Intelligent Machinery, A Heretical Theory (c.1951)”. In: *The Essential Turing*. Ed. by B J Copeland. Oxford University Press, Sept. 2004. doi: `10.1093/oso/9780198250791.003.0018`. url: `https://doi.org/10.1093/oso/9780198250791.003.0018` (visited on 02/23/2023).

[31] Richard Bellman et al. *Dynamic programming*. eng. 1. Princeton Landmarks in Mathematics ed., with a new introduction. Princeton Landmarks in mathematics. Princeton, NJ: Princeton University Press, 2010.

[32] A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM Journal of Research and Development* 3.3 (July 1959), pp. 210–229. doi: `10.1147/rd.33.0210`.

[33] Andrew G. Barto et al. “Neuronlike adaptive elements that can solve difficult learning control problems”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (Sept. 1983), pp. 834–846. doi: `10.1109/TSMC.1983.6313077`.

[34] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. en. In: *Machine Learning* 3.1 (Aug. 1988), pp. 9–44. doi: `10.1007/BF00115009`. url: `https://doi.org/10.1007/BF00115009` (visited on 03/02/2023).

[35] Ian H. Witten. “An adaptive optimal controller for discrete-time Markov environments”. en. In: *Information and Control* 34.4 (Aug. 1977), pp. 286–295. doi: `10.1016/S0019-9958(77)90354-0`.

url: https://www.sciencedirect.com/science/article/pii/S0019995877903540 (visited on 03/01/2023).

[36] MyeongSeop Kim et al. “Adaptive Discount Factor for Deep Reinforcement Learning in Continuing Tasks with Uncertainty”. en. In: *Sensors* 22.19 (Jan. 2022), p. 7266. doi: 10.3390/s22197266. url: https://www.mdpi.com/1424-8220/22/19/7266 (visited on 03/03/2023).

[37] John Schulman et al. *Proximal Policy Optimization Algorithms*. Aug. 2017. doi: 10.48550/arXiv.1707.06347. url: http://arxiv.org/abs/1707.06347 (visited on 03/09/2023).

[38] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. Dec. 2013. doi: 10.48550/arXiv.1312.5602. url: http://arxiv.org/abs/1312.5602 (visited on 03/08/2023).

[39] Lucian Busoniu, ed. *Reinforcement learning and dynamic programming using function approximators*. Automation and control engineering. Boca Raton, FL: CRC Press, 2010.

[40] Pragati Baheti. *The Essential Guide to Neural Network Architectures*. en. July 2021. url: https://www.v7labs.com/blog/neural-network-architectures-guide,%20https://www.v7labs.com/blog/neural-network-architectures-guide (visited on 05/13/2023).

[41] *Hidden Layer*. Publication Title: DeepAI. May 2019. url: https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning (visited on 05/13/2023).

[42] Xavier Glorot et al. “Understanding the difficulty of training deep feedforward neural networks”. en. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256. url: https://proceedings.mlr.press/v9/glorot10a.html (visited on 05/30/2023).

[43] Diederik P. Kingma et al. *Adam: A Method for Stochastic Optimization*. Jan. 2017. doi: 10.48550/arXiv.1412.6980. url: http://arxiv.org/abs/1412.6980 (visited on 05/13/2023).

[44] Kathiravan Srinivasan et al. “An Efficient Implementation of Artificial Neural Networks with K-fold Cross-validation for Process Optimization”. In: *Journal of Internet Technology* 20 (June 2019), pp. 1213–1225. doi: 10.3966/160792642019072004020.

[45] Shipra Saxena. *Introduction to Softmax for Neural Network*. en. Publication Title: Analytics Vidhya. Apr. 2021. url: https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/ (visited on 05/13/2023).

[46] Jiuxiang Gu et al. “Recent advances in convolutional neural networks”. en. In: *Pattern Recognition* 77 (May 2018), pp. 354–377. doi: 10.1016/j.patcog.2017.10.013. url: https://www.sciencedirect.com/science/article/pii/S0031320317304120 (visited on 05/13/2023).

[47] Jason Brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. en-US. Publication Title: MachineLearningMastery.com. Apr. 2019. url: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (visited on 05/13/2023).

[48] Larry Medsker. *Recurrent Neural Networks: Design and Applications*. en. CRC-Press, Feb. 2000.

[49] Jason Brownlee. *A Gentle Introduction to Long Short-Term Memory Networks by the Experts*. en-US. Publication Title: MachineLearningMastery.com. May 2017. url: https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/ (visited on 05/13/2023).

[50] Jimmy Lei Ba et al. *Layer Normalization*. July 2016. doi: 10.48550/arXiv.1607.06450. url: http://arxiv.org/abs/1607.06450 (visited on 05/13/2023).

[51] Antoine Wehenkel et al. *Unconstrained Monotonic Neural Networks*. Mar. 2021. doi: 10.48550/arXiv.1908.05164. url: http://arxiv.org/abs/1908.05164 (visited on 05/31/2023).

[52] Thibaut Théate et al. “Distributional Reinforcement Learning with Unconstrained Monotonic Neural Networks”. In: *Neurocomputing* 534 (May 2023), pp. 199–219. doi: 10.1016/j.neucom.2023.02.049. url: http://arxiv.org/abs/2106.03228 (visited on 04/10/2023).

[53] Hongming Zhang et al. “Taxonomy of Reinforcement Learning Algorithms”. en. In: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Ed. by Hao Dong et al. Singapore:

Springer, 2020, pp. 125–133. doi: 10.1007/978-981-15-4095-0_3. url: https://doi.org/10.1007/978-981-15-4095-0_3 (visited on 05/15/2023).

[54] Mostafa Hussien. "Exploring the Maze of Reinforcement Learning". en-US. In: *Substance ÉTS* (July 2021). url: https://substance.etsmtl.ca/en/exploring-maze-reinforcement-learning (visited on 05/15/2023).

[55] Brett Daley et al. *Improving the Efficiency of Off-Policy Reinforcement Learning by Accounting for Past Decisions*. Dec. 2021. doi: 10.48550/arXiv.2112.12281. url: http://arxiv.org/abs/2112.12281 (visited on 05/15/2023).

[56] *Welcome to Spinning Up in Deep RL!* May 2023. url: https://github.com/openai/spinningup (visited on 05/18/2023).

[57] Ivo Grondman et al. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (Nov. 2012), pp. 1291–1307. doi: 10.1109/TSMCC.2012.2218595.

[58] R. Sutton et al. "Comparing Policy-Gradient Algorithms". In: 2001. url: https://www.semanticscholar.org/paper/Comparing-Policy-Gradient-Algorithms-Sutton-Singh/ (visited on 05/15/2023).

[59] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". en. In: *Machine Learning* 8.3 (May 1992), pp. 229–256. doi: 10.1007/BF00992696. url: https://doi.org/10.1007/BF00992696 (visited on 05/15/2023).

[60] John Schulman et al. *Trust Region Policy Optimization*. Apr. 2017. doi: 10.48550/arXiv.1502.05477. url: http://arxiv.org/abs/1502.05477 (visited on 05/15/2023).

[61] Amir Yazdanbakhsh et al. "TPO: TREE SEARCH POLICY OPTIMIZATION FOR CONTINUOUS ACTION SPACES". en. In: (May 2023). url: https://openreview.net/forum?id=HJew70NYvH (visited on 05/15/2023).

[62] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". en. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. doi: 10.1038/nature14236. url: https://www.nature.com/articles/nature14236 (visited on 03/03/2023).

[63] Marc G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279. doi: 10.1613/jair.3912. url: http://arxiv.org/abs/1207.4708 (visited on 05/16/2023).

[64] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. Oct. 2017. doi: 10.48550/arXiv.1710.02298. url: http://arxiv.org/abs/1710.02298 (visited on 05/16/2023).

[65] Tom Schaul et al. *Prioritized Experience Replay*. Feb. 2016. doi: 10.48550/arXiv.1511.05952. url: http://arxiv.org/abs/1511.05952 (visited on 05/16/2023).

[66] David Silver et al. "Deterministic Policy Gradient Algorithms". en. In: ().

[67] *Welcome to Spinning Up in Deep RL! — Spinning Up documentation*. url: https://spinningup.openai.com/en/latest/index.html (visited on 05/15/2023).

[68] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. July 2019. doi: 10.48550/arXiv.1509.02971. url: http://arxiv.org/abs/1509.02971 (visited on 05/16/2023).

[69] Sebastian Thrun et al. "Issues in Using Function Approximation for Reinforcement Learning". en. In: ().

[70] Scott Fujimoto et al. *Addressing Function Approximation Error in Actor-Critic Methods*. Oct. 2018. doi: 10.48550/arXiv.1802.09477. url: http://arxiv.org/abs/1802.09477 (visited on 05/17/2023).

[71] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. Aug. 2018. doi: 10.48550/arXiv.1801.01290. url: http://arxiv.org/abs/1801.01290 (visited on 05/18/2023).

[72] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. Jan. 2019. doi: 10.48550/arXiv.1812.05905. url: http://arxiv.org/abs/1812.05905 (visited on 05/18/2023).

[73] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Dec. 2019. doi: 10.48550/arXiv.1912.01703. url: http://arxiv.org/abs/1912.01703 (visited on 05/18/2023).

[74] Silvia Ferrari et al. “Online Adaptive Critic Flight Control”. In: *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM* 27 (Sept. 2004), pp. 777–786. doi: 10.2514/1.12597.

[75] Pieter Abbeel et al. “An Application of Reinforcement Learning to Aerobatic Helicopter Flight”. In: *Advances in Neural Information Processing Systems*. Vol. 19. MIT Press, 2006. url: https://papers.nips.cc/paper_files/paper/2006/hash/98c39996bf1543e974747a2549b3107c-Abstract.html (visited on 05/20/2023).

[76] Tommaso Mannucci et al. *SHERPA: a safe exploration algorithm for Reinforcement Learning controllers*. Jan. 2015. doi: 10.2514/6.2015-1757.

[77] Javier García et al. “A comprehensive survey on safe reinforcement learning”. In: *The Journal of Machine Learning Research* 16.1 (Jan. 2015), pp. 1437–1480.

[78] Yizhang Dong et al. “Self-learned suppression of roll oscillations based on model-free reinforcement learning”. en. In: *Aerospace Science and Technology* 116 (Sept. 2021), p. 106850. doi: 10.1016/j.ast.2021.106850. url: https://www.sciencedirect.com/science/article/pii/S1270963821003606 (visited on 05/20/2023).

[79] Vlad Gavra. “Evolutionary Reinforcement Learning: A Hybrid Approach for Safety-informed Intelligent Fault-tolerant Flight Control”. en. In: (2023). url: https://repository.tudelft.nl/islandora/object/uuid%3A46989854-bcf0-4d3d-a6bf-f35cbb559d49 (visited on 05/31/2023).

[80] Tengyang Xie et al. *Policy Finetuning: Bridging Sample-Efficient Offline and Online Reinforcement Learning*. Feb. 2022. doi: 10.48550/arXiv.2106.04895. url: http://arxiv.org/abs/2106.04895 (visited on 05/20/2023).

[81] Casper Teirlinck. “Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance”. en. In: (2022). url: https://repository.tudelft.nl/islandora/object/uuid%3Adae2fdae-50a5-4941-a49f-41c25bea8a85 (visited on 05/31/2023).

[82] Matthias Hutsebaut-Buysse et al. “Hierarchical Reinforcement Learning: A Survey and Open Research Challenges”. en. In: *Machine Learning and Knowledge Extraction* 4.1 (Mar. 2022), pp. 172–221. doi: 10.3390/make4010009. url: https://www.mdpi.com/2504-4990/4/1/9 (visited on 05/19/2023).

[83] Jonathan Hoogvliet. “Hierarchical Reinforcement Learning for Model-Free Flight Control: A sample efficient tabular approach using Q(lambda)-learning and options in a traditional flight control structure”. en. In: (2019). url: https://repository.tudelft.nl/islandora/object/uuid%3Ad66efdb7-d7c7-4c44-9b50-64678ffdf60d (visited on 05/19/2023).

[84] Jiachen Xiong. “Safe Hierarchical Reinforcement Learning for Fixed-Wing Flight Control”. en. In: (2021). url: https://repository.tudelft.nl/islandora/object/uuid%3A2cbd7734-98a3-4260-867d-b828a16c1bbb (visited on 05/19/2023).

[85] André Ferreira Lemos. “Explainable Reinforcement Learning in Flight Control through Reward Decomposition”. en. In: (2022). url: https://repository.tudelft.nl/islandora/object/uuid%3Add7325fa-c615-4df5-8ce3-e3d6ef9cb33c (visited on 05/31/2023).

[86] Gabriel de Haro Pizarroso. “Explainable Artificial Intelligence Techniques for the Analysis of Reinforcement Learning in Non-Linear Flight Regimes”. en. In: (2022). url: https://repository.tudelft.nl/islandora/object/uuid%3Ad278202d-fe0b-4679-8b74-63d3c2f57495 (visited on 05/20/2023).

[87] Lucian Busoniu et al. “Approximate Dynamic Programming and Reinforcement Learning”. In: *Studies in Computational Intelligence*. Vol. 281. Mar. 2010, pp. 3–44. doi: 10.1007/978-3-642-11688-9_1.

[88] J. Si et al. *Handbook of Learning and Approximate Dynamic Programming*. Sept. 2004. doi: 10.1109/9780470544785.

[89] D.V. Prokhorov et al. “Adaptive critic designs”. In: *IEEE Transactions on Neural Networks* 8.5 (Sept. 1997), pp. 997–1007. doi: 10.1109/72.623201.

[90] 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 (Jan. 2021), pp. 142–160. doi: 10.1109/TSMC.2020.3042876.

[91] Chu-I Yang et al. “Explainable uncertainty quantifications for deep learning-based molecular property prediction”. In: *Journal of Cheminformatics* 15.1 (Feb. 2023), p. 13. doi: 10.1186/s13321-023-00682-3. url: https://doi.org/10.1186/s13321-023-00682-3 (visited on 05/06/2023).

[92] Owen Lockwood et al. *A Review of Uncertainty for Deep Reinforcement Learning*. Aug. 2022. doi: 10.48550/arXiv.2208.09052. url: http://arxiv.org/abs/2208.09052 (visited on 05/03/2023).

[93] Prakhar Kaushik et al. *Understanding Catastrophic Forgetting and Remembering in Continual Learning with Optimal Relevance Mapping*. Feb. 2021. doi: 10.48550/arXiv.2102.11343. url: http://arxiv.org/abs/2102.11343 (visited on 05/07/2023).

[94] Jarryd Martin et al. *Count-Based Exploration in Feature Space for Reinforcement Learning*. June 2017. doi: 10.48550/arXiv.1706.08090. url: http://arxiv.org/abs/1706.08090 (visited on 05/04/2023).

[95] Alexander L. Strehl et al. “An analysis of model-based Interval Estimation for Markov Decision Processes”. en. In: *Journal of Computer and System Sciences*. Learning Theory 2005 74.8 (Dec. 2008), pp. 1309–1331. doi: 10.1016/j.jcss.2007.08.009. url: https://www.sciencedirect.com/science/article/pii/S0022000008000767 (visited on 05/04/2023).

[96] Peter Auer et al. “Finite-time Analysis of the Multiarmed Bandit Problem”. In: *Machine Learning* 47 (May 2002), pp. 235–256. doi: 10.1023/A:1013689704352.

[97] Haoran Tang et al. *#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning*. Dec. 2017. doi: 10.48550/arXiv.1611.04717. url: http://arxiv.org/abs/1611.04717 (visited on 05/04/2023).

[98] Marc G. Bellemare et al. *Unifying Count-Based Exploration and Intrinsic Motivation*. Nov. 2016. doi: 10.48550/arXiv.1606.01868. url: http://arxiv.org/abs/1606.01868 (visited on 05/31/2023).

[99] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. url: http://jmlr.org/papers/v15/srivastava14a.html (visited on 05/04/2023).

[100] Yarin Gal et al. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. Oct. 2016. doi: 10.48550/arXiv.1506.02142. url: http://arxiv.org/abs/1506.02142 (visited on 04/26/2023).

[101] Thomas M. Moerland et al. *Efficient exploration with Double Uncertain Value Networks*. Nov. 2017. doi: 10.48550/arXiv.1711.10789. url: http://arxiv.org/abs/1711.10789 (visited on 05/04/2023).

[102] Yue Wu et al. *Uncertainty Weighted Actor-Critic for Offline Reinforcement Learning*. May 2021. doi: 10.48550/arXiv.2105.08140. url: http://arxiv.org/abs/2105.08140 (visited on 05/04/2023).

[103] Ian Osband et al. *Deep Exploration via Bootstrapped DQN*. July 2016. doi: 10.48550/arXiv.1602.04621. url: http://arxiv.org/abs/1602.04621 (visited on 05/03/2023).

[104] Gabriel Kalweit et al. “Uncertainty-driven Imagination for Continuous Deep Reinforcement Learning”. en. In: *Proceedings of the 1st Annual Conference on Robot Learning*. PMLR, Oct. 2017, pp. 195–206. url: https://proceedings.mlr.press/v78/kalweit17a.html (visited on 05/06/2023).

[105] *Reacher - Gym Documentation*. url: https://www.gymlibrary.dev/environments/mujoco/reacher/ (visited on 05/06/2023).

[106] Oren Peer et al. *Ensemble Bootstrapping for Q-Learning*. Apr. 2021. doi: 10.48550/arXiv.2103.00445. url: http://arxiv.org/abs/2103.00445 (visited on 05/31/2023).

[107] Marc G. Bellemare et al. *A Distributional Perspective on Reinforcement Learning*. July 2017. doi: 10.48550/arXiv.1707.06887. url: http://arxiv.org/abs/1707.06887 (visited on 04/08/2023).

[108] Marc G. Bellemare et al. *Distributional Reinforcement Learning*. MIT Press, 2023. url: http://www.distributional-rl.org.

[109] Adam Lowet et al. "Distributional Reinforcement Learning in the Brain". In: *Trends in Neurosciences* 43 (Dec. 2020), pp. 980–997. doi: 10.1016/j.tins.2020.09.004.

[110] Will Dabney et al. *Distributional Reinforcement Learning with Quantile Regression*. Oct. 2017. doi: 10.48550/arXiv.1710.10044. url: http://arxiv.org/abs/1710.10044 (visited on 04/08/2023).

[111] Will Dabney et al. *Implicit Quantile Networks for Distributional Reinforcement Learning*. June 2018. doi: 10.48550/arXiv.1806.06923. url: http://arxiv.org/abs/1806.06923 (visited on 04/08/2023).

[112] Gabriel Barth-Maron et al. *Distributed Distributional Deterministic Policy Gradients*. Apr. 2018. doi: 10.48550/arXiv.1804.08617. url: http://arxiv.org/abs/1804.08617 (visited on 05/26/2023).

[113] Mark Rowland et al. *An Analysis of Categorical Distributional Reinforcement Learning*. Feb. 2018. doi: 10.48550/arXiv.1802.08163. url: http://arxiv.org/abs/1802.08163 (visited on 05/10/2023).

[114] Ke Sun et al. *Interpreting Distributional Reinforcement Learning: A Regularization Perspective*. Sept. 2022. url: http://arxiv.org/abs/2110.03155 (visited on 05/24/2023).

[115] Clare Lyle et al. "A Comparative Analysis of Expected and Distributional Reinforcement Learning". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 4504–4511. doi: 10.1609/aaai.v33i01.33014504. url: https://ojs.aaai.org/index.php/AAAI/article/view/4365 (visited on 05/01/2023).

[116] Xiaoteng Ma et al. *DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning*. June 2020. doi: 10.48550/arXiv.2004.14547. url: http://arxiv.org/abs/2004.14547 (visited on 05/31/2023).

[117] Laurent Valentin Jospin et al. "Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users". In: *IEEE Computational Intelligence Magazine* 17.2 (May 2022), pp. 29–48. doi: 10.1109/MCI.2022.3155327. url: http://arxiv.org/abs/2007.06823 (visited on 05/31/2023).

[118] *SAS Help Center: Prior Distributions*. url: https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.4/statug/statug_introbayes_sect004.htm (visited on 05/31/2023).

[119] Vincent Fortuin et al. *Bayesian Neural Network Priors Revisited*. Mar. 2022. doi: 10.48550/arXiv.2102.06571. url: http://arxiv.org/abs/2102.06571 (visited on 05/25/2023).

[120] Mohammad Ghavamzadeh et al. "Bayesian Reinforcement Learning: A Survey". In: *Foundations and Trends® in Machine Learning* 8.5-6 (2015), pp. 359–483. doi: 10.1561/2200000049. url: http://arxiv.org/abs/1609.04436 (visited on 05/31/2023).

[121] Marco Taboga. *Conjugate prior \textbar Definition, explanation and examples*. Publication Title: StatLect. 2021. url: https://www.statlect.com/fundamentals-of-statistics/conjugate-prior (visited on 04/25/2023).

[122] Richard Dearden et al. "Bayesian Q-learning". In: *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*. AAAI '98/IAAI '98. USA: American Association for Artificial Intelligence, July 1998, pp. 761–768. (Visited on 04/14/2023).

[123] John Asmuth et al. *A Bayesian Sampling Approach to Exploration in Reinforcement Learning*. May 2012. doi: 10.48550/arXiv.1205.2664. url: http://arxiv.org/abs/1205.2664 (visited on 05/07/2023).

[124] Augustine Mavor-Parker et al. "How to stay curious while avoiding noisy tvs using aleatoric uncertainty estimation". In: *International Conference on Machine Learning*. PMLR, 2022, pp. 15220–15240.

[125] Takuya Hiraoka et al. *Dropout Q-Functions for Doubly Efficient Reinforcement Learning*. Mar. 2022. doi: 10.48550/arXiv.2110.02034. url: http://arxiv.org/abs/2110.02034 (visited on 05/09/2023).

[126] Yufeng Zhang et al. *On the Properties of Kullback-Leibler Divergence Between Multivariate Gaussian Distributions*. Jan. 2023. doi: 10.48550/arXiv.2102.05485. url: http://arxiv.org/abs/2102.05485 (visited on 05/02/2023).

[127] Marc G. Bellemare et al. *The Cramer Distance as a Solution to Biased Wasserstein Gradients*. May 2017. doi: 10.48550/arXiv.1705.10743. url: http://arxiv.org/abs/1705.10743 (visited on 05/10/2023).

[128] By James McCaffrey et al. *Wasserstein Distance Using C# and Python* -. en-US. Publication Title: Visual Studio Magazine. url: https://visualstudiomagazine.com/articles/2021/08/16/wasserstein-distance.aspx (visited on 05/31/2023).

[129] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *The Annals of Mathematical Statistics* 35.1 (Mar. 1964), pp. 73–101. doi: 10.1214/aoms/1177703732. url: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-35/issue-1/Robust-Estimation-of-a-Location-Parameter/10.1214/aoms/1177703732.full (visited on 05/11/2023).

[130] Derek Yang et al. *Fully Parameterized Quantile Function for Distributional Reinforcement Learning*. Aug. 2020. doi: 10.48550/arXiv.1911.02140. url: http://arxiv.org/abs/1911.02140 (visited on 05/02/2023).

[131] Cheng Liu et al. *Adaptive Risk-Tendency: Nano Drone Navigation in Cluttered Environments with Distributional Reinforcement Learning*. Sept. 2022. doi: 10.48550/arXiv.2203.14749. url: http://arxiv.org/abs/2203.14749 (visited on 04/12/2023).

[132] Borislav Mavrin et al. *Distributional Reinforcement Learning for Efficient Exploration*. May 2019. doi: 10.48550/arXiv.1905.06125. url: http://arxiv.org/abs/1905.06125 (visited on 04/18/2023).

[133] Marc G. Bellemare et al. “Autonomous navigation of stratospheric balloons using reinforcement learning”. en. In: *Nature* 588.7836 (Dec. 2020), pp. 77–82. doi: 10.1038/s41586-020-2939-8. url: https://www.nature.com/articles/s41586-020-2939-8 (visited on 05/27/2023).

[134] Danial Kamran et al. *Minimizing Safety Interference for Safe and Comfortable Automated Driving with Distributional Reinforcement Learning*. July 2021. doi: 10.48550/arXiv.2107.07316. url: http://arxiv.org/abs/2107.07316 (visited on 05/27/2023).

[135] Jiawei Wang et al. “Robust Dynamic Bus Control: A Distributional Multi-agent Reinforcement Learning Approach”. In: *IEEE Transactions on Intelligent Transportation Systems* 24.4 (Apr. 2023), pp. 4075–4088. doi: 10.1109/TITS.2022.3229527. url: http://arxiv.org/abs/2111.01946 (visited on 05/27/2023).

[136] Xiaoyu Tan et al. “Robot-assisted flexible needle insertion using universal distributional deep reinforcement learning”. en. In: *International Journal of Computer Assisted Radiology and Surgery* 15.2 (Feb. 2020), pp. 341–349. doi: 10.1007/s11548-019-02098-7. url: https://doi.org/10.1007/s11548-019-02098-7 (visited on 05/27/2023).

[137] Jinyoung Choi et al. *Risk-Conditioned Distributional Soft Actor-Critic for Risk-Sensitive Navigation*. Apr. 2021. doi: 10.48550/arXiv.2104.03111. url: http://arxiv.org/abs/2104.03111 (visited on 05/27/2023).

[138] Julian Bernhard et al. “Addressing Inherent Uncertainty: Risk-Sensitive Behavior Generation for Automated Driving using Distributional Reinforcement Learning”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. June 2019, pp. 2148–2155. doi: 10.1109/IVS.2019.8813791.

[139] William R. Thompson. “On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples”. In: *Biometrika* 25.3/4 (1933), pp. 285–294. doi: 10.2307/2332286. url: https://www.jstor.org/stable/2332286 (visited on 04/27/2023).

[140] Emilie Kaufmann et al. “Thompson Sampling: An Asymptotically Optimal Finite-Time Analysis”. en. In: *Algorithmic Learning Theory*. Ed. by Nader H. Bshouty et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 199–213. doi: 10.1007/978-3-642-34106-9_18.

[141] Shipra Agrawal et al. “Optimistic posterior sampling for reinforcement learning: worst-case regret bounds”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran As-

sociates, Inc., 2017. url: https://papers.nips.cc/paper_files/paper/2017/hash/3621f1454cacf995530ea53652ddf8fb-Abstract.html (visited on 04/27/2023).

[142] Siddharth Aravindan et al. *State-Aware Variational Thompson Sampling for Deep Q-Networks*. Feb. 2021. doi: 10.48550/arXiv.2102.03719. url: http://arxiv.org/abs/2102.03719 (visited on 04/18/2023).

[143] Daniel J. Russo et al. “A Tutorial on Thompson Sampling”. English. In: *Foundations and Trends® in Machine Learning* 11.1 (July 2018), pp. 1–96. doi: 10.1561/2200000070. url: https://www.nowpublishers.com/article/Details/MAL-070 (visited on 04/27/2023).

[144] Pascal Poupart et al. “An analytic solution to discrete Bayesian reinforcement learning”. In: *Proceedings of the 23rd international conference on Machine learning*. ICML ’06. New York, NY, USA: Association for Computing Machinery, June 2006, pp. 697–704. doi: 10.1145/1143844.1143932. url: https://doi.org/10.1145/1143844.1143932 (visited on 05/21/2023).

[145] J. Zico Kolter et al. “Near-Bayesian exploration in polynomial time”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. New York, NY, USA: Association for Computing Machinery, June 2009, pp. 513–520. doi: 10.1145/1553374.1553441. url: https://dl.acm.org/doi/10.1145/1553374.1553441 (visited on 05/31/2023).

[146] Raphael Fonteneau et al. *Optimistic Planning for Belief-Augmented Markov Decision Processes*. Apr. 2013. doi: 10.1109/ADPRL.2013.6614992.

[147] Jochen Görtler et al. “A Visual Exploration of Gaussian Processes”. In: Oct. 2018, p. 1. url: https://www.jgoertler.com/visual-exploration-gaussian-processes/ (visited on 05/31/2023).

[148] Marc Peter Deisenroth et al. “Gaussian Processes for Data-Efficient Learning in Robotics and Control”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (Feb. 2015), pp. 408–423. doi: 10.1109/TPAMI.2013.218. url: http://arxiv.org/abs/1502.02860 (visited on 04/25/2023).

[149] Ali Younes et al. “Toward Faster Reinforcement Learning for Robotics: Using Gaussian Processes”. In: *Artificial Intelligence*. Berlin, Heidelberg: Springer-Verlag, Mar. 2022, pp. 160–174. url: https://doi.org/10.1007/978-3-030-33274-7_11 (visited on 05/31/2023).

[150] Carl Edward Rasmussen et al. *Gaussian processes for machine learning*. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 2006.

[151] *Covariance matrix - Encyclopedia of Mathematics*. url: https://encyclopediaofmath.org/wiki/Covariance_matrix (visited on 04/25/2023).

[152] David Duvenaud. *Kernel Cookbook*. url: https://www.cs.toronto.edu/~duvenaud/cookbook/ (visited on 05/31/2023).

[153] Charles Gadd et al. *Sample-efficient reinforcement learning using deep Gaussian processes*. Nov. 2020. doi: 10.48550/arXiv.2011.01226. url: http://arxiv.org/abs/2011.01226 (visited on 05/31/2023).

[154] Haitao Liu et al. *When Gaussian Process Meets Big Data: A Review of Scalable GPs*. Apr. 2019. doi: 10.48550/arXiv.1807.01065. url: http://arxiv.org/abs/1807.01065 (visited on 04/26/2023).

[155] Matthias Bauer et al. *Understanding Probabilistic Sparse Gaussian Process Approximations*. May 2017. url: http://arxiv.org/abs/1606.04820 (visited on 04/26/2023).

[156] *1.7. Gaussian Processes*. en. Publication Title: scikit-learn. url: https://scikit-learn/stable/modules/gaussian_process.html (visited on 04/26/2023).

[157] Adrien Corenflos et al. *Temporal Gaussian Process Regression in Logarithmic Time*. May 2021. doi: 10.48550/arXiv.2102.09964. url: http://arxiv.org/abs/2102.09964 (visited on 05/22/2023).

[158] Akella Ravi Tej et al. *Deep Bayesian Quadrature Policy Optimization*. Dec. 2020. doi: 10.48550/arXiv.2006.15637. url: http://arxiv.org/abs/2006.15637 (visited on 05/22/2023).

[159] Mohammad Ghavamzadeh et al. *Bayesian Policy Gradient Algorithms.* Jan. 2006.

[160] *Bayesian Quadrature*. Publication Title: Emukit. June 2016. url: https://emukit.github.io/bayesian-quadrature/ (visited on 05/22/2023).

[161] Mohammad Ghavamzadeh et al. “Bayesian policy gradient and actor-critic algorithms”. In: 17 (Aug. 2016).

[162] Bahareh Tasdighi et al. *PAC-Bayesian Soft Actor-Critic Learning*. May 2023. url: http://arxiv.org/abs/2301.12776 (visited on 05/31/2023).

[163] Pierre Alquier. *User-friendly introduction to PAC-Bayes bounds*. Mar. 2023. doi: 10.48550/arXiv.2110.11216. url: http://arxiv.org/abs/2110.11216 (visited on 05/22/2023).

[164] Qin Yang et al. *BSAC: Bayesian Strategy Network Based Soft Actor-Critic in Deep Reinforcement Learning*. Aug. 2022. url: http://arxiv.org/abs/2208.06033 (visited on 04/01/2023).

[165] Devin Soni. *Introduction to Bayesian Networks*. en. Publication Title: Medium. July 2019. url: https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e (visited on 05/22/2023).

[166] Erick Delage et al. “Percentile Optimization for Markov Decision Processes with Parameter Uncertainty”. In: *Operations Research* 58.1 (2010), pp. 203–213. url: https://www.jstor.org/stable/40605970 (visited on 05/23/2023).

[167] Katherine Chen et al. “Tractable Objectives for Robust Policy Optimization”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. url: https://papers.nips.cc/paper_files/paper/2012/hash/ce5140df15d046a66883807d18d0264b-Abstract.html (visited on 05/23/2023).

[168] *Maximin criterion*. en. Publication Title: CEOpedia \textbar Management online. url: https://ceopedia.org/index.php/Maximin_criterion (visited on 05/23/2023).

[169] Alex J. Chan et al. *Scalable Bayesian Inverse Reinforcement Learning*. Mar. 2021. doi: 10.48550/arXiv.2102.06483. url: http://arxiv.org/abs/2102.06483 (visited on 05/31/2023).

[170] Jacob Buckman. *A Sober Look at Bayesian Neural Networks*. en. url: https://jacobbuckman.com/2020-01-17-a-sober-look-at-bayesian-neural-networks/ (visited on 05/24/2023).

[171] Ibai Roman et al. “Evolution of Gaussian Process kernels for machine translation post-editing effort estimation”. en. In: *Annals of Mathematics and Artificial Intelligence* 89.8 (Sept. 2021), pp. 835–856. doi: 10.1007/s10472-021-09751-5. url: https://doi.org/10.1007/s10472-021-09751-5 (visited on 05/24/2023).

[172] William T. Stephenson et al. *Measuring the robustness of Gaussian processes to kernel choice*. Mar. 2022. doi: 10.48550/arXiv.2106.06510. url: http://arxiv.org/abs/2106.06510 (visited on 05/24/2023).

[173] Pascal Poupart. *CS885 Lecture 10: Bayesian RL*. 2018. url: https://www.youtube.com/watch?v=ar9RLwgUvVQ (visited on 03/29/2023).

[174] Max Jaderberg et al. *Reinforcement Learning with Unsupervised Auxiliary Tasks*. Nov. 2016. doi: 10.48550/arXiv.1611.05397. url: http://arxiv.org/abs/1611.05397 (visited on 05/25/2023).

[175] Marc G. Bellemare et al. *A Geometric Perspective on Optimal Representations for Reinforcement Learning*. June 2019. doi: 10.48550/arXiv.1901.11530. url: http://arxiv.org/abs/1901.11530 (visited on 05/25/2023).

[176] Davor Runje et al. *Constrained Monotonic Neural Networks*. May 2023. doi: 10.48550/arXiv.2205.11775. url: http://arxiv.org/abs/2205.11775 (visited on 08/25/2023).

[177] J. A. Mulder et al. *Flight Dynamics*. Delft: Delft University of Technology, Mar. 2013.

[178] Siddharth Mysore et al. *Regularizing Action Policies for Smooth Control with Reinforcement Learning*. May 2021. doi: 10.48550/arXiv.2012.06644. url: http://arxiv.org/abs/2012.06644 (visited on 08/22/2023).

[179] Killian Dally. “Deep Reinforcement Learning for Flight Control: Fault-Tolerant Control for the PH-LAB”. en. In: (2021). url: https://repository.tudelft.nl/islandora/object/uuid%3Afcef2325-4c90-4276-8bfc-1e230724c68a (visited on 05/31/2023).

[180] Fabian Grondman et al. “Design and flight testing of incremental nonlinear dynamic inversion based control laws for a passenger aircraft”. en. In: *AIAA Guidance, Navigation, and Control* 210039 (2018). Publisher: American Institute of Aeronautics and Astronautics Inc. (AIAA). doi: 10.2514/6.2018-0385. url: https://repository.tudelft.nl/islandora/object/uuid%3A964e1942-5c83-45e7-8ace-507a33ea3146 (visited on 10/23/2023).

[181] ICAO. *ICAO Safety Management Manual Doc 9859 | SKYbrary Aviation Safety*. 2018. url: https://skybrary.aero/articles/icao-safety-management-manual-doc-9859 (visited on 10/24/2023).

[182] Yongshuai Liu et al. “Policy Learning with Constraints in Model-free Reinforcement Learning: A Survey”. en. In: vol. 5. ISSN: 1045-0823. Aug. 2021, pp. 4508–4515. doi: 10.24963/ijcai.2021/614. url: https://www.ijcai.org/proceedings/2021/614 (visited on 10/22/2023).

[183] Haeun Yoo et al. “A dynamic penalty approach to state constraint handling in deep reinforcement learning”. In: *Journal of Process Control* 115 (July 2022), pp. 157–166. doi: 10.1016/j.jprocont.2022.05.004. url: https://www.sciencedirect.com/science/article/pii/S0959152422000816 (visited on 11/17/2023).

[184] C. a. a. M. van der Linden. “DASMAT - Delft University Aircraft Simulation Model and Analysis Tool: A Matlab/Simulink Environment for Flight Dynamics and Control Analysis”. en. In: *Series 03: Control and Simulation 03* (1998). Publisher: Delft University Press. url: https://repository.tudelft.nl/islandora/object/uuid%3A25767235-c751-437e-8f57-0433be609cc1 (visited on 09/14/2023).

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