Distributional Reinforcement Learning for Flight Control

A risk-sensitive approach to aircraft attitude control using Distributional RL

Peter Seres

TU Delft

Distributional Reinforcement Learning for Flight Control

A risk-sensitive approach to aircraft attitude control using Distributional RL

Thesis report

by

Peter Seres

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on November 9, 2022 at 9:00

Thesis committee:

Chair:	Dr. Coen de Visser
Supervisors:	Dr. Erik-Jan van Kampen
	Cheng Liu
External examiner:	Dr. Erwin Mooij
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	January, 2022 - October, 2022
Student number:	4370716

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

Faculty of Aerospace Engineering · Delft University of Technology

Preface

In the next few decades, the field of reinforcement learning (RL) and other machine learning techniques are expected to solve increasingly complex real-world problems and might prove to be vital to solving engineering, economical, medical and climate-related challenges of the 21st century. The foundation of these algorithms and their implementations are inspired by nature and mimic the way humans learn. The ability of such algorithms to learn without human input is what inspired me throughout the research project, together with the hope that this work makes a contribution towards making RL-based flight control safer and one step closer to real-world application. In order to serve the reproducibility of results obtained during this thesis, and to assist future students and researchers, the code repository has been made publicly available.[1]

This thesis marks the end of my studies at the Delft University of Technology and I want to thank those who made this project possible. First and foremost, thank you to my supervisor, Dr. Erik-Jan van Kampen for his continued support and guidance. Thank you to Cheng Liu for providing his expertise and constantly steering me towards better research and better results. Thank you to my family - especially my dad - for your unconditional support and encouragement.

[1] Code publicly available at: `https://github.com/peter-seres/dsac-flight`

Contents

Nomenclature

List of Abbreviations

ACD	Adaptive Critic Design
ADP	Action-dependent HDP
ADP	Approximate Dynamic Programming
AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CPW	Cumulative Probability Weighting
CVaR	Conditional Value at Risk
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
DHP	Dual Heuristic Dynamic Programming
DNN	Deep Neural Network
DOF	Degree of freedom
DP	Dynamic Programming
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DSAC	Distributional Soft Actor-Critic
EOM	Equations of Motion
FCSD	Flight Control System Design
GDHP	Global DHP
GPI	Generalized Policy Iteration
HDP	Heuristic Dynamic Programming
HRL	Hierarchical Reinforcement Learning
IBS	Incremental Backstepping
IDHP	Incremental DHP
IID	Independent and identically distributed
INDI	Incremental Nonlinear Dynamic Inversion
IQN	Implicit Quantile Network
LOC	Loss of Control
LQR	Linear Quadratic Regulator
MC	Monte Carlo
ML	Machine Learning
MSE	Mean Squared Error
nMAE	Normalized Mean Absolute Error
OBM	On-board Model
PID	Proportional-Integral-Derivative
QR	Quantile regression
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RQ	Research Question
RTV	Right Truncated Variance
RV	Random Variable
SAC	Soft Actor-Critic
SGD	Stochastic Gradient Descent
TCV	Tail Conditional Variance
TD	Temporal Difference
TD3	Twin-delayed DDPG
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
VTOL	Vertical Take-off and Landing

List of Symbols

$\mathcal{A}$	Set of Actions
α	Step-size / Angle of attack

$\mathcal{B}$	Mini-batch of transitions	$\mathcal{S}$	Set of States
β	Distortion function / Angle of sideslip	$\mathcal{T}$	MDP transition
$:=$	By definition	τ	Quantile fraction
$\mathcal{D}$	Memory buffer	θ	Parameter vector / Pitch angle
δ_a	Aileron deflection	$\underline{u}$	Control input vector
δ_e	Elevator deflection	$\underline{\delta}$	Control surface deflections
δ	TD-error	w	Policy parameter vector
$\overset{\mathcal{D}}{=}$	Equality by distribution	$\underline{x}$	Dynamic state vector
δ_r	Rudder deflection	ξ	Risk-distortion parameter
ϵ	Exploration coefficient	ζ	Polyak step-size
η	Entropy Temperature	A	Advantage function
γ	Discount factor	a	Action
$\mathcal{H}$	Entropy	D_{KL}	Kullback-Leibler divergence
k	Parameter vector	G	Discounted Return
k	Value-function parameter vector	J	Objective function
κ	Huber-loss parameter	Q	Action-value function
$\mathcal{L}$	Loss function	q	Pitch rate
λ	Decay-rate	R	Reward function
λ_S	Spatial smoothness loss weight	r	Reward
λ_T	Temporal smoothness loss weight	s	State
$\mathcal{M}$	Markov Decision Process	s'	Next State
μ	Behavioural Policy	t	Time-step
$\mathcal{N}$	Normal distribution	U	Uniform distribution
Ω	Angular velocity	u	Forward velocity component
$\mathcal{P}$	State Transition	V	Value function
ϕ	Roll angle	v	Sideways velocity component
π	Policy	w	Vertical velocity component
Ψ	Distortion risk-measure	W_p	p-Wasserstein metric
ψ	Yaw angle	Z	Value distribution function
$\mathcal{R}$	Set of Rewards	$\mathbb{T}$	Transition tuple $\langle s, a, r, s' \rangle$

List of Figures

List of Tables

1 Introduction

1.1 Autonomous Control Systems for Flight Control

In recent years, technological advancements have resulted in significantly increased complexity in both the design and operations of aerospace systems. These systems include a variety of designs outside of conventional aircraft, such as Unmanned Aerial Vehicles (UAVs), flapping wing systems such as the DelFly [1, 2], aircraft with morphing wings [3, 4], flying-v aircraft [5, 6], convertiplanes [7] and vertical take-off landing (VTOL) systems [8, 9]. Often, these systems have to perform safety-critical multi-objective control tasks without the possibility of human intervention in non-static, partially observable environments. These factors drive the need for an increased level of intelligence and autonomy in the control of aerospace systems, for both low-level control tasks and high-level decision making.

The added complexity of modern aerospace systems and the desire for increased levels of autonomy has posed great challenges for classical flight control system design (FCSD) methods. Traditional control law synthesis requires a pre-determined set of operating conditions and rely on accurate models of the plant dynamics [10], resulting in controllers that are unable to respond to unexpected scenarios. Furthermore, obtaining high-fidelity models of modern aerospace systems is a costly, non-trivial process that involves specialized flight test instrumentation systems, carefully designed experiments, skilled test pilots and a complex process of model parameter estimation and validation [11]. Therefore, there is a need for model-independent controller synthesis, that can produce strategies with high levels of generalization power.

There has been a great deal of research on advanced control techniques to deal with the aforementioned need for adaptability and autonomy. Modern model-based methods rely on high accuracy model representations of aerospace systems. Robust control methods, such as H_∞ control [12] assess the effect of model mismatches and model uncertainties at the cost of degraded control performance [13]. Control methods based on Nonlinear Dynamic Inversion (NDI) rely on an on-board model (OBM) representation of the aircraft dynamics and successfully address most modern flight control design needs, however struggle with control law complexity and robustness to model uncertainties [14, 15]. Sensor-based approaches, such as Incremental Nonlinear Dynamic Inversion (INDI) require a reduced control effectiveness model of the input dynamics, as opposed to a globally accurate model, making the implementation process significantly easier and providing improved robust performance [16]. INDI control laws have demonstrated adaptability [17] and fault-tolerance [18, 19], however they face additional challenges with sensor synchronization and filtering [20].

The last decade has shown great advancements in machine learning (ML) techniques and bio-inspired artificial intelligence (AI). The field of decision making and control is connected to these AI techniques by the field of reinforcement learning (RL), which is a class of methods capable of finding near-optimal strategies to solve complex problems with minimal priori knowledge of the process and the environment. Such model-independent, self-learning algorithms can be beneficial for the aforementioned ever-increasingly complex design of aerospace control systems, and can achieve the increasing requirements of adaptability and autonomy.

1.2 Reinforcement Learning for Aerospace Systems

Reinforcement learning is a class of goal-directed, bio-inspired computational methods that learn using direct interaction with the environment [21]. In an RL task, the agent learns to choose from a set of actions that maximize the amount of expected cumulative reward. For real-world tracking control and robotics tasks, the RL agents learn to choose control actions to manipulate the dynamic states of the system, such as velocity, position, and rotational rates. Traditional RL methods like Q-learning [22] and SARSA [21] had been developed for finite, discrete state-action space problems, where initially lookup-tables were used (tabular methods). Scaling these methods up to real-world control tasks and using high-dimensional observations and actions leads to an exponential growth of the state-action space. This phenomenon is often regarded as the ”curse of dimensionality” [23] and a great deal of RL research is directed at overcoming such limitations.

The ability of RL algorithms to solve real-world tasks has significantly improved since the demonstration of the Deep Q-Network (DQN) algorithm [24, 25], where deep learning (DL), i.e. the use of deep neural networks (DNNs) as universal function approximators was utilized to find optimal control strategies on a discrete action space. The field of Deep Reinforcement Learning (DRL) has demonstrated significant scalability and showed superhuman performance on problems that had previously been thought unsolvable, such as the game of Go [26, 27], which was originally though to require human intuition and creativity. In addition to discrete actions and decision making, the use of DNNs as function approximators in a policy-based and actor-critic algorithms enables the application of DRL to continuous control tasks.

These advancements make RL an attractive alternative to traditional FCSD methods, provided that the aforementioned need for adaptability, fault-tolerance and autonomy can be achieved. There are two primary challenges of applying RL to flight control. Firstly, RL algorithms require a large amount of samples to converge, due to the complexity and dimensionality of the system. Given that flight control tasks are complex, high-dimensional problems, the curse of dimensionality is one of the biggest challenges to overcome to achieve complete 6 degree of freedom (DOF) flight control. Secondly, since flight control is a safety-critical process, there are safety concerns both in the training phase during exploration, and post-training when the agent encounters unexplored states.

It has been shown that sample efficient, online algorithms, such as Increment Dual-Heuristic Programming (IDHP) can achieve fault-tolerant, adaptive control. Such algorithms struggle with generalization power and can only be applied to control tasks with limited dimensionality. Previous work on DRL-based flight control has shown that fault-tolerant control of a jet aircraft can be achieved using state-of-the-art actor-critic methods, such as the soft actor-critic (SAC) algorithm [28]. Even though the robust capability of SAC has been demonstrated for flight control applications, it was found that the training process results in inconsistent outcomes and a low success rate.

A promising class of RL algorithms is the field of distributional RL[1], where the total cumulative rewards are represented by their full probability distribution functions, as opposed to their expectations like in traditional RL [29]. This approach of adding a distributional representation has shown increased sample efficiency and state-of-the-art learning performance [30]. Furthermore, having a representation of the return distribution enables the use of risk-sensitive policies [31], which is an essential aspect of safe flight control applications. Further advancements in distributional RL research have shown the feasibility of risk-sensitive continuous control [32] and have applied the distributional methods to a variety of control tasks, such as the station-keeping control of under-actuated balloons [33], minimally-invasive surgery guidance [34] and robust bus traffic control [35].

[1]The abbreviation DRL is customarily used for Deep Reinforcement Learning, therefore this report refers to the class of distributional methods as distributional RL.

1.3 Research Formulation

As presented above, there is a variety of promising RL methods that have the potential to improve the adaptability and autonomy of modern aerospace systems and to reduce the time and development costs of the traditional FCSD process. The scope of this research project is within investigating the sample efficiency and learning characteristics of RL algorithms applied to flight control tasks, and investigating the ability of RL methods to overcome the challenge of safety.

In order to establish a clear research objective and research questions, a few aforementioned concepts must be defined explicitly. Firstly, the challenge of efficiency relates to the data intensive learning process and the amount of data samples required to train an RL agent. Therefore, an RL method with high *sample efficiency* needs a low amount of data points during training in order to reach a certain level of tracking performance.

When discussing performance, this report distinguishes between learning performance and tracking performance. A good *learning performance* corresponds with not only high sample efficiency, but also with a stable learning process that is robust to hyperparameter changes and partial observability. On the other hand, *tracking performance* is specific to the control tracking task that is required for flight control applications. Good tracking performance describes a controller with low error between the reference and the controlled dynamic state of the plant.

Research Objective

The primary objective of this research project is to improve the *learning performance* of distributional Reinforcement Learning applied to a continuous flight control task and compare its *tracking performance* to state of the art traditional RL methods.

Numerous RL algorithms exist with countless additional extensions that are application specific. It is of primary interest to survey promising state of the art techniques that can be used for solving the flight control RL task, where high-dimensional continuous state-action spaces and the need for safe learning pose additional challenges. Therefore, the first research question (RQ) can be defined as the following:

Research Question 1

What state-of-the-art RL methods are most suitable for flight control tasks?

Section 1.2 mentioned that distributional RL is a promising extension of reinforcement learning that not only comes with learning performance benefits, but also allows the use of risk-sensitive policies. For this research the primary interest is the use of distributional RL for a control task and therefore the following research question can be posed:

Research Question 2

What state-of-the-art distributional RL techniques are the most applicable to flight control tasks?

Distributional RL research has shown that estimating the value distribution increases the sample efficiency and also results in final performance improvements [29]. Applying distributional RL methods to continuous action spaces is essential to study their effectiveness at flight control tasks. Therefore, RQ. 3 can be defined:

Research Question 3

How does the learning performance of distributional RL methods compare to traditional value-based RL methods when applied to flight control tasks?

Due to the dimensionality of the control task, one of the main challenges of applying RL to flight control is achieving 6 DOF control. A simulation gap exists between offline and online RL and improving both classes of methods is required to cross this gap. Improvements are necessary not only in terms of sample efficiency, but also robustness, stability, generalization power and tracking performance. Studying how the performance of distributional RL scales when applied to high-dimensional tasks is of primary interest and therefore RQ. 4 can be defined:

Research Question 4

How does the learning and tracking performance of distributional RL compare to traditional RL methods when applied to high-dimensional flight control tasks?

Since distributional RL methods estimate the entire value distribution, a wide class of control strategies can be defined, which opens the possibility for risk-sensitive policies and control. Such policies have the ability to respond to different uncertainty levels when interacting with the environment.

Research Question 5

How do risk-sensitive distributional RL agents respond to uncertainties in high-dimensional flight control tasks?

Recent developments in distributional RL applications have proposed methods to synthesize risk-averse policies. It is an open question how such risk-sensitive / risk-averse policies can best be exploited to reduce the risk associated with applying RL to flight control tasks.

Research Question 6

How can risk-sensitive agents best be applied to high-dimensional flight control tasks to improve learning performance, tracking performance and safety?

1.4 Structure of the Report

The purpose of this report is to present the findings of the research project and answer the research questions listed above. State-of-the-art traditional and distributional RL methods are discussed and the most suitable algorithms are selected to investigate the relative learning and tracking performance of the RL agents trained in a controlled stochastic environment. This report also includes a literature review, which builds a foundation of RL theory and discusses recent advances in deep RL, distributional RL and flight control applications.

The structure of the report is as follows. Firstly, Chapter 2 presents the scientific article and the primary findings of the research project. Secondly, the preliminary analysis is discussed in Chapters 3 and 4, which present the literature review and preliminary work respectively. Then, additional results are discussed in Chapter 5 about the robustness of the trained agents, followed by a discussion of the verification and validation approach in Chapter 6. Moreover, a brief discussion of the wall clock time of training is given in Chapter 7. Lastly, concluding remarks are given in Chapter 8 followed by recommendations for future research in Chapter 9.

Part I

Scientific Article

Distributional Reinforcement Learning for Flight Control

Peter Seres *

* *Aerospace Engineering, Delft University of Technology, 2629HS Delft (e-mail: peter.seres.ae@gmail.com).*

Abstract:
With the recent increase in the complexity of aerospace systems and autonomous operations, there is a need for an increased level of adaptability and model-free controller synthesis. Such operations require the controller to maintain safety and performance without human intervention in non-static environments with partial observability and uncertainty. Deep Reinforcement Learning (DRL) algorithms have the potential to increase the safety and autonomy of aerospace control systems. It has been shown that the soft actor-critic (SAC) algorithm can achieve robust control of a CS-25 certified aircraft and has the generalization power to react to failure scenarios. Traditional DRL approaches, such as the state-of-the-art SAC algorithm struggle with inconsistent learning in high-dimensional tasks and fall short of modelling uncertainty and risk in the environment. In contrast, distributional RL algorithms estimate the entire probability distribution of rewards, improve the learning characteristics and enable the synthesis of risk-sensitive policies. This paper demonstrates the improved learning characteristics of distributional soft actor-critic (DSAC) compared to traditional SAC and discusses the benefits of risk-sensitive learning applied to flight control. We show that the addition of distributional critics significantly improves learning consistency, and successfully approximates the uncertainty when applied to a fully-coupled attitude control task of a jet aircraft.

Keywords: Reinforcement Learning, Computational Intelligence in Control, Aerospace, Intelligent Autonomous Vehicles

1. INTRODUCTION

In recent years, technological advancements have resulted in significantly increased complexity in the design and dynamics of aerospace systems. Such systems include a variety of designs such as various vertical take-off and landing (VTOL) systems, aircraft with morphing wings (Weisshaar, 2013; Ajaj et al., 2016), flapping wing systems (de Croon et al., 2009, 2016), flying-V aircraft (Faggiano et al., 2017; Ruiz Garcia et al., 2022) and convertiplanes (Saeed et al., 2015). Often these complex systems have to perform multi-objective control tasks while maintaining safety and performance without the possibility of human intervention. Furthermore, state-of-the-art applications of aerospace systems require autonomous operations in challenging, non-static environments, often with partial observability and unforeseen circumstances. These factors drive the need for an increased level of intelligence and autonomy in the control of aerospace systems for both low-level control tasks and high-level decision-making problems.

Traditional approaches to flight control synthesis rely on the costly identification of high-fidelity aerodynamic models (Morelli and Klein, 2016) and a predefined set of operating conditions (Stevens et al., 2015). Such classical control approaches and gain-scheduling architectures reduce the adaptability and robustness of control systems, and are not able to handle unexpected scenarios. Advanced incremental techniques such as incremental non-linear dynamic inversion (INDI) (Sieberling et al., 2010), backstepping control (IBS) (van Gils et al., 2016) reduce modelling requirements, show improved robust performance, and have shown fault-tolerant capability (Sun et al., 2021; Wang and Sun, 2021), but introduce challenges with sensor synchronization and filtering (Pollack and Van Kampen, 2022).

With the rapid advancements in bio-inspired machine learning (ML) techniques, Deep Reinforcement Learning (DRL) methods have shown promising capability to solve large-scale real-world problems in decision making and control. Reinforcement Learning (RL) is a goal-oriented model-free approach to synthesize near-optimal policies for complex systems which has the potential to reduce the model-dependency of flight control system design and to increase the autonomy of aerospace systems. State-of-the-art algorithms, such as the soft actor-critic (SAC) (Haarnoja et al., 2018) have been shown to be capable of fault-tolerant flight control of fully coupled aircraft dynamics while maintaining robustness to varying initial flight conditions (IFC) and sensor noise (Dally and Van Kampen, 2022).

Even though such offline algorithms show great control performance and generalization power, the learning behaviour of these agents is inconsistent and sensitive to hyperparameters. In order to facilitate the eventual application of DRL algorithms on real-world safety-critical systems, it is desired to improve the reliability of these approaches.

Unlike traditional RL methods, distributional RL algorithms (Bellemare et al., 2017; Dabney et al., 2017, 2018) represent the full probability distribution of the reward and achieve improved learning characteristic as a result. Additionally, distributional RL unlocks the training of risk-sensitive policies and the synthesis of risk-averse control laws. Liu et al. (2022) has shown that the use of risk-sensitive distributional RL agents improves the safety of drone navigation in uncertain environments. The efficient exploration approach of Mavrin et al. (2019) demonstrated that distributional value-based methods successfully estimate the intrinsic uncertainty of the environment. The approach proposed by Ma et al. (2020) applies distributional critics to the SAC architecture and enables risk-sensitive learning in continuous action spaces.

The contribution of this paper is three-fold. Firstly, we demonstrate that using a distributional soft actor-critic (DSAC) for a flight control task improves on the sample efficiency, and significantly improves on the consistency and stability of learning. Secondly, we demonstrate that the tracking performance of DSAC is similar to state-of-the-art SAC flight controllers, while modelling the full probability distribution of returns. Lastly, we demonstrate that distributional RL successfully models the uncertainty in the environment, unlocking the synthesis of safer, risk-averse RL-based flight controllers.

The structure of the paper is as follows. Firstly, Section 2 provides a brief formulation of the RL task, the SAC architecture, distributional RL and the flight control environment. Secondly, Section 3 discusses the methodology used to construct risk-sensitive distributional attitude controllers for the validated model of a CS-25 certified research aircraft. Then, Section 4 presents the results conducted on the learning and tracking performance comparison of the traditional and distributional approaches. Lastly, concluding remarks are given in Section 5.

2. BACKGROUND

This section provides a formal definition of RL tasks, introduces maximum entropy RL and distributional RL concepts, and discusses the flight control task formulated under RL.

2.1 Reinforcement Learning

Reinforcement learning is a bio-inspired machine learning technique where an agent learns by interacting with the environment. The sequential decision making task that RL agents solve is formulated as a Markov Decision Process (MDP) described by the structured set $\mathcal{M} \sim \langle \mathcal{S}, \mathcal{A}, R, \mathcal{P}, \gamma \rangle$, with state-space $\mathcal{S} \subset \mathbb{R}^n$, action-space $\mathcal{A} \subset \mathbb{R}^m$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, stochastic state transition dynamics $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$ and discount factor $\gamma \in (0, 1)$. The RL agent chooses action $a_t \in \mathcal{A}$ according to policy $a \sim \pi(a|s)$ at time-step t, and observes the transition tuple $\mathbb{T}_t = \langle s, a, r, s' \rangle$, where r is the immediate reward and s' is the simplified notation for the next-state s_{t+1}. The goal of traditional model-free RL is to find the optimal policy π^* that maximizes the expected return, i.e. the expected cumulative rewards of this sequential decision making task. The action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the expected return of the agent choosing action a in state s and following policy π thereafter, as given by (1):

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

In order to find the optimal policy π^*, RL algorithms repeatedly apply the contractive Bellman operator $\mathcal{T}^\pi$ shown in (2):

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

Where s' and a' denote the next-state and next-action respectively.

The field of deep reinforcement learning (DRL), introduces deep learning concepts to RL and uses deep neural networks (DNNs) as universal function approximators to estimate the action-value function (value-based methods), the policy directly (policy-based methods) or both (actor-critic methods).

Value-based critic-only DRL approaches, such as DQN (Mnih et al., 2015), approximate the action-value function $Q_k(s,a) \approx Q(s,a)$ with parameter vector k and define the policy implicitly using ϵ-greedy action selection, where the action chooses a random action with probability $\epsilon \in [0, 1]$ to facilitate exploration. When the environment requires continuous actions, critic-only methods require an additional optimization step to find the action corresponding to the highest action-value estimate.

In order to handle continuous action spaces, policy-based methods can be used which parameterize the policy directly, such that $\pi_w(s) \approx \pi^*(s)$, where w is the parameter vector of the approximation and π^* is the optimal policy. Policy-based, actor-only methods such as (Silver et al., 2014) require unbiased samples of the return, which in turn results in high-variance, slow convergence and limits the use of such method to episodic environments. State-of-the-art algorithms, such as Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2016) improve on the convergence characteristics of actor-only approaches.

Actor-critic methods combine value-based and policy-based methods in a joint architecture where the critic estimates the action-value function and the actor estimates the optimal policy, combining the advantages of both approaches. Actor-critic methods can handle continuous action spaces due to the direct parameterization of the policy, and the action-value function can be used as a biased estimate of the return, improving the variance of the policy. State-of-the-art actor-critic algorithms, such as Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), Twin-Delayed DDPG (TD3) (Fujimoto et al., 2018) and soft actor-critic (SAC) (Haarnoja et al., 2018)

have been shown to be capable of tackling high-dimensional complex control and robotics tasks.

2.2 Soft Actor-Critic

The state-of-the-art soft actor-critic algorithm (SAC) originally proposed by Haarnoja et al. (2018, 2019) makes use of the *maximum entropy RL* (Ziebart, 2010) framework which introduces an additional entropy term to the objective function to maximize not only the expected cumulative rewards, but to also encourage the diversity of actions.

Maximizing the entropy of the policy $\mathcal{H}_\pi$ results in more efficient exploration, as the entropy serves as a metric of randomness of the stochastic policy over the state space. The entropy is given by the log-likelihood of the policy, as shown in (3), and the resulting soft Bellman operator is shown in (4):

$$\mathcal{H}(\pi_w(\cdot|s)) = \mathbb{E}_{a\sim\pi_w}\left[-\log\left(\pi_w(a|s)\right)\right] \tag{3}$$

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

Where η is the temperature parameter introduced to scale the entropy term $\mathcal{H}_\pi$ and balance the prioritization of future rewards and the information content of the policy.

Since SAC is an off-policy method, the actor and critic networks are trained by sampling a mini-batch $\mathcal{B}$ of transitions from an experience replay buffer $\mathcal{D} = \{\mathbb{T}_0, \mathbb{T}_1, ...\}$. The algorithm proposed by Haarnoja et al. (2019) utilizes a double-critic architecture to prevent the overestimation of the action-value function (Fujimoto et al., 2018). This means that two soft Q-functions are trained in parallel with subscripts $l = 1, 2$: $Q_{k_{1,2}}(s,a)$ and the minimum over the two estimates is taken to determine the temporal-difference (TD) error. In order to stabilize learning, the target Q-networks are fixed relative to the behavioural Q-networks and Polyak averaging (Polyak and Juditsky, 1992) is used with step size ζ to interpolate the parameters of the target-network towards the local network. The parameter vectors of the fixed Q-networks are denoted using $\bar{k}$. The critic loss function minimizes the mean-squared TD-error δ_l given in (5) for both Q-networks $l = 1, 2$:

$$\begin{aligned}\delta_l &= r + \gamma\Big(\min_{l=1,2} Q_{\bar{k}_l}(s',a') - \eta\,\log\,\pi_w(a'|s')\Big) - Q_{k_l}(s,a) \\ \mathcal{L}_Q^{\mathcal{B}}(k_l) &= \mathbb{E}_{\mathcal{B}}\left[\delta_l^2\right]\end{aligned} \tag{5}$$

Where the mini-batch $\mathcal{B}$ of transitions $\{\langle s,a,r,s'\rangle, ...\}$ is sampled from the experience replay buffer $\mathcal{D}$, the next action a' is sampled from the policy $\pi_w(\cdot|s)$, the term $Q_{k_l}(s,a)$ is the local action-value function estimate and $Q_{\bar{k}_l}(s',a')$ is the one-step ahead prediction to calculate the TD-error, which is corrected by the entropy term $\eta\mathcal{H}(\pi_w(a'|s'))$.

SAC uses a stochastic policy as an actor to ensure improved exploration. Namely, the actor is an m-dimensional multivariate Gaussian distribution with a diagonal covariance matrix. The mean vector $\mu_w \in \mathbb{R}^m$ and the covariance diagonal $\sigma_w \in \mathbb{R}^m$ are estimated by a fully-connected feed-forward DNN. The actions sampled from the distribution are passed through a *tanh* squashing function to ensure they are defined on a finite bound. Even though the actor is stochastic during training, the behaviour of the agent is deterministic during evaluation, when the mean μ_w vector is chosen as the action. The action selection of policy $\pi_w(a|s)$ is shown in (6):

$$\begin{aligned}\tilde{a}_w(s) &\sim \mathcal{N}(\mu_w(s), \sigma_w(s)) \\ a_w(s) &= \tanh(\tilde{a}_w(s))\end{aligned} \tag{6}$$

Where in practice an additional reparameterization step is implemented to ensure that the Gaussian policy is differentiable with respect to the parameter vector w for the stochastic gradient descent (SGD) optimization step (Haarnoja et al., 2019). The loss function of the SAC policy can be formulated to maximize not just the return but also the entropy term, as shown in (7):

$$\mathcal{L}_\pi^{\mathcal{B}}(w) = \mathbb{E}_{\mathcal{B}}\left[\eta\,\log\,\pi_w(a_w|s) - \min_{l=1,2} Q_{k_i}(s, a_w(s))\right] \tag{7}$$

An approach proposed by Haarnoja et al. (2019) adapts the temperature parameter η dynamically to achieve a target entropy $\bar{\mathcal{H}}$ in order to improve exploration and reduce the sensitivity to the hyperparameter η. The target entropy is often chosen to be related to the dimension of the action space, such that: $\bar{\mathcal{H}} = -m$. Equation (8) shows the loss function that optimizes the adaptive temperature coefficient:

$$\mathcal{L}^{\mathcal{B}}(\eta) = \mathbb{E}_{\mathcal{B}}\left[\eta\bar{\mathcal{H}} - \eta\,\log\,\pi_w(a|s)\right] \tag{8}$$

The resulting baseline SAC architecture is depicted in Figure (1).

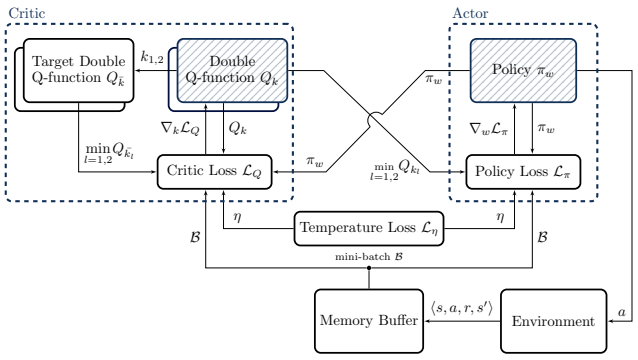


Fig. 1. Soft actor-critic (SAC) architecture with adaptive temperature η optimization and double critic soft Q-networks.

2.3 Distributional RL

While traditional RL aims to maximize the expected cumulative rewards, distributional RL instead gets rid of the expectation and uses information about the entire probability distribution of returns to improve the policy of the agent. This probability distribution of cumulative rewards is called the return distribution function and maps

$Z : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$, where $\mathcal{Z}$ is the action-value distribution space with finite moments for all state-action pairs, as shown in (9) (Bellemare et al., 2023):

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

The action-value function as defined in (1) is the first moment of the return distribution, s.t.: $Q^\pi(s,a) \coloneqq \mathbb{E}\left[Z^\pi(s,a)\right]$ and the random variable Z is the discounted cumulative reward:

$$Z^\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t R(s,a) \tag{10}$$

As shown by Bellemare et al. (2017), the distributional Bellman operator $\mathcal{T}_D^\pi$ can be formulated as given in (11):

$$\mathcal{T}_D^\pi Z(s,a) \overset{\mathcal{D}}{\coloneqq} R(s,a) + \gamma Z(s',a') \tag{11}$$

Where $\overset{\mathcal{D}}{=}$ denotes equality by distribution, i.e. the notion that two random variables are equal when their distributions are equal.

A distributional RL method is primarily defined by two attributes: the probability metric used to measure distances between distributions and the parameterization of the approximate return distribution.

Bellemare et al. (2017) showed that the distributional Bellman operator is a contraction under a p-Wasserstein metric defined in terms of the inverse cumulative distribution function (c.d.f.) of the random return. Given random variable Z, the c.d.f. is defined as $F_Z(z) \coloneqq \mathbb{P}\left[Z < z\right]$ and the quantile function is $F_Z^{-1}(\tau) \coloneqq \inf\left\{z \in \mathbb{R} : \tau \leq F_Z(z)\right\}$, where τ is the quantile fraction. The p-Wasserstein metric is shown in (12) for random variables u and v, and a visualization of the metric is shown in Figure (2) for two example return distribution functions. Hereinafter, the notation $Z_\tau(s,a;k)$ is used to denote the approximate quantile function of the return with parameter vector k.

$$W_p(u,v) = \left(\int_0^1 \left\| F_u^{-1}(\tau) - F_v^{-1}(\tau) \right\|_p d\tau \right)^{1/p} \tag{12}$$

There are several ways to parameterize the return distribution. The C51 algorithm (Bellemare et al., 2017) uses $N = 51$ discrete atoms, whereas QR-DQN (Dabney et al., 2017) uses quantile regression (QR) to approximate the location of a uniform mixture of N diracs. The implicit quantile network (IQN) representation proposed by Dabney et al. (2018) further improves the QR approach by approximating the continuous quantile function implicitly by passing quantile fractions sampled from a uniform distribution $\tau \sim U([0,1])$ through a DNN. Using IQN is a parametrically and computationally efficient way of representing the return distribution.

Figure (3) shows the difference between a Deep Q-Network estimating the scalar expectation and an Implicit Quantile Network estimating the return distribution for randomly sampled quantile fractions.

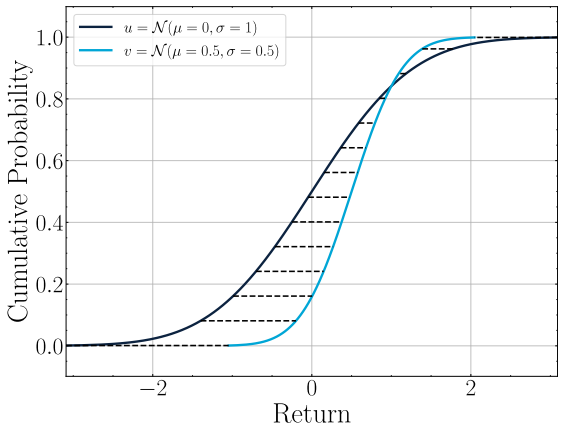


Fig. 2. Adapted from (Bellemare et al., 2023); Illustration of the p-Wasserstein metric between the quantile function of two normal distributions.

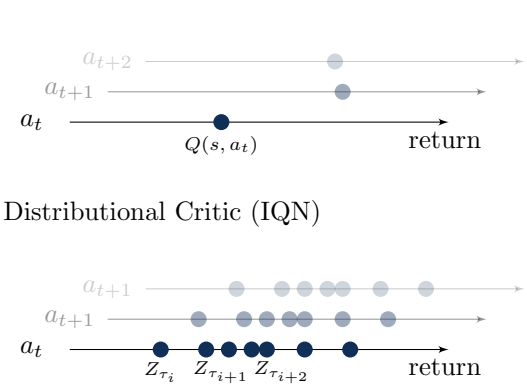


Fig. 3. Traditional and distributional representation of the return for a given state s and choosing action a_t at time-step t; Adapted from Dabney et al. (2018)

2.4 Flight Control as an RL Task

When RL is applied to flight control tasks, the environment often includes partially observable flight dynamics and stochastic processes such as turbulence and sensor noise. It is important to make a distinction between the RL state vector s, which is the observation of the agent, and the dynamic state vector $\underline{x}$, which describes the dynamics of the generalized non-linear non-affine system shown in (13):

$$\dot{\underline{x}} = f(\underline{x}, \underline{u}, t) \approx f(\underline{x}, \underline{u}) \tag{13}$$

Where f is the non-linear state transition function, $\underline{x} \in \mathbb{R}^{n'}$ is the dynamic state vector of the aircraft and $\underline{u} \in \mathbb{R}^{m'}$ is the control input vector. Equation (13) makes the assumption that the aircraft dynamics are stationary on short time scales.

A *tracking control task* requires the agent to track desired trajectories and minimize the tracking error between reference signal and certain controlled states of the dynamic system. To formulate a tracking control task as an MDP, a

subset of the dynamic state vector $\underline{x}$ has to be augmented with the reference signals y_r. The reward function is often defined as a penalty proportional to either the absolute tracking error $R_a \propto \|y_r - \underline{x}_c\|_1$ or the squared tracking error $R_s \propto \|y_r - \underline{x}_c\|_2$, where y_r is the reference trajectory and $x_c \subset \underline{x}$ is the vector of controlled states.

A major assumption of traditional RL is that the MDP $\mathcal{M} \sim \langle \mathcal{S}, \mathcal{A}, R, \mathcal{P}, \gamma \rangle$ has the Markov-property, i.e. the observation vector $s \in \mathcal{S}$ fully predicts the outcome of the transition $\mathcal{P}(s'|a, s)$. Flight control tasks on the other hand often contain intrinsic uncertainty, either due to stochastic processes such as turbulence and sensor noise, or due to the influence of unobservable states on the dynamics of the aircraft.

There are two primary challenges when it comes to applying RL for flight control tasks. Firstly, the large dimensionality of the state-action space demands algorithms to efficiently use transition samples and be able to generalize learned behaviour across the flight envelope. Secondly, since flight control is a safety-critical task, exploration poses a challenge of safety both during the training phase and the post-training state when the RL agent encounters unexplored states (Pollack and Van Kampen, 2019).

Due to the challenges of safety and sample efficiency, there exists a simulation gap between RL agents trained in an offline simulated environment and the online, real-world application of RL-based flight controllers. In order to reduce dependence on a high-fidelity model of the system, it is desirable to synthesize robust control laws that are capable of handling uncertainty in the environment.

DRL methods have the generalization power to achieve robust fault-tolerant control in high-dimensional environments (Dally and Van Kampen, 2022), although may present safety concerns when trained online due to the continually changing policy. Teirlinck and Van Kampen (2022) showed how a hybrid approach of SAC (Haarnoja et al., 2018, 2019) and Incremental Approximate Dynamic Programming (IADP) methods, such as IDHP (Zhou et al., 2018) can cross the gap between offline, simulation-trained methods and online, adaptive methods. However, they observed inconsistent convergence of the offline training of SAC agents with high sensitivity to hyperparameters and stochastic processes.

3. METHODOLOGY

This section discusses the risk-sensitive distributional soft actor-critic algorithm, the aircraft control task considered for this research and the implementation details.

3.1 Distributional Soft Actor-Critic (DSAC)

In order to use distributional RL for continuous control tasks, it is necessary to parameterize the policy directly and resort to policy-based or actor-critic approaches. A distributional actor-critic framework proposed by Ma et al. (2020) extends the SAC algorithm to distributional RL architecture by using critics that approximate the return distribution. This combination of maximum entropy RL and distributional RL facilitates not only effective exploration, but also allows the use of risk-sensitive policies.

The distributional soft actor-critic (DSAC) algorithm adopts the quantile regression approach (Dabney et al., 2017) and uses the quantile Huber loss (Huber, 1992) as a substitute for the Wasserstein metric. The pairwise TD-error between two quantile fractions τ_i and τ_j is given by (14):

$$\begin{aligned}\delta_{ij}^{l} = r + \gamma \left(\min_{l=1,2} Z_{\tau_i}(s', a'; \bar{k}_l) - \eta \ \mathsf{log} \ \pi_w(a'|s') \right) \\ - Z_{\tau_j}(s, a; k_l)\end{aligned} \tag{14}$$

Where the quantile fractions are sampled independently $\tau_i, \tau_j \sim U([0, 1])$, $a' \sim \pi_w(\cdot|s')$, and the subscript $l = 1, 2$ denotes each Z-network in the double-critic architecture. The Huber loss for quantile fraction τ is given by (15):

$$\begin{aligned}\rho_\tau^\kappa(\delta) &= |\tau - \mathbb{I}\{\delta < 0\}| \cdot \mathcal{L}^\kappa(\delta), \text{with} \\ \mathcal{L}^\kappa(\delta) &= \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq \kappa \\ \kappa \left(|\delta| - \frac{1}{2}\kappa \right) & \text{otherwise} \end{cases}\end{aligned} \tag{15}$$

Where $\mathbb{I}$ is the indicator function, and the Huber loss $\mathcal{L}^\kappa(\delta)$ provides smooth gradient-clipping with threshold parameter κ.

In order to estimate approximate quantile loss for all quantile levels, a set of N independent quantile fractions is sampled for both the target and local networks. Thus, the critic loss function for the return distribution estimator $Z_\tau(s, a; k)$ can be written as:

$$\mathcal{L}_Z^{\mathcal{B}}(k_l) = \mathbb{E}_{\mathcal{B}} \left[\frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \rho_{\tau_j}^\kappa(\delta_{ij}^l) \right] \tag{16}$$

Adapting the approach from (Dabney et al., 2018), risk-sensitive learning can be achieved by maximizing a distorted expectation of the soft action-value distribution. Let $\Psi : [0, 1] \to [0, 1]$ be a continuous monotonic distortion function under which the distorted expectation is given by:

$$Q_k^\Psi(s, a) = \mathbb{E}_\tau \left[Z_{\Psi(\tau)}(s, \pi_w(\cdot|s); k) \right] \tag{17}$$

Where Ψ acts as a *distortion risk measure*. The Wang risk-distortion function (Wang, 2000) provides a straightforward way to parameterize the risk-distortion towards risk-averse or risk-seeking learning, and is shown in (18):

$$\text{Wang}(\xi, \tau) = \Phi(\Phi^{-1}(\tau) + \xi) \tag{18}$$

Where Φ is the c.d.f. of the normal distribution and ξ is the risk-distortion parameter that determines the learning behaviour, such that $\xi < 0$ results in risk-averse, and $\xi > 0$ results in risk-seeking learning. For the synthesis of risk-neutral policies the risk-distortion function is the identity mapping. Figure (4) shows the difference in the risk-distorted expectation of two random variables with different variances. A value distribution with smaller uncertainty under risk-averse distortion results in a higher relative expected reward.

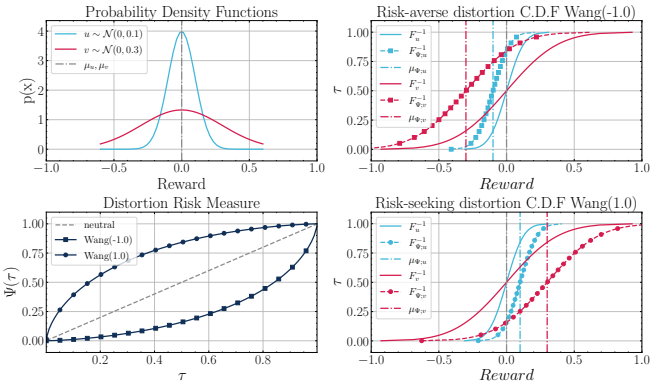


Fig. 4. Expectations under risk-averse $\xi < 0$ and risk-seeking $\xi > 0$ distortions for two normal distributions with the same mean and different variances.

Figure (5) shows the DSAC architecture where the critic networks are distributional Z-function approximators, and an additional risk-distortion step is introduced in the policy loss function to train risk-sensitive agents.

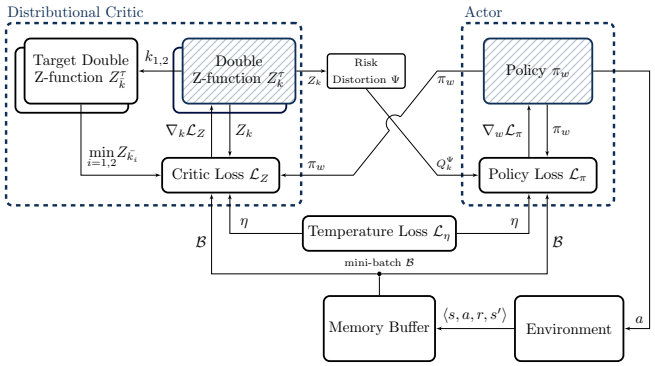


Fig. 5. Distributional Soft Actor-Critic (DSAC) architecture with adaptive temperature and risk-sensitive learning.

3.2 Policy regularization

A common phenomenon of converged DRL policies is the lack of smoothness of control actions, especially in continuous control and robotics settings. The oscillatory behaviour and high-frequency content in the actions causes undesirable behaviour and reduces the practical utility of synthesized control laws (Mysore et al., 2021). In the control of real robotics and aerospace systems, such oscillations may degrade the tracking performance, cause overheating, high power usage, and actuator failure.

In order to avoid such oscillatory converged policies, Dally and Van Kampen (2022) used an incremental control architecture, where the action of the RL agent is an incremental change of control action. This approach further augments the observation space with the current actuator state and increases the dimensionality of the problem. Early experiments in this research showed that the incremental approach does not guarantee control law smoothness and therefore this paper uses a policy regularization technique proposed by Mysore et al. (2021).

The approach of Conditioning for Action Policy Smoothness (CAPS) adds two regularizing terms to the policy loss function: a spatial and a temporal loss term. The spatial loss term $\mathcal{L}_S$ given by (19) ensures that similar states result in similar actions and the temporal term $\mathcal{L}_T$ given by (20) ensures that two neighbouring time-steps produce similar actions. Two additional hyperparameters λ_S, λ_T tune the prevalence of the smoothness regularization term in (21).

$$\mathcal{L}_S = \|\pi_w(s) - \pi_w(\tilde{s}\ )\|_2 \tag{19}$$
$$\mathcal{L}_T = \|\pi_w(s) - \pi_w(s')\|_2 \tag{20}$$
$$\mathcal{L}_\pi^{\mathrm{C}} = \lambda_S \mathcal{L}_S + \lambda_T \mathcal{L}_T \tag{21}$$

Where the transition tuple is sampled from the mini-batch $\langle s, a, r, s' \rangle \sim \mathcal{B}$, and the proximal states $\tilde{s}$ are sampled from a normal distribution $\tilde{s} \sim \mathcal{N}(s, \tilde{\sigma})$.

Thus, the final objective function of the policy maximizes the entropy term $\eta\mathcal{H}_\pi$ to facilitate diverse actions, maximizes the risk-distorted soft action-value Q_k^Ψ to facilitate risk-sensitive action selection, and ensures smooth, practical control laws using the regularization term $\mathcal{L}_\pi^{\mathrm{C}}$. The combined policy objective is formulated as a loss in (22):

$$\mathcal{L}_\pi^{\mathcal{B}}(w) = \mathbb{E}_{\mathcal{B}}\Big[\eta\ \log\ \pi_w(a_w(s)|s) - Q_k^\Psi(s,a) + \mathcal{L}_\pi^{\mathrm{C}}\Big] \tag{22}$$

3.3 Attitude Control

The tracking task investigated in this paper is the attitude control of a validated high-fidelity model of the PH-LAB Cessna Citation II research aircraft with fully-coupled nonlinear dynamics (Van den Hoek et al., 2018). The attitude control task is to track pitch θ_r and roll angle ϕ_r reference trajectories, and to regulate the sideslip angle to $\beta_r = 0$.

3.3.1 Aircraft Model

The model has 12 dynamic states visualized in Figure (6), which include the airspeed components: true airspeed V, angle of attack α and angle of sideslip β, the angular velocities p, q and r for roll, pitch and yaw rate respectively, and the angular orientation of the aircraft is represented using Euler-angles: ϕ, θ and ψ for roll, pitch and yaw (heading), respectively. Additionally, the translational positions are given with respect to a local tangent plane reference frame: X_e and Y_e for the horizontal position and altitude h for the vertical position.

Similarly to the methodology of (Dally and Van Kampen, 2022), the actions of the agent are limited to aerodynamic surface deflections and the thrust control is delegated to an inner control loop that regulates velocity, thus reducing both the size of the state-action space and the overall control complexity. The available control surfaces are the elevator deflection δ_e, the aileron deflection δ_a and the rudder deflection δ_r. This approach results in the following dynamic state vector $\underline{x} \in \mathbb{R}^{12}$ given by (23) and control input vector $\underline{u} \in \mathbb{R}^3$ given by (24). The dynamic states and control inputs of the aircraft are shown in Figure (6).

$$\underline{x} = [p, q, r, V, \alpha, \beta, \phi, \theta, \psi, h, X_e, Y_e]^T \tag{23}$$
$$\underline{u} = [\delta_e, \delta_a, \delta_r]^T \tag{24}$$

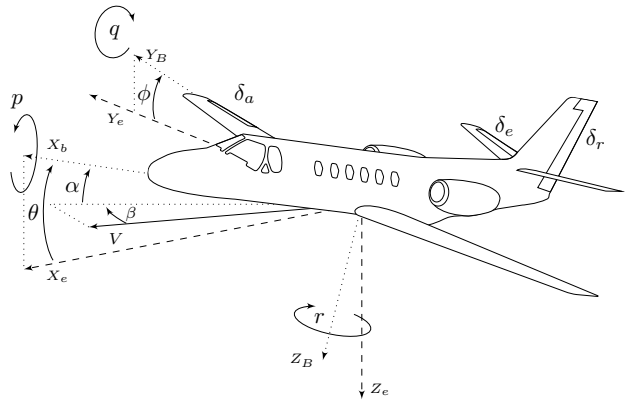


Fig. 6. Visualization of the dynamic states of the aircraft, where X_b, Y_b and Z_b define the body frame with attitude $[\phi, \theta, \psi]$ relative to the inertial frame. The heading ψ is depicted as zero for visual clarity.

At the start of the simulation, the aircraft is initialized from a trimmed condition at an altitude of $h = 2,000$ (m) and airspeed of $V = 90$ (m/s). The refresh rate of the simulation is 100 (Hz). Additionally, ideal sensors are assumed and the actuators are modeled using low-pass filter dynamics with fixed deflection saturation limits. The actuator deflection limits are given by (25) (Konatala et al., 2021) and define the action space of the control policy:

$$\mathcal{A} = \{[-17°, 15°] \times [-19°, 15°] \times [-22°, 22°]\} \subset \mathbb{R}^3 \quad (25)$$

($\longleftarrow\delta_e\longrightarrow$ $\longleftarrow\delta_a\longrightarrow$ $\longleftarrow\delta_r\longrightarrow$)

3.3.2 Controller Architecture

Previous studies by Dally and Van Kampen (2022) and Teirlinck and Van Kampen (2022) have shown the robust fault-tolerant capability of the SAC architecture using a cascaded hierarchy of RL agents to control the attitude and altitude of the PH-LAB aircraft model.

In order to investigate the isolated effect of distributional RL and risk-averse learning, this paper only considers the safety-critical inner-loop attitude control task without the additional complexity of a multi-agent control system. Figure (7) shows the control diagram considered in this research, where the RL agent controls the actuator deflections directly and the observation vector is varied to investigate the effect of dimensionality and observability.

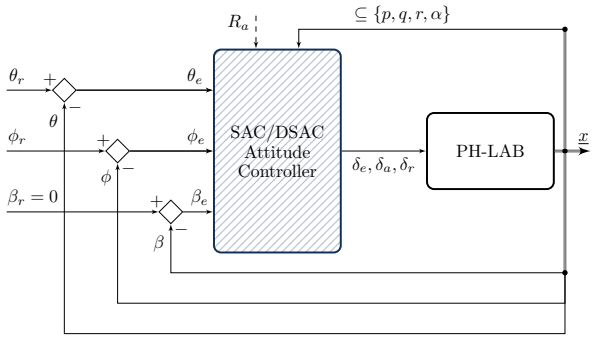


Fig. 7. SAC/DSAC attitude controller architecture with direct deflection control $a = \underline{u} = [\delta_e, \delta_a, \delta_r]^T$.

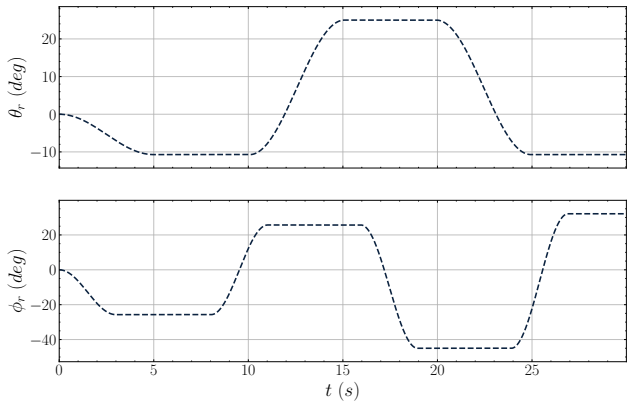


Fig. 8. Pitch θ_r and roll ϕ_r reference signals for training are randomly generated sequences of cosine smoothed steps, shown for a single episode.

The baseline observation vector s_1 is given in (26) and was used by Dally and Van Kampen (2022) to synthesize fault-tolerant SAC attitude controllers. This paper investigates the use of an augmented vector s_2 given in (27), which adds angle of attack to the observation of the agent with the purpose of reducing the partial observability of the MDP.

$$s_1 = [\theta_e, \phi_e, \beta_e, p, q, r]^T \in \mathbb{R}^6 \quad (26)$$

$$s_2 = [\theta_e, \phi_e, \beta_e, p, q, r, \alpha]^T \in \mathbb{R}^7 \quad (27)$$

Where θ_e, ϕ_e, and β_e are the pitch, roll and sideslip tracking errors respectively. The tracking error vector is given by (28):

$$e = [\theta_e, \phi_e, \beta_e]^T = [\theta_r - \theta, \phi_r - \phi, \beta_r - \beta]^T \quad (28)$$

The reward function used by Dally and Van Kampen (2022) is adopted to penalize the weighted L1 norm of the tracking error:

$$R(s, a) = -\frac{1}{3}\left\| \text{clip}\left[c \odot error\right], -1, 0\right] \right\|_1 \quad (29)$$

Where $c \in \mathbb{R}^3$ is the associated relative cost of the controlled states, set higher for the sideslip angle due to its lower magnitude, as seen in (30):

$$c = \frac{6}{\pi}[1, 1, 4]^T \quad (30)$$

The agents are trained using 30 (s) episodes with randomly generated pitch θ_r and roll ϕ_r reference signals. The reference signals are sequences of cosine-smoothed step inputs with amplitudes that are uniformly sampled from a set of discrete levels. The amplitudes and step widths are chosen based on previous work (Dally and Van Kampen, 2022). Figure (8) shows an example realization of the training signals.

3.4 Critic and Actor Networks

As described in (3.1), the distributional critic is an implicit quantile network which estimates the continuous quantile function of the return. In practice, N randomly sampled

quantile fractions $\tau \sim U([0,1])$ are passed through a cosine embedding layer, given in (31):

$$C_j(\tau) \coloneqq \mathbb{F}\left(\sum_{i=1}^{N} cos(\pi i \tau) w_{ij} + b_j\right) \tag{31}$$

Where $\mathbb{F}$ is a nonlinear activation function, and w_{ij}, b_j are the individual weight and bias parameters of the cosine embedding layer.

Interaction between the input features, i.e. the state-action pair $[s,a]^T \in \mathbb{R}^{n+m}$ and the sample embedding is achieved using the multiplicative Hadamard product, as outlined by Dabney et al. (2018), and layer normalization is utilized Ba et al. (2016) to ensure well-bounded quantile magnitudes. Figure (9) shows the DNN architecture of the Z-function approximator networks.

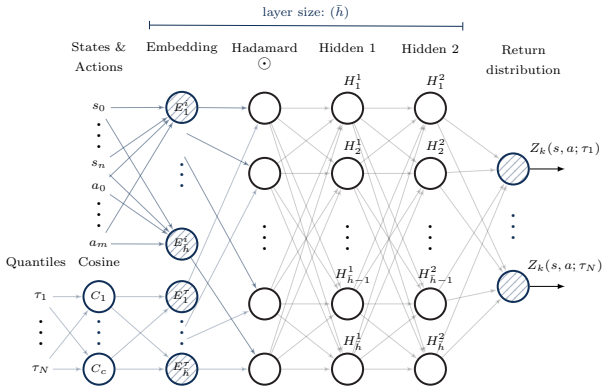


Fig. 9. Depiction of the implicit quantile network (IQN) based on Dabney et al. (2018), with 2 hidden fully-connected feed-forward layers of size $\bar{h}$, cosine embedding layer size C and N generated quantiles fractions.

In order to optimize the parameters of the critic, stochastic gradient descent (SGD) optimization is used (Kingma and Ba, 2017) with the quantile Huber loss function defined in (16).

Figure (10) shows the multivariate Gaussian network used to approximate the policy of SAC agents. In practice, the DNN estimates the $\log(\sigma)$ terms for more efficient computation of the entropy term defined in (3). In order to compute back-propagation, the required reparameterization step (`rsample`) is provided by popular machine learning frameworks. The *tanh* squashing function is applied to the sampled actions as described by (6) and the mean μ is taken for deterministic action selection in post-training evaluations.

3.5 Experiment Design

In order to assess the effect of distributional RL, the performance of DSAC agents is compared to baseline SAC agents. For the purpose of assessing the risk-sensitive capabilities of the DSAC algorithm, risk-neutral and risk-averse agents are trained using the Wang distortion risk measure, shown in (18) and the risk-distorted action-value (17). The comparison of the three agents, i.e. baseline SAC, risk-neutral (R.N.) DSAC and risk-averse (R.A.) DSAC is done for two environments: the baseline environment with

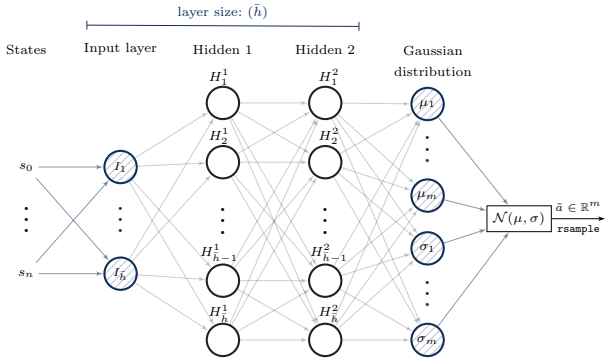


Fig. 10. Depiction of the multivariate *tanh*-Gaussian policy.

observation vector s_1 (26), and the environment with the α-augmented observation vector s_2 (27).

Both observation vectors result in partially observable environments, as the angle of attack α has a great effect on the longitudinal dynamics of the aircraft, whereas the airspeed V and altitude h influence all dynamic modes of the aircraft, due to the different dynamic pressure and Mach number. This partial observability appears as an intrinsic uncertainty to an agent with limited knowledge of all dynamic states and the distributional critic estimates this uncertainty (Mavrin et al., 2019).

The performance of the algorithms is assessed with respect to two aspects. Firstly, the *learning performance* of the algorithms is evaluated in terms of sample efficiency, final converged rewards and consistency. Secondly, the *tracking performance* of the agents is assessed, with the use of a normalized mean absolute error (nMAE) metric, where the θ_e, ϕ_e tracking errors are normalized with respect to the maximum amplitude of training signals and β_e is normalized in $[-5°, 5°]$, similarly to the methodology of Teirlinck and Van Kampen (2022).

Additionally, risk-neutral and risk-averse agents trained using different ξ risk-parameters are compared with respect to their ability to model uncertainty and handle high-risk manoeuvres.

The hyperparameters used for the experiments are shown in Table (1), a subset of which are shared across all agent types. The linearly decreasing learning rate, discount factor γ, network size, batch size $|\mathcal{B}|$, and Polyak step-size ζ parameters have been set to the baseline SAC attitude controller investigated by Dally and Van Kampen (2022). The size of the memory buffer $|\mathcal{D}|$ is increased to 1e6 in order to stabilize learning and discourage catastrophic forgetting behaviour. The CAPS smoothing parameters $\lambda_{S,T}$ are set to 400 based on early experiments by Teirlinck and Van Kampen (2022).

The parameters of the IQN critics are chosen according to the findings of Dabney et al. (2018), who showed that $N = 8$ quantiles are sufficient to estimate the return distribution. Ma et al. (2020) found that using the `Sigmoid` activation function achieves smoother embedding of the quantile information compared to the original `ReLU` activation. The distortion risk measure is set to identity for the risk-neutral agents and is set to $\xi = -0.5$ for the risk-averse agents.

Table 1. Hyperparameters for the SAC and DSAC agents, with shared parameter subsets.

Hyperparameter	Notation	Value
Shared		
Optimizer		`Adam`
Learning rate		4.4e-4 $\rightarrow$ 0
Entropy target	$\bar{\mathcal{H}}$	$-m = -3$
Discount factor	γ	0.99
Dense network activation		`ReLU`
Hidden neurons	$\bar{h}$	64×64
Memory buffer size	$\lvert\mathcal{D}\rvert$	$1,000,000$
Mini-batch size	$\lvert\mathcal{B}\rvert$	256
Polyak step-size	ζ	0.995
Policy-smoothing	$\lambda_{S,T}$	400
Distributional critic		
Nr. of quantiles	N	8
Nr. of cosine neurons	C	64
Embedding activation	$\mathbb{F}$	`Sigmoid`
Nr. of quantiles for Q_k^Ψ	T	16
Huber threshold	κ	1.0
Risk-averse learning		
Distortion risk measure	Ψ	Wang
Risk parameter	ξ	-0.5

In order to ensure reproducibility, the pseudo-random stochasticity during training is controlled for both environment and agent.

4. RESULTS AND DISCUSSION

This section presents the results of the baseline (SAC) and distributional (DSAC) control law synthesis for both risk-neutral and risk-averse settings.

4.1 Learning performance

The number of training samples required to train SAC agents for flight control is on the order of 1e6 transitions (Dally and Van Kampen, 2022). The agents are trained for 7.5e5 samples, i.e. 250 episodes with each episode lasting 30 (s).

The learning curves are shown in Figures (11, 12). Due to the early convergence, the first 2e5 samples are shown and Savitzky-Golay smoothing is applied for visual clarity. It can be seen that for the baseline environment the distributional agents achieve earlier convergence, demonstrating an increase in sample efficiency of 56.3% for the risk-neutral, and 43.7% for the risk-averse distributional agents. The improvement in sample efficiency is slightly lower for the augmented observation vector with 33.9% for the risk-neutral and 28.5% for the risk-averse agents. The difference in convergence between the two environments indicates that the early relative learning performance of distributional agents is improved for environments with more intrinsic uncertainty.

Furthermore, the sample efficiency of all three types of agents have increased by approximately 95% relative to the baseline SAC approach by Dally and Van Kampen (2022), which required 1e6 samples to converge. This significant improvement can be attributed to the omission of the incremental control approach, which reduces the size of the observation space from $\mathbb{R}^9$ to $\mathbb{R}^{6,7}$ and the addition of policy regularization, which eliminates the convergence to local-optima that result in oscillatory control laws.

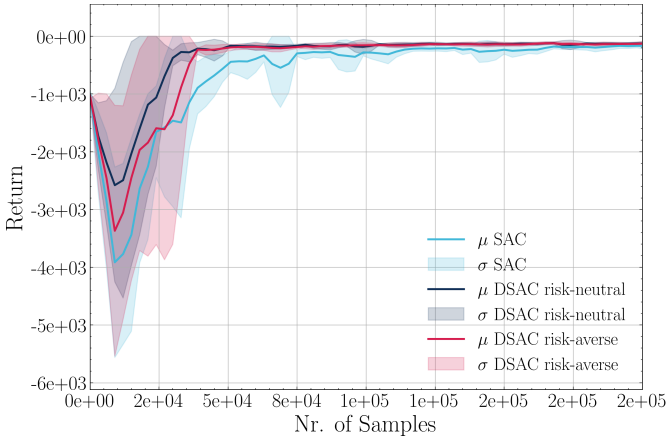


Fig. 11. Mean learning curve and standard deviation (n=10) for each agent type trained in the baseline environment (s_1).

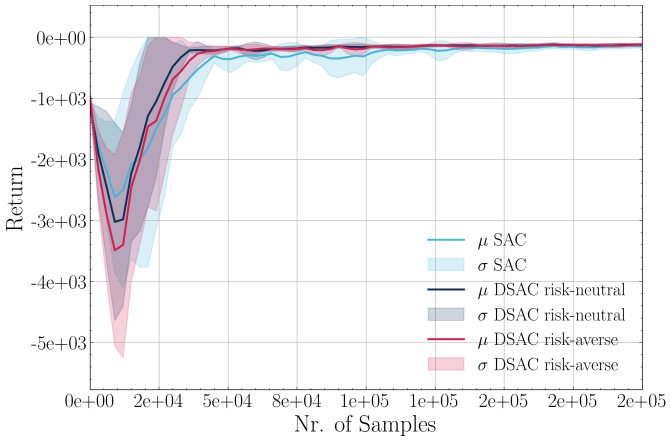


Fig. 12. Mean learning curve and standard deviation (n=10) for each agent type trained in the augmented environment (s_2).

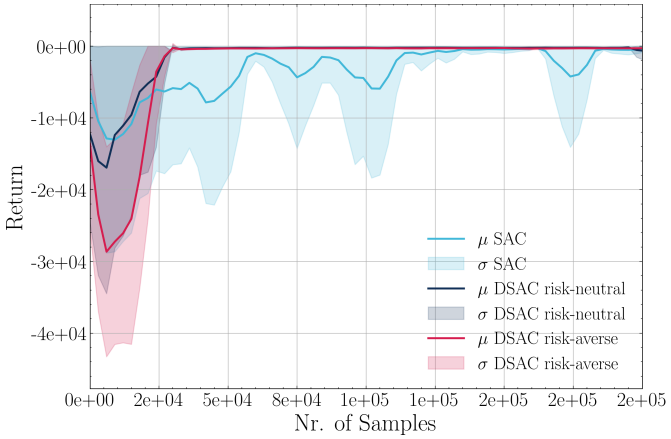


Fig. 13. Mean learning curve and standard deviation (n=10) for each agent type trained in the augmented environment (s_2) using a modified set of hyperparameters.

In addition to the improved sample efficiency, two further improvements can be observed: higher converged rewards and a reduction in variance.

Table (2) summarizes the converged average return μ and converged standard deviation of return σ for both environments and the two distributional agent types, indicating the relative improvement as a percentage for both the variance and mean return. The p-value is shown for the hypothesis that the mean return of distributional agents is higher, given two independent distributions of agents with non-equal variance. Additionally, the p-value for the lower achieved variance is shown, calculated using an f-test of different variances.

Due to the sample size (n=10), the higher achieved rewards are not statistically significant and early experiments showed that the improvement of returns is dependent on the environment. However, the reduced variance of distributional agents is statistically significant and is highlighted in Table (2). The reduced variance indicates improved learning stability and a reduction in catastrophic forgetting, as well as improved robustness to the stochasticity of the environment.

In order to ensure that the conclusions can be generalized, an additional batch of agents were trained using a modified hyperparameter set. The hyperparameter modifications consist of reduced smoothing coefficients $\lambda_{S,T} = 300$, increased network sizes 128×128, and an increased number of quantiles used ($T = 32$) to estimate the risk-distorted expectation of the return Q_k^r.

Table 2. Converged rewards of the three agents for both environment settings and for the modified hyperparameter set. The p-value shows the significance of mean μ reward improvement and the variance σ improvement. Bold values show statistically significant differences with 5e-2 threshold.

	SAC	R.N. DSAC		R.A. DSAC	
		Value (Rel.)	p	Value (Rel.)	p
Baseline env.					
μ	-148	-138 (+7%)	4e-1	-145 (+1.7%)	5e-1
σ	78	**41 (-46%)**	**3e-2**	69 (-12%)	4e-1
Augmented env.					
μ	-149	-122 (+18%)	3e-1	-118 (+21%)	3e-1
σ	177	**32 (-82%)**	**5e-6**	**31 (-82%)**	**5e-6**
Augmented env. & Modified hyperparameters					
μ	-1778	-292 (-93%)	2e-1	-295 (-83%)	2e-1
σ	6170	**608 (-90%)**	**5e-8**	**117 (-98%)**	**2e-14**

The mean learning curves with the modified hyperparameters are shown in Figure (13) and indicate a clear improvement in the stability of learning relative to the baseline SAC learning curves. Furthermore, Table (2) indicates the mean reward improvements and significant variance improvements. This shows that the DSAC agents have outstanding stability in the augmented environment, even with larger network sizes. Moreover, the risk-averse agents trained using the modified hyperparameter set show an additional 81% improvement in variance compared to risk-neutral agents, with a p-value of 2e−5.

4.2 Tracking performance

An evaluation routine similar to that of Dally and Van Kampen (2022) is used to show the attitude tracking control response of the converged controllers. Since a batch of agents (n=10) was trained for each algorithm variant and environment type, the time-domain response is plotted using the mean across all agents of the type, with a 2σ ($\approx 95\%$) confidence interval. Example time-domain responses are shown in Figure (14) and (15) for the baseline SAC agents and risk-averse DSAC agents, respectively.

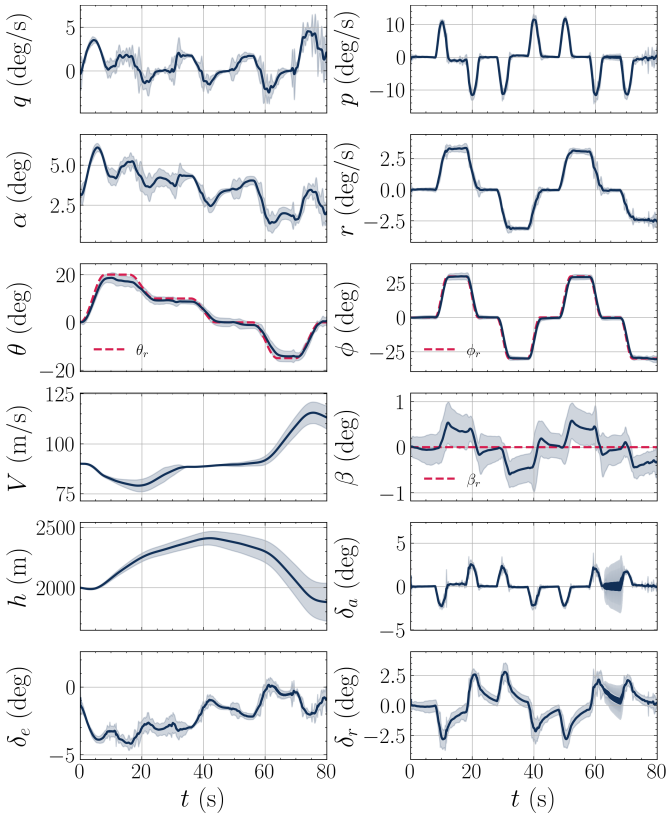


Fig. 14. Evaluation of SAC agents trained using the baseline observation s_1. Red dashed lines show the reference signals, solid blue lines show the mean response and the shaded areas show the 95% confidence interval.

In the tracking response, adequate attitude tracking can be observed for both ϕ and θ, while β is successfully regulated within the $[-1°, 1°]$ range. At $t \approx 10$ (s), the pitch tracking error increases as a banking manoeuvre is initiated. The oscillatory behaviour of the policy is eliminated for most states. High frequency control actions can be observed on the lateral control surfaces, however the confidence interval indicates that the magnitude of such actions is limited to a few degrees and the oscillations do not propagate to the controlled states.

As a metric of control tracking performance, the normalized Mean Absolute Error (nMAE) is used to compare the distributional agents to the baseline. The resulting nMAE values are shown in Table (3), where it can be seen that similar tracking performance is achieved for all agent types. Table (3) also shows the relative percentage improvements compared to the baseline SAC performance and the p-value indicates the statistical significance of the improvements.

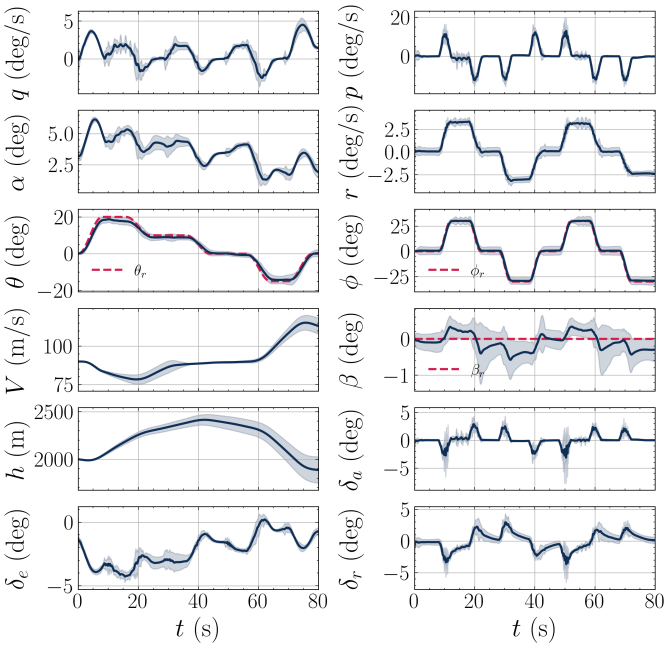


Fig. 15. Evaluation of risk-averse DSAC agents trained using the augmented observation s_2. Red dashed lines show the reference signals, solid blue lines show the mean response and the shaded areas show the 95% confidence interval.

Table 3. nMAE values for each agent type, with indicated relative improvement (Rel.). The p-value shows the t-test confidence level for the nMAE improvement. Bold values show significant differences with 5e-2 threshold.

SAC	R.N. DSAC		R.A. DSAC	
nMAE	nMAE	p	nMAE	p
Baseline env.				
12.5	**12.1 (-3.0%)**	**4e-3**	12.5 (-0.2%)	9e-1
Augmented env.				
12.5	12.4 (-1.5%)	5e-1	**12.0 (-4.0%)**	**8e-3**
Augmented env. & Mod. hyperparameters				
14.0	12.8 (-8.2%)	1e-1	**12.6 (-9.9%)**	**4e-2**

It can be seen that risk-averse DSAC agents slightly improve on the tracking performance compared to SAC when trained using the augmented observation vector s_2. This shows that DSAC agents perform similarly and often outperform SAC agents in tracking performance depending on the environment.

4.3 Risk-sensitive learning

In order to investigate the effect of risk-averse learning on the synthesized control laws, the reference signals are modified to assess the behaviour of the control laws in near-stall situations. In addition to the modified evaluation, a set of risk-averse DSAC agents are trained with varying ξ risk parameters.

The learning curves of the risk-averse agents are shown in Figures (16) and (17) for the baseline environment s_1 and the augmented environment s_2, respectively. It can be seen that highly distorted distributions result in neither degraded, nor improved learning performance.

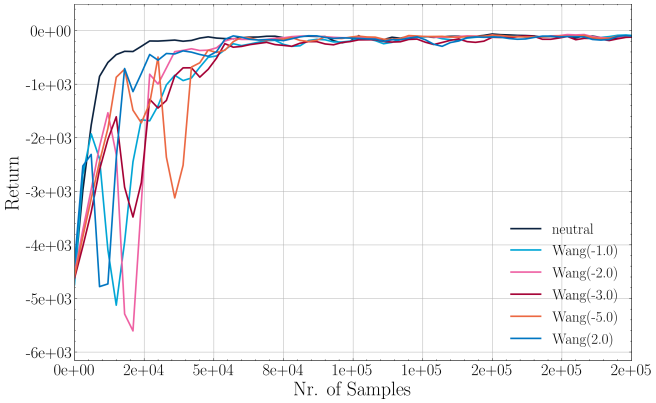


Fig. 16. Learning curves with varying Wang(ξ) distortion parameters; baseline environment s_1.

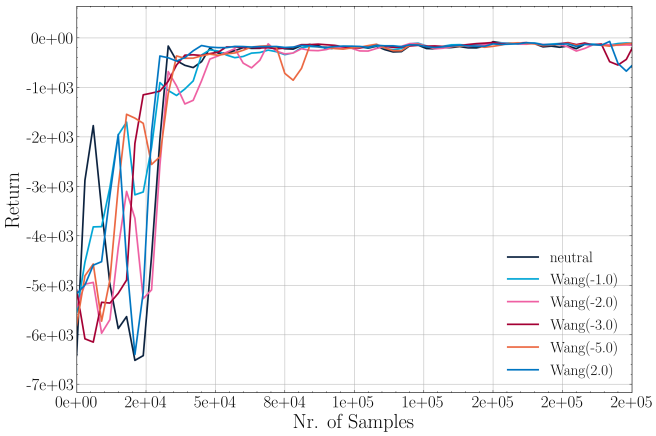


Fig. 17. Learning curves with varying Wang(ξ) distortion parameters; augmented environment s_2.

The aforementioned modified evaluation task of the risk-neutral and risk-averse agents is to follow a high pitch-up manoeuvre to near-stall conditions. Such a situation is chosen for two reasons. Firstly, it is expected that the uncertainty of such near-stall conditions is high, due to the lack of exploration and due to the unobservable dynamics that depend on airspeed and altitude. Secondly, such situations connect the uncertainty of return directly to *flight risk* (ICAO, 2018), as such flight conditions are considered hazardous and may lead to loss-of-control (LOC).

The tracking response to a near-stall condition is shown in Figure (18) for both risk-neutral (left) and risk-averse (right) agents. Figure (18) only depicts the longitudinal states, as the reference roll and sideslip angles are both kept at zero. Throughout the manoeuvre, the sustained high pitch-angle reference causes the aircraft to lose airspeed and gain altitude. The 45° pitch angle reference at $t = 60$ (s) is unattainable by the agent without entering stall-induced oscillations and instability. The risk-neutral DSAC agent (left) responds to the reference signal by further deflecting the elevator, inducing undesirable oscillatory behaviour. On the other hand, the risk-averse agent (right) avoids the stall-induced oscillations and keeps α at $\sim 10°$.

The variance of the return distribution estimated by the trained critic is also shown for each state-action pair during

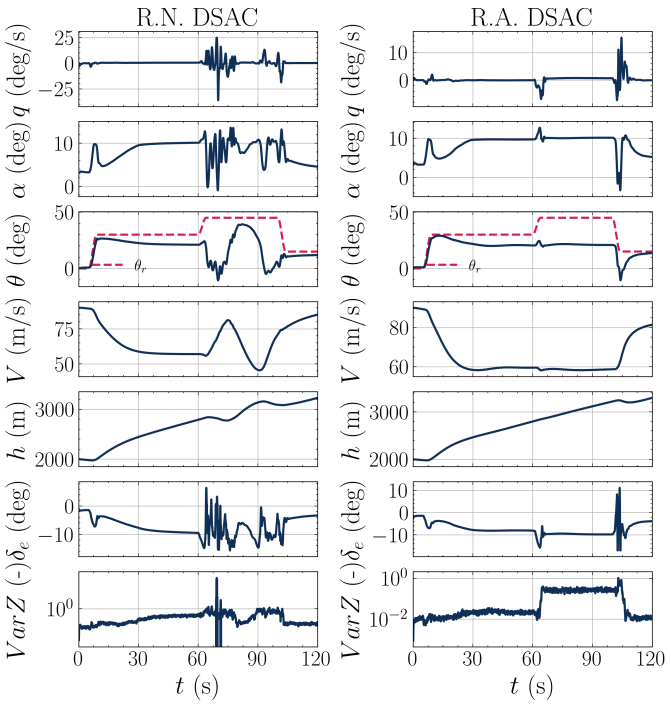


Fig. 18. Comparison of near-stall attitude control task of risk-neutral (left) and risk-averse (right) DSAC controllers; augmented observation s_2; Risk-averse trained using Wang(-2.0); Red dashed lines show the reference signals.

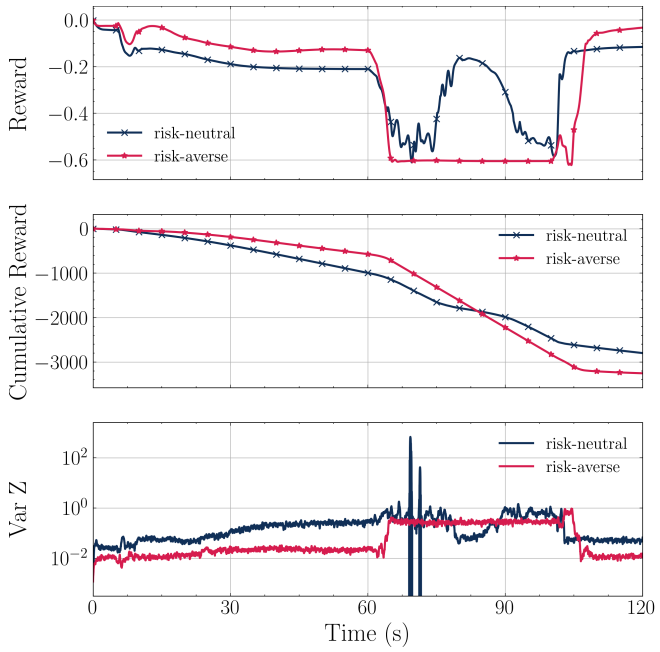


Fig. 19. Rewards, cumulative rewards and the distributional critic's variance estimate during the near-stall manoeuvre evaluation for both risk-neutral and risk-averse DSAC agents.

the high-risk manoeuvre. The variance of return can be used as a metric of estimated uncertainty (Mavrin et al., 2019). Figure (19) shows both the immediate and cumulative rewards achieved by the risk-neutral and risk-averse agents for each time-step throughout the manoeuvre. It can be seen that the risk-averse controller sacrifices immediate rewards to avoid states with high-uncertainty.

Even though the risk-averse controller slightly outperforms the risk-neutral agent in the first half of the episode, it chooses more conservative actions following the 45° pitch angle reference at $t = 60$ (s). The risk-averse agent achieves a reduced end-of-episode return, but manages to avoid stall-induced oscillations. This indicates that training the actor to prioritize state-action pairs with low variance in the return distribution inherently increases the safety of flight.

The risk-averse behaviour is achieved without the addition of human-domain knowledge, such as reward shaping. Whether the critic's estimate of uncertainty is due to parametric or intrinsic uncertainty in the environment is not pertinent to the safety of control and decision making, as both unexplored and highly stochastic state-action are to be assigned a lower risk-distorted action-value, in order to reduce flight risk.

5. CONCLUSION

This research contributes to the synthesis of risk-averse model-free flight controllers for non-linear fully-coupled aerospace systems, and lays the foundation for deep reinforcement learning flight controllers that approximate the uncertainty of the environment. The distributional soft actor-critic algorithm used in this paper does not optimize for higher expectation of rewards, but rather a trade off between rewards, uncertainty, action diversity, and action smoothness.

Previously, Dally and Van Kampen (2022) found low training reliability in the training of the baseline SAC algorithm. We show that the DSAC algorithm significantly improves on the learning characteristics by combining maximum entropy RL with distributional critics, which proves to be effective at exploration and provides more stable and consistent learning for partially observable, high-dimensional control tasks.

The estimation of the return distribution enables the synthesis of risk-sensitive policies. This paper implements the method of distorted expectations, as a risk-measure to train risk-averse control laws. The resulting risk-neutral and risk-averse DSAC agents show similar tracking performance compared to the baseline SAC agents.

Additionally, we show that training risk-averse policies using highly distorted expectations results in control laws that prioritize state-action pairs with low variance, increasing the safety of RL-based flight control.

5.1 Significance

Improving the robustness of offline RL algorithms with regards to uncertainty is a crucial step in bridging the simulation gap in flight control applications of RL and reducing the model-dependency of adaptive controller synthesis. It is expected that DRL controllers trained in simulated environments will provide the initial starting point for online, adaptive controllers, as demonstrated by Teirlinck and Van Kampen (2022). Modelling the uncertainty of the environment and/or the uncertainty

of model parameters is an essential step of applying offline simulated RL controllers to real-world aerospace systems.

As mentioned above, the synthesized risk-averse control laws increase the safety of flight by avoiding uncertainty in the return. This is achieved using minimal human-domain knowledge and is purely a result of goal-oriented interaction. Such approaches are needed to reduce the model-dependence of control synthesis and to enable the real-world application of continually learning RL-based controllers for safety critical systems.

5.2 Recommendations

Variety of Risk-measures This paper only considers the risk-distorted expectation as a mode of synthesizing risk-sensitive control laws. Having access to the return distribution provides wide array of risk-sensitive approaches.

Unused Trained Critic More importantly, the DSAC algorithm outlined in this paper does not utilize the return distribution estimate post-training, even though it is demonstrated that the variance of the return is a valuable predictor of flight risk. Future work is needed to make full use of the trained critic networks, in a potentially adaptive, continually learning setting.

Monotonic Constraints During early stages of learning, the monotonicity of the quantile function is not guaranteed and the return distribution estimate is ill-defined. An approach by Théate et al. (2021) could improve the early learning performance and improve the early estimate of uncertainty.

Hyperparameter Sensitivity While the sensitivity of DSAC to hyperparameter changes was outside the scope of the research, early experiments suggest robustness to hyperparameter changes. Additional experiments and parameter sweeps are needed to investigate the robustness to hyperparameters explicitly.

Recurrent Neural Networks Since most DRL flight controllers train using partially observable MDPs, the use of recurrent neural networks could improve the estimation of uncertainty, and the risk-averse response at the cost of more difficult learning.

6-DOF Flight Control This paper focuses on the isolated effect of adding distributional RL to low-level flight control and trained attitude controllers in a limited flight envelope. With the observed improvements in learning performance, it is expected that 6-DOF control of an aircraft may be achieved using a hierarchy of DRL agents and ensuring thorough exploration of the flight envelope.

REFERENCES

Ajaj, R.M., Beaverstock, C.S., and Friswell, M.I. (2016). Morphing aircraft: The need for a new design philosophy. Aerospace Science and Technology, 49, 154–166. URL https://doi.org/10.1016/j.ast.2015.11.039.

Ba, J.L., Kiros, J.R., and Hinton, G.E. (2016). Layer Normalization. arXiv:1607.06450 [cs, stat]. URL http://arxiv.org/abs/1607.06450.

Bellemare, M.G., Dabney, W., and Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. International Conference on Machine Learning, 449–458. URL https://arxiv.org/abs/1707.06887.

Bellemare, M.G., Dabney, W., and Rowland, M. (2023). [under review] Distributional Reinforcement Learning. MIT Press. http://www.distributional-rl.org.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018). Implicit Quantile Networks for Distributional Reinforcement Learning. URL http://arxiv.org/abs/1806.06923.

Dabney, W., Rowland, M., Bellemare, M.G., and Munos, R. (2017). Distributional Reinforcement Learning with Quantile Regression. URL http://arxiv.org/abs/1710.10044.

Dally, K. and Van Kampen, E.J. (2022). Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control. In AIAA SCITECH 2022 Forum. San Diego, CA & Virtual. URL https://arc.aiaa.org/doi/10.2514/6.2022-2078.

de Croon, G., de Clercq, K., Ruijsink, R., Remes, B., and de Wagter, C. (2009). Design, Aerodynamics, and Vision-Based Control of the DelFly. International Journal of Micro Air Vehicles, 1(2), 71–97. URL http://journals.sagepub.com/doi/10.1260/175682909789498288.

de Croon, G., Perçin, M., Remes, B., Ruijsink, R., and De Wagter, C. (2016). The DelFly. Springer Netherlands, Dordrecht. URL http://link.springer.com/10.1007/978-94-017-9208-0.

Faggiano, F., Vos, R., Baan, M., and Van Dijk, R. (2017). Aerodynamic Design of a Flying V Aircraft. In 17th AIAA Aviation Technology, Integration, and Operations Conference. Denver, Colorado. URL https://arc.aiaa.org/doi/10.2514/6.2017-3589.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the 35th International Conference on Machine Learning, 1587–1596. PMLR. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning, 1861–1870. PMLR. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2019). Soft Actor-Critic Algorithms and

Applications. URL http://arxiv.org/abs/1812.05905.

Huber, P.J. (1992). Robust estimation of a location parameter. In Breakthroughs in statistics, 492–518. Springer.

ICAO (2018). Doc 9859: Safety management manual, fourth edition, 2018. URL https://skybrary.aero/bookshelf/books/5863.pdf.

Kingma, D.P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. URL http://arxiv.org/abs/1412.6980.

Konatala, R., Van Kampen, E.J., and Looye, G. (2021). Reinforcement learning based online adaptive flight control for the cessna citation ii (ph-lab) aircraft. In AIAA Scitech 2021 Forum, 0883. URL https://arc.aiaa.org/doi/10.2514/6.2021-0883.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. URL http://arxiv.org/abs/1509.02971.

Liu, C., van Kampen, E.J., and de Croon, G.C.H.E. (2022). Adaptive Risk Tendency: Nano Drone Navigation in Cluttered Environments with Distributional Reinforcement Learning. URL https://arxiv.org/abs/2203.14749.

Ma, X., Xia, L., Zhou, Z., Yang, J., and Zhao, Q. (2020). DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning. URL http://arxiv.org/abs/2004.14547.

Mavrin, B., Yao, H., Kong, L., Wu, K., and Yu, Y. (2019). Distributional Reinforcement Learning for Efficient Exploration. International Conference on Machine Learning, 4424–4434. URL http://proceedings.mlr.press/v97/mavrin19a.html.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533. URL http://www.nature.com/articles/nature14236.

Morelli, E.A. and Klein, V. (2016). Aircraft system identification: theory and practice, volume 2. Sunflyte Enterprises Williamsburg, VA.

Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K. (2021). Regularizing Action Policies for Smooth Control with Reinforcement Learning. IEEE International Conference on Robotics and Automation (ICRA), 1810–1816. URL http://arxiv.org/abs/2012.06644.

Pollack, T. and Van Kampen, E.J. (2019). Safe Curriculum Learning for Primary Flight Control. Master Thesis, Technical University of Delft, Delft. URL http://resolver.tudelft.nl/uuid:1b2becfd-c2db-43fc-a273-a3ff6a9ba50a.

Pollack, T. and Van Kampen, E.J. (2022). Robust Stability and Performance Analysis of Incremental Dynamic Inversion-based Flight Control Laws. In AIAA SCITECH 2022 Forum. San Diego, CA & Virtual. URL https://arc.aiaa.org/doi/10.2514/6.2022-1395.

Polyak, B.T. and Juditsky, A.B. (1992). Acceleration of Stochastic Approximation by Averaging. SIAM Journal on Control and Optimization, 30(4), 838–855. URL https://epubs.siam.org/doi/abs/10.1137/0330046. Publisher: Society for Industrial and Applied Mathematics.

Ruiz Garcia, A., Brown, M., Atherstone, D., Arnhem, N.v., and Vos, R. (2022). Aerodynamic Model Identification of the Flying V from Sub-Scale Flight Test Data. In AIAA SCITECH 2022 Forum. San Diego, CA & Virtual. URL https://arc.aiaa.org/doi/10.2514/6.2022-0713.

Saeed, A.S., Younes, A.B., Islam, S., Dias, J., Seneviratne, L., and Cai, G. (2015). A review on the platform design, dynamic modeling and control of hybrid UAVs. In 2015 International Conference on Unmanned Aircraft Systems (ICUAS), 806–815. IEEE, Denver, CO, USA. URL http://ieeexplore.ieee.org/document/7152365/.

Schulman, J., Levine, S., Moritz, P., Jordan, M.I., and Abbeel, P. (2015). Trust Region Policy Optimization. URL http://arxiv.org/abs/1502.05477. ArXiv: 1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2016). Proximal Policy Optimization Algorithms. URL http://arxiv.org/abs/1707.06347.

Sieberling, S., Chu, Q.P., and Mulder, J.A. (2010). Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction. Journal of Guidance, Control, and Dynamics, 33(6), 1732–1742. URL https://arc.aiaa.org/doi/10.2514/1.49978.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In International conference on machine learning, 387–395. PMLR. URL http://proceedings.mlr.press/v32/silver14.html.

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

Sun, S., Wang, X., Chu, Q., and Visser, C.d. (2021). Incremental Nonlinear Fault-Tolerant Control of a Quadrotor With Complete Loss of Two Opposing Rotors. IEEE Transactions on Robotics, 37(1), 116–130. URL https://ieeexplore.ieee.org/document/9160894/.

Teirlinck, C. and Van Kampen, E.J. (2022). Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance. Master Thesis, Technical University of Delft. URL http://resolver.tudelft.nl/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85.

Théate, T., Wehenkel, A., Bolland, A., Louppe, G., and Ernst, D. (2021). Distributional Reinforcement Learning with Unconstrained Monotonic Neural Networks. URL http://arxiv.org/abs/2106.03228.

Van den Hoek, M., de Visser, C., and Pool, D. (2018). Identification of a cessna citation ii model based on flight test data. In Advances in Aerospace Guidance, Navigation and Control, 259–277. Springer. URL https://doi.org/10.1007/978-3-319-65283-2_14.

van Gils, P., Van Kampen, E.J., de Visser, C.C., and Chu, Q.P. (2016). Adaptive incremental backstepping flight control for a high-performance aircraft with uncertainties. In AIAA Guidance, Navigation, and Control Conference, 1380. URL https://arc.aiaa.org/doi/10.2514/6.2016-1380.

Wang, S.S. (2000). A class of distortion operators for pricing financial and insurance risks. Journal of risk and insurance, 15–36.

Wang, X. and Sun, S. (2021). Incremental fault-tolerant control for a hybrid quad-plane UAV subjected to a complete rotor loss. Aerospace Science and Technology, 107105. URL https://linkinghub.elsevier.com/retrieve/pii/S1270963821006155.

Weisshaar, T.A. (2013). Morphing Aircraft Systems: Historical Perspectives and Future Challenges. Journal of Aircraft, 50(2), 337–353. URL https://arc.aiaa.org/doi/10.2514/1.C031456.

Zhou, Y., van Kampen, E.J., and Chu, Q.P. (2018). Incremental model based online dual heuristic programming for nonlinear adaptive control. Control Engineering Practice, 73, 13–25. URL https://linkinghub.elsevier.com/retrieve/pii/S096706611730285X.

Ziebart, B.D. (2010). Modeling purposeful adaptive behavior with the principle of maximum causal entropy. Carnegie Mellon University.

Appendix A. DSAC ALGORITHM

Algorithm 1: Distributional soft actor-critic with policy regularization

Data: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma \rangle$, N_{ep}
Result: $\pi_w(s)$, $Z_k(s,a)$
Hyperparameter: $|\mathcal{B}|, N, \kappa, T, \zeta, \sigma, \lambda_S, \lambda_T, \bar{\mathcal{H}}$

Init: memory $\mathcal{D} \leftarrow \{\}$
Init: critics $Z_{k_{1,2}}$, $Z_{\bar{k}_{1,2}}$ and policy π_w
Init: temperature $\eta \leftarrow 1$

for episode $n \leftarrow 1$ **to** N_{ep} **do**
  Initialize s
  while s not terminal **do**
    sample action $a \sim \pi_w(\cdot|s)$
    observe transition $r \leftarrow R(s,a)$, $s' \sim \mathcal{P}(s'|s,a)$
    store transition $\mathcal{D} \leftarrow \mathcal{D} \cup \langle s, a, r, s' \rangle$

    Update networks
    sample mini-batch $\mathcal{B} \overset{i.i.d}{\sim} \mathcal{D}$
    $\langle s, a, r, s' \rangle \sim \mathcal{B}$
    sample next action $a' \sim \pi_w(\cdot|s)$
    sample quantile fractions $\tau_i, \tau_j \overset{N}{\sim} U([0,1])$
    for $l \leftarrow 1$ **to** 2 (each critic) **do**
      for $i \leftarrow 1$ **to** N **do**
        for $j \leftarrow 1$ **to** N **do**
          $\mathbb{Z} \leftarrow \min_{l=1,2} Z_{\tau_i}(s', a'; \bar{k}_l)$
          $\mathcal{H}_\pi \leftarrow -\log \pi_w(a'|s')$
          $\delta_{ij}^l = r + \gamma\left(\mathbb{Z} + \eta\mathcal{H}_\pi\right) - Z_{\tau_j}(s, a; k_l)$
        end
      end
      quantile huber loss
      $\mathcal{L}_Z(k) \leftarrow \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\rho_{\tau_j}^{\kappa}(\delta_{ij}^l)$
      update critic weights
      $k_l \leftarrow \texttt{Adam}\left(\nabla_k \mathcal{L}_Z\right)$
      $\bar{k}_l \leftarrow \zeta\bar{k}_l + (1-\zeta)k_l$
    end
    sample new action $\bar{a} \sim \pi(\cdot|s)$
    risk-distorted soft action-value
    $\tau_i^\Psi \leftarrow \Psi(\tau)$, with $\tau_i \overset{T}{\sim} U([0,1])$
    $Q_k^\Psi \leftarrow \min_{l=1,2}\left[\frac{1}{T}\sum_{i=1}^{T} Z_{\tau_i^\Psi}(s, \bar{a}; k_l)\right]$

    smoothness loss terms
    $\mathcal{L}_S \leftarrow \left\|\pi(s) - \pi_w(\tilde{s})\right\|_2$, with $\tilde{s} \sim \mathcal{N}(s, \sigma)$
    $\mathcal{L}_T \leftarrow \left\|\pi(s) - \pi_w(s')\right\|_2$

    update policy
    $\mathcal{L}_\pi(w) \leftarrow \eta \log \pi_w(\bar{a}|s) - Q_k^\Psi + \lambda_S\mathcal{L}_S + \lambda_T\mathcal{L}_T$
    $w \leftarrow \texttt{Adam}\left(\nabla_w \mathcal{L}_\pi(w)\right)$

    update temperature
    $\mathcal{L}(\eta) \leftarrow \eta\bar{\mathcal{H}} - \eta\log \pi_w(\bar{a}|s)$
    $\eta \leftarrow \texttt{Adam}\left(\nabla_\eta \mathcal{L}(\eta)\right)$
  end
end

Part II

Preliminary Analysis

*This part has been assessed for the course AE4020 Literature Study.

3

Literature Review

3.1 Fundamentals of Reinforcement Learning

Reinforcement learning is a bio-inspired algorithm that facilitates learning by interaction. An agent must discover the best action to take by interacting with the environment in order to reach a certain goal, i.e. to maximize the reward it gets from the environment. Reinforcement learning as a concept is quite mature, as even Alan Turing theorized a bio-inspired ”pleasure-pain system” [36, 21], however recent advancements in machine learning methods have laid the foundation for a vast field of modern techniques and approaches to RL.

Reinforcement learning deals with finding a solution to a sequential decision making process. As opposed to other machine learning techniques, RL has no supervisor, as the agent learns from the samples it collects via direct interaction with the environment. In a sequential decision making process, the gathered data sequences are not independent and identically distributed (IID), which means that the state, action and reward signals the RL agent learns from are highly correlated. One of the major attractive properties of RL algorithms is that no priori knowledge is required about the environment or the underlying process. Additionally, the learning is goal-oriented and the optimal strategies can be found by the agent without ever having been given hints about a solution.

The purpose of this chapter is to establish a foundation of RL methods and to provide an overview of existing approaches. Firstly, Subsection 3.1.1 presents the core elements of reinforcement learning and provides a formal framework to define the RL problem. Secondly, dynamic programming (DP) is introduced in Subsection 3.1.2 followed by a discussion of model-free prediction methods like Monte-Carlo (MC) and Temporal-difference (TD) in Subsection 3.1.3. Then, Subsection 3.1.4 presents model-free tabular methods that find near-optimal policies. Moreover, Subsection 3.1.5 discusses ways to categorize RL methods, in order to provide a better overview of the variety of approaches. Lastly, Subsection 3.1.6 provides a brief overview and synthesis of the chapter.

3.1.1 Formulation of the RL Problem

Reinforcement learning methods aim to solve sequential decision tasks on a set of problems. This section presents the framework for defining RL tasks using Markov Decision Processes (MDPs). This chapter assumes that the state and action spaces of the RL problems are finite and countable for simpler formulations and to allow the use of tabular methods. Section 3.2 then extends these concepts to real-world scalable scenarios. Most of the fundamental concepts of this chapter are adapted from [21] and [37].

Elements of Reinforcement Learning

The main concept that separates reinforcement learning from other types of machine learning, namely supervised and unsupervised learning is direct interaction. Reinforcement learning is formulated as a sequential decision making problem, where the *agent* is the decision making entity that repeatedly interacts with a process, the *environment*, and subsequently receives a certain numeric *reward*. Additionally, the *actions* chosen by the agent influence the state of the environment, which provides measurements, i.e.

observations of its state. The goal of this sequential decision making process is for the agent to maximize the expected cumulative reward received from the environment.

This interaction between agent and environment is shown in Fig. 3.1, where an action at discrete time-step t: A_t is chosen by the agent, which determines the next state S_{t+1} and reward in the sequence R_{t+1}.

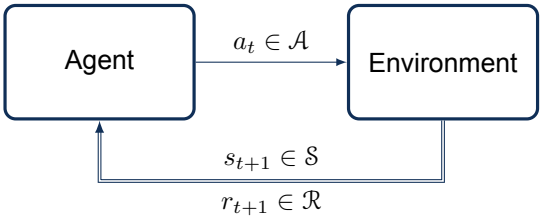


Figure 3.1: Adapted from [21]; Agent-environment interaction in a reinforcement learning task.

Often, the realizations of the random variables S, A and R at time-step t are denoted by: s, a and r respectively. Another common short-hand notation is the next-state S_{t+1} being represented by s'. With this notation, the one-step transition of the process depicted in Fig. 3.1 is $\mathbb{T}_t = \langle s, a, r, s' \rangle$.

Reinforcement learning defines the boundary between agent and environment somewhat differently with respect to control theory, where a controller determines the inputs in order to stabilize or control a certain process. In RL, the environment contains not only the plant dynamics, but also all of the sensors, actuators, disturbances and even the reward signal. Therefore, the environment often describes a stochastic, nonlinear process with time-varying elements and possible delayed rewards.

An agent may be composed of several elements, namely a *policy* which can be a deterministic $a = \pi(s)$, or a stochastic mapping $a \sim \pi(a|s)$ from state to action, a *value function* $v(s)$ which describes the expected total reward for being in certain states and lastly an agent may contain a model representation of the environment.

A crucial and unique element of reinforcement learning is the constant trade-off between *exploration* and *exploitation*. When the agent has a limited knowledge of the effect of certain actions and states, a greedy choice of exploiting immediate rewards can have negative effects, when a better option exists outside of the knowledge of the agent. This means that in the early stages of learning, choosing exploration at the cost of immediate reward can increase the total future reward received by the agent.

There is an important distinction between two types of problems in sequential decision making: *planning* and *reinforcement learning*. In a planning problem, the agent has access to a model of the environment and can improve its policy using calculations with the model without any external interaction with the environment. On the other hand, in a reinforcement learning problem the environment is initially unknown and the agent improves its policy by interaction. There are certain model-based RL methods that successfully combine interaction with an intermediate planning step [38].

An additional remark on the definition of RL concepts is the distinction between *prediction* and *control*. In the context of reinforcement learning, prediction refers to an evaluation problem, i.e. determining how successful a certain policy is. RL control[1] on the other hand is an optimization problem, where the best policy is to be found to reach a certain goal. Most often RL control methods are recursive iterative two-step algorithms: first an evaluation step is made, followed by a policy improvement step.

Markov Decision Process

In order to formulate a framework for the sequential decision making problem, Markov Decision Processes (MDPs) can be defined. A Markov process is a memory-less process where the current state is a sufficient statistic of the future. Almost all RL problems can be formulated as a *Markov Decision Process*, or in case of partially observability, as a Partially Observable MDP.

[1] Control in RL refers to finding the optimal policy, whereas control theory and control system design refer to designing algorithms that can determine the actions to take to regulate dynamic systems.

A Markov process is described by two attributes: the set of possible states $\mathcal{S} \subset \mathbb{R}^n$ and the stochastic transition dynamics of the environment $\mathcal{P}$. The state of the environment is the random variable $S_t^e \in \mathcal{S}$, with realization s at discrete time step t, and the state transition is described by the distribution function $\mathcal{P}$.

For the process to have the Markov property, the state signal must contain enough information such that the state of the system at time step $t+1$ only depends on state S_t, as opposed to depending on the entire history of states. Therefore, the state S_t is Markov if and only if:

$$\mathbb{P}\left[S_{t+1} \mid S_t\right] = \mathbb{P}\left[S_{t+1} \mid S_1, ..., S_t\right] \tag{3.1}$$

The Markov process can be defined as the tuple $\langle \mathcal{S}, \mathcal{P} \rangle$, where $\mathcal{P}$ is the state transition probability distribution function described by:

$$\mathcal{P}_{s,s'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right] \tag{3.2}$$

Where s' is used to denote a realization of the next state at time $t+1$.

In order to describe the full MDP, the formulation is extended with the set of rewards $\mathcal{R}$ with reward function R and with the actions made by the agent $A_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of all actions. The reward function can be defined as:

$$R_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right] \tag{3.3}$$

With actions introduced, the state distribution function depends on the actions made by the agent, therefore the transition dynamics can be written as:

$$\mathcal{P}_{s,s'}^a = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right] \tag{3.4}$$

The signal R_t is an immediate reward. In order for an agent to be able to maximize not only the reward at the next time step, the *discounted return* G_t at time-step t is defined as the following:

$$G_t = R_{t+1} + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+1} + ... = \sum_{k=0}^{T-t-1} \gamma^k \cdot R_{t+k+1} \tag{3.5}$$

Where $\gamma \in [0, 1]$ is the *discount rate*. This formulation is generalized for both episodic tasks with terminal states and continuing tasks. With each component defined, the full MDP can written as the structured set $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

The agent operating in the MDP uses policy π to make decisions. The policy can be deterministic $a = \pi(s)$ or more often a stochastic mapping:

$$\pi(a \mid s) = \mathbb{P}\left[A_t = a \mid S_t = s\right] \tag{3.6}$$

Since the next state s' fully characterizes future rewards due to the Markov property, policy π does not depend on the history of states and actions.

With the MDP fully defined, as well as the policy that chooses the action, the sequential decision making process can be depicted in Fig. 3.2, where state s_n is either terminal or the process is continuing, i.e. $n \to \infty$.

Bellman Equations

Since the goal of the agent is to maximize the total cumulative rewards, the formulation of the MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ makes it possible to define the *value* of a certain state: $V_\pi(s)$, which represents the total expected discounted return when following policy π from state s thereafter:

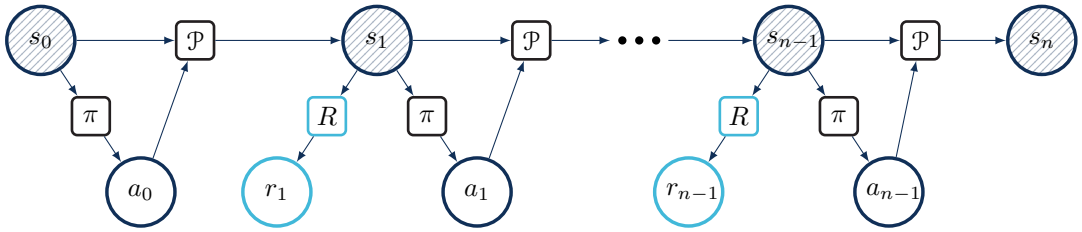


Figure 3.2: Depiction of the MDP following policy $\pi(a \mid s)$, with Markov Process $\langle \mathcal{P}, R \rangle$.

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{3.7}$$

Similarly, the action-value function can be defined as the total expected return of state s, choosing actions a and following policy π thereafter:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{3.8}$$

Noticing the recursive nature of this sequential process, the Bellman expectation equation defines the value function[2] as the expected sum of the next immediate reward and the discounted value of the next state, while following policy π [23]:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \, V_\pi(S_{t+1}) \mid S_t = s \right] \tag{3.9}$$
$$Q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \, Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right] \tag{3.10}$$

The maximum value function over all policies is the optimal value function $V_*(s) = \max\limits_{\pi} V_\pi(s)$ and corresponds to the best possible performance in the MDP and therefore the MDP is considered solved when $V_*(s)$ is known. Similarly, the optimal action-value function is $Q_*(s,a)$. Given that there exists an optimal policy π_* that achieves the optimal value function $V_*(s)$, this policy can be found by maximizing over $Q_*(s,a)$ which can be written in the deterministic form:

$$\pi_*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q_*(s,a) \tag{3.11}$$

Similar to the Bellman expectation equations, this relationship between optimal value functions follows a recursive pattern. The Bellman optimality equations for $V_*(s)$ and $Q_*(s,a)$ can be written as:

$$V_*(s) = \max_a \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a V_*(s') \right) \tag{3.12}$$
$$Q_*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \max_{a'} \; Q_*(s',a') \tag{3.13}$$

Due to the non-linearity of the optimality equations no closed form solutions exist, but several iterative approaches can be taken which are discussed in subsequent sections.

[2]Value function often refers to both the value function $V(s)$ and action-value function $Q(s,a)$.

3.1.2 Dynamic Programming

Dynamic Programming (DP) is a general solution method of solving complex problems which have two properties: optimal substructures that can be solved in an optimal way and overlapping substructures that recur multiple times. The reason dynamic programming is prevalent in RL methods is because MDPs possess both of these properties. The Bellman equations provide a recursive decomposition of the MDP and the value function stores and reuses previous solutions. Dynamic Programming in RL solves the planning problem, i.e. performing calculations with a given model of the MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ to either evaluate a policy (prediction) or to find an optimal policy (control). Since the Bellman equations pose a nonlinear optimization problem, most often an iterative approach is constructed as a closed-form solution does not exist. In generalized terms, Dynamic Programming creates full-width backups of the decision tree using all states and actions and utilizes *value iteration* (VI) or *policy iteration* (PI) to propagate information of the rewards through the states. The two approaches mentioned, i.e. value iteration and policy iteration are discussed in Subsection 3.1.2 and Subsection 3.1.2.

Value Iteration

Value iteration estimates the optimal value function $V_*(s)$ by iteratively applying the Bellman optimality equations defined in Eq. (3.12):

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left(R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{s,s'}^a V_k(s') \right) \tag{3.14}$$

Note that value iteration can also use the action-value function instead. Since the value iteration operates purely in value space, there is no explicit definition of policy in this process, however at any time-step a policy can be defined implicitly by choosing the action with the highest value, i.e. acting *greedily* against the value function estimate.

Policy Iteration

Policy iteration is a two-step iterative approach, in which first a policy evaluation step is done to evaluate the value-function $V_\pi(s)$, followed by a policy improvement step. This comes at additional computation cost as several non-optimal intermediate policies have to be evaluated. In order to evaluate the policy, the value of each state $V_k(s)$ is updated after k iterations using the value of the corresponding successor state, utilizing the Bellman expectation equation defined in Eq. (3.9):

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right) \tag{3.15}$$

Then, a new policy π' can be obtained by acting greedily with respect to $V_\pi(s)$, which ensures that the optimal policy is found with sufficient iterations as $k \to \infty$. This iterative process is depicted in Fig. 3.3 and defines Generalized Policy Iteration (GPI), which is a core concept within reinforcement learning.

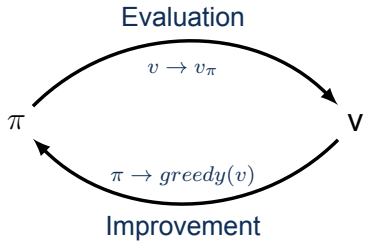


Figure 3.3: Adapted from [37]; Generalized Policy Iteration (GPI) iterates through an evaluation step followed by a policy improvement step.

Remarks on Dynamic Programming

As mentioned, Dynamic programming does full-width backups, considering all states and all actions, which results in a computational complexity of $\mathcal{O}(mn^2)$ for n states and m actions. When action-value functions are used ($Q_\pi(s,a)$ or $Q_*(s,a)$) instead of value functions ($V_\pi(s)$ or $V_*(s)$) the complexity grows to $\mathcal{O}(m^2n^2)$. This causes DP to suffer from the curse of dimensionality for large problems.

Some approaches use a function approximator $V(s;\theta)$ to estimate the value function in order to generalize and scale the method, whereas other algorithms use samples from the environment to reduce the computational complexity of the backups.

3.1.3 Model-Free Prediction

Whereas dynamic programming solves a known MDP, model-free approaches require the agent to learn from samples taken from interacting with the environment. This section describes two commonly used approaches, namely Monte-Carlo (MC) and Temporal-Difference (TD) methods to solve the prediction problem, i.e. to estimate the value function of interacting with an unknown MDP under policy π.

Monte-Carlo Methods

In Monte-Carlo approaches, the agent collects the end-of-episode returns to estimate the value function from its experience and interactions with the environment. The returns are collected at the end of the episodes and therefore provide an unbiased sample of the true return. One of the downsides of MC is that the end of the episode must be reached before an update step can occur, which also means that it can only be applied to episodic environments.

In essence, Monte-Carlo methods use an empirical mean to estimate the expected return. This empirical mean can be calculated using the update rule given by:

$$\hat{V}_\pi(s) = \frac{1}{N}\sum_{n=0}^{N} G(s) \tag{3.16}$$

Where state s has been encountered N times during the episode and $G(s)$ represents the end of episode return observed after encountering state s. With $N \to \infty$, this simple update rule is guaranteed to converge to the true value function $V_\pi(s)$. The incremental form of the Monte-Carlo update is also useful as it is able to learn non-stationary problems:

$$V(s) \leftarrow V(s) + \alpha\left(G_t - V(s)\right) \tag{3.17}$$

Where α is the step-size towards the actual return G_t.

MC methods provide an unbiased estimate of the value function as the true return is used at the end of each episode, however they often encounter higher variance during learning which is associated with the high variance of the return.

Temporal-Difference Methods

Temporal-difference methods update the value function instantly after each step by taking the next step's expected return as a target. This process of updating a guess towards another guess is called bootstrapping and introduces bias into the learning process, however TD-methods can be applied to continuing environments that have no terminal states. Furthermore, updating the value function every step lowers the variance and stabilizes the learning process.

The estimated return of the next step is given as the sum of the next immediate reward and discounted expected return of the next state: $r + \gamma V(s')$, which is the so-called TD-target. TD-methods update the value function towards this next step estimate by the update rule:

$$V(s) \leftarrow V(s) + \alpha \left(r + \gamma V(s') - V(s) \right) \tag{3.18}$$

Where the term $r + \gamma V(s') - V(s)$ is often denoted using δ, called the TD-error.

As mentioned, this update rule can be used on continuing episodes, as the final outcome is not required to complete the backup. TD-methods exploit the Markov property and are therefore generally more efficient in Markov environments.

Eligibility Traces

In most cases it is desirable to be able to tune the amount of bootstrapping the method uses and hence set the method to use the reward from multiple steps taken in the environment. For this, n-step predictions can be used to as a TD-target and a tunable decay parameter $\lambda \in [0, 1]$ is defined to average over each consecutive n-step return. The multi-step λ-return is then:

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t;n} \tag{3.19}$$

Where $G_{t;n}$ is the n-step return and is calculated by considering each consecutive expected return:

$$G_{t;n} = r + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n}) \tag{3.20}$$

With G_t^{λ} as the n-step TD-target, the update step for forward-view TD(λ) can be defined:

$$V(s) \leftarrow V(s) + \alpha \left(G_t^{\lambda} - V(s) \right) \tag{3.21}$$

In practice, it is often more practical to use a backward-view of the equivalent update rule using the eligibility trace signal $E(s)$. The eligibility trace keeps track of two heuristics: the frequency and recency, by keeping track of the state-visitation count and decaying the signal with each time step:

$$E_t(s) = \gamma \lambda E_{t-1}(s) + 1(S_t = s) \tag{3.22}$$

Where $\lambda \in [0, 1]$ is the aforementioned decay-rate, the eligibility trace is initialized with $E_0(s) = 0$ and $E(s)$ is updated for all states for each time-step t. This update rule produces the equivalent backward-view of the TD(λ) method. At the two extremes TD(0) is equivalent to the previously defined TD-method, whereas TD(1) approaches the every-visit Monte-Carlo method.

3.1.4 Model-free Control

Previous sections have considered either planning tasks where the MDP is known, or prediction approaches where the objective is to determine the value function under a given policy π. Model-free RL control deals with finding the optimal policy for an *unknown* MDP.

RL algorithms utilize the GPI paradigm from dynamic programming, where an evaluation step is followed by a policy improvement step as shown in Fig. 3.3. The evaluation step can be adapted from the model-free prediction methods discussed in Subsection 3.1.3, whereas the policy improvement can similarly be a greedy-action selection w.r.t the updated value-function. The modification that is required for the model-free control task is to use the action-value function $Q(s, a)$ instead of the value function $V(s)$, in order to be able to define the greedy policy improvement:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, Q(s, a) \tag{3.23}$$

In order to ensure correct exploration of certain state-action pairs the ϵ-$greedy$ policy can be defined to choose a random[3] action with probability $\epsilon \in [0, 1]$. Often, ϵ is initialized at a high value to ensure early exploration and decayed to allow the RL algorithm to converge to a near-optimal policy. The subsequent subsections discuss the application of previously mentioned approaches to a model-free control problem.

Monte-Carlo Control

A natural way to define an algorithm for control is to collect end of episode returns and update the action-value function based on the visited state-action pairs during the episode. Similarly to the method in Subsection 3.1.3, Monte-Carlo control waits until the terminal state to update the value function at each visited state-action pair and then does a policy improvement step by acting greedily towards the updated action-value function.

SARSA

Another way to find the optimal policy is to use bootstrapping as defined in Subsection 3.1.3. Just like in the prediction case, this approach is able to learn online using the one-step or multi-step backups. Taking the TD-target to be the next-step predicted action-value, results in the following update rule [21]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)) \tag{3.24}$$

Where a' is the action chosen in the next state s' using the ϵ-$greedy$ policy implicitly derived from Q.

Q-Learning

Whereas SARSA uses the current policy to make the next step prediction, Q-learning [22] follows an off-policy architecture: the agent uses a behavioural policy $\mu(a \mid s)$ to make decisions but uses target policy $\pi(a \mid s)$ to evaluate the action-value function.

In this architecture, the update target uses the alternate action a' taken by the greedy target policy $\pi(s) = \underset{a'}{\text{argmax}}\, Q(s', a')$ instead of the behavioural policy $\mu(s)$ which chooses the exploratory actions to interact with the environment. The update rule defined in Eq. (3.24) then becomes the off-policy update given by Eq. (3.25):

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', \underset{a'}{\text{argmax}}\, Q(s', a')) - Q(s, a)) \tag{3.25}$$

Where action a is chosen using $\mu(s) = \epsilon$-$greedy\,(Q(s, a))$.

Q(λ)-Learning

Similarly to Subsection 3.1.3, the amount of bootstrapping can be controlled by considering multi-step updates and using the decay-rate parameter λ to control the amount of effect consecutive past states have on the received reward. For this method definition, only the backward view is considered which uses the concept of eligibility traces defined in Subsection 3.1.3 as a way to achieve the multi-step updates.

For each one-step transition in the episode, the eligibility trace of the visited state-action pair is incremented $E(s, a) \leftarrow E(s, a) + 1$ and the trace is decayed for all state-action pairs by the discount factor and the decay rate: $E(s, a) \leftarrow \gamma\lambda E(s, a)$ with $\gamma \in [0, 1]$ and $\lambda \in [0, 1]$. After observing the transition, the TD-error is the same as with Q-learning:

$$\delta = r + \gamma Q(s', \underset{a'}{\text{argmax}}\, Q(s', a')) - Q(s, a) \tag{3.26}$$

[3]Most commonly sampled uniformly.

However, the action-value function update step now depends on the eligibility trace and is done for all state-action pairs:

$$\begin{aligned} Q(s,a) &\leftarrow Q(s,a) + \alpha\delta E(s), && \forall \langle s,a \rangle \in \langle \mathcal{S}, \mathcal{A} \rangle \\ E(s,a) &\leftarrow \gamma\lambda E(s,a), && \forall \langle s,a \rangle \in \langle \mathcal{S}, \mathcal{A} \rangle \end{aligned} \tag{3.27}$$

3.1.5 Categorization of RL Methods

The approaches mentioned so far discussed several ways to design algorithms: whether on-policy or off-policy updates are used, whether an approach is model-free or model-based or whether end-of-episode rewards should be used as opposed to bootstrapping. This section discusses these aspects as a way to categorize RL methods and to provide a better overview of the variety of RL approaches.

Backup depth and width

The approaches listed above, namely Dynamic Programming, Monte Carlo and Temporal Difference approaches are all extreme variations of two parameters: the depth and the width of each update step. In the fourth corner of the extremes is Exhaustive Tree Search (ETS), which entails a brute-force search through the decision tree. This distinction between the four extremes is visualized in Fig. 3.4.

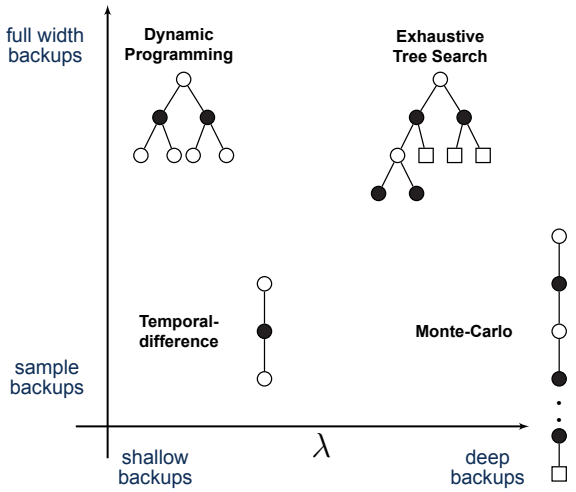


Figure 3.4: Adapted from [37]; overview of common approaches in terms backup depth and width.

The backup depth is tuned with the decay rate λ, as presented in Subsection 3.1.3, and is a measure of how many steps the agent takes before updating the value function of the visited states. On one end of the extreme is TD and DP, where an update of the current state is done after a single step; on the other end of the spectrum is MC and ETS where the agent must run through the entire episode to collect the unbiased return and complete the backup. In order to achieve the best sample efficiency, the decay rate hyperparameter often lies somewhere between the two extremes: $\lambda \in \ ]0, 1[$ [37].

The other parameter is the backup width and deals with the number of state-action choices to update in a single step across the breadth of the state-action tree. On one end of the extreme are the sample-based backups TD and MC, where only a single trajectory is taken into account; on the other end of the spectrum is DP and ETS, where all states and actions are backed up during a single step. Most often the MDP is given for the planning problem solved by DP and ETS, however due to computational complexity, sample based methods tend to be more applicable in practice.

Value-based and Policy-based methods

Previous sections have so far discussed value-based methods, where the policy of the agent is defined implicitly using an estimate of the value function $V(s)$ or action-value function $Q(s, a)$. As opposed to value-based methods, the class of policy-based methods use an explicit definition of the policy using a parametric function approximation and use iterative approaches like gradient descent to find the optimal policy directly.

Policy-gradient methods have better convergence properties, i.e. stable learning, and are effective for high-dimensional and continuous action space, however they often suffer from low sample efficiency, high variance and tend to converge to local optima. An additional option they offer is the ability to define both deterministic and stochastic policies. Policy-based methods are discussed in Subsection 3.2.3 in more detail.

To combine the benefits of both approaches, a third category can be constructed: the class of actor-critic methods, which try to find both an estimate of the value function (critic) and an estimate of the optimal policy (actor). Actor-critic methods are discussed in more detail in Subsection 3.2.4. This categorization approach is depicted in Fig. 3.5.

Figure 3.5: Adapted from [21]; classes of RL approaches based on agent type.

Model-free vs Model-based

Model-free RL methods learn the value function and/or the policy directly from gathered experience. Model-based methods learn a model approximation of the underlying MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ and use planning methods to construct a value function and policy. This integrates learning and planning into a single architecture. The primary advantage of this approach is that it efficiently learns a model by supervised learning methods and that it can utilize model uncertainty to define complex policies. The disadvantage of model-based methods is the added complexity and that another source of approximation error is introduced to the learning process.

When discussing model-free and model-based methods in the context of flight control, a distinction has to be made between the methodology of controller design and the model-usage of the RL method. *Model-free controller design* can be defined such that no system identification work is required beforehand; no a-priori model is provided for the RL learning process. On the other hand, *Model-based* RL methods use an internal representation of the MDP built from the agent's own experience and therefore fall within this definition of model-free controller synthesis.

As mentioned model-based RL methods add another layer of complexity to the RL architecture, however they allow the use Dynamic Programming as a planning step that uses the approximate MDP $\hat{\mathcal{M}}_\eta = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$, where η are the model parameters. Using sample-based forward-search algorithms provides increased sample efficiency by using imagined samples generating using the approximate internal model. Furthermore, a combination of both imagined and real-world samples can be used in architectures like Dyna-2 [38], which utilizes a long-term memory sampled from real-world experiences to estimate general domain knowledge and a short-term memory sampled from the model to estimate specific knowledge about the current situation.

On-policy vs Off-policy

Another aspect of RL categorization is the concept of on-policy vs off-policy learning, which has been briefly mentioned in Subsection 3.1.4 when discussing the off-policy Q-Learning method. Off-policy methods allow the agent to learn from transition samples generated by other agents or policies which enables

the re-use of gathered experience. This is essential in stabilizing Deep Reinforcement Learning (DRL) approaches as the non-IID transition samples made by the active behavioural policy must be decorrelated for the stochastic gradient descent optimization step. Furthermore, the replay and re-use of experience is beneficial to increase the sample efficiency of the algorithm via the re-use of samples.

3.1.6 Synthesis

This chapter presents the fundamental elements of RL and provides a framework to define an RL task, within a Markov Decision Process. Since MDPs have a recursive nature, concepts from Dynamic Programming can be adopted to a model-free setting to find optimal policies using Generalized Policy Iteration. The tabular methods presented are capable of solving relatively small-scale RL problems with countable finite state and action spaces. A brief categorization of these approaches was also presented which classifies agents based on how the transition samples are used, whether the agent estimates the action-value function or the policy directly or whether the agent uses an internal representation of the MDP.

The most important take-away for subsequent chapters is how temporal-difference errors can be used to learn from samples taken by the agent exploring the environment. Most real-world problems often have high-dimensional state and action spaces and RL methods must be generalized in order to scale these approaches to larger, more complex tasks. While the core concepts of using GPI and using bootstrapping remain widespread in many RL methods, state of the art algorithms solve this generalization by using deep learning concepts. Therefore, the following chapters elaborate on Deep Reinforcement Learning algorithms, in order identify methods that can overcome the challenge of solving the high-dimensional flight control task.

3.2 Deep Reinforcement Learning

The RL methods discussed can be applied to relatively small problems with discrete action and state spaces. However, in real-world applications the tabular approaches do not scale in neither their memory and computational complexity, nor their ability to generalize their solution across high-dimensional state and action spaces.

In order to scale the approaches mentioned in Section 3.1, several relatively recent advancements in adjacent machine learning techniques can be used to extend RL algorithms, such as the use of deep neural networks (DNNs) as universal function approximators and stochastic gradient descent (SGD) methods. The use of deep learning methods in combination with RL is commonly referred to as the field of deep reinforcement learning (DRL) and refers to a wide array of RL approaches.

As opposed to supervised learning, RL agents do not have access to true labels, therefore the experience gathered from interacting with the environment is used to gradually learn either the value function, the policy or both. When the action-value function is approximated it is often referred to as the *critic* of the RL architecture; when the policy is approximated directly, it is generally referred to as the *actor*. A combination of value-based and policy-based methods yields the actor-critic architecture that aims to utilize the advantages of both approaches.

The goal of this chapter is to present the variety of methods used in state of the art DRL research that aim at tackling large-scale real-world problems and to survey methods suitable for flight control tasks as posed by RQ 1. Firstly, function approximation approaches are introduced in Subsection 3.2.1 along with a brief summary of deep learning concepts. Secondly, value-based deep RL methods are discussed in Subsection 3.2.2 which presents recent advances in applying deep learning to RL methods along with improvements that increase the sample efficiency and performance of DRL agents. Then, the family of policy-based methods is presented in Subsection 3.2.3 where instead of estimating a value-function, the agent optimizes the policy directly. Lastly, the class of actor-critic methods is discussed in Subsection 3.2.4 followed by an overview of the chapter in Subsection 3.2.5.

3.2.1 Function Approximation and Optimization

As mentioned above, in order to apply RL to real-world scenarios, generalization is required due to the size of the state and action spaces and due to the complexity of real-world MDPs. Learning the value of each state-action pair individually is too inefficient in terms of both time and memory, therefore function approximation is needed to generalize either the value function or the policy directly.[4]

The true value function while following policy π is $V_\pi(s)$, while the value function approximate is denoted: $V(s;\theta)$, where θ is the parameter vector of the approximation. Similarly, the action-value function approximate can be written as: $Q(s,a;\theta) \approx Q(s,a)$.

There are numerous choices for function approximators, such as linear combinations of features or the use of artificial neural networks (ANNs) as universal black-box methods. It is preferable to use function approximators that are differentiable with respect to the parameter vector θ, so that gradient descent methods can be used to estimate the parameters.

Subsection 3.2.1 discusses widely used gradient descent methods, then the use of ANNs as function approximators are briefly presented in Subsection 3.2.1.

Gradient Descent

There are several approaches to estimate the parameters of function approximators. Incremental methods are popular among ML algorithms, such as gradient descent which iteratively reduce the error by updating the parameters in the direction that incrementally lowers a given loss function. For any differentiable objective function $J(\theta)$, the gradient can be defined using the notation:

[4] This section often discusses function approximation in the context of estimating the action-value function (critic) however the same concepts apply to policy-based and actor-critic methods.

$$\nabla_\theta J(\theta) = \left[\frac{\partial J(\theta)}{\partial \theta_1}, ..., \frac{\partial J(\theta)}{\partial \theta_n} \right]^T \tag{3.28}$$

Often the objective is formulated in terms of loss $\mathcal{L}$, and therefore the goal of gradient descent is to find a local minimum of the loss function $L(\theta)$. The loss function $L(\theta)$ measures how well the model predicts the true values the function estimate must adhere to. A simple gradient descent step can be written as shown in Eq. (3.29), where the parameter vector θ is incrementally moved following the gradient using step-size α for each time-step t:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{3.29}$$

Where the step-size hyperparameter α is also referred to as the *learning rate* and is often set to follow schedule, for instance a linear decay to ensure better convergence properties.

As opposed to supervised learning, RL methods do not have access to true labels at the start and therefore learn from labeled samples collected via interacting with the environment. The optimization of the loss function $J(\theta)$ must be done on a finite available training set. For this reason deep RL method commonly rely on *stochastic gradient descent* (SGD) methods which use finite samples of the gradient $\nabla_\theta J(\theta_t)$. This approach is common for high-dimensional problems, as it reduces the computational complexity of a single iteration at the cost of slower convergence.

Modern sophisticated optimizers like Adam [39] utilize stochastic gradient approaches with further optimizations added: namely root mean square propagation (RMSP) and gradient descent with momentum. For practical implementation of RL methods, the requirement to use such automated SGD methods is that the model must be differentiable with respect to the parameter vector.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are bio-inspired universal function approximators that use several *layers of neurons* with a large number of weighted connections. These weighted connections mimic the biological analog of neurons strengthening their connections and *activation functions* mimic the firing behaviour of biological neurons. The latter allows the use nonlinear activation functions which provides ANNs great nonlinear approximation power [40].

In general terms, ANNs are black-box models with a large number of parameters. Using multiple hidden layers renders them Deep Neural Networks (DNNs). Most often, deep RL methods use deep feed-forward networks that are referred to as Multi-layered Perceptrons (MLPs) [41, 42].

A single neuron unit of the network is shown in Fig. 3.6, where the trainable parameters of this particular connection are the weights $w \in \mathbb{R}^m$, where $x \in \mathbb{R}^m$ is the input vector of the layer. An additional bias parameter b is used to be able to shift the neuron activation by a constant. Often hidden neurons of ANNs are output-normalized in order to keep the values bounded against diverging values.

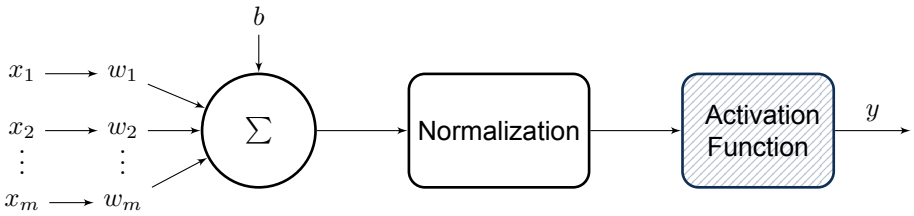


Figure 3.6: Adapted from [28]; Depiction of a single neuron unit, the weight and bias parameters and the optional normalization and activation functions.

These units are organized into layers and multi-layered networks, like the MLPs mentioned above. A two-layered feed-forward network is shown in Fig. 3.7, where all layers are fully connected.

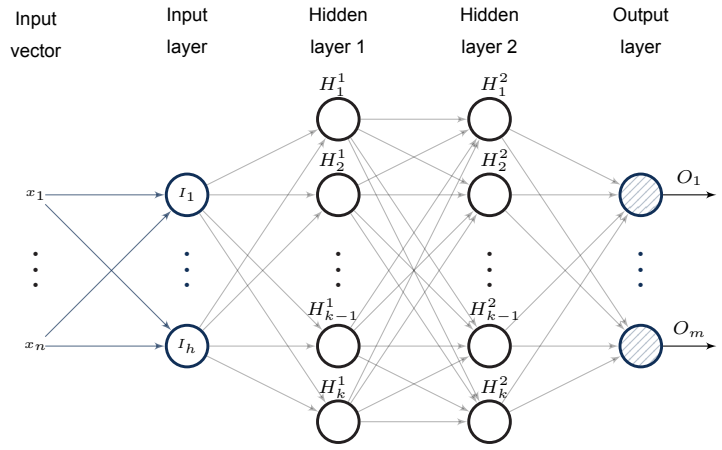


Figure 3.7: Depiction of a multi-layered perceptron (MLP) with two hidden fully-connected layers.

Activation Functions As mentioned before, a bio-inspired feature of ANNs is the fact that each unit neuron can be assigned certain activation functions which determine how and when certain neuron can 'fire', similar to their biological analog. Certain activation functions serve specific purposes, but generally they enable the nonlinear function approximation power of the ANNs.

One of the most common activation functions is a rectifier function, often called *Rectified Linear Unit* (ReLU). The ReLU activation function passes the positive part of its argument, as is shown in Eq. (3.30):

$$\text{ReLU}(x_i) = \max(x_i, 0); \quad \frac{\partial \text{ReLU}(x_i)}{\partial x} = \begin{cases} 1 & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.30}$$

The widespread use of ReLU activation functions in deep learning is due to its computational efficiency, scale-invariance and sparse activation. Some variations of ReLU either allow some parts of the negative arguments to be activated (Leaky ReLU, PReLU) or create a smooth transition between negative and positive arguments (Softplus).

The *Sigmoid* activation function is often used as a squashing function, to clip large values and to keep the layer responses bounded, as given in Eq. (3.31).

$$\text{Sig}(x_i) = \frac{e_i^x}{e_i^x + 1} \tag{3.31}$$

Another noteworthy activation function is the *Softmax* function given by Eq. (3.32), which is commonly used to normalize the output of a layer to denote a probability distribution over the possible outputs.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \quad x \in \mathbb{R}^K \tag{3.32}$$

Each component of the Softmax output vector are $f(x_i) \in [0, 1]$ and the sum of all components is 1, and can therefore be interpreted as a vector of probabilities.

Layer Types In deep learning methods, there are numerous layer types with their specific purposes. The choice of DNN layers is largely dependent on the types of input & output data that is available and the desired behaviour of the network.

The most common type of layer is the dense, *fully-connected linear layer*, where the output is easily differentiable w.r.t. both the input and the parameter vector as shown in Eq. (3.33). Linear layers are widely used due their universal approximation power and computational simplicity and efficiency.

$$f_i(x) = A_i x + b_i; \qquad \frac{\partial f_i(x)}{\partial A_i} = x; \qquad \frac{\partial f_i(x)}{\partial x} = A_i; \tag{3.33}$$

Normalization layers [43] focus on fixing the mean and variance of each hidden unit and bounding the input-output behaviour of the DNN, preventing too large or too small runaway values.

Convolutional layers are commonly used to process image data as their primary strength is detecting features and sub-features of smaller sub-grids, i.e. kernels of the image. A convolutional layer is an image convolution of the previous layer and the weights specify the behaviour of the resulting convolution filter. The use of convolutional layers in DNNs defines the class of *convolutional neural networks* (CNNs) that are most commonly used for image processing and computer vision. CNNs are often used in RL applications when the state of the MDP can be represented by pixel-data.

Pooling layers are commonly used to sub-sample image data for convolutional networks, by combining the sub-grid of data samples into a single value. Several methods of choosing that value exist, but a commonly used layer is a so-called max-pooling layer, that takes the maximum value of the sub-grid.

Recurrent layers add temporal behaviour to the model by adding memory to the network. Recurrent units may use variable lengths of stored state/memory and process signals dynamically which makes *recurrent neural networks* (RNNs) especially useful in the fields of speech and handwriting recognition. In the context of RL, the use of RNNs is limited due to the underlying Markov-process assumption, however they have recently been shown to be powerful in RL applications that involve partially observable MDPs [44, 45].

Synthesis For the scope of this research, fully-connected linear layers with non-linear activation functions and layer normalization are the most applicable architecture. Such feed-forward networks have great universal approximation power and are capable of estimating nonlinear action-value functions or policies. Since the continuous control problem revolves around receiving a state vector $\underline{x} \in \mathbb{R}^n$ and deciding on the control input vector $\underline{u} \in \mathbb{R}^m$, there is rarely a need to introduce recurrent memory or to handle image data.

Additionally, there are several common problems that deep learning approaches have to deal with in practice. Firstly, gradient descent methods often encounter the so-called *vanishing gradient* problem, when the local gradient becomes too small, preventing the network to further change its parameters. Secondly, *catastrophic forgetting* occurs when SGD methods applied to neural networks 'unlearn' data previously encountered, often caused by receiving sets of highly correlated samples. Lastly, a general problem of all function approximation methods is the occurrence of *over-fitting*, which occurs when the input-output behaviour of the model is perfect on the provided learning data set, but its generalization power is lost in-between samples.

3.2.2 Value-based Deep RL

This section discusses the class of deep RL methods that focus on estimating the action-value function, i.e. the Q-function using deep learning. Deep Q-Networks are introduced in Subsection 3.2.2 followed by a discussion of improvements in Subsection 3.2.2.

Deep Q-Networks (DQN)

In order to extend the off-policy Q-learning concepts discussed in Section 3.1 to DRL, so-called Deep Q-Networks (DQN) [24] can be used. DQN methods had shown super-human capability in Atari gaming environments [25], where the pixel-information of each frame was used to estimate the action-value function using a convolutional feed-forward network. The mentioned set of Atari games today serves as a standardized benchmark for deep RL algorithms [46].

The DQN algorithm is off-policy, similar to its tabular Q-learning equivalent discussed previously in Subsection 3.1.4. The action-value function is estimated by the parameterized approximator $Q(s, a; k) \approx Q(s, a)$, where the parameter vector is k. This report often simplifies the notation of the approximate action-value function $Q(s, a; k)$ to $Q_k(s, a)$.

The TD-error for DQN is similar to the Q-learning TD-update, with the action-value replaced with the parameterized Q-network estimates, shown in Eq. (3.34)

$$\delta = r + \gamma \max_{a'} Q_{\bar{k}}(s', a') - Q_k(s, a) \tag{3.34}$$

Where k is the parameter vector of behavioural Q-network and $\bar{k}$ is the parameter vector of the target Q-network and the action a is distributed according to a so-called *behaviour distribution* $a' \sim \mu(s, a)$, hence the off-policy nature of the algorithm.

In practice, the behaviour distribution $\mu(s, a)$ is often an ϵ-greedy action selection w.r.t. the Q-function estimate. The Q-value next-step prediction used in bootstrapping chooses the action greedily w.r.t. the Q-function, therefore in Eq. (3.34) the next-action is $a' = \underset{a'}{\text{argmax}}\, Q_{\bar{k}}(s', a')$.

In order to estimate the parameter vector k of the Q-network, a mean-squared error (MSE) based loss function $\mathcal{L}(k)$ can be defined that the SGD optimizer can use to train the network parameters and approach the true action-value function.

$$\mathcal{L}(k) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q_{\bar{k}}(s', a') - Q_k(s, a)\right)^2\right] \tag{3.35}$$

Where $\langle s, a, r, s' \rangle$ are samples from interacting with the environment, following behavioural policy $a \sim \mu(s, a)$. In practice however, two additional improvements are required to stabilize the SGD optimization process of the DNN approximator, namely the use of a fixed Q-target network and the use of experience replay.

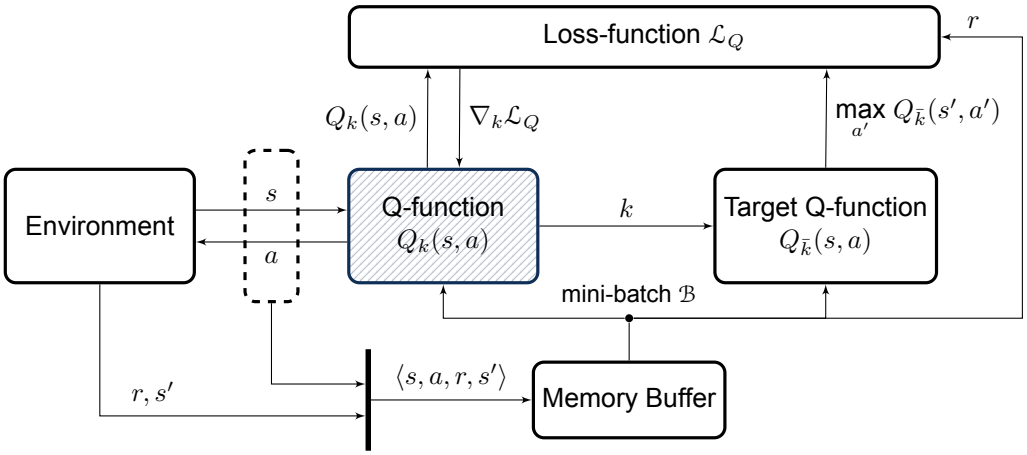


Figure 3.8: Depiction of the DQN algorithm

Fixed Q-targets Due to the off-policy behaviour of the algorithm, it is possible to stabilize the learning process by freezing the target Q-network parameters and only updating its parameters every N update-step. There are two methods used to update the frozen network: a hard update that synchronizes the parameters directly or a soft update which linearly interpolates the parameters of the target network towards the local network. The latter approach is commonly known as Polyak-averaging [47] and is shown in Eq. (3.36):

$$\bar{k}_{i+1} = \zeta \bar{k}_i + (1 - \zeta) k_i \tag{3.36}$$

Where $\zeta \in [0, 1]$ is the Polyak step-size, k is the parameter vector of behavioural Q-network and $\bar{k}$ is the parameter vector of the target Q-network.

Experience Replay Due to the process of decision making and the nature of the MDP, RL methods deal with time-series data where the signals are heavily correlated. Deep learning methods rely on IID data, which means the sampled experience from the agent must be decorrelated for each learning step.

The off-policy nature of DQN allows the stabilization of learning by collecting the experienced transitions into a memory buffer and sampling a random mini-batch $\mathcal{B}$ of transition samples for the stochastic gradient update step to improve the Q-function estimate. This concept is called *experience replay* and is widely used to stabilize the learning process for deep RL methods.

When the learning process is unstable, DNNs tend to struggle with the phenomenon of *catastrophic forgetting*. For instance, when the replay buffer is full and the policy has converged to a local optimum with no exploration, the buffer is filled with highly correlated data that only focuses on a limited state and action space due to the convergence of the policy. This causes the DNN to 'forget' the previously visited states that are at that point outside the buffer. In order to counteract such scenarios, it is essential to maintain a minimum exploration rate.

This report refers to nominal DQN with the two concepts of using fixed target Q-networks and using a memory buffer added to the RL architecture. This architecture is shown in Fig. 3.8, where the behavioural policy is implicitly defined by the local Q-function $Q_\theta(s, a)$. The transition samples $\langle s, a, r, s' \rangle$ are collected by the memory buffer where a uniformly sampled batch of transitions with size $|\mathcal{B}|$ is used to train the DNN the represents the Q-function using the MSE loss L_Q.

Extensions of DQN

This subsection discusses further extensions to the DQN method framework that focus on improving sample efficiency, stabilizing the learning process and improving the overall performance w.r.t. the goal of the agent.

Double DQN The nominal DQN algorithm is known to overestimate the action-value function. Double DQN (DDQN) architectures [48] overcome this overestimation bias by using the online network to evaluate the greedy policy but using the target network to estimate its value. This changes the TD-error to the following:

$$\begin{aligned} a' &= \underset{a'}{\text{argmax}}\, Q_k(s', a') \\ \delta^{\text{double}} &= r + \gamma \max_{a'} Q_{\bar{k}}(s', a') - Q_k(s, a) \end{aligned} \tag{3.37}$$

This formulation comes at the benefit of increased efficiency and reduction of overestimation bias without having to add additional networks / parameters.

Some state of the art deep RL algorithms use the double DQN architecture in a way where two separate networks are trained and a pessimistic minimum is taken over the two networks, followed by SGD optimization of both Q-networks. Such a double value estimate is used in the algorithms TD3 and SAC [49, 50], discussed below in Subsection 3.2.4.

Prioritized Experience Replay Nominal experience replay samples mini-batches from the memory buffer uniformly. Prioritized experience replay methods [51] improve upon the sample efficiency by replaying important transitions more frequently. The relative importance of a transition is approximated by using the magnitude of the TD-error.

Prioritizing transitions introduces diversity issues due to the fact that early visit to low TD-error transitions will have a lower chance to be encountered again and that greedy prioritization focuses on a small subset of experiences. In order to overcome diversity issues, *stochastic prioritization* is introduced to interpolate between greedy prioritization and random sampling. Additionally, prioritization changes the distribution of the stochastic updates that the value estimation relies on. This introduced bias is corrected by *importance sampling* which fully corrects for the change in distribution.

Applying prioritized experience replay improves the sample efficiency of DQN substantially and is complementary to the other extensions discussed in this subsection.

Dueling architectures It often occurs that the action-value function estimate is similar for two actions in the same state. In those situations choosing any of the high value actions would suffice and it is of interest how much better certain actions are relative to each other.

Dueling architectures [52] separate the action-value function into a state dependent action-advantage term and a state-value term, such that $Q(s,a) = A(s,a) + V(s)$. Using this formulation, the agent is able to learn which states are not valuable without having to learn the effect of each action in those states and therefore it is particularly effective for problems with large action spaces.

Rainbow The extensions discussed so far focus on speeding up and stabilizing learning. The approaches are modular improvements and can be combined into a single DQN-based method [53], which the authors call Rainbow. The Rainbow algorithm combines the double architecture, prioritized replay and the dueling architecture.

Furthermore, it adds learning from multi-step bootstrapping targets, adds stochastic layers to the Q-network to encourage exploration [54] and adds Distributional RL [29] to further increase sample efficiency. The latter is discussed in detail in Section 3.3. The combined architecture of [53] results in a state of the art off-policy RL method that can generalize the continuous state-spaces and choose between a set of discrete actions.

3.2.3 Policy-based Deep RL

Previously discussed value-based methods, also called critic-only approaches have to resort to an optimization procedure to determine the best action given the value-function estimate. This is computationally expensive and can be especially troublesome for continuous action spaces, therefore most value-based approaches use discretized action spaces [55].

Another approach is to parameterize the policy directly as $\pi(s,a;w) = \mathbb{P}\left[a_t = a \mid s_t = s; w\right]$ [5] [56], which leads to the class of policy-based approaches. Given that the policy is differentiable, i.e. $\partial\pi_w/\partial w$ exists, gradient descent methods can be used to find the optimal parameter vector w given the objective function $J(w)$ and its gradient $\nabla_w J(w)$.

The class of policy gradient methods come with several advantages over value-based approaches. Firstly, infinitesimal changes in the action-value function estimate often push the value of one action over another and therefore cause discontinuities in the implicitly defined policy and the overall performance [57]. Secondly, the direct parameterization allows the use of stochastic policies, which can be highly effective for certain types of environments, such as when certain states are aliased or when the MDP is partially observable. The third advantage is the fact that value-based methods must find the action corresponding to the maximum value which can be problematic for high-dimensional and/or continuous action spaces.

This section establishes the fundamentals of policy-based methods in Subsection 3.2.3, then briefly presents extensions and improvements to the policy gradient approaches in Subsection 3.2.3. The extensions discussed in this section are also commonly used within actor-critic architectures, i.e. they often make use of a value function estimate, but the algorithms themselves focus on improving the policy gradient approach and are therefore presented in an actor-only context.

Policy Gradient Theorem

The goal of policy gradient methods is to change the parameter vector w to find a local (or possibly global) maximum of the differentiable objective function $J(w)$. The objective is to increase the expected cumulative rewards received from the environment, which for state s_t at time-step t is the value function by definition: $V_\pi(s_t)$. Therefore, the policy gradient ascent step can be defined, similarly to Eq. (3.29), as the following:

$$w_{t+1} = w_t + \alpha \nabla_w V_\pi(s_t). \tag{3.38}$$

[5] Occasionally, this report simplifies the notation of the policy $\pi(s,a;w)$ to π_w.

Where α is the step-size and $\nabla_w V_\pi(s_t)$ is the policy gradient. The policy gradient theorem exploits the log likelihood ratio trick to define the above mentioned policy gradient in terms of the so-called *score function* $\nabla_w \log \pi_w$ [56]. The theorem is shown in Eq. (3.39):

$$\nabla_w V_\pi(s_t) = \mathbb{E}_{\pi_w}\left[\sum_{t=0}^{T} \nabla_w \log \pi_w(a \mid s) V_t\right] \tag{3.39}$$

Since the definition of the policy structure is up to the designer of the RL algorithm, the parameterization can be chosen such that the score function gradient can be analytically solved for and no numerical estimations are needed, such as estimating the gradient via perturbations.

The simplest actor-only RL algorithm that can be defined using the policy gradient is to use the end-of-episode returns as unbiased samples of the value V_t and update the parameters for every transition encountered throughout the episode. This forms the Monte-Carlo policy gradient method, also known as REINFORCE [58] and its pseudo-code algorithm is shown in Algorithm 1:

Algorithm 1: Monte-Carlo Policy Gradient (REINFORCE) [58]

Data: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle, N_{ep}$
Result: $\pi_w(s, a)$
for *episode* $i \leftarrow 1$ **to** N_{ep} **do**
  Initialize s
  while s *not terminal* **do**
    observe transitions
    $\{\langle s, a, r, s' \rangle, ...\} \sim \pi_w$
  end
  for $t \leftarrow 1$ **to** T **do**
    $w \leftarrow w + \alpha \nabla_w \log \pi_w(s, a) \cdot V_t$
  end
end

The resulting algorithm generally has great convergence qualities and learning stability at the cost of high variance and slow, inefficient learning. The following subsection provides examples of policy-based algorithms that improve upon the baseline concepts of policy gradient methods.

Policy Gradient Extensions

As mentioned policy gradient methods are inefficient and take a large number of samples to converge. Furthermore, when using DNNs to train a certain parameterized policy, regular policy gradient approaches are highly sensitive to the step-size hyperparameter α. Additionally, an early bad parameter update may cause the rest of the data received from the interaction to lead to further policy deterioration. In order to solve these issues, policy gradient extensions focus on increasing the efficiency and improving robustness by either constraining the policy gradient update step or optimizing the step-size.

Trust region Policy Optimization (TRPO) Trust Region Policy Optimization (TRPO) [59] improves the robustness of policy gradient methods by constraining the policy update step using the Kullback-Leibler (KL) divergence [60] between the current policy and the improved policy. The algorithm denotes *trust regions* as regions of the state-action space where the local function approximation is accurate.

In addition to the KL-divergence constrain, TRPO makes use of the advantage function, previously mentioned in the context of dueling architectures in Subsection 3.2.2. The state-dependent action-advantage function is shown in Eq. (3.40) and describes the difference in value of choosing a certain action compared to choosing the action of the policy.

$$A(s, a) = Q(s, a) - V(s) \tag{3.40}$$

Let $\hat{A}$ denote an approximator of the advantage function $A(s, a)$. The constrained update rule of TRPO is shown in Eq. (3.41):

$$\begin{aligned} \underset{w}{\text{maximize}} \quad & J(w) = \mathbb{E}\left[\frac{\pi_w(a \mid s)}{\pi_{w_{old}}(a \mid s)}\hat{A}\right] \\ \text{subject to} \quad & \mathbb{E}\left[D_{KL}\left(\pi_{w_{old}} \mid\mid \pi_w\right) \leq \delta\right] \end{aligned} \tag{3.41}$$

Where w_{old} is the parameter vector before the update and D_{KL} is the KL-divergence. The solution to such an optimization involves a conjugate gradient algorithm and linear & quadratic approximations of the objective function and constraint respectively.

Proximal Policy Optimization (PPO) In practice, the optimization problem with the KL-divergence constraint in TRPO is computationally expensive to solve. Instead of using a hard constraint, the theory of TRPO in [59] allows the use of a penalty term, such that:

$$\underset{w}{\text{maximize}} \quad \mathbb{E}\left[\frac{\pi_w(a \mid s)}{\pi_{w_{old}}(a \mid s)}\hat{A}\right] - \eta\mathbb{E}\left[D_{KL}\left(\pi_{w_{old}} || \pi_w\right)\right] \tag{3.42}$$

Where η denotes some tunable coefficient. In practice however, it has been found that it is difficult to choose a single η such that the policy update performs well across multiple learning task or even across the duration of the same learning task.

Proximal Policy Optimization (PPO) [61] increases the computational efficiency w.r.t. TRPO by removing the KL-divergence and improves upon the ease of implementation and tuning. The changed objective function used in PPO is shown in Eq. (3.43):

$$J_{CLIP}(w) = \mathbb{E}\left[\text{min}(r(w) \cdot \hat{A},\ \text{clip}\left(r(w), 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}\right] \tag{3.43}$$

Where the notation $r(w)$ is used to denote the probability ratio: $r(w) = \frac{\pi_w(a \mid s)}{\pi_{w_{old}}(a \mid s)}$, and ϵ is a coefficient that clips the probability ratio within the bounds $[1-\epsilon, 1+\epsilon]$. The minimum operator is chosen to take a pessimistic bound of the objective.

3.2.4 Actor-Critic Methods

The class of actor-critic methods aim to combine the advantages of both value-based and policy-based methods, where the policy approximator is referred to as the *actor* and the value function approximator as the *critic* [62]. The primary advantage of including the actor is to be able to discard the optimization procedure of finding the best action, whereas the critic provides knowledge about the performance of the agent. The action-value function estimate of the critic helps reduce the variance of the gradient, which speeds up the learning process and improves on the convergence characteristics.

The ability to generalize not-only over the state space but the action space has made actor-critic methods widespread in RL and many of the algorithms are considered state of the art in their ability to solve real-world problems. Actor-critic methods are derived from the concept of Generalized Policy Iteration (GPI) discussed in Subsection 3.1.2, which consists of a policy evaluation step and a policy improvement step, based on the obtained value function. The agent-environment interaction for actor-critics is shown in Fig. 3.9.

Deep Deterministic Policy Gradient (DPG)

As opposed to the stochastic policy gradient methods discussed in Subsection 3.2.3, the policy can also be defined in a deterministic fashion, such that: $a = \pi_w(s)$. This formulation is called Deterministic Policy Gradient (DPG) and was formulated in an actor-critic architecture [63]. DPG dramatically improves on efficiency compared to stochastic policy gradient methods, however uses an off-policy formulation where

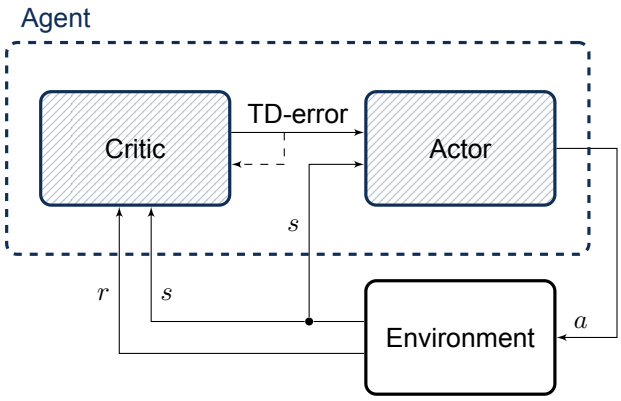


Figure 3.9: RL Agent interacting with the environment in an actor-critic architecture.

the behavioural policy uses an added noise to ensure sufficient exploration. This approach is analogous to Q-learning [64], as both methods learn a deterministic policy, while following the behaviour of a stochastic version of the greedy policy to explore [63].

DPG relies on the critic's value function approximate by updating the parameters along the action-value gradient. This parameter update is done by applying a chain rule to the expected return with respect to the actor parameters, as shown in Eq. (3.44):

$$\Delta w \propto \frac{\partial Q_\pi(s,a)}{\partial w} = \frac{\partial Q_\pi(s,a)}{\partial \pi_w(s)} \frac{\partial \pi_w(s)}{\partial w} \tag{3.44}$$

DPG is therefore also commonly known as *gradient-ascent on value* and this parameter update concept is equivalent [65] to *action-dependent heuristic dynamic programming* (ADHDP)[6] [66].

In order to eliminate function approximation bias, DPG limits the critic to use linear approximators. This guarantees that the policy gradient update step will follow the true gradient, however this limits the nonlinear generalization power of the algorithm. Deep deterministic policy gradient (DDPG) [67] extends DPG by using DNNs as function approximators and is the deep RL variant of DPG [63].

Similarly to DPG, DDPG uses a Q-function to enable off-policy learning and uses a deterministic actor that maximizes this Q-function. DDPG makes use of the DNN stabilization techniques used by nominal DQN [24, 25], namely the addition of experience replay and a fixed target network. This combination of DQN and DPG allows the algorithm to generalize not only over large observation spaces, but also continuous action spaces.

However, this interplay between the deterministic actor and the Q-function makes DDPG difficult to stabilize and makes it sensitive to hyperparameter tuning. Two independent improvements have been made to the DDPG algorithm: namely the twin-delayed deep deterministic policy gradient (TD3) [49] and the soft-actor critic (SAC) [50, 68] algorithm.

TD3 [49] improves upon DDPG in three aspects. Firstly, double DQN architecture (twin) is used to improve on the overestimation bias discussed in Subsection 3.2.2. The minimum of two action-value function estimates is used in the TD-update as a pessimistic estimation. Secondly, delayed policy updates are added, such that the Q-function approximator updates are more frequent than the policy gradient step updates, in order to improve stability and convergence. Lastly, so-called *target policy smoothing regularization* is added as a means to tackle over-fitting of the policy and is achieved by adding noise to the otherwise deterministic actions of the policy.

Both TD3 and SAC are considered state of the art DRL methods that can generalize over high-dimensional continuous state and action spaces and have similar learning and tracking performances. A benchmark by [69] has found SAC to outperform TD3 in terms of sample efficiency in most complex control tasks. Furthermore, SAC models the distribution of actions which makes it an attractive architecture for studying distributional RL.

[6] ADHDP is a model-free variation of Adaptive Critic Designs (ACDs), discussed in more detail in Subsection 3.4.2.

Soft Actor-Critic

Soft actor-critic (SAC), originally proposed by [50] to deal with large continuous action spaces, is an extension of DDPG that uses stochastic policies to enable better exploration and to prevent premature convergence to local optima. Similarly the DDPG and TD3, SAC is off-policy and makes use of experience replay and fixed Q-networks in a double architecture to reduce overestimation bias. SAC uses to concept of maximum entropy RL that aims at maximizing not only the action-value function, but also the entropy of the policy and introduces the so-called soft action-value function.

Maximum entropy RL The formulation of SAC is derived from *Soft Policy Iteration* (SPI), a general algorithm that uses the maximum entropy framework. An additional maximum entropy objective is defined [70] that augments the expected return with an an additional entropy term $\mathcal{H}(\pi)$. In this framework the optimal policy π_* aims to maximize not only the expected cumulative reward, but also the entropy at each visited state. Such maximum entropy methods result in significantly more stable and scalable algorithms [68]. The entropy term serves as a metric of randomness in the probability distribution of the policy and shows how much the policy is spread out over the action-space. The entropy is given by the log-likelihood of the policy, as shown in Eq. (3.45):

$$\mathcal{H}(\pi_w(\cdot \mid s)) = \mathbb{E}_{a' \sim \pi_w}\left[-\log\left(\pi_w(a' \mid s)\right)\right] \tag{3.45}$$

The temperature parameter term η is introduced that scales the entropy term $\mathcal{H}(\pi)$ to balance between the prioritization of future rewards and entropy and therefore adjusts the stochasticity of the optimal policy. Then, maximum entropy RL defines the optimum policy π_* as shown in Eq. (3.46):

$$\pi_* = \underset{\pi}{\text{argmax}}\, \mathbb{E}\left[\sum_{i=0}^{N} \gamma^i (R(s_{t+i}, a_{t+i}) + \eta \mathcal{H}(\pi(\cdot \mid s_{t+i})))\right] \quad \forall s_t \in \mathbb{R}^m \tag{3.46}$$

Since the maximum objective of entropy RL in this framework differs from the objective of traditional RL, the conventional objective can be recovered as $\eta \to 0$ [68].

Critic As mentioned, SAC adapts several stabilizing concepts from value-based methods in order to approximate the action-value function $Q(s, a; k)$, where k is used to denote the parameter vector of the critic. Firstly, experience replay is added to sample mini-batches $\mathcal{B}$ of transitions to decorrelate the training data. Secondly, double Q-function architecture is used to prevent the overestimation of the action-value function. This means that two function approximators are trained in parallel with subscripts $l = 1, 2$: $Q_{k_{1,2}}(s, a)$ to estimate the value function and the minimum over the two estimates is taken to determine the TD-error. Thirdly, the target Q-network is fixed w.r.t. the behavioural Q-network and Polyak averaging is used with step-size ζ to interpolate the parameters of the target-network towards the local network. The parameter vector of the target network is denoted with $\bar{k}$. The loss-function of the critic is an MSE loss over the mini-batch of transition samples using the off-policy TD-error and is given in Eq. (3.47):

$$\mathcal{L}_Q^{\mathcal{B}}(k_l) = \mathbb{E}\left[\left(Q_{k_l}(s, a) - \left(r + \gamma(\min_{l=1,2} Q_{\bar{k}_l}(s', a') - \eta \log \pi_w(a' \mid s'))\right)\right)^2\right] \tag{3.47}$$

Where the transition $\langle s, a, r, s' \rangle$ is sampled from $\mathcal{B}$, the next action a' is sampled from the policy $\pi_w(\cdot|s)$, the term $Q_{k_i}(s, a)$ is the local action-value function estimate and $Q_{\bar{k}_i}(s', a')$ is one-step ahead prediction to calculate the TD-error, which is corrected by the entropy term $\eta\mathcal{H}(\pi_w(a' \mid s'))$.

Policy SAC uses a stochastic policy to ensure better exploration, namely an m-dimensional multivariate Gaussian distribution with a diagonal covariance matrix. The actions of the policy are passed through a *tanh* squashing function to ensure they are defined on a finite bound. The mean vector and the covariance matrix (in this case, covariance vector) are estimated for the entire state space by a DNN, usually an feed-forward MLP with parameter vector θ, such that output of the DNN is: μ_θ and σ_θ as shown in Fig. 3.10:

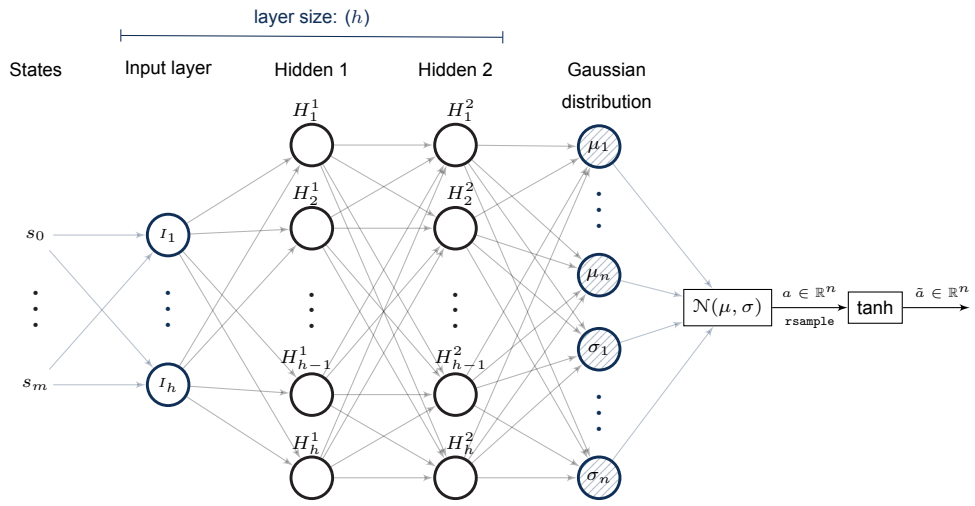


Figure 3.10: Depiction of the m-dimensional multivariate tanh-Gaussian stochastic policy network used by the SAC algorithm.

The actions of the policy are then sampled from this Gaussian distribution, however in order for the objective function, as given in Eq. (3.46) to be differentiable a reparameterization of the sampling of actions is needed [68], by using an additional independent noise vector. In practice, automatic gradient calculation tools (such as Pytorch [71]) provide the reparameterization needed to sample from the normal distribution, so the noise vector ξ defined in [68] is omitted in this report for notational simplicity. Then, the actions chosen by the policy can be written as:

$$\tilde{a}_w(s) = \tanh(a_w(s)) \quad \text{with} \quad a_w(s) \sim \mathcal{N}(\mu_w(s), \sigma_w(s)) \tag{3.48}$$

The policy loss function can be formulated based on Eq. (3.46) by using the critic's estimate of the action-value function as shown in Eq. (3.49):

$$L_\pi^{\mathcal{B}}(w) = \mathbb{E}\left[\eta \log \pi(\tilde{a}_w(s) \mid s) - \min_{l=1,2} Q_{k_l}(s, \tilde{a}_w(s))\right] \tag{3.49}$$

Where s is sampled from mini-batch $\mathcal{B}$, $\tilde{a}_w(s)$ is the action sampled from the distributional policy and η is the entropy temperature.

Adaptive Entropy Temperature It was found that the nominal SAC algorithm is brittle w.r.t. the temperature parameter η [50]. A proposed adaptive temperature adjustment can be made [68, 28] to dynamically find the lower temperature that aims to achieve a target entropy. The temperature parameter can be found using the following loss function:

$$\mathcal{L}^{\mathcal{B}}(\eta) = -\mathbb{E}\left[\eta \cdot \left(\log \pi_w(a \mid s) + \bar{\mathcal{H}}\right)\right] \tag{3.50}$$

Where $\bar{\mathcal{H}}$ is the target entropy that has empirically been found to be set such that $\log \bar{\mathcal{H}} = -m$, where m is number of action-space dimensions, i.e. $\mathcal{A} \subset \mathbb{R}^m$.

SAC Architecture Combining the discussed policy, critic and adaptive temperature into an RL algorithm results in the architecture shown in Fig. 3.11. Similar to other deep RL methods, an experience replay memory buffer is used to decorrelate the transition samples, thus providing a mini-batch of $\langle s, a, r, s' \rangle$ transitions to both the critic and the actor.

Application-oriented research has shown the SAC algorithm to be particularly effective at continuous control & flight control tasks and is discussed in more detail in Section 3.4.

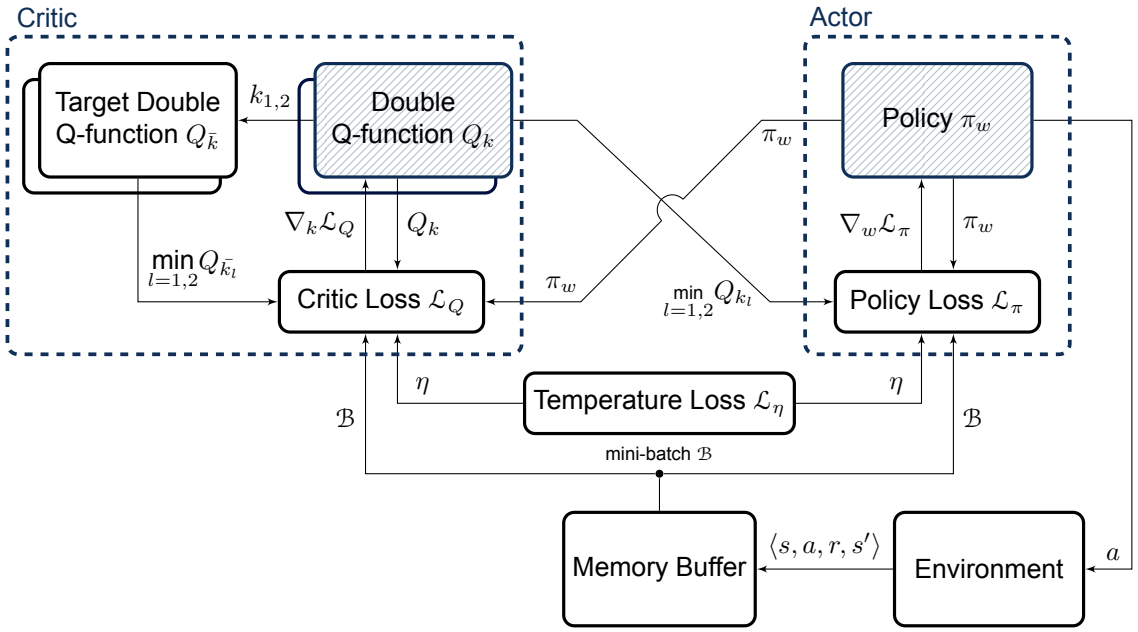


Figure 3.11: Adapted from [28]; Depiction of the SAC algorithm.

3.2.5 Synthesis

This chapter gave an overview of the most commonly used, state of the art deep reinforcement learning approaches. The generalization of value functions and policies has given RL the ability to tackle problems that had previously been thought unsolvable. Recent advancements in RL have shown its ability to find near-optimal solutions to real-world tasks in robotics and automation using a model-free framework. Many improvements to the original Deep Q Network [24, 25] method have been developed that not only speed up and stabilize learning but also improve on the final performance of the trained agent, which is shown in the Rainbow agent [53] that combines the most widely used extensions.

Purely value-based, critic-only methods tend to struggle with the high-dimensional, continuous action-spaces due to the optimization step required to find the best action, and they often resort to a discrete set of action. Policy-based, actor-only methods directly parameterize the policy and enable the use of continuous state-action spaces, which is crucial for complex control tasks. Actor-only policy gradient algorithms have stable learning properties, however struggle with inefficiency, robustness and high variance.

Actor-critic architectures combine the best features of the two classes and use the value estimate of the critic to improve the learning characteristics of the actor. Recent algorithms derived from DDPG [67], namely TD3 [49] and SAC [68] have shown state of the art efficiency and effectiveness at complex tasks in high-dimensional continuous state and action spaces and are therefore identified as candidate algorithms suitable for flight control tasks, as posed by RQ 1.

The classes of RL methods discussed so far have all focused on improving the expected cumulative reward, whereas the action-value function is an expectation of a probability distribution that can be exploited to further increase sample efficiency and to introduce the ability of applying risk-sensitive policies. The following chapter discusses RL methods that estimate the full distribution of the action-value function.

3.3 Distributional Reinforcement Learning

Traditional reinforcement learning frameworks use the expectation of cumulative future rewards as the primary method of value representation for the learning task. The value function is an expected value taken over all sources of randomness while interacting with the often intrinsically random environment. The field of distributional RL captures not only the first moment, but the entire distribution of the action-value function [29].

An often used analogy is comparing the difference between traditional and distributional value representation to that of a gray-scale image and a colored image [72]. The hue of each pixel in the colored image contains more information that otherwise cannot be recovered from the luminance signal alone, however the black and white image can easily be obtained from the colored photo. Similarly, the expectation of any distribution is easily obtained, but not vice versa.

The reasons to study distributional RL for flight control is two-fold. Firstly, numerous studies have shown empirical results that distributional methods produce state of the art learning performance and increased sample efficiency [29, 30, 73]. This difference in performance has been shown to be especially beneficial when using nonlinear approximation [74]. Secondly, return distributions allow the use of a variety of risk-sensitive policies that are crucial to achieve safety in RL-based flight control. Implicit representation of the return distribution [31] allows a simple way to apply risk-distortions to skew the risk-tendency of the agent.

Distributional RL methods are characterized by three things: the parameterization of the return distribution, the distance metric that the algorithm uses to optimize the parameters and the risk-sensitivity method applied to either the learning process or the post-training strategy. The purpose of this chapter is to provide an overview of existing distributional RL methods and identify candidate algorithms to be used for flight control tasks, as posed by RQ 2.

First, the fundamentals and the core algorithms of distributional RL are presented in Subsection 3.3.1. Secondly, Subsection 3.3.2 expands on the risk-sensitive methods that can be used in conjunction with distributional algorithms. Then, an overview of additional methods is provided in Subsection 3.3.3 with a focus on applying distributional RL to continuous control settings. Lastly, synthesis is provided on the findings of this chapter in Subsection 3.3.4.

3.3.1 Distributional RL Algorithms

The field of RL has put most of its efforts into modelling the mean of the cumulative rewards and therefore maximizing the expectation of the random return. However, distributional RL extends this axiomatic approach, as modelling only the expectation does not account for many complex phenomena in an RL task. Value function representations in traditional RL methods are scalars for a given state, whereas distributional algorithms operate on collections of probability distributions, so-called *return distribution functions*.

In traditional RL, where real-valued vectors are used to represent the value function, the quality of an agent's prediction can be measured by an absolute difference as a distance metric. With return distribution functions, a probability metric has to be defined in order to quantify the distance between distributions. The choice of metrics is important, as it is desirable for the distributional Bellman operator to be a contraction using the chosen distance metric between distributions.

This section discusses the core distributional RL methods that arise from the several ways the return distribution can be parameterized and the kind of distance metric and loss function used to apply the Bellman updates. First, the distributional version of the Bellman equation is provided in Subsection 3.3.1. Then, several core algorithms are discussed in Subsection 3.3.1 that established practical implementations of the distributional RL framework. Lastly, Subsection 3.3.1 provides a brief overview of the core distributional RL algorithms.

Distributional Bellman Equations

As presented in Subsection 3.1.1, the Bellman equations represent the expected value of a state in a compact way utilizing the recursive nature of the underlying MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. The *return*, similarly defined in Eq. (3.5) is the sum of discounted future rewards while following policy π.

$$Z(s_t, a_t) \coloneqq \sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i}) \tag{3.51}$$

The notation Z is used to denote the action-value distribution function that maps $Z : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$, where $\mathcal{Z}$ is the action-value distribution space. The space $\mathcal{Z}$ is defined with finite moments for all state-action pairs, formally written in Eq. (3.52):

$$\mathcal{Z} = \left\{ Z : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathbb{R}) \mid \mathbb{E}\left[\|Z(s,a)\|_p \right] < \infty, \forall (s,a), p \geq 1 \right\} \tag{3.52}$$

Traditional RL defines the action-value function $Q_\pi(s,a)$ as the expected cumulative reward collected along a certain trajectory following policy π and aims at maximizing this expectation. With the sum of discounted rewards given by the distribution function $Z_\pi(s,a)$ as shown in Eq. (3.51), the value $Q_\pi(s,a)$ is the first moment of Z, i.e. the expectation:

$$Q_\pi(s,a) \coloneqq \mathbb{E}\left[Z_\pi(s,a) \right] \tag{3.53}$$

In essence, distributional methods take the approach of removing the expectation and consider the full return distribution Z_π instead of the action-value Q_π. Using the previously defined Bellman equation in Eq. (3.13), its distributional version can be formulated, however one must utilize the concept of *equality by distribution*, such that two random variables Z_1, Z_2 are equal in distribution when their probability distributions are equal:

$$Z_1 \overset{\mathcal{D}}{=} Z_2 \quad \iff \quad \mathbb{P}\left[Z_1 \in U \right] = \mathbb{P}\left[Z_2 \in U \right] \qquad \text{for any } U \subseteq \mathbb{R} \tag{3.54}$$

This formulation is necessary to be able to compare random variables solely based on their probability distributions and avoid directly equating random variables. Then, using the definition of Z_π given above, the distributional Bellman equation can be defined as the following:

$$Z_\pi(s,a) \overset{\mathcal{D}}{=} R(s,a) + \gamma Z(s', a') \tag{3.55}$$

Where the sample transition model $\langle s, a, r, s' \rangle$ is used to denote a single transition in the MDP trajectory, with $s' \sim \mathcal{P}(\cdot \mid s, a)$, $a \sim \pi(\cdot \mid s)$ and $a' \sim \pi(\cdot \mid s')$. Considering the optimal policy $\pi_*(s)$, the distributional version of the Bellman optimality equation can also be defined, shown in Eq. (3.56).

$$Z(s,a) \overset{\mathcal{D}}{=} R(s,a) + \gamma Z\left(s', \underset{a'}{\text{argmax}}\, \mathbb{E}_{\mathcal{P},R}\left[Z(s', a') \right] \right) \tag{3.56}$$

The return distribution captures three sources of randomness: the randomness of the reward function R, the stochastic transition $\mathcal{P}$ and the randomness of the next return distribution $Z(s', a')$. Similar to traditional RL, the distributional optimality equation can be used to define distributional RL algorithms that make use of generalized policy iteration to find the optimal policy using model-free control concepts.

In order to define distributional RL algorithms, two additional formulations are needed: the way the return distribution function is parameterized, and the way the distance metric is defined between two distributions. The algorithms detailed in subsequent subsections primarily differ in these two aspects.

Distance metrics The choice of the distance metric is essential for the agent to converge on a near-optimal policy. In [29] it was shown that the distributional Bellman update is a contraction operator, i.e. the repeated application of the operator converges to a fixed-point when the p-Wasserstein metric is used as a distance metric between distributions. Occasionally, the Wasserstein metric is also called the earth-mover metric as it indicates the minimum energy cost needed to convert one 'pile' of a distribution into another, i.e. the amount of earth that needs to be moved times the distance.

The p-Wasserstein metric is the L_p metric on inverse cumulative distribution functions (c.d.f.) [31]. The c.d.f. F_U of the random variable U describes the probability: $F_U(u) = \mathbb{P}[U \leq u]$. Its inverse, $F_U^{-1}(p)$ often called the *quantile function* gives the unique real number u, such that $F_U(u) = p$. The p-Wasserstein metric is given by Eq. (3.57), where U and V are random variables with quantile functions F_U^{-1} & F_V^{-1} and $\|\cdot\|_p$ is the p-norm metric:

$$W_p(U,V) = \left(\int_0^1 \left\| F_U^{-1}(\omega) - F_V^{-1}(\omega) \right\|_p d\omega \right)^{1/p} \tag{3.57}$$

Even though the use of the p-Wasserstein distance metric was shown to be a contraction, some distributional RL algorithms make use of the KL-divergence as a distance metric in their practical implementations. The KL-Divergence is given in Eq. (3.59) for both discrete and continuous random variables:

$$D_{KL}^D(P^D \,||\, Q^D) = \sum_{x \in \mathcal{X}} P^D(x) \cdot log\left(\frac{P^D(x)}{Q^D(x)} \right) \tag{3.58}$$

$$D_{KL}^C(P^C \,||\, Q^C) = \int_{-\infty}^{\infty} p(x) \cdot log\left(\frac{p(x)}{q(x)} \right) dx \tag{3.59}$$

Where P^D and Q^D are discrete probability distributions on the same probability space, P^C and Q^C are continuous random variables and p and q are the probability density functions of P^C and Q^C respectively.

Subsection 3.3.1 provide a brief overview on how the core RL algorithms implement the value distribution representation, distance metric and loss function.

Categorical DQN

Early distributional RL research have used Gaussian parameterization of the return distribution [75] and applied the Q-learning and SARSA frameworks to approximate value distribution functions. In [29], the distributional representation is expanded to a categorical approach and is combined with value-based deep RL concepts.

The categorical representation is described in the formulation of the algorithm C51 [29] where $N \in \mathbb{N}$ ($N = 51$) discrete atoms are used within the domain $[V_{\text{min}}, V_{\text{max}}]$ to approximate the distribution function, where the supports are defined using Eq. (3.60) and their parameterized probabilities are defined using Eq. (3.61).

$$z_i = V_{\text{min}} + i \cdot \Delta z, \qquad \text{where } \Delta z = \frac{V_{\text{max}} - V_{\text{min}}}{N-1} \tag{3.60}$$

$$q_i(s,a) \coloneqq \frac{e^{k_i(s,a)}}{\sum_j e^{k_j(s,a)}} \tag{3.61}$$

This discretization of the approximate distribution is computationally efficient however requires an additional projection step, as the supports become disjointed between two Bellman updates. The C51 method [29] uses the DQN architecture with the categorical representation approach and calls the framework *Categorical DQN*. Similarly to DQN [25], ϵ-greedy policy is used. The Bellman update is computed for each atom z_j and a projection step is done to align the supports.

While [29] presented proof of contraction of the distributional Bellman operator under the p-Wasserstein metric defined in Eq. (3.57), the C51 algorithm itself did not directly make use of this metric to define the loss function, and instead used KL-divergence as a metric between the Bellman updated distribution $Z_{\tilde{k}}(s,a)$ and the old distribution $Z_k(s,a)$.

The approach presented in [29] established the fundamentals of distributional RL and showed state of the art performance on Atari benchmarks, i.e. the Arcade Learning Environment (ALE) [76]. Even though

the distributional information is available, the C51 algorithm maximizes the expected value much like nominal DQN. Therefore, the author attributed the observed sample efficiency and performance improvements solely to the addition of the distributional representation.

Quantile Regression

An improved framework is defined in [30], where *quantile regression* (QR) is introduced to stochastically adjust the return distribution to minimize the Wasserstein distance towards the target distribution. The parameterization of the distribution is changed from variable probabilities of N fixed locations, as in Eq. (3.60) and Eq. (3.61), to uniform probabilities of N variable locations. This discrete representation is a uniform mixture of N Diracs and is given in Eq. (3.62). A visual representation of the discrete quantile representation is shown below in Fig. 3.13.

$$Z_k(s,a) \overset{\mathcal{D}}{=} \frac{1}{N}\sum_{i=1}^{N} \delta_{k_i(s,a)} \tag{3.62}$$

Where δ_x denotes a Dirac at location $x \in \mathbb{R}$. This *quantile distribution* Z_θ assigns uniform weights $q_i = 1/N$ for $i = 1, ..., N$ and denotes the discrete cumulative probabilities of the c.d.f. by $\tau_i = 1/N$ for $i = 1, ..., N$. This formulation does not require a projection step, as with C51 and no domain knowledge has to be added via the parameters $[V_{\text{min}}, V_{\text{max}}]$.

Lastly, quantile regression can be used to minimize the 1-Wasserstein distance between the distribution at time-step t and the Bellman updated distribution. The quantile regression framework in [30] is adopted from value-at-risk economic settings and is proposed to train the quantile estimations using a Huber quantile regression loss presented in Eqs. (3.63) to (3.66).

$$\mathcal{L}^{\tau}_{\text{QR}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\rho^{\kappa}_{\tau_i}(r + \gamma\, k_j(s',a') - k_i(s,a))\right] \tag{3.63}$$

$$\text{with} \quad a' = \underset{a'}{\text{argmax}} \sum_{i=1}^{N} k_i(s',a') \tag{3.64}$$

Where $\mathcal{L}^{\tau}_{\text{QR}}$ is the QR loss at time-step t, the next action is selected according to Eq. (3.64) and $\rho^{\kappa}_{\tau_i}$ is given by the following:

$$\rho^{\kappa}_{\tau_i}(x) = |\tau_i - \mathbb{I}\{x < 0\}| \cdot \mathcal{L}^{\kappa}(x) \tag{3.65}$$

Where $\mathbb{I}$ is the indicator function and $L^{\kappa}(x)$ is the Huber-loss with parameter κ presented in Eq. (3.66).

$$L^{\kappa}(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| \leq \kappa \\ \kappa\left(|x| - \frac{1}{2}\kappa\right) & \text{otherwise} \end{cases} \tag{3.66}$$

Minimizing this loss for each quantile τ_i and parameter k_i via SGD results in the minimization of the 1-Wasserstein distance. The Huber-loss defined in Eq. (3.66) is a squared loss in the interval $[-\kappa, \kappa]$, an absolute loss otherwise and retains smoothness where $x = \kappa$. It is used in [30] in the quantile regression framework to ensure smoothness as $x \to 0^+$.

Equation (3.63) and Eq. (3.64) describe the distributional Q-learning update step, where the next action a' is the greedy action w.r.t the mean of the next value distribution. Lastly, the DQN architecture is adopted to establish the QR-DQN distributional algorithm.

Implicit Quantiles

The approach in [31] extends QR-DQN from learning a discrete set of quantiles to learning the full continuous quantile function of the return distribution. By sampling each quantile τ_i from a uniform distribution $U([0, 1])$ and passing it through an *implicit quantile network* (IQN), a continuous mapping is created between probabilities and return. This generalization does not depend on the discrete N number of quantiles, but rather the size and training of the network itself and adds ease of tuning on the amount τ samples used per update. Lastly, this formulation makes it simple to implement risk-sensitive policies by adding a distortion mapping to the uniform distribution the samples are taken from. Such distortion functions are discussed in Subsection 3.3.2.

The IQN deep neural network is a deterministic parametric function trained to reparameterize the τ quantile samples from a base distribution $\tau \sim U([0, 1])$ to respective quantile values of a target distribution. The framework in [31] also adapts the DQN architecture and uses the off-policy TD-targets to train the network. The independently sampled quantiles are passed through a so-called cosine-embedding layer in the following way: $\phi_j(\tau) = \sum_{i=0}^{C-1} cos(\pi i \tau) w_{ij} + b_j$, where C is the embedding dimension ($C = 64$), and w, b are the weights and biases of the ANN layer. In [31], a ReLU activation function is used, however in [32] it was found that a Sigmoid activation function applied to the cosine-embedding layer stabilizes learning in practice. The output of the embedding layer is multiplied element-wise using the so-called Hadamard product denoted with $\odot$. Given that the state input layer is $\psi(s)$, the input to the hidden layers is $\psi(s) \odot \phi(\tau)$.

Figure 3.12 presents the implicit quantile network as described in [31] for an m-dimensional state-space, a discrete action space with size k, with quantile samples of size n and two fully-connected hidden layers of size h.

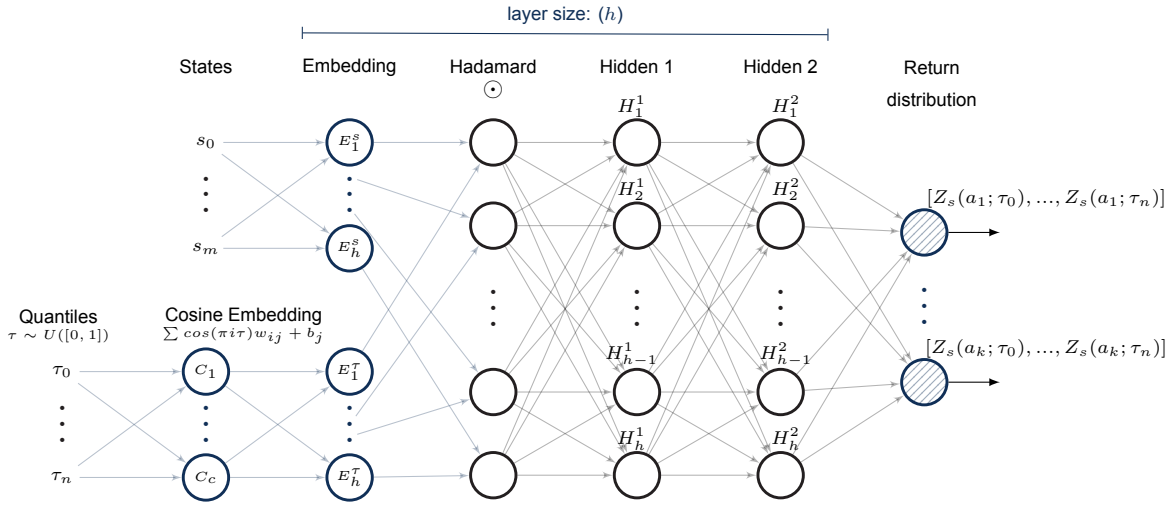


Figure 3.12: Depiction of the implicit quantile network (IQN) based on [31].

Synthesis

Three major advances in distributional RL have been identified that each showed state of the art performance w.r.t. traditional RL methods by exploiting the return distribution. As mentioned, in order to define a distributional RL method, two frameworks have to be specified: the way the distributions are parameterized and the way the distance metric is defined between the value distribution and the target distribution. The core algorithms and the frameworks they use are summarized in Table 3.1.

Table 3.1: Core algorithms developed for Distributional RL.

Algorithm	Distribution Representation	Distance Metric	Paper
C51	Categorical	KL-divergence	[29]
QR-DQN	Variable location quantiles	Quantile Huber loss	[30]
IQN	Stochastic quantiles	Quantile Huber loss	[31]

The different representation methods are shown in Fig. 3.13, where value distributions are depicted for a given state. The traditional deep RL method DQN estimates the expected return $Q(s,a) = \mathbb{E}[Z(s,a)]$, whereas C51 uses variable probabilities for N atoms separated by Δz distance. QR-DQN inverts the categorical formulation by using fixed probabilities and parameterizing variable quantile locations. This allows the algorithm to use quantile regression to get rid of the projection step and achieve better convergence. Lastly, IQN uses an implicit definition of the continuous return distribution and uses n quantile samples that are passed through the value network.

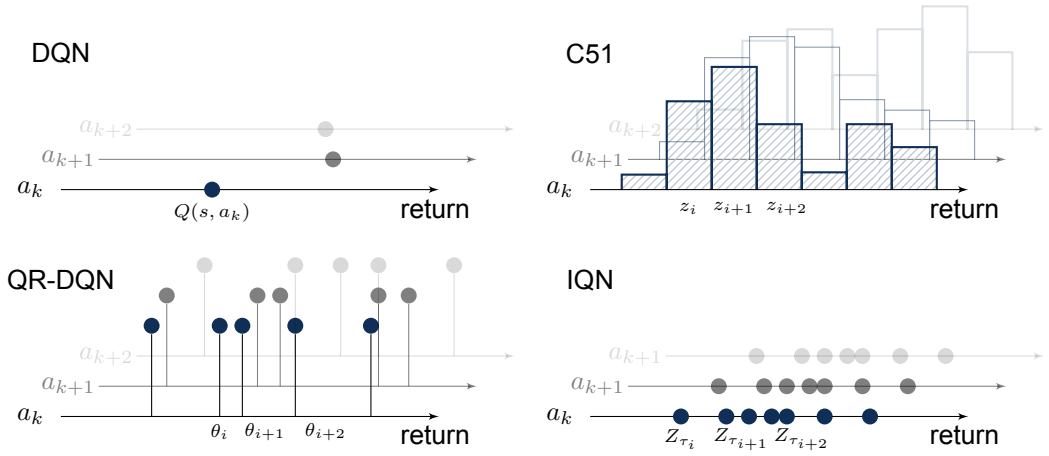


Figure 3.13: Adapted from [31]; Depiction of the different parameterization methods to estimate the return distribution $Z_k(s, a_k)$ for a given state s.

The quantile generation of IQN is especially convenient for specifying a richer class of policies, by distorting the original distribution that the τ quantile-levels are sampled from. Subsection 3.3.2 discusses how to use such distortions to achieve risk-sensitive behaviour in distributional RL.

3.3.2 Risk-sensitivity in Distributional RL

Traditional RL policies always maximize the expectation of returns. However, when the expected returns are replaced by value distribution functions, more complex strategies can be constructed that depend on the full distribution, which enables the use of risk-sensitive policies. The results of C51 and QR-DQN had shown significant improvements in sample efficiency, final performance and stability over traditional RL methods, however both algorithms defined the policy purely based on the expectation as in conventional value-based RL algorithms. The access to the distributional information enables a wide variety of policy definitions that can be used as risk-sensitive strategies. In Subsection 3.3.2 several risk-distortion functions are presented that can be used in risk-sensitive distributional RL methods. Then, Subsection 3.3.2

discusses potential approaches that can be used to deal with uncertainties when interacting with the environment.

Risk Distortions

The implicit formulation in IQN [31] makes it convenient to define so-called *risk distortions* that are applied to the distribution that τ quantiles are sampled from. The distortion operator is denoted using β and is a mapping that follows: $\beta : [0, 1] \rightarrow [0, 1]$. When β is identity, the policy is considered risk-neutral.

Several candidate distortion functions are listed in [31], namely *conditional value-at-risk* (CVaR), *cumulative probability weighting* (CPW), *Wang* and a standard power formula (Pow), shown in Eqs. (3.67) to (3.70) respectively. These are considered utility functions and the resulting policy follows the expectation of the utility function applied to the sampling distribution, such that if the risk-distortion is convex, the policy is risk-seeking and if it is concave, the resulting policy is risk-averse. A linear risk-distortion corresponds with the risk-neutral tendency.

$$\text{CVaR}(\xi, \tau) = \xi\tau \tag{3.67}$$

$$\text{CPW}(\xi, \tau) = \frac{\tau^{\xi}}{\left(\tau^{\xi} + (1+\tau)^{\xi}\right)^{1/\xi}} \tag{3.68}$$

$$\text{Wang}(\xi, \tau) = \Phi(\Phi^{-1}(\tau) + \xi) \tag{3.69}$$

$$\text{Pow}(\xi, \tau) = \begin{cases} \tau^{\frac{1}{1+|\xi|}} & \text{if } \xi \geq 0 \\ 1 - (1-\tau)^{\frac{1}{1+|\xi|}} & \text{otherwise} \end{cases} \tag{3.70}$$

Where ξ is a risk-tendency parameter, and Φ is the standard Normal cumulative distribution function. The way CVaR [77] modifies the sampling distribution is that it changes $\tau \sim U([0, 1])$ to $\tau \sim U([0, \xi])$. CPW is a unique choice as it is locally concave for small τ values and locally convex for large τ values, therefore provides a convenient risk-tendency manipulation with the distortion parameter ξ. The Wang distortion results in risk-averse behaviour for $\xi < 0$ and therefore provides an easy switch between the two behaviours. The same applies to the Pow formula, with $\xi < 0$ resulting in a risk-seeking and $\xi > 0$ resulting in a risk-averse setting.

In the study of these different policies [31] found that the best choice of risk-seeking / risk-averse setting and the choice of distortion function significantly depends on the environment choice. However, a tendency they observe is that risk-averse policies commonly outperform risk-neutral settings.

Handling uncertainties with Distributional RL

This subsection briefly discusses possible metrics to use within the MDP, which may drive complex risk-sensitive strategies. The risk-tendency of a distributional RL policy can be dynamically set using a choice of distortion function presented in Subsection 3.3.2. For instance, ART-IQN [78] adjusts the risk-tendency of the RL agent dynamically based on a measure of uncertainty in the partially observable environment. In model-based RL architectures the parametric uncertainty is explicit as an internal parametric representation of the MDP is used to solve the RL task. In model-free RL, this parametric uncertainty is implicit and relates to the either the parameters of the value-function or the parameters of the policy.

Such risk-sensitive strategies can be applied either post-learning during the deployment / application of the RL agent, or during training. It is an open question whether the adaptive risk-tendency can be exploited during the training process to increase sample efficiency of the agent. The results in [31] show that risk-averse policies have a tendency to outperform risk-neutral policies. Another motivation to study such strategies is the challenge of increasing safety for flight control applications. Risk-averse or adaptive

risk-tendency strategies have the potential to find near-optimal policies while retaining safety based on some measure of risk.

One of the most intuitive metrics to use for risk-sensitivity manipulation is some measure of uncertainty within the MDP. Two primary sources of uncertainty can be defined within in the deep distributional RL framework [79]. Firstly, interacting with the MDP comes with *intrinsic uncertainty* due to the randomness of the transition dynamics, the rewards and observations. Additionally, a dominant source of intrinsic uncertainty can be the partial observability of the MDP. Secondly, *parametric uncertainty* comes from incomplete information about the environment and the finite amount samples gathered throughout the learning process.

In [80] it was shown that the separation of intrinsic and parametric uncertainties is possible and can be exploited to improve the exploration characteristics of QR-DQN. The paradigm of *optimism in the face of uncertainty* is applied by using the *Tail Conditional Variance* (TCV) of the estimated distribution as an exploration bonus metric. The general formula for TCV is given by Eq. (3.71) and the TCV for the QR-DQN distributional representation is given by Eq. (3.72). Assigning a schedule to the upper tail variance σ_+^2 in Eq. (3.72) [80] improves the sample efficiency of QR-DQN.

$$\text{TCV}_x(k) = Var(k - \bar{k} \mid k > x) \tag{3.71}$$

$$\sigma_+^2 = \frac{1}{2N} \sum_{i=\frac{N}{2}}^{N} (k_{\frac{N}{2}} - k_i)^2 \tag{3.72}$$

In [78] it was shown that the intrinsic uncertainty of a partially observable MDP can be used to dynamically adapt the risk-tendency of the RL agent. ART-IQN [78] uses the lower half tail of the return distribution as a measure of intrinsic uncertainty. This measure is then used to adapt the risk-tendency using a CVaR distortion function. Instead of using the variance as an absolute measure of uncertainty, ART-IQN uses *exponentially weighted average forecasting* to relate the variance of different states to each other to account for the difference in varying value distributions across states.

3.3.3 Other Distributional RL Methods

This section discusses additional distributional RL frameworks that have either been derived from the core algorithms or propose novel parameterization and distribution estimation methods. Additionally numerous distributional RL methods are derived by combining the core algorithms mentioned in Subsection 3.3.1 with the deep RL architectures discussed in Section 3.2.

This section also surveys research about the application of distributional RL to continuous actions and therefore the methods are divided into two groups. Firstly, several improvements and additional distributional RL methods are briefly discussed, followed by approaches that focus on continuous action-spaces in Subsection 3.3.3.

Following the success of DQN extensions detailed in Section 3.2 and the novel distributional RL approach of C51, [53] combines six improvements to the DQN algorithm to produce *Rainbow* and achieve state of the art performance. The performance of the Rainbow algorithm shows that many of the deep RL extensions are compatible not only with each other, but also with the distributional RL framework.

While IQN samples n quantiles randomly from a uniform distribution, the algorithm proposed by [81] adds an additional fraction proposal network that is trained to provide optimal τ quantiles for the implicit distribution definition. These *fully parameterized quantile functions* (FQF) therefore add a second DNN to the architecture that is trained jointly together with the implicit quantile network to minimize the Wasserstein distance.

The framework in [82] introduces unconstrained monotonic neural networks (UMNN) as a novel way to parameterize the return distributions. This method exploits the fact that the probability density functions and quantile functions of random variables share the property of being strictly monotonic. Combining

this type of parameterization with DQN defines the distributional RL method of Unconstrained Monotonic DQN (UMDQN). The flexibility of the UMDQN framework in [82] allowed the testing of different types of probability metrics, namely KL-divergence, Cramer distance and Wasserstein distance. Their results show no clear best metric to use and state that the best metric depends heavily on the environment.

In [83] an argument is made that a monotonic constraint is necessary, as C51, QR-DQN and IQN all rely on the assumption that the quantile curve is non-decreasing. Since no global constraint is used in the core algorithms the monotonicity cannot be ensured. Non-decreasing quantile function networks (NDQFNs) are proposed in [83] for distributional RL methods in order to enforce the monotonicity and to increase the efficiency of exploration.

Distributional RL for Continuous Actions

The algorithms discussed in Subsection 3.3.1 are all value-based methods, as they define their policies implicitly based on the value distribution. Similar to traditional value-based RL methods, it is troublesome to generalize such policies over high-dimensional action spaces, as an additional optimization problem has to be solved to find the greedy action. In order to extend distributional RL algorithms to continuous action spaces, actor-critic architectures can be used, where the role of the critic is filled by a distributional approach that estimates the entire value distribution function.

The distributional deep deterministic policy gradient (D4PG)[7] by [73] have shown the ability to estimate rich return distributions and apply the distributional actor-critic framework to complex continuous control tasks. The algorithm of D4PG is constructed by using the Categorical DQN approach as a critic within the DDPG framework by [67]. The findings of the D4PG study show that the most significant sample efficiency improvements can be attributed to the addition of the distributional representation, as opposed to other extensions used, e.g. prioritized experience replay or n-step updates. In addition to formulating a distributional version of DPG, [84] proposes distributional Maximum a-priori Optimization (DMPO). Real-world comparisons have been conducted by [85] where DMPO [84] is compared to D4PG [73] in a real-world setting, with added challenges of constraints, partial observability sparse & delayed rewards and delayed observation feedback. In general distributional actor-critics have shown to be effective in solving continuous control problems, therefore they are considered a candidate class of approaches to apply distributional RL for flight control tasks.

Distributional Soft Actor-critic The distributional actor-critic approach proposed by [32] has shown that the addition of a distributional critic to the state of the art SAC algorithm improves on the learning performance of the algorithm. The motivation to use distributional soft actor-critic (DSAC) for risk-sensitive flight control research is three-fold. Firstly, [32] has shown that modelling both the distributional aspect of the action-space (as in SAC), and the distribution of returns is highly effective in learning high-dimensional continuous control tasks. Secondly, DSAC has shown increase in sample efficiency of adding distributional critics compared to nominal SAC and the results of [32] have also shown that SAC outperforms TD3 using both traditional and distributional architectures[8]. Thirdly, DSAC defines a generalized framework for risk-sensitive learning within the distributional soft policy iteration architecture which makes it possible to apply risk-sensitive approaches. These factors elevate SAC and DSAC as candidate algorithms to study their ability tackle continuous flight control tasks.

The formulation of DSAC [32] identifies three major contributions. Firstly, they define the distributional soft Bellman equation and use quantile regression to estimate the soft discounted return. Second, they formulate a generalized framework for risk-sensitive learning. Lastly, they demonstrate model-free RL on continuous state-action spaces and outperform current state of the art SAC.

The distributional soft Bellman operator is given in Eq. (3.73), where an additional term is added to maximize the entropy as well as the return. Similar to nominal SAC, the parameter η is used as the entropy temperature parameter to trade-off stochasticity and maximizing the return.

$$Z_\pi(s,a) \overset{\mathcal{D}}{=} R(s,a) + \gamma \left[Z(s',a') - \eta \log \pi(a' \mid s') \right] \tag{3.73}$$

[7] The fourth 'D' denotes the distributed parallelization that reduces wall-clock time.

[8] The distributional version of TD3 $\rightarrow$ TD4 is constructed in [32] to compare the distributional versions of SAC and TD3.

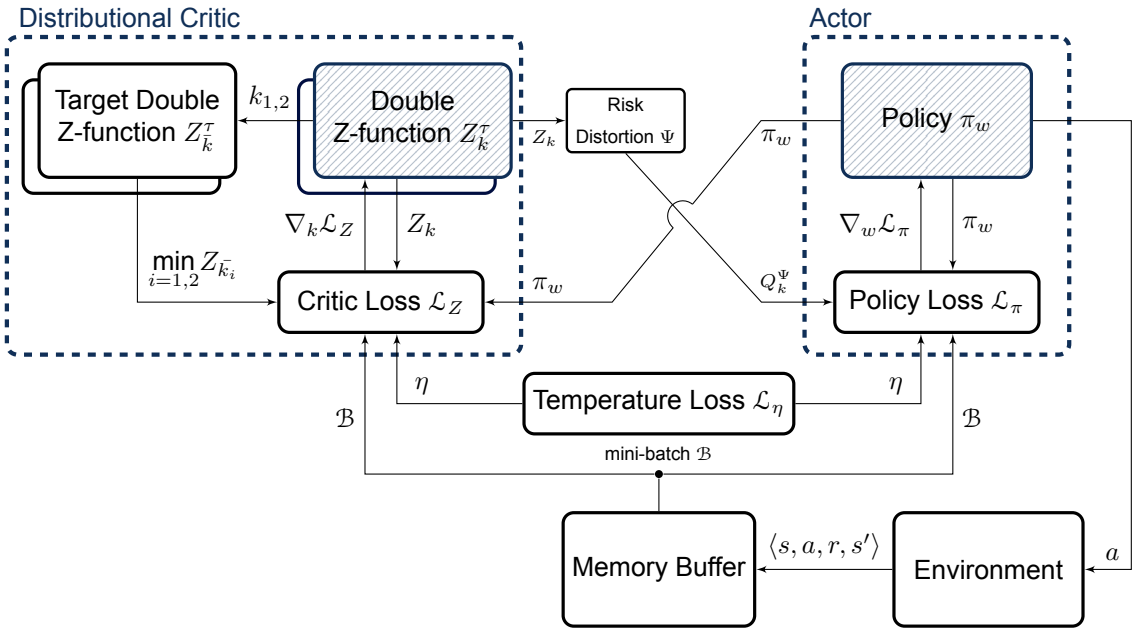


Figure 3.14: Depiction of the distributional SAC (DSAC) algorithm architecture.

The value distribution is estimated using a parameterized quantile value network $Z_\tau(s, a; k)$ and similar to TD3 [49] and SAC [68], double critic networks are used with parameters $k_{1,2}$ to prevent overestimation of the value distribution. The critic is the parameterized policy $\pi(a|s; \theta)$ with parameter vector θ. Using the distributional soft Bellman operator, the pairwise TD-update rule of SAC can be modified for the distributional framework as presented in Eq. (3.74):

$$\delta^l_{ij} = r + \gamma \left[\min_{l=1,2} Z_{\tau_i}(s', a'; \bar{k}_l) - \eta \log \pi(a' \mid s'; w) \right] - Z_{\tau_j}(s, a; k_l) \tag{3.74}$$

Where $\bar{k}$ is the parameter vector of the target quantile value distribution network and the subscript $l = 1, 2$ is used to denote the two networks in the double-critic architecture. Quantile regression is used to estimate the parameters as presented in [30], where quantile Huber loss is used as a probability metric. The critic loss function can therefore be written as shown in Eq. (3.75)

$$\mathcal{L}_Z(k) = -\sum_{i=0}^{N-1} (\tau_{i+1} - \tau_i) Z_{\tau_i}(s, a; k) \tag{3.75}$$

The generalized risk-sensitivity framework in DSAC defines a risk measure function $\Psi : \mathcal{Z} \to \mathbb{R}$, where $\mathcal{Z}$ is the action-value distribution space, such that in a risk-neutral setting $\Psi[\cdot] = \mathbb{E}[\cdot]$. The risk-measure Ψ is applied to the value distribution to obtain the so-called *risk soft action-value*: $Q^r(s, a) = \Psi[Z(s, a)]$. In order to achieve risk-sensitive policies, [32] applies the variety of risk-distortion functions mentioned previously in Subsection 3.3.2. The distributional form of the policy loss function uses the risk soft action-value Q^r and is given by Eq. (3.76):

$$\mathcal{L}_\pi(w) = \mathbb{E}\left[Q^r_k(s, \tilde{a}_w(s)) - \eta \log \pi_w(\tilde{a}_w(s) \mid s) \right] \tag{3.76}$$

$$\text{with} \qquad Q^r_k(s, \tilde{a}_w(s)) = \Psi \left[\min_{i=1,2} Z_{k_i}(s, \tilde{a}_w(s)) \right] \tag{3.77}$$

Where $\tilde{a}_w(s)$ is sampled from the Gaussian policy network as defined in nominal SAC in Eq. (3.48).

Similar to SAC, the target double networks are soft-updated towards the current networks for training stability using the Polyak step-size ζ. The architecture of DSAC is shown in Fig. 3.14, where the changes w.r.t. nominal SAC in Fig. 3.11 are the distributional critic and the risk-measure applied to the Z-function estimate. The critic is extended to the distributional representation and the loss function of the policy uses a risk-distorted expectation $Q^r(s, a; k)$.

3.3.4 Synthesis

This chapter has presented an extension of deep RL methods that estimate the entire distribution of returns. The parameterization of the value distribution not only increases sample efficiency of w.r.t. traditional RL agents, but also enables the use of risk-sensitive policies. Three core algorithms have been identified, namely C51 [29], QR-DQN [30] and IQN [31], each of which improves on the representation and estimation of the value distribution. IQN is capable of implicitly estimating a continuous value distribution while using few additional parameters and hyperparameters. Additionally, IQN provides a simple implementation of risk-distortion functions that can be applied to change the risk-tendency of the policy.

In order to apply distributional RL to high-dimensional continuous state-action spaces, as posed by Research Question 2, it was found that the distributional soft actor-critic (DSAC) [32] is a promising candidate algorithm. The application of DSAC to flight control tasks can be compared to the nominal SAC algorithm [68] as described in Section 3.2. The generalized risk-sensitivity framework of DSAC enables the use of risk-distortions which makes it possible to study the effects of risk-sensitive learning.

Research Question 5 was posed to review how the risk-sensitivity of distributional RL affects their performance at solving continuous control tasks. It has been found that risk-averse or adaptive risk-tendency policies tend to outperform traditional risk-neutral settings. It is still an open question how the risk-tendency during the learning process influences the sample efficiency or how the risk-tendency affects the tracking performance of distributional RL controllers.

So far, this report has focused on providing background on RL methods and surveying state of the art RL literature. Next, application oriented research is discussed that focuses on the use of deep and distributional RL to solve flight control and relevant robotics control tasks.

3.4 Reinforcement Learning for Flight Control

The goal of applying reinforcement learning to flight control tasks is to introduce intelligent control systems without requiring priori knowledge of the plant dynamics. In order to solve the flight control problem using model-free RL, the agent must be able to tackle high-dimensional continuous action-spaces with highly nonlinear transition dynamics and high levels of coupling in a 6 DOF environment. As discussed in Chapter 1, traditional FCSD approaches have traditionally relied on known plant dynamics and have configured automatic control systems to work in predetermined flight conditions. Recent research has demonstrated that RL approaches improve the control system's ability to handle unexpected scenarios and have the ability to handle varying degrees of risk without the need of priori knowledge about the plant dynamics.

Previous chapters so far have provided background in the fundamentals of RL approaches and recent advances in deep RL algorithms. Additionally, the class of distributional RL methods has been shown to increase sample efficiency and provide the ability to define complex risk-sensitive behaviours. The goal of this chapter is to place RL approaches within the context of flight control, survey application oriented RL research that can be applied to high-dimensional continuous action spaces and to identify the state of the art of RL flight control, as posed by RQ 1 and RQ 2.

Firstly, Subsection 3.4.1 formulates the continuous flight control as an RL problem and provides an overview of the main challenges. Secondly, the class of adaptive critic designs (ACDs) is introduced in Subsection 3.4.2 within the context of flight control research. Then, control applications of distributional RL are presented in Subsection 3.4.3 that illustrate the applicability of distributional RL to real-world control tasks. Lastly, state of the art RL research is discussed for flight control tasks in Subsection 3.4.4 follow by a synthesis of the chapter in Subsection 3.4.5.

3.4.1 Flight Control as an RL Task

The motion of aircraft is described by nonlinear dynamics, where the state-action space is often high-dimensional and continuous. The generalized nonaffine form can be written as shown in Eq. (3.78) where the assumption is made that the dynamics are stationary on short time scales.

$$\dot{\underline{x}} = f(\underline{x}, \underline{u}, t) \approx f(\underline{x}, \underline{u}) \tag{3.78}$$

Where f is the nonlinear state transition function of the system, $\underline{x} \in \mathbb{R}^n$ is the state vector of the aircraft and $\underline{u} \in \mathbb{R}^m$ is the vector of control inputs. The state transition function f denotes the equations of motion (EOM), where the nonlinearities are often due to the complex aerodynamics of the aircraft. The state vector for 6 DOF may consist of position $\underline{p} \in \mathbb{R}^3$, velocity $\underline{v} = [u, v, w]^t \in \mathbb{R}^3$, unit-quaternion rotation $\underline{q} \in \mathbb{R}^4$ or Euler-angle rotation $[\phi, \theta, \psi]^T \in \mathbb{R}^3$ and angular velocity $\Omega = [p, q, r]^T \in \mathbb{R}^3$. The velocity vector is often represented using the total airspeed $V = \|v\|$, angle of attack $\alpha = \text{atan}\left(\frac{w}{u}\right)$ and angle of sideslip $\beta = \text{atan}\left(\frac{v}{\sqrt{u^2+w^2}}\right)$ due to to their aerodynamic significance. In the case of traditional fixed-wing aircraft, the control input $\underline{u}$ is often the deflection of the control surfaces and the thrust setting. An example control input is $\underline{u} = [\delta_e, \delta_a, \delta_r, \delta_T]^T$ depicting the elevator, aileron and rudder deflections and the thrust setting respectively. Additional control inputs may be secondary control surface deflections, such as flaps and slates that can change the aerodynamic properties of the aircraft.

Additionally, when dealing with fixed-wing aircraft the dynamics and the state-space can be separated, i.e. de-coupled into longitudinal and lateral degrees of freedom. This de-coupling often allows RL research to reduce the DOF of the control task while maintaining fidelity.

MDP Formulation

Regulation problems, i.e. control tasks may entail either state regulation, where the objective is to keep the state near equilibrium or tracking control, where the objective is to track desired trajectories. Modelling tracking control tasks as an MDP usually requires the use of an augmented system definition, i.e. the system states are augmented with the tracking error signals and desired trajectories. For this reason, a distinction has to be made between the *observation* vector s, that previous chapters referred to as the

state of the MDP, and the state vector $\underline{x}$ which describes the dynamic state of the plant. Often, especially in a flat-RL case, $\underline{x} \subset s$ due to the augmentation, however this relation may vary depending on what information is available to the RL agent in a given architecture. Generally, the actions available a are either the same as the control inputs $\underline{u}$ or formulated in an incremental architecture: $a = \Delta\underline{u}$, in which case the control input u_t at time-step t is part of the observation vector [86].

An assumption is made that the observation vector s has the Markov-property for the plant dynamics in question. The transition probability function was previously formulated as a stochastic mapping, such that $\mathcal{P}_{s,a}^{s'} = \mathbb{P}\left[a' \mid a, s\right]$ for a given state-action pair. This directly translates to the state transition function f, which describes the plant dynamics. The source of stochasticity in aircraft dynamics are in large part due to the external disturbances acting on the system, such as wind gusts. The reward function is often formulated, such that it is proportional to either the absolute tracking error $R_{\text{abs}}(s,a) \propto |y_{ref} - \underline{x}|$ or the squared tracking error $R_{\text{sq}}(s,a) \propto (y_{ref} - \underline{x})^2$ where y_{ref} is the reference trajectory [28]. Several applications of RL to robotics and control tasks also add an action penalty term to the reward function that penalizes large action demands by the agent.

With the trajectory augmented action-space definition, the stochastic state transition given by the plant dynamics and the tracking error-based reward function the flight control task can be formulated as an MDP using the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, \mathcal{P}, \gamma \rangle$.

Challenges

As mentioned in Chapter 1, the two primary challenges of applying RL to flight control are the *safety* and *efficiency* of learning and both of these challenges merit their own fields of research within flight control RL applications.

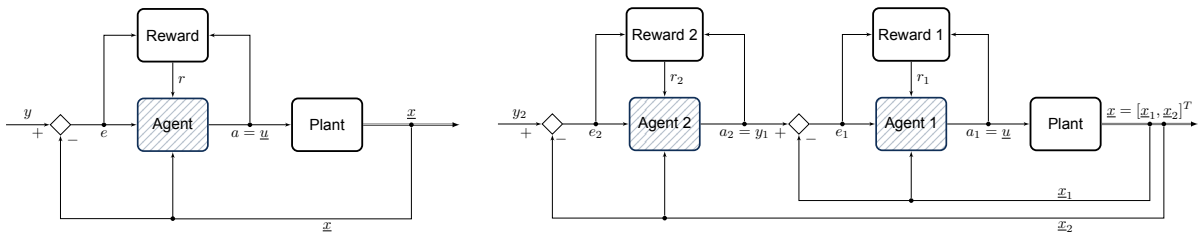
Safety of Learning The application of RL to safety-critical systems poses the risk of the agent choosing actions that lead to catastrophic outcomes. The agent exhibiting dangerous behaviour can take place not only during exploration and training, but also post-training when a near-optimal policy has been found and the agent encounters unexplored states [87]. Methods that focus on tackling such issues are Safe Learning algorithms and often focus on constraining the actions of the flight control agent both during and after training, such that the flight envelope of the aircraft is not breached and loss of control (LOC) is avoided [88].

For flight control applications SHERPA was developed by [88, 89] which stands for: Safety Handling Exploration with Risk Perception Algorithm. The SHERPA safety filter keeps track of a so-called *fatal state space* and ensures that the agent stays away from states that have a high risk associated with them, i.e. they lead towards the fatal state space. The set of safe states is gradually expanded during exploration. SHERPA has been demonstrated to facilitate safe learning for low level flight control of UAVs in [90].

Efficiency of Learning The efficiency of learning is a challenge for aircraft control due to high-dimensionality of the state-action space. The curse of dimensionality makes it exponentially more difficult to explore such high-dimensional state-action spaces which additionally may have sharp nonlinearities due to complex aerodynamic interactions.

In order to tackle the curse of dimensionality, concepts from Hierarchical Reinforcement Learning (HRL) [91] are often adopted to use a cascaded structure of RL agents that each control a smaller, more specific task within the control architecture. Hierarchical Reinforcement Learning (HRL) is an extension of RL which attempts to solve the curse of dimensionality by decomposing the original learning task into several smaller RL problems. The RL agents in a cascaded hierarchy use a subset of the augmented state vector that corresponds with separated dynamics that the agent is controlling.

Such cascaded architectures make use of the concept of time-scale separation and allow the inner-loop agents to control a subset of target states. This reduces the dimensionality of the RL problem and increases the sample efficiency of the architecture. This paradigm of time-scale separation is often utilized in traditional FCSD, as well. For example traditional airplanes, as well as multi-rotor or VTOL UAVs generally exhibit rotational rate dynamics that have an order of magnitude faster time-scales w.r.t. attitude dynamics, which allows the use of cascaded controller architectures. Figure 3.15 shows a simplified



(a) Flat-RL **(b)** Cascaded-HRL

Figure 3.15: Example of a flat-RL and a cascaded hierarchical RL control architecture.

depiction of two control architectures. The *flat*-RL approach is shown in Fig. 3.15a, where one agent uses the whole augmented observation. The *cascaded* HRL architecture is shown in Fig. 3.15b, where the inner-loop dynamics is assumed to have much faster time-scales than the outer-loop dynamics. This way, the state-space can be separated into $\underline{x}_1 \in \mathcal{X}_1$ and $\underline{x}_2 \in \mathcal{X}_2$, such that $\mathcal{X}_1, \mathcal{X}_2 \subset \mathcal{X}$ and multiple agents can be trained to control a smaller dimensional state-action space.

In a more general context, HRL creates abstractions within the MDP, such as *temporal abstractions* (Options) by [92] where the actions chosen by the agent may last for multiple time-steps, or *hierarchies of abstract machines* (HAM) [93] where a hierarchy of agents use exploit priori knowledge to solve high-dimensional RL tasks. Additionally, the approach of MAXQ [94] decomposes the value function of the MDP into a combination of value-functions from sub-MDPs.

Options has been demonstrated to increase sample efficiency when applied to the flight control of an F-16 [95]. Cascaded control architectures have been shown to result in more efficient and stable learning for controlling CS-25 certified aircraft [86, 28].

Simulation Gap The concept of a simulation gap refers to the phenomenon of the discrepancy between the simulated environment used to train RL agents or design controllers and the real-world environment used to deploy the algorithms. Due to the fact that aircraft are safety-critical system, using online learning alone is often not an option due to risk posed by stochastic exploration strategies and the cost associated with complex systems. Therefore, it is standard practice to first train RL controllers offline and then transfer the trained, or partially trained agent to the real-world system.

The performance of the RL agent degrades when the discrepancies between simulated environment and real-world application are dominant. Such discrepancies are in part due to model-errors, measurement, sensor and control input delays, and noisy/biased state estimates. This list is non-exhaustive and it is an active research area to bridge this gap between simulation and real-world application without degradation of control performance and loss of safety [96, 84]. The simulation gap also shows the importance of studying the robustness of RL methods w.r.t. model errors, studying their ability to deal with failures and their ability to generalize control laws over large state-action spaces. Distributional RL and risk-sensitive approaches have the ability to improve on closing the simulation gap by providing risk-averse policies for both exploration and post-learning deployment.

3.4.2 Adaptive Critic Designs

Adaptive Critic Designs (ACDs) are a class of Approximate Dynamics Programming (ADP) methods. Both ADP and RL provide approximate solutions to the dynamic programming problem and have been used to denote the same class of methods interchangeably. A distinction is made between Adaptive Critic Designs and the RL methods discussed so far, due to the fact that ADP research has mostly focused on the optimal control of continuous systems, while RL research had been extensive on discrete systems [97]. Since ACD methods approach the RL problem from a control theory viewpoint, they are discussed primarily in the context of flight control research, separately from deep RL methods. The reason ADP research has been prevalent in flight control is two-fold. Firstly, ACD methods describe online adaptive methods which is essential to achieve autonomous RL-based control systems. Secondly, the primary focus of ACDs is

the optimal control of continuous dynamic systems of the nonaffine form shown in Eq. (3.78) or the affine form: $\dot{\underline{x}} = f(\underline{x}) + g(\underline{x})\underline{u}$.

Three primary classes of ACDs can be defined. The most extensively researched class is Heuristic Dynamic Programming (HDP), which uses a critic ANN to approximate the value function. The second architecture is Dual Heuristic Dynamic Programming (DHP), where the critic estimates the derivative of the value function with respect to the state variables. A combination of HDP and DHP results in Global Dual Heuristic Dynamic Programming (GDHP) methods, which estimate both the value function and the derivative at the cost of higher computational complexity for the benefit of high approximation accuracy. All of these methods have an action-dependent variant, where the control policy is added to the inputs of the critic network [66]. Such action-dependent ADP variants have similar architecture to that of DPG approaches mentioned in Subsection 3.2.4. Most of the ACD algorithms estimate an online model of the plant dynamics and are therefore model-based RL approaches. The incremental forms of ACDs aim to reduce model-dependence through the use of linearized incremental model identification. Table 3.2 presents the mentioned variants of ACD architectures.

Table 3.2: Table of ACD methods

	Heuristic	Global	Dual-Heuristic
	HDP	GDHP	DHP
Action-dependent	ADHDP	ADGDHP	ADDHP
Incremental (iADP)	IHDP	IGDHP	IDHP

Traditional HDP methods update the model network by minimizing the difference between state prediction and state measurement. The incremental approaches of ADP (iADP) [98] provide a framework for model-free, adaptive control of nonlinear systems by updating a linearized model and exploiting incremental model-identification techniques. Methods such as IHDP, IDHP and IGDHP[18] are model-free in the sense that no priori model identification is needed, however use an internal incremental linear model of the system that is identified online.

Even though IDHP and similar iADP methods are great at online learning and adaptation, due to the local linearization tend to struggle with their generalization. The converged control policy of IDHP is often a localized linear mapping between states and control actions which the algorithm adjusts online.

Furthermore, their effectiveness has only been shown for limited degrees of freedom and small state-action spaces. ADP methods do not yet have the sample efficiency to achieve outer-loop control of coupled 6DOF dynamics [28]. It is an open research how these methods can perform well in high-dimensional state-action spaces. The combined use of IDHP with traditional stability augmentation systems (SAS) [99] has proved to be a promising approach to increase the scalability of ADP methods.

Lastly, the continual learning behaviour iADP methods pose an additional challenge for safety [98, 100]. Extended time periods with little excitation can cause continually growing weights which can destabilize learning. While IDHP and similar ACD methods are considered state of the art for online adaptive control, their application is not considered for this research project due to their lack of nonlinear generalization power, small DOF applicability and a need for persistent excitation. However, such methods are a vital part of RL-based flight control due to the fact that online learning is essential for crossing the simulation gap.

3.4.3 Distributional RL for Flight Control

In Section 3.3 the class of distributional RL was presented as an extension of traditional RL methods. Distributional RL has shown promising results in terms of sample efficiency and performance and several studies have shown the benefits of risk-sensitive RL. This section presents application-oriented research that applies distributional RL to real-world tasks in robotics, control and navigation.

A project with real-world application by [33] has shown that QR-DQN can successfully learn the control of under-actuated stratospheric balloons, whose task is to maintain position around a reference location.

While their research is not focused on comparing distributional RL to traditional RL methods, their results show a direct application of distributional RL in a control task for which traditional control system design is non-trivial due to the complexity of under-actuated control and navigation.

The QT-OPT algorithm applied by [101] is a version of distributional RL that can be applied to continuous actions. The algorithm was tested that on a robot grasping task that used computer vision and results showed that the distributional tests achieve higher success rate using fewer samples. Furthermore, they tested risk-sensitive policies using the distortion functions listed in Subsection 3.3.2, and found that often risk-sensitive behaviours result in better learning performance w.r.t. risk-neutral and risk-seeking settings. In special cases, such as extremely conservative risk-averse strategies the learning can be unstable.

In [34], a path planning framework is developed to assist in minimally-invasive surgeries using distributional RL. Universal Distributional Q-Learning (UDQL) is proposed to learn the action-value distribution function and assist the surgeons in planning and risk-management during single-insertion point multi-objective tasks. Their results show significantly improved accuracy and robustness achieved by the distributional agent w.r.t. the compared nominal DQN approaches.

Robust bus control has been achieved by [35] using a continuous variant of the IQN framework. The multi-agent RL architecture used risk-distortions to apply risk-sensitivity and make the system more robust to uncertainties, anomalies and perturbations. Furthermore, risk-sensitive robot navigation has been demonstrated by [102], using their proposed *risk-conditioned DSAC* method. which is a distributional actor-critic approach capable of adjusting its risk-tendency post-learning. The risk-sensitive approach resulted in the increased safety of a robot navigating the partially observable environment. In addition to the increased safety, the authors observed increase sample efficiency and performance.

Recent developments have shown that distributional RL agents can also *adapt* and adjust their risk-tendency to deal with uncertainties or unexpected situations. A safe RL architecture proposed by [103] uses distributional RL to adapt the conservatism of policies based on desired safety levels and desired comfort level of the passenger in an autonomous vehicle. The safety verification layer in [103] penalizes the distributional RL agent which in turn learns a risk-aware value distribution function. Their architecture is based on the IQN parameterization and demonstrated the capability to perform safer lane merging and turning manoeuvres with a self-driving car.

A similar approach was used in the proposal of a risk-adaptive IQN [78], where distributional RL framework is used to navigate a multi-rotor UAV through obstacles in a partially observable environment. The *adaptive risk-tendency* (ART-IQN) approach is proposed by [78] where the variance of the return distribution is used as an uncertainty metric to dynamically adjust the risk-tendency of the IQN agent.

3.4.4 State of the Art

This section presents recent research conducted on applying deep reinforcement learning to flight control tasks. This part of the survey focuses on the application of the state of the art actor-critic methods SAC & TD3 to continuous flight control tasks.

SAC and TD3 for Flying-wing Control A recent study by [104] has shown the successful use SAC and TD3 methods to control and suppress high-angle of attack roll oscillations prevalent in flying-wing designs. Such roll oscillations are inherently unsafe and increase the risk of LOC. Using traditional FCSD techniques is difficult for such situations as the relevant aerodynamic phenomena are highly nonlinear and complex resulting in non-trivial controller synthesis.

The model-free RL techniques of SAC and TD3 were applied in [104] to successfully suppress these roll oscillations. Their findings shows a significant difference in learning stability, as the SAC algorithm exploits its stochastic policy to ensure efficient exploration which makes the learning curve of TD3 more stable in comparison with reduced variance.

When applying the agents to a real-world test setup, the authors have identified the presence of latencies which render the dynamics of the aircraft non-Markovian. In order to counteract these delays and to ensure the Markov-property, the observation space of the agents was extended to include a history of previous time-steps, a memory that allows the agents to learn an implicit model of the complex aerody-

namic interactions. The results of [104] show the need for high nonlinear approximation and generalization power and the need for methods that are capable of modelling complex dynamics of aerospace systems.

SAC for Fault-tolerant Flight Control Due to the prevalence of loss of control failures in flight accidents, studying the fault-tolerant capabilities of RL is central to increasing the safety of complex aerospace systems. In a study by [28], the SAC algorithm was applied to achieve fault-tolerant flight control of the validated model of the PH-LAB aircraft.

In order to achieve a robust control architecture, [28] defines a hierarchical cascade of multiple SAC controllers that makes use of the time-scale separation of aircraft rotational dynamics, as shown in Fig. 3.16. Altitude h^R and attitude references ϕ^R, β^R are given to two SAC controllers, which determine the incremental actions: change in pitch angle reference $\Delta\theta^R$ and change in control surface deflections $\Delta\underline{\delta}$, where $\underline{\delta}$ denotes the vector of control surface deflections: $\underline{\delta} = [\delta_e, \delta_a, \delta_r]^T$ for shorter notation. This incremental formulation requires the agents to store the current control input as part of the observation vector as seen by the feedback of θ^R and $\underline{\delta}$ in Fig. 3.16. Such incremental formulation produces smoother control policies at the cost of increasing the dimension of the observation space.

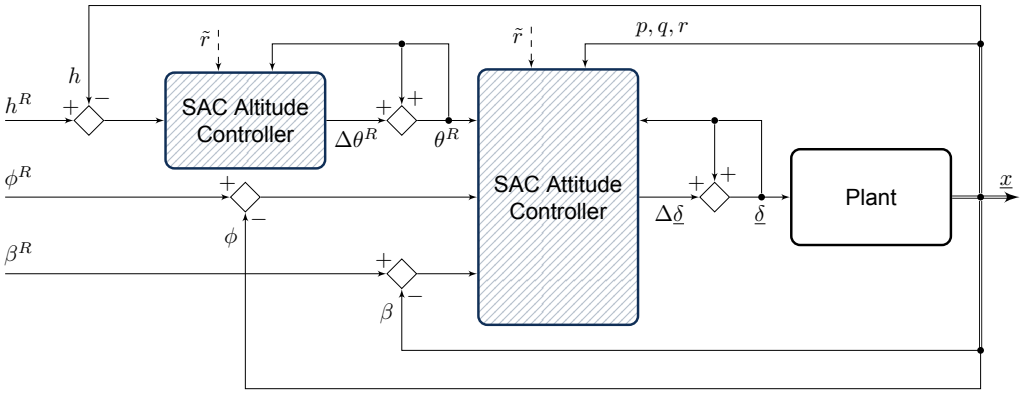


Figure 3.16: Adapted from [28]; Cascaded HRL architecture to control several DOF of the PH-LAB aircraft using multiple SAC agents.

Six failure-cases / changes to the plant dynamics are introduced as an unexpected change not seen during training. In addition to fault-tolerance, the robustness of the SAC controllers is tested for varying initial flight conditions, sensor bias and noise and atmospheric disturbances.

The results of [28] demonstrate SAC's fault-tolerance as the agent maintains stability and achieves the tracking task in the face of induced unexpected failures. This robust, fault-tolerant capability of SAC is attributed to its high generalization power and better exploration and the authors point out that unlike model-based controllers, this approach can adapt to failed flight conditions.

Future work recommended by [28] focuses on increasing the training reliability of SAC to increase the offline training consistency of the algorithm.

3.4.5 Synthesis

This chapter presented flight control as an RL task and surveyed recent developments in flight control related RL research. The primary challenges of the application of RL to real-world aerospace systems is the concern of safety, the efficiency of learning and the simulation gap that is currently present in flight control research between the offline and online training of RL agents.

While numerous advancements have been made in the application of ACDs for online adaptive flight control, state of the art ADP methods are limited in their generalization power and their sample efficiency to learn large state-action spaces and to implicitly model complex aerodynamic phenomena.

It has been found that distributional RL has not been directly applied to low-level flight control, however there are numerous robotics and flight control related examples of successful applications of risk-sensitive distributional RL methods. Furthermore, there are multiple proposals of an adaptive risk-tendency framework that allows the distributional RL agent to adjust its risk-sensitive tendency dynamically based on uncertainties in the environment. A consensus between application oriented distributional RL research papers is the fact that the parameterization of the return distribution improves the sample efficiency and tracking performance of the agent.

Moreover, state of the art actor-critic DRL methods have been successfully applied to high-dimensional flight control tasks. SAC has been shown to achieve fault-tolerant and robust flight control at the cost of inconsistent offline learning. A hierarchical cascade of SAC agents have the generalization power to control high-dimensional continuous systems, which makes SAC a promising candidate to study its extension to distributional RL. This chapter investigated the most suitable DRL methods for flight control tasks as posed by Research Question 1. The most recent developments in flight control RL research show that continuous control methods such as SAC and TD3 excel at the high-dimensional control tasks and SAC has been shown to be able to maintain stability and tracking in the face of model errors, sensors biases and failures.

Next, the methodology and results of the preliminary analysis are presented that aim to support the findings of the literature review using empirical evidence and to show the feasibility of using distributional RL for a continuous flight control task.

4

Preliminary Work

In order to further investigate Research Questions 1, 2 and 3, preliminary analysis has been done using simple RL environments that emulate the task of continuous flight control. The primary goal of the preliminary analysis is to assess feasibility of applying distributional methods to flight control tasks. The secondary goal is to support the literature driven findings of this report with empirical data on the learning and tracking performance of certain traditional RL and distributional RL architectures.

The two environments used are a simple pendulum environment [105] with a continuous action space and a linear model of a Cessna 500 aircraft [106] that has similar properties to the PH-LAB research aircraft [107] which is the target aircraft model of this research project.

First, the methodology of the preliminary analysis is presented in Section 4.1 which elaborates on the environments, the algorithm choice (SAC and DSAC) and the approach used to train and evaluate the agents. Then, the results are summarized for both environments in Section 4.2, followed by a synthesis of the preliminary work in Section 4.3.

4.1 Methodology

This section discusses the methodology of the preliminary analysis. First, the environment used for the continuous control task is presented in Subsection 4.1.1. Secondly, the algorithms used for the analysis are discussed in Subsection 4.1.2. Then, the training and evaluation methods are briefly elaborated in Subsection 4.1.3.

4.1.1 Environments

The preliminary analysis is conducted on two environments. Firstly, the effectiveness of applying distributional RL is shown on the simple *openAI* pendulum environment [105]. Secondly, a linear time-invariant (LTI) model of the short-period dynamics of a Cessna 500 is used to demonstrate the feasibility of applying distributional RL to flight control.

Pendulum

The goal of the pendulum gym control task is to keep the pendulum in a vertical position. The observation vector provided to the agent encodes three states: the sine and cosine of θ and the angular velocity of the pendulum, i.e. the time-derivative $d\theta/dt$, where θ is the orientation of the pendulum w.r.t. the vertical as shown in Fig. 4.1.

The action available to the agent is a continuous torque signal $T \in [-2, 2]$ [Nm] and the reward function applies a penalty to larger deviations from the vertical, large angular velocities and large actions. This results in the observation vector, action vector and reward function shown in Eqs. (4.1) to (4.3):

$$s = \left[\cos(\theta), \sin(\theta), \frac{d\theta}{dt}\right] \tag{4.1}$$

$$a = [T] \tag{4.2}$$

$$R(s,a) = -\left(\theta^2 + 0.1 \cdot \frac{d\theta}{dt} + 0.001 \cdot T^2\right), \quad \theta \in [-\pi, \pi] \tag{4.3}$$

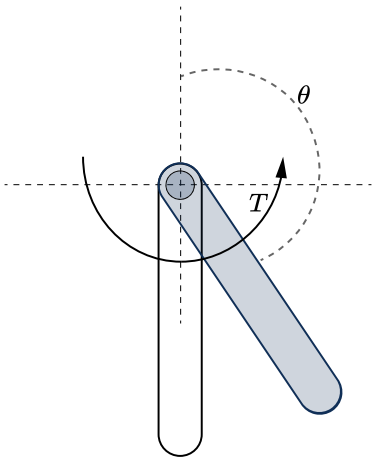


Figure 4.1: Adapted from [105]; openAI gym Pendulum-v1 environment.

Cessna 500

The primary objective of the research project is to control the validated simulation model of the CS-25 certified PH-LAB Cessna Citation II [107] research aircraft. To approximate the dynamics of the PH-LAB research aircraft, an LTI model of the similar Cessna 500 (Ce500 henceforth) is used from [106]. The state space model approximates the short period motion of the aircraft, trimmed at $\approx 60\ m/s$ airspeed.

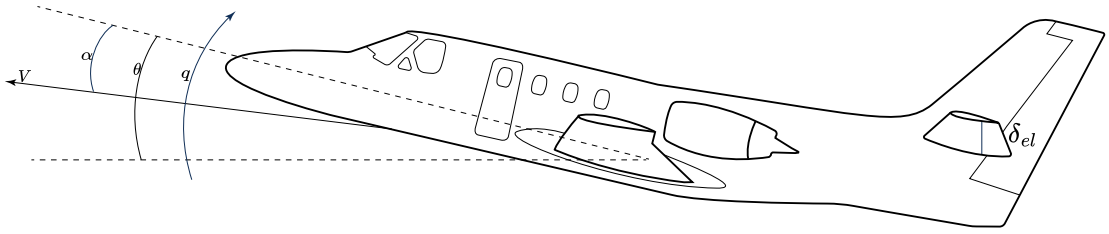


Figure 4.2: Cessna 500 (Ce500) short period model with two states: angle of attack (a.o.a) α and pitch rate q. The control input is the elevator deflection δ_e.

The state space model has two states, namely the angle of attack α [rad] and the pitch rate q [rad/s], and has a single continuous action that controls the elevator deflection δ_{el} [rad]. These states and the elevator deflection are shown in Fig. 4.2. Only the state transition dynamics of the state space: A and B are considered., i.e.: $C = I$ and $D = 0$. The state space is given by:

$$A = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{Z_\alpha}}{2\mu_c - C_{Z_{\dot{\alpha}}}} & \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V}{\bar{c}} \frac{C_{M_\alpha} + C_{Z_\alpha} \frac{C_{M_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} & \frac{V}{\bar{c}} \frac{C_{M_q} + C_{M_{\dot{\alpha}}} \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix}, \quad B = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{Z_{\delta_e}}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V}{\bar{c}} \frac{C_{M_{\delta_e}} + \frac{C_{M_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix} \tag{4.4}$$

Where the parameters are given in Table 4.1.

Table 4.1: Ce500 LTI model parameters

Geometry & Trim		Stability & Control Derivatives			
V	$59.9\ m/s$	C_{Z_α}	-5.16	C_{M_α}	-0.43
$\bar{c}$	$2.02\ m$	$C_{Z_{\dot{\alpha}}}$	-1.43	$C_{M_{\dot{\alpha}}}$	-3.7
μ_c	-1.55	C_{Z_q}	-3.86	C_{M_q}	-7.04
K_Y^2	0.98	$C_{Z_{\delta_e}}$	-0.62	$C_{M_{\delta_e}}$	102.7

4.1.2 Algorithms used

Previous chapters have discussed state of the art RL methods that can be used for the continuous control of high-dimensional state-action spaces. Section 3.2 has identified actor-critic methods, such as SAC as the most promising class of approaches to control continuous action space. While purely value-based approaches, such as DQN have great performance for discrete spaces, their application to control task requires either an optimization step to find the best action or the discretization of the action space. Some research in flight control application of RL, such as [95] has shown the method of discretizing the action space. The required resolution to achieve high tracking performance comes with dimensionality costs to purely value-based methods.

Furthermore, Section 3.4 discussed the prevalence of hierarchical cascades of RL agents used for high-dimensional flight control tasks. Flat-RL architectures use the entire state-action space to control complex systems whereas HRL architectures divide the control task into smaller sub-tasks, as mentioned in Subsection 3.4.1. The use of a hierarchical cascade of agents is most likely necessary to achieve successful control of a high-dimensional aircraft model.

Lastly, the objective of this research is to study the use of distributional agents. Section 3.3 has identified that IQN is an efficient way to parameterize the continuous return distribution. The DSAC algorithm, which can make use of implicit quantile networks as critics has shown state of the art performance in risk-sensitive control of continuous systems.

These 3 variations of RL methods, i.e. the type of action-space, the hierarchical structure of the control system and the addition of distributional methods is depicted in the experiment matrix in Fig. 4.3.

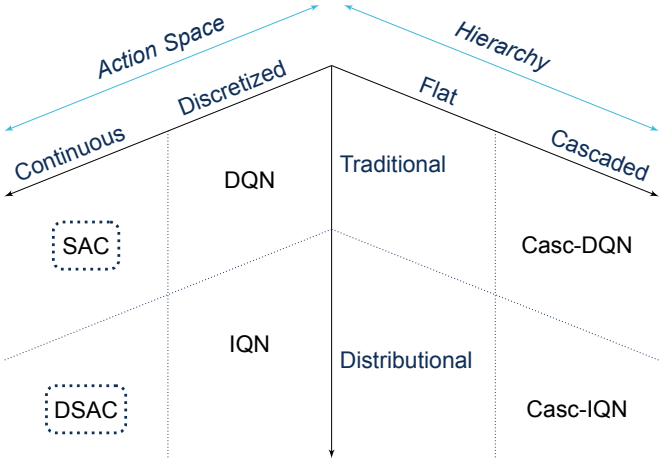


Figure 4.3: Experiment matrix for the preliminary analysis of applying applying Distributional RL to a continuous control task

The most promising algorithms that were identified are the soft actor-critic method proposed by [50, 68], which is discussed in detail in Subsection 3.2.4. SAC has great generalization power and has been found to outperform TD3 for certain control tasks [74]. More importantly, it models the action-space as a Gaussian distribution and [32] has found that the combination of a distributional critic and a distributional actor to significantly improve on performance and robustness.

For these reasons SAC and DSAC algorithms are studied for the two environments mentioned above. Since the environments are relatively simple and there is no obvious way to subdivide the MDP based on time-scale separation, the preliminary analysis only considers flat-RL architectures, however the use of HRL will be necessary for future work on high-dimensional models.

The pendulum environment is used to support the findings of literature about the increased sample efficiency of distributional RL. The control of the Ce500 LTI model is used to show the feasibility of using DSAC for a flight control task.

4.1.3 Training & Evaluation

In order to train the agents for the Ce500 control task, a randomized training signal is generated at the start of each episode. The goal of the control task is to track the angle of attack (a.o.a) α^R reference signal throughout the episode. The generated random reference signal is a series of random steps with amplitudes that are sampled from a uniform distribution $U([-A_{tr}, A_{tr}])$ [deg]. The width of each step block is h_{tr} [s] with a slight offset in timing, in order to ensure the agent cannot take advantage of the periodicity. An example of the randomized step sequence is shown in Fig. 4.4:

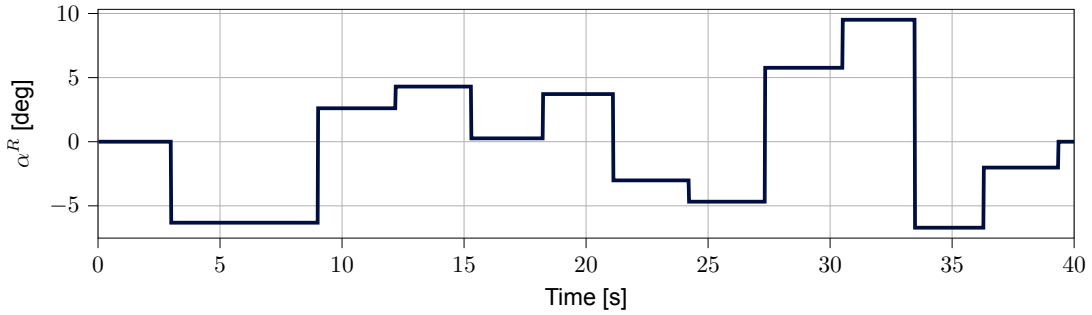


Figure 4.4: Example of a randomized step sequence used as an α-tracking reference signal for training.

For the control of the Ce500 plant, incremental elevator deflection control inputs are used similarly to the implementation in [28]. The agent controls the elevator deflection by applying incremental $\Delta\delta_{el}$ changes, that are limited by deflection rate saturations. In order to train the agent, the smallest possible observation space of the control task MDP is produced by using the observation vector $s = [\alpha^R - \alpha, \delta_{el}]$, where α^R is the reference angle of attack and $\alpha^R - \alpha$ is the tracking error. This results in a control architecture shown in Fig. 4.5.

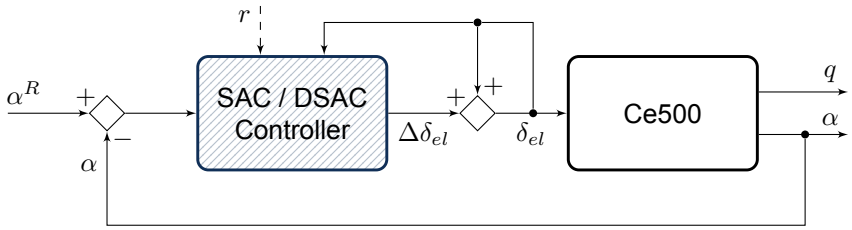


Figure 4.5: RL architecture to control the angle of attack of the Ce500 LTI model.

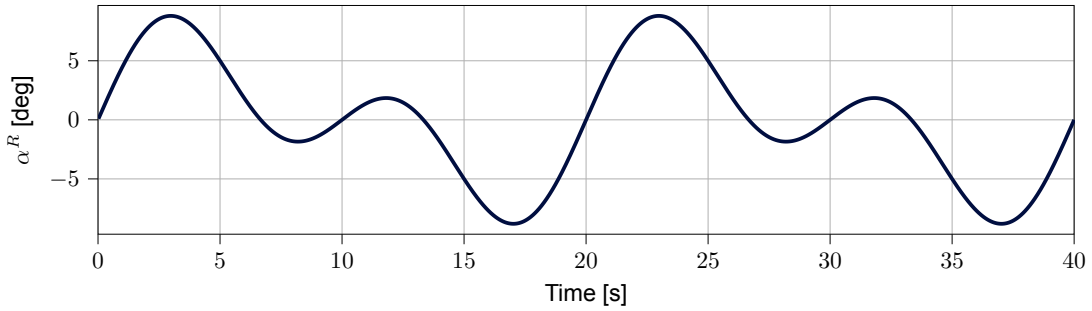
The reward function is proportional to the tracking error and is given by Eq. (4.5):

$$R(s,a) = -\frac{\Delta t}{T_{\text{max}}}\left[C_e \cdot \left|\alpha^R - \alpha\right| + C_{\delta_e} \cdot \delta_e + C_{\Delta\delta_e} \cdot \Delta\delta_e\right] \tag{4.5}$$

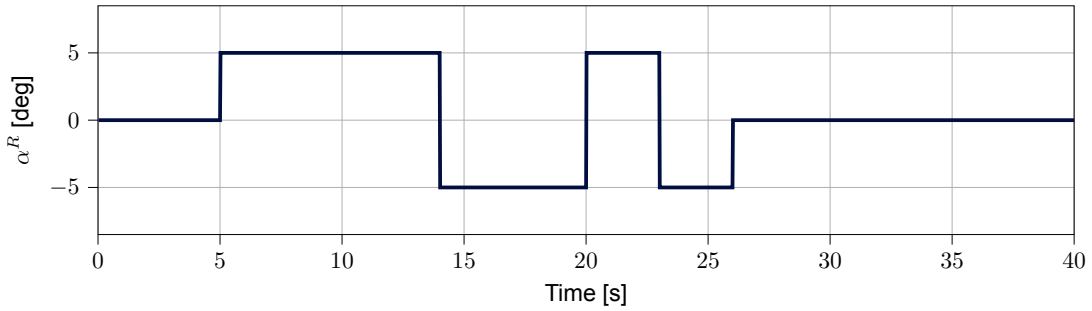
Where C_e is a tracking error penalty weight, Δt is the sampling time of the simulation, T_{max} is the maximum time of the episode. The term $\frac{\Delta t}{T_{\text{max}}}$ is used to make the scale of the reward signal invariant to different time settings.

The trained agents are evaluated on two types of reference signals. The first signal is a composite sinusoid that has previously been used to study the performance of RL agents controlling similar aircraft [28]. Furthermore, the sinusoid evaluation signal tests the capability of the agent to respond to reference signals not encountered during training. The sinusoid is given by Eq. (4.6) using two frequencies chosen to be $f_1 = 0.1$ and $f_2 = 0.05$. The amplitude A_{ev} is chosen to be half of the maximum amplitude encountered during training, similarly to the approach of [95]. The second evaluation reference signal used is a 3-2-1-1 signal which is a sequence of blocks with varying time duration. Both signals are shown in Fig. 4.6

$$\alpha^R_{\text{sine}} = A_{ev}\left(\sin(2\pi f_1) + \sin(2\pi f_2)\right) \tag{4.6}$$



(a) Sinusoidal evaluation reference signal



(b) 3-2-1-1 evaluation reference signal

Figure 4.6: Angle of attack reference signals for evaluating trained SAC and DSAC agents.

The architecture shown in Fig. 4.5, the reward function shown in Eq. (4.5) and the different types of training signal results in a set of hyperparameters that define the control task that the RL agents learn. The parameters used for the preliminary analysis are shown in Subsection 4.1.3.

Table 4.2: Hyperparameters for the training signals, simulation, reward signal and evaluation signal definitions

Parameter	Notation	Value	Unit	Parameter	Notation	Value	Unit
Time settings				*Training-signal*			
Sampling time	Δt	0.02	[s]	Random amplitude	A_{tr}	10.0	[deg]
Episode duration	T_{max}	30	[s]	Random block width	h_{tr}	3.0 ± 0.2	[s]
Reward-function				*Evaluation-signals*			
Error weight	C_e	1	[-]	Evaluation amplitude	A_{ev}	5.0	[deg]
Deflection weight	C_{δ_e}	0.001	[-]	Sine-frequencies	$[f_1, f_2]$	$[0.1, 0.05]$	[Hz]
Action weight	$C_{\Delta\delta_e}$	0.005	[-]	3-2-1-1 block width	h_{ev}	3.0	[s]

4.2 Results

This section presents the results of training SAC and DSAC agents on the two simplified continuous control task environments. Firstly, the learning performance of the two methods is compared using the training on the pendulum environment in Subsection 4.2.1. Then, the feasibility of applying SAC and DSAC on the Ce500 α-tracking flight control task is presented in Subsection 4.2.2.

The same hyperparameter set was used for the two algorithms SAC and DSAC. The hyperparameters are slightly different for the two environments as shown in Table 4.3 and were tuned to produce stable learning curves. The DSAC algorithm has 3 additional hyperparameters that tune the size of the quantile embedding layer of the implicit quantile network and the size of the quantile tensor used to approximate the return distribution. The values chosen are based on the original IQN paper [31] and on ART-IQN [78].

Table 4.3: Hyperparameters for training the SAC and DSAC agents for the pendulum task.

Parameter	Notation	Pendulum	Ce500
Nr. of hidden layers	N_{layers}	5	3
Nr. of hidden units	N_{units}	64	64
Learning rate	α_{LR}	1e$-$3	3e$-$4
Discount rate	γ	0.99	0.99
Batch size	$\lvert\mathcal{B}\rvert$	128	256
Buffer size	$\lvert\mathcal{D}\rvert$	1e6	1e5
Polyak step-size	ζ	0.995	0.995
IQN parameters (DSAC only)			
Nr. of cosine embedding units	C	64	64
Nr. of quantiles	N	8	8
Nr. of quantiles for expectation	K	16	32

4.2.1 Learning performance

In order to study the learning performance, 30 agents were trained for both SAC and DSAC on the pendulum environment. The learning curve for the pendulum environment is presented in Fig. 4.7 for both agents. The lines represent the mean curves over the 30 trained agents of each algorithm and the color filled areas represent the standard deviations.

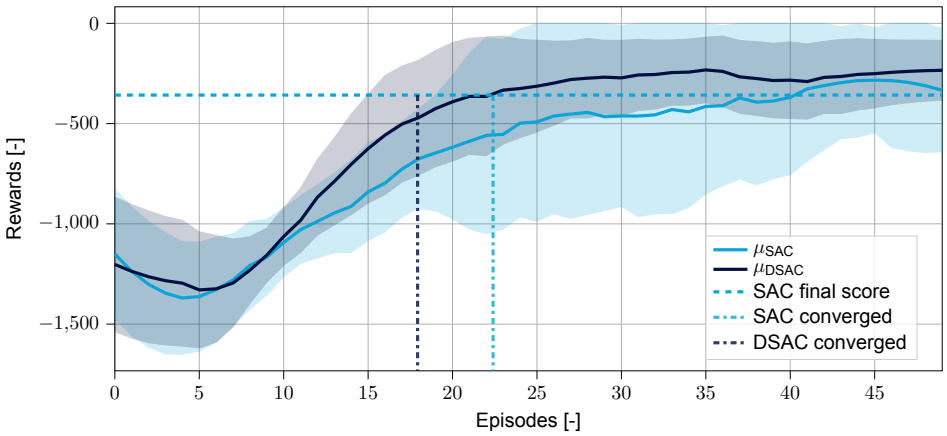


Figure 4.7: Learning curve (N=30) of the SAC and DSAC agents (Pendulum-v1) with sample efficiency markers.

It can be seen in Fig. 4.7 that the distributional agent generally has better final tracking performance, a learning curve with less variance and the distributional agents tend to converge using fewer samples, which points to an increased sample efficiency.

In order to quantify the observed improvements, the trained agents are evaluated on a set of $k = 10$ episodes and the average scores are taken to represent the final trained tracking performances. The box plot in Fig. 4.8 shows a significant decrease in variance of results and a general increase in median performance for the DSAC algorithm. The final tracking score is increased by 86.6, which is a 32% relative improvement, with a two-tailed p-value of 2.14%.

In addition to the reduced variance of the learning curve, the final tracking scores also exhibit lower variances, as shown in Fig. 4.8. A 55% reduction in final score variance can be observed in the performance of the trained agents and a 72% reduction in the variance of end-of-episode rewards can be observed during training. This reduction in variance points to the distributional agent's robustness to stochastic processes and shows increased consistency in the outcomes of converged policies.

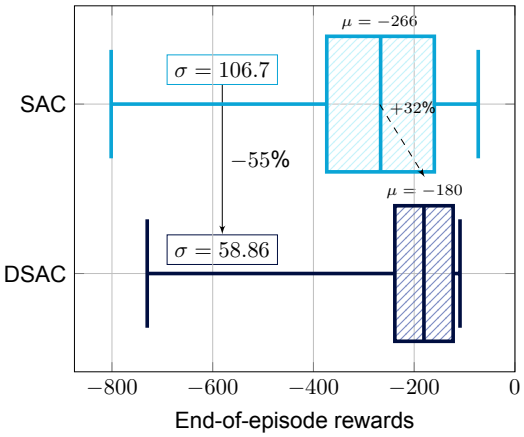


Figure 4.8: Average (k=10) score statistics of SAC and DSAC agents after training.

In order to quantify the sample efficiency improvement of the DSAC agent w.r.t. the SAC method, the amount of samples each algorithm requires to reach a given performance has to be determined. Due to the 32% relative increase in final score, the converged end-of-episode return of SAC (the lower of the two) is used as a point of interest to compare the sample efficiency of the two methods. As shown in Fig. 4.9 the DSAC agent reaches the converged final score 4.6 episodes sooner, which translates to an increased sample efficiency of 20% with a two-tailed p-value of $1.6\text{e}{-6}$.

For the Ce500, the learning curve of the two agents is shown in Fig. 4.9, with a smaller sample size ($N = 15$). Additionally, due to the environment and the control task of tracking angle of attack resulted in a relatively fast learning performance from both methods with little variance in results.

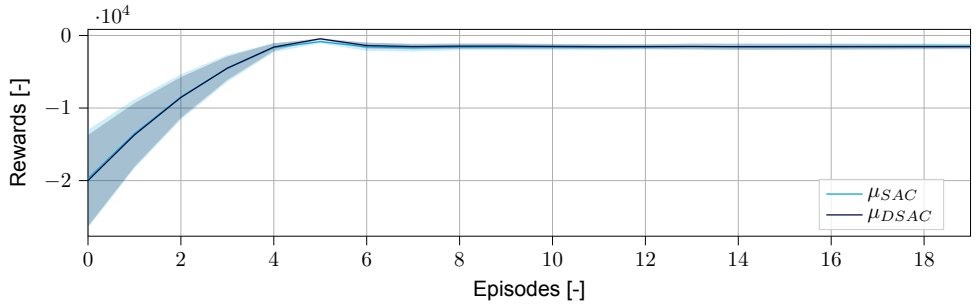
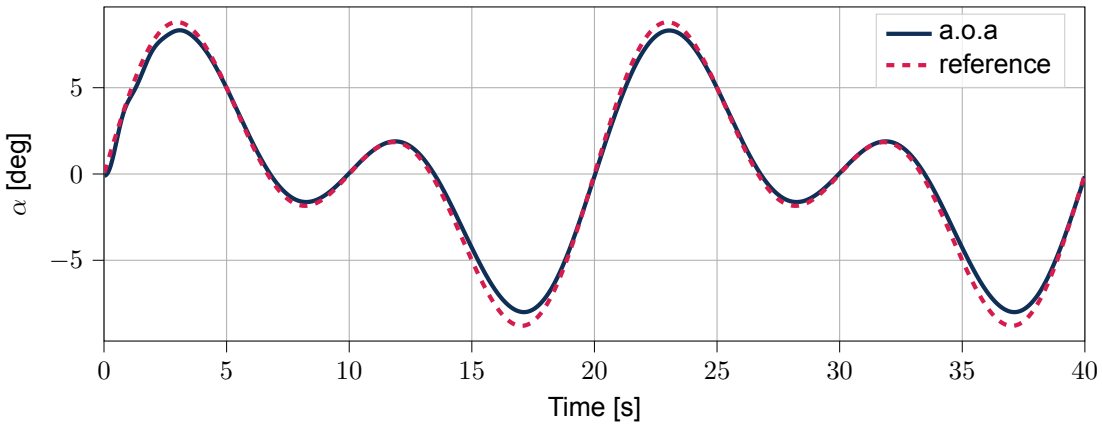


Figure 4.9: Learning curve (N=15) of the SAC and DSAC agents for the Ce500 control task, with mean μ and standard deviation.

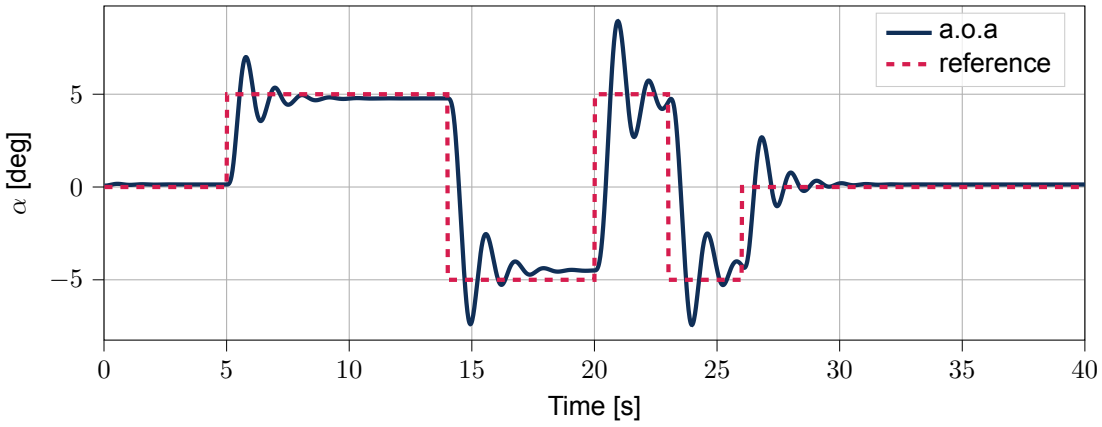
4.2.2 Flight Control Feasibility

As shown in Fig. 4.9 a number of agents were trained for the angle of attack tracking control task. This subsection presents examples of trained agents and their control tracking performance evaluated on the two reference signals mentioned in Section 4.1. With the selected hyperparameter set shown in Table 4.3, both methods have a high success rate at angle of attack tracking.

The angle of attack tracking results of an example trained SAC agent are shown in Fig. 4.10 and the results of en example DSAC agent are shown in Fig. 4.11 for both a sinusoidal and a 3-2-1-1 evaluation signal. For the evaluation task, the aircraft model starts in trimmed conditions, i.e. $[\alpha, q]^T = 0$. Both agents achieve adequate tracking performance for the evaluation signals, however the converged policy often results in poor damping for step inputs.



(a) Sinusoidal reference

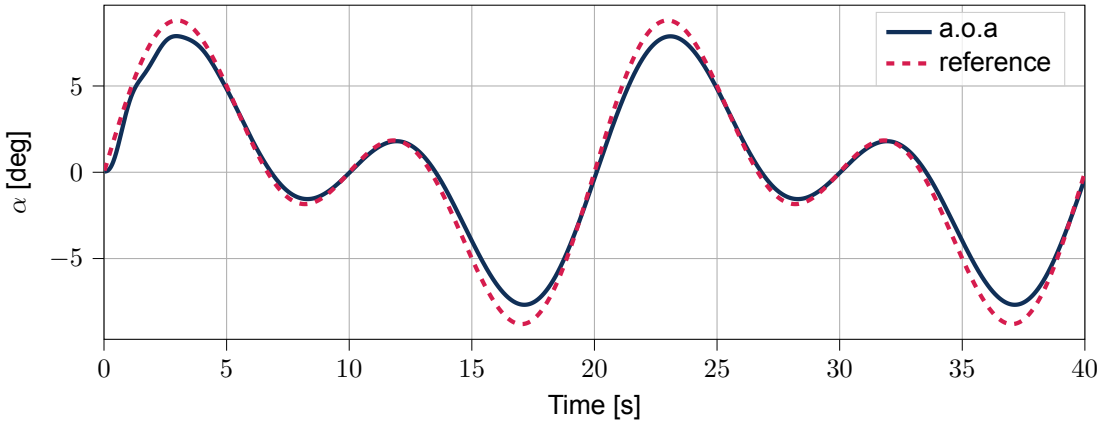


(b) 3-2-1-1 reference

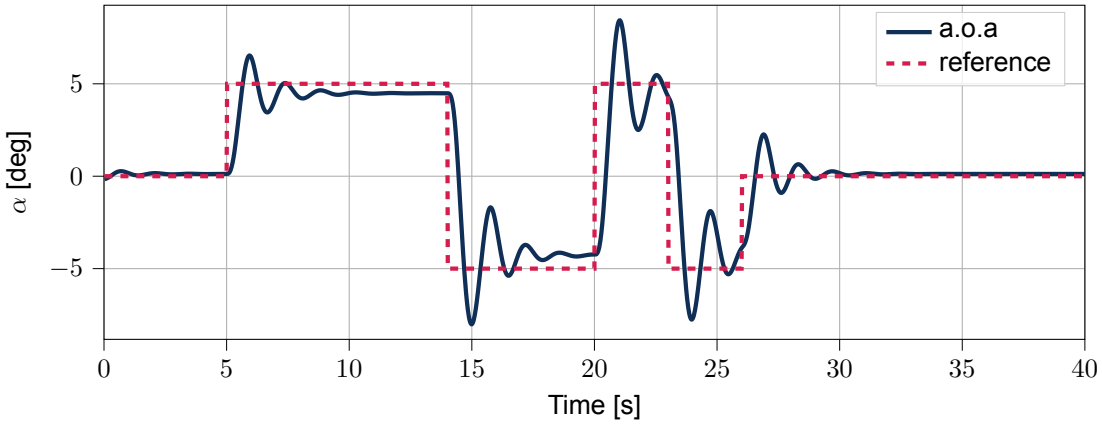
Figure 4.10: SAC post-training control tracking performance evaluation.

High-frequency oscillations

In some cases, (in about 30% of agents) the converged policy contains high-frequency high-amplitude oscillations that manage to achieve high returns on the a.o.a. tracking task. However, such policies are not practical for real-world use cases and robotics/flight control applications. The tendency to converge on such binary policies can be reduced by a number of approaches. Firstly, increasing the penalty weight of high actions (C_{δ_e}, C_a) in the reward function reduces the occurrence of such policies. Secondly, a regularization approach can be implemented as proposed by [108], where additional loss terms are added to the policy objective function to encourage the policy to be smooth both in a temporal sense (ensure subsequent actions are similar) and in a spatial sense (ensure that similar states result in similar actions). The implementation of realistic control laws can be investigated in future work with higher fidelity models



(a) Sinusoidal reference



(b) 3-2-1-1 reference

Figure 4.11: DSAC post-training control tracking performance evaluation.

where actuator dynamics plays a more vital role in the behaviour of the aircraft.

Wall-clock time

Due to the offline nature of the training it is also important to briefly discuss the computational complexity of the two methods. Although, the analysis of wall-clock time was not the focus of the preliminary analysis, no significant increase in computation time was observed with the addition of the distributional representation. The IQN representation of the return distribution function adds a low amount of additional parameters to the DNN and therefore doesn't come with a significant computational burden to

There are three sources of additional computational requirements that accompany the addition of the double Z-function critic. Firstly, the SGD step has to be performed on the parameters of the cosine embedding layer. Secondly, N quantile levels have to be generated which means that the size of the input tensor for each forward pass is multiplied by factor of $N = 8$. Thirdly, the expectation (or risk-distorted expectation) is obtained by performing a forward pass using $K = 32$ quantiles. In general, these additional computations do not increase the overall complexity of the algorithm however future investigation is required when dealing high-dimensional control tasks, such as the PH-LAB model.

4.3 Synthesis

This chapter gave an overview of the preliminary analysis done on applying a distributional actor-critic method, namely DSAC to continuous control task. The openAI gym pendulum environment was used to show that adding the distributional representation increases the sample efficiency (20%) and final tracking performance (32%) of the agent and achieves more consistent outcomes, when the comparison is controlled for all hyperparameters and stochastic processes. A short period LTI model of the Cessna 500 aircraft was used to demonstrate the feasibility of using distributional SAC for flight control tasks. Both methods consistently produce agents that are capable of tracking the angle of attack of the short period, even if the agent did not encounter the evaluation reference signal.

During the process of hyperparameter tuning it was found that the choice of control task settings is as important as the hyperparameters of the agent, which this report categorizes as the weights of the reward function and the training & evaluation signals. A reward function with incorrect weights results in inadequate control laws, or control laws that achieve adequate tracking control but use high-frequency binary policies to do so.

Furthermore, this chapter has provided empirical evidence to support the findings of the literature, i.e. that the addition of distributional RL increases the learning and tracking performance of the agent. Furthermore, it has shown that DSAC is a viable architecture for flight control, with the ability to produce risk-sensitive policies. Next, the findings of the literature review and preliminary work are concluded and the future work of the research project is outlined to answer the remaining research questions.

Part III

Additional Results

5

Robustness Analysis

The simulated PH-LAB RL environment presented in Chapter 2 initializes the aircraft model at the same trimmed initial flight condition (IFC) for each training episode and assumes ideal sensors and observations. The generalization power and robust performance of a cascaded SAC controller architecture has been demonstrated by [28] for varying IFCs, model errors, and biased sensors polluted with noise. This research project investigates the isolated effect of distributional RL and presents the findings that distributional SAC achieves similar tracking performance, while significantly improving the learning characteristics of the agent.

In order to generalize the findings regarding achieved tracking performance, this chapter discusses the response of the SAC and DSAC controllers when subjected to different IFCs and subjected to non-ideal sensor measurements. Section 5.1 presents the evaluations performed at two additional flight conditions, whereas Section 5.2 presents the tracking performance of the agents with noisy and biased measurements.

5.1 Varying Initial Flight Conditions

As mentioned, the nominal training environment is initialized at the same trimmed flight condition for each episode: at $2,000\ [m]$ altitude with an airspeed of $90\ [m/s]$. In order to investigate the robust performance of each agent variant, i.e. SAC, risk-neutral (R.N.) DSAC and risk-averse (R.A.) DSAC, the trained agents are subjected to evaluation attitude tracking tasks at two additional IFCs shown in Table 5.1.

	Altitude $[m]$	Airspeed $[m/s]$
training	2,000	90
IFC 1	2,000	150
IFC 2	10,000	90

Table 5.1: Initial Flight Conditions used for training and additional post-training evaluation.

The same reference signals outlined in Chapter 2 are used to evaluate tracking performance and the nMAE metric is used to compare the tracking error. The nMAE values of the three agent variants are shown in Table 5.2 for the two additional IFCs. Since, batches (n=10) of agents have been trained for each variant, the mean nMAE performance metric is shown, along the with a p-value of the statistical significance of distributional RL achieving better results.

It can be seen that all agent variants show degraded tracking performance for both additional IFCs, compared to the results shown in Section 2.4. This is expected due to the lack of exploration in different regions of the flight envelope. Additionally, the tracking performance of all variants is notably worse at IFC 2, as there are significant differences in aircraft dynamics at an altitude of $10,000\ [m]$.

Table 5.2: Evaluation at varying IFCs; nMAE values for each agent type with indicated relative improvements and p-values.

	SAC	R.N. DSAC		R.A. DSAC	
IFC	nMAE	nMAE (Rel.)	p	nMAE (Rel.)	p
h=2,000 $[m]$, V=150 $[m/s]$	*Baseline env.*				
	12.8	12.4 (3.0%)	1.6e-1	12.8 (-0.2%)	9.2e-1
	Augmented env.				
	13.6	13.5 (-0.4%)	9e-1	13.0 (-4.2%)	1.4e-1
h=10,000 $[m]$, V=90 $[m/s]$	*Baseline env.*				
	107.4	30.2 (-71.8%)	3.7e-1	14.4 (-86.6%)	2.7e-1
	Augmented env.				
	20.1	13.9 (30.6%)	2.7e-1	22.7 (+12.9%)	8.2e-1

Figure 5.1 shows example time-domain responses at IFC 2, where the tracking performance changes are significant. Figure 5.1(a, b) each show the mean response of a batch of agents where the shaded area shows the 95% (2σ) confidence interval. It can be seen that the tracking response of SAC agents is unreliable in all degrees of freedom, whereas risk-averse DSAC agents retain adequate lateral control in most cases, with some oscillatory behaviour. The longitudinal response of risk-averse DSAC agents shown in Fig. 5.1(b) also shows improvement relative to SAC, which is highlighted by the smaller variance of end-of-episode altitude.

In addition to the batch visualizations, Fig. 5.1(c, d) show individual responses of the best performing agents. Despite degraded longitudinal tracking performance, it can be seen that distributional agents are capable of generalizing across unobservable states when considering lateral degrees of freedom.

5.2 Biased & Noisy Sensors

An assumption made in the training of SAC and DSAC agents, is that the PH-LAB simulated aircraft model uses ideal sensors without noise and bias. In order to investigate the robustness of agents trained using ideal state observations, Gaussian white noise and stationary bias is added to the RL state vector, similarly to the methodology of [28]. The sensor noise statistics from [109] are used and are summarized in Table 5.3.

State observation	$p, q, r\ [rad/s]$	$\theta, \phi\ [rad]$	$\alpha, \beta\ [rad]$
Noise STD	6.3e-4	3.2e-5	2.7e-4
Sensor Bias	3.0e-5	4.0e-3	1.8e-3

Table 5.3: Adopted from [28]; Sensor noise and bias characteristics of the Cessna Citation PH-LAB research aircraft.

The nMAE results of evaluation are summarized in Table 5.4. Slightly degraded performance can be observed for all agent variants, as expected for noisy observation. Despite the added noise and bias, all variants achieve adequate tracking throughout the evaluation attitude control task.

When the sensors are polluted with noise and bias, Table 5.4 indicates the augmenting the observation with noisy measurements of α can result in degraded performance for the SAC and risk-neutral DSAC agents. However, since risk-averse DSAC agents are better at handling uncertainty in the environment, a small relative improvement can be observed.

Table 5.4: Evaluation with sensor noise and bias nMAE values for each agent type, with indicated relative improvement (Rel.). The p-value shows the t-test confidence level for the mean nMAE improvement. Bold values show statistically significant differences with 5e-2 threshold.

SAC	R.N. DSAC		R.A. DSAC	
nMAE	nMAE (Rel.)	p	nMAE (Rel.)	p
Baseline env.				
13.2	12.95 (-2.02%)	4e-1	13.10 (-0.93%)	7e-1
Augmented env.				
13.6	13.15 (-3.16%)	3e-1	12.89 (-5.07%)	6e-2

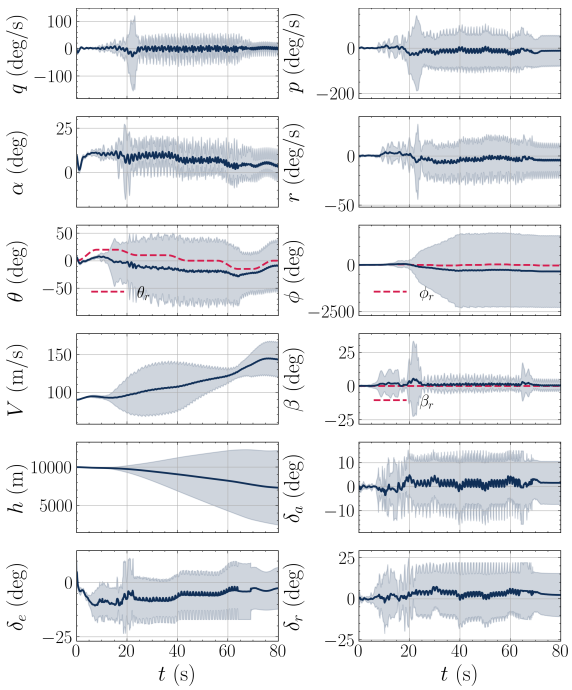
Example time-response of the agent variants are shown in Fig. 5.2 where the observation of the agent is polluted with noise and bias according to Table 5.3. Figure 5.2 visualizes the mean response with a 95% (2σ) confidence interval. As expected, the variance of trajectories is increased with the added stochasticity of observations. No significant improvement of DSAC can be observed relative to the trained SAC agents, however this investigation of added sensor noise shows the robust capability of the constructed algorithm.

5.3 Synthesis

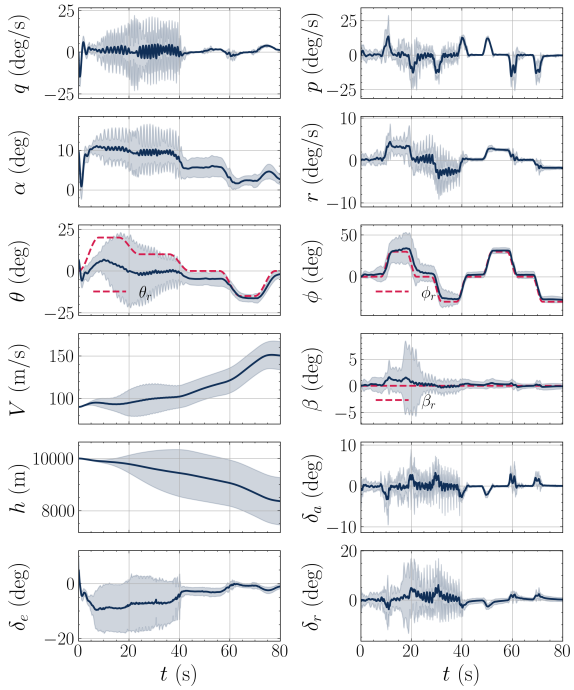
In order to synthesize RL-based control laws that can be used in real-world aircraft, it is important to show the generalization power of these algorithms, and to show their ability to handle real-world phenomena, such as sensor noise and bias. Furthermore, the findings of Chapter 2 are specific to the assumptions used to train the different agent variants. Two scope-limiting assumptions are the limited exploration of the flight envelope and the use of ideal, non-biased observations of the dynamic states of the aircraft.

This chapter investigates the tracking performance of the baseline SAC and DSAC agents when evaluated at flight conditions not explored during training. Additionally, sensor noise and bias is added to the observation of the agents that mimic the noise levels of the real-world aircraft.

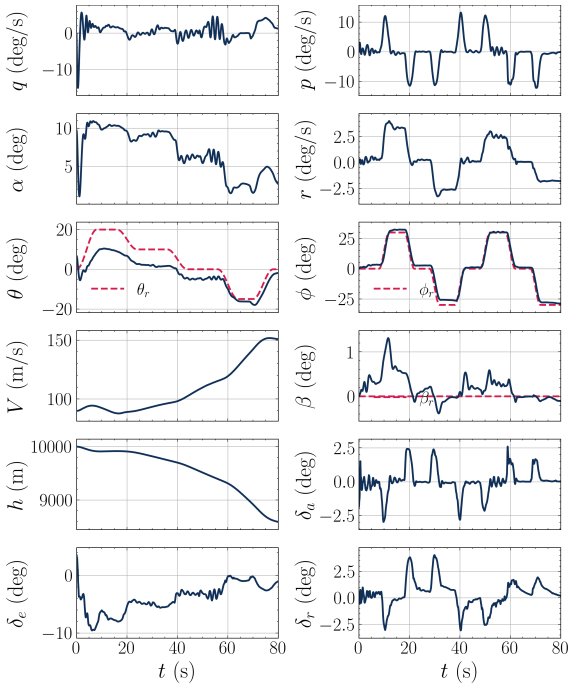
It is shown that the trained agents are capable of handling realistic levels of measurement noise and bias, resulting in a slight increase in variance. Furthermore, it is shown that all agent variants can generalize control laws to regions of the flight envelope where the aircraft dynamics are similar to that of the training conditions. However, initializing the simulation at an altitude of $10,000\ [m]$ drastically changes to the longitudinal dynamics resulting in degraded control performance and reduced consistency. Both risk-neutral and risk-averse DSAC agents demonstrated the ability to retain adequate lateral control even at the unexplored high altitudes. In order to obtain better generalization and reliability, thorough exploration of the flight envelope is required.



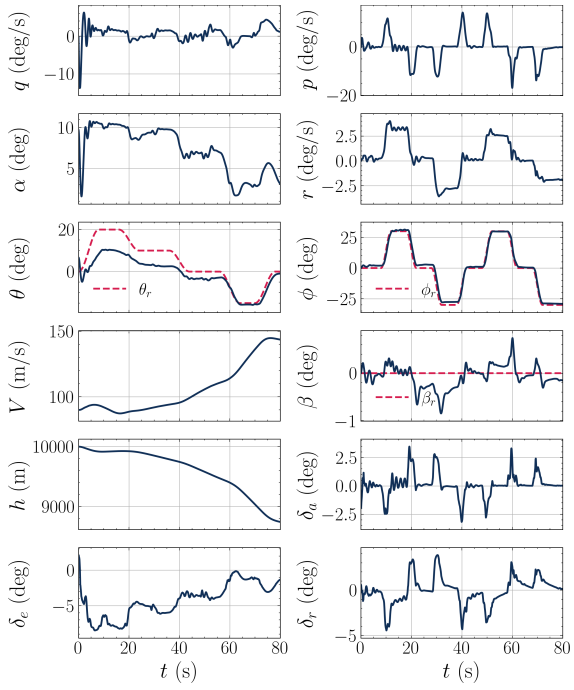
(a) SAC; baseline observation; mean response with 2σ confidence interval



(b) Risk-averse DSAC; baseline observation; mean response with 2σ confidence interval

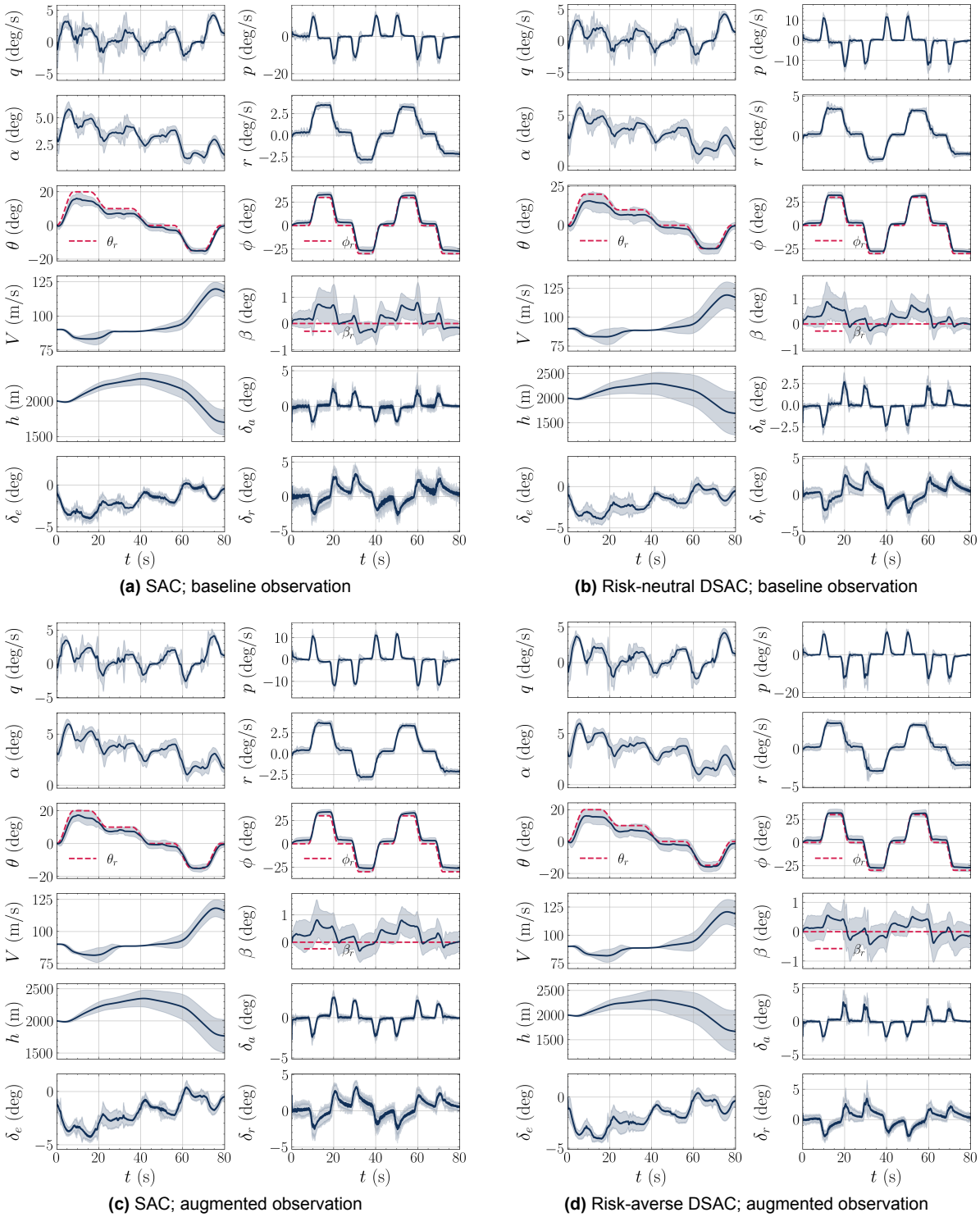


(c) Risk-neutral DSAC; augmented observation; response of best-performing agent



(d) Risk-averse DSAC; augmented observation; response of best-performing agent

Figure 5.1: Evaluation of agents at a different initial flight condition; IFC: $h = 10,000\ [m],\ V = 90\ [m/s]$



(a) SAC; baseline observation

(b) Risk-neutral DSAC; baseline observation

(c) SAC; augmented observation

(d) Risk-averse DSAC; augmented observation

Figure 5.2: Evaluation of agents trained using ideal sensors with sensor noise and bias.

6

Verification & Validation

When conducting machine learning experiments in a simulated environment, it is important to draw conclusions in the context of the assumptions and limitations of the algorithm, the simulation and the experiments. The purpose of this chapter is to summarize the steps taken to control the quality of method implementations and the conducted experiments, and to summarize the validity of the results of the research project. Section 6.1 discusses the verification steps, followed by a summary of validation details in Section 6.2.

6.1 Verification

6.1.1 Implementation of RL algorithms

The RL algorithms used in this research project are built upon several years of deep reinforcement learning research and include numerous individual, modular tools, approaches and components. Such approaches are for example the use experience replay buffers, double critic networks, the widespread use of SGD optimizers and the implementation of DNNs. In order to ensure that algorithms, such as the soft actor-critic have been implemented correctly, preliminary experiments have been conducted to show that the performance of the agent stands up to state-of-the-art baselines when compared against popular benchmarks.

The benchmark environment presented in Chapter 4 is the simple pendulum environment, which provides a continuous action space to be able to seamlessly transfer the implementation to flight control problems. Both traditional SAC and distributional SAC achieve high rewards and converge to optimal policies on the pendulum environment, indicating a correct integration of state-of-the-art methods and continuous action-space environments.

The flight control problem implementation is tested against the simple LTI model of the Ce500 aircraft, which verifies the augmentation of the observation vector with reference signals and verifies the correct implementation of common reward signals used in automatic control and robotics. Additionally, the implementation of sub-components is verified by the ensuring that the learning and tracking behaviour of the agents responds to hyperparameter changes as expected.

6.1.2 Simulated Environment

Throughout the primary research phase, the high-fidelity validated aerodynamic model of the PH-LAB aircraft has been used as the simulated environment. The model was constructed using the Delft University Aircraft Simulation Model and Analysis Tool (DASMAT) [110] from real-world flight data of a Cessna Citation 500 and is primarily available as a simulated environment in Matlab/Simulink.

The correct implementation of the simulated environment has been verified by comparing the step response of the aircraft to that of the DASMAT Citation Simulink model. The verification method of [86] is adopted and the aircraft dynamics are excited using pulse signals on both the elevator and aileron control surfaces to verify expected behaviour. Figure 6.1 shows the comparison of the simulated RL-environment to the original validated Simulink model. As expected, negative elevator deflections results in positive pitch

rate q and pitch angle θ values. The rather aggressive positive aileron deflection at $t = 5\ [s]$ results in an immediate roll rate p and roll angle ϕ response, with a slightly delayed yaw rate response. This is expected behaviour that shows the excitation of the dutch roll eigenmode of the aircraft.

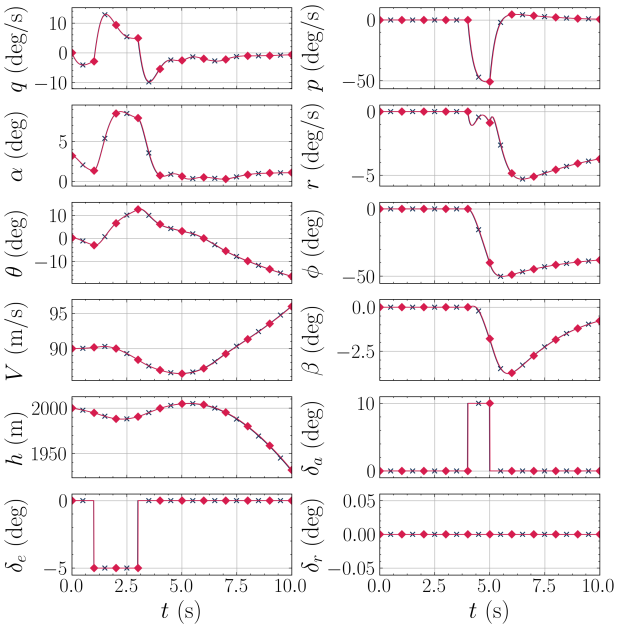


Figure 6.1: Pulse response of the simulated PH-LAB environment (red diamond markers) and the original DASMAT Simulink environment (dark cross markers) show identical trajectories.

6.2 Validation

Validation steps are required to ensure the quality of the conclusions and findings of the research project and to highlight the specific circumstances in which some of the findings may not hold true. This section discusses the validity of the simulation model, the assumptions used when training the RL agents and discusses the repeatability and reproducibility of the findings, which is crucial for ML-based research.

6.2.1 PH-LAB Simulation Model

One of the primary tools used to investigate the ability of SAC and DSAC to handle flight control tasks has been the high-fidelity aerodynamic model of the PH-LAB research aircraft. As mentioned in Section 6.1, the DASMAT tool [110] was used to construct the model based on flight data from a Cessna Citation 500. The Cessna 500 is a similar aircraft to the Cessna Citation II PH-LAB research aircraft, however there are differences in its wing design, fuselage length and engine performance. Despite the differences in design, the DASMAT Citation 500 model shows good prediction power when compared to flight data of the PH-LAB research aircraft, with a Root Mean Squared Error (RMSE) shown in Table 6.1, for both longitudinal, and lateral forces and moments. This shows that the validated aerodynamic model used to the ML experiment predicts realistic aircraft dynamics in nominal flight conditions.

Table 6.1: Relative RMSE of force and moment coefficients of the DASMAT Citation 500 model compared to PH-LAB flight data.

	Longitudinal	Lateral
Forces	9%	7%
Moments	13%	9%

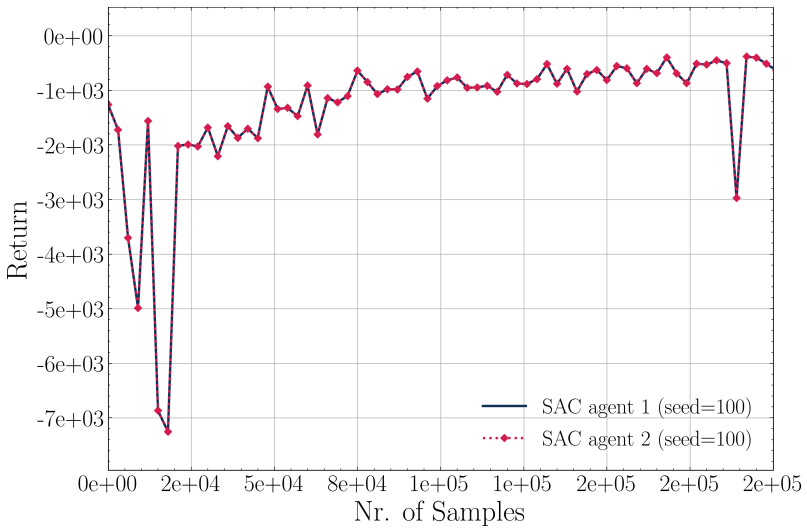


Figure 6.2: Same pseudo-random seed used for the training of two agents results in identical learning curves and repeatable experiments.

6.2.2 Assumptions

Several assumptions and scope limiting simplifications have been made, which pose limitations on of the findings. Firstly, the aircraft is initialized form the same trimmed flight condition at the start of each episode. This simplifies the learning task, however limits the generalization capability of the RL agents. In order to show the robustness of the SAC and DSAC algorithms, Chapter 5 presents additional evaluations conducted post-training at different IFCs and shows how significant changes in aircraft dynamics may result in inconsistent results.

Secondly, it is assumed that the RL agents receive perfect observations of the dynamic states of the aircraft. In real-world flight tasks, the sensors are biased, noisy with additional time-delays and possible difficulties with synchronization. Chapter 5 investigates the post-training of SAC and DSAC agents using noisy and biased sensors and shows that adequate tracking performance is achieved with slightly degraded performance.

Thirdly, the DASMAT Simulink model used in this research includes an inner-loop PID yaw damper and thrust controller, which simplifies the RL task. The resulting RL agents are only trained to control the surface actuators and have no access to thrust control, flap deflections and trim tabs. Moreover, the control surface actuators are modelled using first-order lag dynamics, with angle saturations. This assumes no signal delays between agent command and actuator response. Lastly, several sources of stochastic processes and uncertainty are excluded from the simulation, such as atmospheric disturbances and model uncertainty between the simulated environment and the post-training evaluation environment.

6.2.3 Repeatability & Reproducibility

As mentioned in Chapter 2, in order to assure reproducible results, the pseudo-random number generators used in both the environment and stochastic agent have been controlled. This is done in order to ensure repeatability and reproducibility.

As a demonstration of repeatable experiments, two SAC agents have been trained using the same pseudo-random seed, which results in identical learning curves shown in Fig. 6.2. The identical learning curve means that each of the episodes are identical, which includes randomized training reference signals, stochastic decisions made by the agent and the random sampling of experiences used for SGD. This results in fully reproducible experiments given that certain software and framework dependencies are met.

7

Wall Clock Time

The addition of distributional RL to the baseline SAC architecture comes with increased computational costs, which can be attributed to two sources: the additional trainable parameters needed to represent the return distribution and the computation of the pairwise TD-error between the randomly generated quantile fractions.

This chapter presents the increase of required wall clock time (WCT) to train distributional agents. Firstly, Section 7.1 discusses the increase in parameter count w.r.t. the chosen hyperparameters and the dimensionality of the environment. Then, Section 7.2 presents empirical measurements of WCT for the chosen hyperparameter set.

7.1 Number of parameters

Traditional DRL approaches use deep feed-forward neural networks to represent the action-value function with mapping $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. When introducing the IQN critics, as proposed by [31], N discrete quantile fractions are passed through the networks the produce the implicit representation of the inverse cumulative distribution function. This results in a DNN that maps: $\mathcal{S} \times \mathcal{A} \times \mathbb{R}^N \rightarrow \mathbb{R}^N$.

The network size of the critic is controlled by the number of hidden layers and the number of hidden neurons per layer. Adding the distributional representation by using IQN critics adds an additional embedding layer of size C. Previous research found that $C = 64$ embedding neurons are sufficient to encode the distributional information [31]. The agents in this research project were trained using two types of observation vectors: $\mathcal{S} \subseteq \mathbb{R}^6$ and $\mathcal{S} \subseteq \mathbb{R}^7$.

The resulting increase in the number of parameters is shown in Table 7.1 below, with the relative increase indicated as a percentage. The hyperparameters used in this research includes network sizes of either 64x64 or 128x128, therefore the relative parameter cost of using DSAC is in the range of $41 - 53\%$.

Table 7.1: Trainable parameter count of the distributional critics relative to the traditional SAC critics with $C = 64$ embedding size.

Network size	Nr. of observations	SAC	DSAC	Relative increase
64×64	6	3.2e4	4.9e4	53.4%
64×64	7	3.2e4	4.9e4	53.1%
128×128	6	1.2e5	1.7e5	41.7%
128×128	7	1.2e5	1.7e5	41.6%
256×256	6	4.7e5	6.4e5	35.3%
256×256	7	4.7e5	6.4e5	35.3%

7.2 Average training time

For the assessment of learning & tracking performance of distributional RL, two batches of agents were trained using two types of observation vectors $s_1 \in \mathbb{R}^6$ and $s_2 \in \mathbb{R}^7$, with $n = 10$ agents trained for each of the three approaches, namely the baseline SAC method, risk-neutral (R.N.) DSAC and risk-averse (R.A.) DSAC.

With all agents simulated and trained on the same hardware, the relative increase in wall clock time can be compared to provide an indication of the time-costs of distributional RL. Table 7.2 provides the empirical measurements of average training time per episode.

Table 7.2: Average training time (mm:ss) per episode (3000 samples) for each type of agent.

Observation	SAC	R.N. DSAC	R.A. DSAC
s_1	01:30	01:54 **(+27.4%)**	01:54 **(+27.1%)**
s_2	01:29	01:52 **(+25.5%)**	01:53 **(+26.0%)**

Previously, traditional SAC has been found to be produce inconsistent results with a success rate of 26% [28]. Chapter 2 shows that distributional RL significantly improves on the learning consistency and variance of converged policies. In terms of wall clock time, the use of distributional RL increases the training time by approximately $25\% - 28\%$, which results in a beneficial trade-off in the practicality of synthesizing DRL control laws.

The measurements are limited to a single set of hardware specifications. Early experiments on different hardware and cloud virtual machines have showed a similar increase in WCT.

Part IV

Closure

8

Conclusion

Recent years have shown significant increase in the complexity of control systems required for aerospace applications, which is driven both by the ever-increasing complexity of system dynamics and the growing demand for higher levels of autonomy and adaptability. Deep reinforcement learning, a class of bio-inspired machine learning techniques have shown promising results in finding complex near-optimal control strategies for high-dimensional problems in a model-free setting.

Applying DRL methods to flight control tasks comes with a multitude of challenges, such as safety, robustness, adaptability and sample efficiency. Incremental improvements to state-of-the-art RL methods are needed to cross the simulation gap that is currently present in flight control RL research and to apply safe and sample efficient RL techniques to real-world aircraft.

Whereas traditional RL methods learn the policy that maximizes the estimated expectation of cumulative rewards received from the environment, the field of distributional RL extends this axiomatic approach to estimate not just the expectation but the entire return distribution function. This parameterization of the return distribution improves on the learning and tracking performance of the algorithms, while enabling the use of complex risk-sensitive strategies that can improve the robustness and safety of the agent.

This chapter provides closing remarks on the findings of this report in Section 8.1. Then, Section 8.2 gives reflections on the findings with regards to the research questions posed in Chapter 1.

8.1 Closing Remarks

State-of-the-art DRL algorithms have been shown to excel at tasks previously deemed unsolvable. These RL approaches learn by direct interaction with the environment which enables the model-free synthesis of complex near-optimal control algorithms. Recent application oriented research has shown that distributional RL can be widely used in robotics and control applications, thus improving the RL agent's ability to deal with uncertainties and risk.

This research project investigates the application of risk-sensitive distributional RL methods for the attitude control a CS-25 certified Cessna Citation II research aircraft (PH-LAB). The attitude control task used in this research contains non-linear fully-coupled dynamics and is considered a high-dimensional RL task due to the large amount of states and control actions within the MDP.

Chapter 2 presents the distributional soft actor-critic (DSAC) framework used to investigate the risk-sensitive approach to controller synthesis. The combination of maximum entropy RL, i.e. the soft actor-critic (SAC) framework and the distributional critic significantly improves on the sample efficiency and consistency of learning, while achieving similar tracking performance. This improvement in consistency is crucial to train reliable and robust DRL agents for the safe control of real-world aircraft.

In addition to the improvements in learning characteristics, DSAC allows the synthesis of risk-averse control laws. Risk-averse agents are trained to trade-off immediate rewards against uncertainty in the environment. Whether the estimated uncertainty is intrinsic to the environment due to unknown stochastic processes or is due to lack of exploration is not pertaining to flight risk. Therefore, risk-averse DSAC agents achieve safer flight control by prioritizing state-action pairs with lower estimated uncertainty.

This is achieved using minimal added human-domain knowledge. The agents are trained using a goal-oriented reward signal that is proportional solely to tracking error. The ability to infer risk from the estimated uncertainty in the environment is a crucial step towards achieving increased levels of adaptability, autonomy and safety in RL-based flight control.

8.2 Research Questions

Chapter 1 poses RQ 1, 2 and 3 to identify candidate traditional RL and distributional RL methods for flight control applications and to compare their learning characteristics which is a major aspect of applying RL algorithms to high-dimensional flight control tasks. Research Questions 4, 5 and 6 focus on investigating the application of risk-sensitive distributional RL to a high-dimensional flight control task. The research questions posed in Chapter 1 are repeated below for convenience.

Research Question 1

What state-of-the-art RL methods are most suitable for flight control tasks?

The most suitable algorithms found focus on implementing an actor to parameterize the policy directly, to be able to deal with continuous action spaces. Algorithms such as SAC [68] and TD3 [49] are not only considered state of the art within the general field of DRL, but also for application-oriented research for fixed-wing [28] and multi-rotor flight [104]. While ADP methods such as IDHP [98, 111] have state of the art performance at online learning and adaptive optimal control, they struggle with generalization power and sample efficiency to achieve high DOF flight control [28]. A recent effort to combine offline and online RL in a hybrid architecture [112] has proved to be a promising direction to apply RL to real-world aircraft.

Research Question 2

What state-of-the-art distributional RL techniques are the most applicable to flight control tasks?

Distributional algorithms are defined by two frameworks: the way they parameterize the return distribution and the metric they use to estimate the difference between two value distributions [29]. This report identifies IQN [31] as a method of parameterization that comes with several advantages. Firstly, IQN uses relatively few additional parameters and uses little additional computations. Secondly, IQN estimates the full continuous distribution function implicitly instead of discretization methods used in C51 [29] and QR-DQN [30]. Lastly, IQN provides a simple way to implement risk-sensitive learning and risk-sensitive strategies by applying risk-distortion functions to the distribution of quantiles.

In order to use distributional RL for continuous control, this report identifies the approach of extending SAC [32] using distributional IQN critics as the most efficient and diverse approach, which also provides a generalized framework to apply risk-distortions to continuous policies. Additionally, DSAC was shown to outperform not only nominal SAC and nominal TD3, but also TD4 [32], the distributional equivalent of TD3.

Research Question 3

How does the learning performance of distributional RL methods compare to traditional value-based RL methods when applied to flight control tasks?

The conclusions about the comparison of traditional and distributional RL agents are drawn from three sources, namely application oriented research, the empirical evidence presented in Chapter 4 and the empirical evidence shown in Chapter 2.

In general, distributional RL agents show an increased sample efficiency and better tracking performance. This improvement is often observed for the risk-neutral case and points to a more efficient use

of transition-samples by the distributional agents. The analogy mentioned in Section 3.3 that compares distributional RL to taking a colored photo vs a greyscale image tends to hold true, as distributional agents exploit more information content about the environment.

The preliminary analysis confirms these findings by showing a statistically significant improvement of 22.4% in sample efficiency of DSAC with respect to traditional SAC on simple benchmark environments. Additionally, this research project controls for the pseudo-random stochastic processes present in both the agents and the environment, which means that the improvements in the learning characteristics can be attributed solely to the addition of the distributional critic.

Research Question 4

How does the learning and tracking performance of distributional RL compare to traditional RL methods when applied to high-dimensional flight control tasks?

The findings of comparing DSAC to SAC in the fully coupled attitude control task show a significant improvement in the variance and stability of learning, while achieving similar tracking performance. Chapter 2 shows that the improved learning consistency is even more pronounced for sub-optimal hyperparameters, and when there are additional observation dimensions.

Research Question 5

How do risk-sensitive distributional RL agents respond to uncertainties in high-dimensional flight control tasks?

This report presents DSAC agents trained using high-levels of risk-distortion to show the effect of risk-averse learning relative to risk-neutral policies. Chapter 2 demonstrates that risk-averse agents sacrifice immediate rewards in order to avoid uncertainty, resulting in safer flight control.

Research Question 6

How can risk-sensitive agents best be applied to high-dimensional flight control tasks to improve learning performance, tracking performance and safety?

In order to answer RQ 6, Chapter 2 presents the framework used to train risk-sensitive policies. A distortion risk-measure was identified as a promising way to trade off uncertainty against reward in the policy loss function, further augmenting the maximum entropy RL framework.

The results of Chapter 2 show that risk-averse agents have better learning consistency for more difficult, high-dimensional tasks. Furthermore, the robust analysis of Chapter 5 shows that risk-averse DSAC agents are capable of generalizing flight control laws to unexplored states, where SAC and risk-neutral DSAC have degraded performance. Lastly, Chapter 2 shows that risk-averse distributional agents achieve safer flight control without the addition of human-domain knowledge, such as reward shaping.

9

Recommendations

This chapter provides a brief overview of the primary recommendations for the future continuation of this research project.

Unused post-training critics

The DSAC algorithm used in this report does not make use of the value distribution estimate post-training, even though Chapter 2 demonstrates that the variance of the return can be a valuable online metric to improve the safety of flight control. Post-training, only the policy is used for inference of the RL-agent. Future work should investigate a method to adapt the policy and the risk-tendency of the agent based on the variance of the critic, similarly to the approach demonstrated by [78].

Risk-sensitive policy synthesis

This research only considers risk-distorted expectations for synthesizing risk-sensitive policies. When an estimate of the value distribution is available, a wide array of risk-measures can be used to trade-off immediate rewards against uncertainty.

Non-monotonic distributions

The DSAC algorithm used in this research project adopts the IQN value distribution estimator network [31], which poses no constraints on the estimated quantile function. The estimated inverse c.d.f. is often non-monotonically increasing and is ill-defined throughout the training, especially during early episodes. Furthermore, due to the random initialization of the critic DNN, the variance estimate of the critic is significantly smaller than expected, given the initially high parametric uncertainty and lack of exploration. The approach proposed by [82] points this out and suggests a new method to estimate monotonically increasing return distribution functions.

Use of Model Uncertainty

With the DSAC architecture, the agent has access to a model of uncertainty, which consists of the parametric uncertainty caused by state visitation and exploration and the uncertainty that is intrinsic to the environment. In order to achieve a safe transition from offline, simulation trained DRL agents to online, applied RL agents in real-world flight control, a third source of uncertainty can be inserted from human domain knowledge. Supplying the distributional critic with model parameter uncertainties and synthesizing risk-averse control laws has the potential to improve the robustness of RL flight control and reduce the fidelity requirements of the simulated environment.

References

[1] G.C.H.E. de Croon et al. “Design, Aerodynamics, and Vision-Based Control of the DelFly”. In: *International Journal of Micro Air Vehicles* 1.2 (2009), pp. 71–97. URL: http://journals.sagepub.com/doi/10.1260/175682909789498288.

[2] G.C.H.E. de Croon et al. *The DelFly*. Dordrecht: Springer Netherlands, 2016. URL: http://link.springer.com/10.1007/978-94-017-9208-0.

[3] Terrence A. Weisshaar. “Morphing Aircraft Systems: Historical Perspectives and Future Challenges”. In: *Journal of Aircraft* 50.2 (2013), pp. 337–353. URL: https://arc.aiaa.org/doi/10.2514/1.C031456.

[4] Rafic M. Ajaj et al. “Morphing aircraft: The need for a new design philosophy”. In: *Aerospace Science and Technology* 49 (2016), pp. 154–166. URL: https://doi.org/10.1016/j.ast.2015.11.039.

[5] Francesco Faggiano et al. “Aerodynamic Design of a Flying V Aircraft”. In: *17th AIAA Aviation Technology, Integration, and Operations Conference*. Denver, Colorado, 2017. URL: https://arc.aiaa.org/doi/10.2514/6.2017-3589.

[6] Alberto Ruiz Garcia et al. “Aerodynamic Model Identification of the Flying V from Sub-Scale Flight Test Data”. In: *AIAA SCITECH 2022 Forum*. San Diego, CA & Virtual, 2022. URL: https://arc.aiaa.org/doi/10.2514/6.2022-0713.

[7] Adnan S. Saeed et al. “A review on the platform design, dynamic modeling and control of hybrid UAVs”. In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. Denver, CO, USA: IEEE, 2015, pp. 806–815. URL: http://ieeexplore.ieee.org/document/7152365/.

[8] Avy. B.V. *Avy - Drones for Good*. Feb. 2022. URL: https://avy.eu/.

[9] Wingcopter GmbH. *Wingcopter - Technology with a Purpose*. Feb. 2022. URL: https://wingcopter.com/.

[10] Brian L Stevens et al. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[11] Eugene A Morelli et al. *Aircraft system identification: theory and practice*. Vol. 2. Sunflyte Enterprises Williamsburg, VA, 2016.

[12] R.A. Nichols et al. “Gain scheduling for H-infinity controllers: a flight control example”. In: *IEEE Transactions on Control Systems Technology* 1.2 (1993), pp. 69–79. DOI: 10.1109/87.238400. URL: http://ieeexplore.ieee.org/document/238400/.

[13] Sigurd Skogestad et al. *Multivariable feedback control: analysis and design*. Vol. 2. Citeseer, 2007.

[14] P. Smith. “A simplified approach to nonlinear dynamic inversion based flight control”. In: *23rd Atmospheric Flight Mechanics Conference*. Boston,MA,U.S.A.: American Institute of Aeronautics and Astronautics, 1998. DOI: 10.2514/6.1998-4461. URL: https://arc.aiaa.org/doi/10.2514/6.1998-4461.

[15] Phill Smith et al. “Flight test experience of a non-linear dynamic inversion control law on the VAAC Harrier”. In: *Atmospheric Flight Mechanics Conference*. Denver,CO,U.S.A.: American Institute of Aeronautics and Astronautics, Aug. 2000. DOI: 10.2514/6.2000-3914. URL: https://arc.aiaa.org/doi/10.2514/6.2000-3914.

[16] 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 (2010), pp. 1732–1742. URL: https://arc.aiaa.org/doi/10.2514/1.49978.

[17] 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 (2016), pp. 450–461. DOI: 10.2514/1.G001490. URL: https://arc.aiaa.org/doi/10.2514/1.G001490.

[18] 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 (2021), pp. 116–130. URL: https://ieeexplore.ieee.org/document/9160894/.

[19] Xuerui Wang et al. “Incremental fault-tolerant control for a hybrid quad-plane UAV subjected to a complete rotor loss”. In: *Aerospace Science and Technology* (2021), p. 107105. URL: https://linkinghub.elsevier.com/retrieve/pii/S1270963821006155.

[20] Tijmen Pollack et al. “Robust Stability and Performance Analysis of Incremental Dynamic Inversion-based Flight Control Laws”. In: *AIAA SCITECH 2022 Forum*. San Diego, CA & Virtual, 2022. URL: https://arc.aiaa.org/doi/10.2514/6.2022-1395.

[21] Richard S Sutton et al. *Reinforcement Learning: An Introduction*. 2nd. The MIT Press, 2015.

[22] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: *King’s College, Cambridge United Kingdom* (1989).

[23] Richard Bellman. “Dynamic Programming”. In: *Science* 153.3731 (July 1966), pp. 34–37. DOI: 10.1126/science.153.3731.34. URL: https://www.science.org/doi/10.1126/science.153.3731.34.

[24] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *arXiv:1312.5602 [cs]* (Dec. 2013). URL: http://arxiv.org/abs/1312.5602.

[25] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533. URL: http://www.nature.com/articles/nature14236.

[26] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *arXiv:1712.01815 [cs]* (2017). URL: http://arxiv.org/abs/1712.01815.

[27] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144.

[28] Killian Dally et al. “Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control”. In: *AIAA SCITECH 2022 Forum*. San Diego, CA & Virtual, Jan. 2022. URL: https://arc.aiaa.org/doi/10.2514/6.2022-2078.

[29] Marc G Bellemare et al. “A Distributional Perspective on Reinforcement Learning”. In: *International Conference on Machine Learning* (2017), pp. 449–458. URL: https://arxiv.org/abs/1707.06887.

[30] Will Dabney et al. “Distributional Reinforcement Learning with Quantile Regression”. In: (Oct. 2017). URL: http://arxiv.org/abs/1710.10044.

[31] Will Dabney et al. “Implicit Quantile Networks for Distributional Reinforcement Learning”. In: (June 2018). URL: http://arxiv.org/abs/1806.06923.

[32] Xiaoteng Ma et al. “DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning”. In: (June 2020). URL: http://arxiv.org/abs/2004.14547.

[33] Marc G. Bellemare et al. “Autonomous navigation of stratospheric balloons using reinforcement learning”. 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.

[34] Xiaoyu Tan et al. “Robot-assisted flexible needle insertion using universal distributional deep reinforcement learning”. In: *International Journal of Computer Assisted Radiology and Surgery* 15.2 (Feb. 2020), pp. 341–349. DOI: 10.1007/s11548-019-02098-7. URL: http://link.springer.com/10.1007/s11548-019-02098-7.

[35] Jiawei Wang et al. “Robust Dynamic Bus Control: A Distributional Multi-agent Reinforcement Learning Approach”. In: *arXiv:2111.01946 [cs]* (Nov. 2021). URL: http://arxiv.org/abs/2111.01946.

[36] Alan M Turing. “Intelligent machinery, a heretical theory”. In: *The Turing test: Verbal behavior as the hallmark of intelligence* 105 (1948).

[37] David Silver. *Lectures on Reinforcement Learning*. url: https://www.davidsilver.uk/teaching/. 2015.

[38] David Silver. “Reinforcement Learning and Simulation-Based Search in Computer Go”. PhD thesis. University of Alberta, 2009.

[39] Diederik P. Kingma et al. “Adam: A Method for Stochastic Optimization”. In: (2017). url: http://arxiv.org/abs/1412.6980.

[40] David Kriesel. “A Brief Introduction to Neural Networks. dkriesel. com”. In: *Online) http://www. dkriesel. com/en/science/neural_networks (last retrieved 30-10-2015* (2005).

[41] Ian Goodfellow et al. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[42] Aston Zhang et al. *Dive into Deep Learning*. https://d2l.ai. 2020.

[43] Jimmy Lei Ba et al. “Layer Normalization”. In: (July 2016). url: http://arxiv.org/abs/1607.06450.

[44] Kevin P. Murphy. “A survey of POMDP solution techniques”. In: *environment* 2 (2000), p. X3.

[45] Gautam Singh et al. “Structured World Belief for Reinforcement Learning in POMDP”. In: *Proceedings of the 38th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2021, pp. 9744–9755. url: https://proceedings.mlr.press/v139/singh21a.html.

[46] Marc G Bellemare et al. “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[47] B. T. Polyak et al. “Acceleration of Stochastic Approximation by Averaging”. In: *SIAM Journal on Control and Optimization* 30.4 (1992). Publisher: Society for Industrial and Applied Mathematics, pp. 838–855. url: https://epubs.siam.org/doi/abs/10.1137/0330046.

[48] Hado Van Hasselt et al. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. Issue: 1. 2016.

[49] Scott Fujimoto et al. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR, July 2018, pp. 1587–1596. url: https://proceedings.mlr.press/v80/fujimoto18a.html.

[50] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR, 2018, pp. 1861–1870. url: https://proceedings.mlr.press/v80/haarnoja18b.html.

[51] Tom Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015).

[52] Ziyu Wang et al. “Dueling network architectures for deep reinforcement learning”. In: *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[53] Matteo Hessel et al. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: *Thirty-second AAAI conference on artificial intelligence* (2017). url: http://arxiv.org/abs/1710.02298.

[54] Meire Fortunato et al. “Noisy networks for exploration”. In: *arXiv preprint arXiv:1706.10295* (2017).

[55] 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). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), pp. 1291–1307. doi: 10.1109/TSMCC.2012.2218595.

[56] Richard S Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999. url: https://papers.nips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

[57] Richard S Sutton et al. “Comparing policy-gradient algorithms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* (2000).

[58] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3 (1992). Publisher: Springer, pp. 229–256.

[59] John Schulman et al. “Trust Region Policy Optimization”. In: (2015). arXiv: 1502.05477. url: http://arxiv.org/abs/1502.05477.

[60] S. Kullback et al. “On Information and Sufficiency”. In: *The Annals of Mathematical Statistics* 22.1 (1951). Publisher: Institute of Mathematical Statistics, pp. 79–86. url: https://www.jstor.org/stable/2236703.

[61] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: (2016). url: http://arxiv.org/abs/1707.06347.

[62] Vijay Konda et al. “Actor-critic algorithms”. In: *Advances in neural information processing systems* 12 (1999).

[63] David Silver et al. “Deterministic policy gradient algorithms”. In: *International conference on machine learning*. PMLR, 2014, pp. 387–395. url: http://proceedings.mlr.press/v32/silver14.html.

[64] Christopher JCH Watkins et al. “Q-learning”. In: *Machine learning* 8.3 (1992). Publisher: Springer, pp. 279–292.

[65] Hado van Hasselt. *DeepMind x UCL RL Lecture Series*. Youtube. 2021. url: https://youtu.be/y3oqOjHilio?t=5384.

[66] D.V. Prokhorov et al. “Adaptive critic designs”. In: *IEEE Transactions on Neural Networks* 8.5 (Sept. 1997). Conference Name: IEEE Transactions on Neural Networks, pp. 997–1007. doi: 10.1109/72.623201.

[67] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: (2015). url: http://arxiv.org/abs/1509.02971.

[68] Tuomas Haarnoja et al. “Soft Actor-Critic Algorithms and Applications”. In: (2019). url: http://arxiv.org/abs/1812.05905.

[69] Joshua Achiam. “Spinning up in deep reinforcement learning”. In: *GitHub repository* (2018). url: https://github.com/openai/spinningup/.

[70] Brian D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

[71] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. url: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[72] Marc G Bellemare et al. *Distributional Reinforcement Learning*. MIT Press, 2022. url: http://www.distributional-rl.org.

[73] Gabriel Barth-Maron et al. “Distributed Distributional Deterministic Policy Gradients”. In: (Apr. 2018). url: http://arxiv.org/abs/1804.08617.

[74] Clare Lyle et al. “A Comparative Analysis of Expected and Distributional Reinforcement Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 4504–4511. doi: 10.1609/aaai.v33i01.33014504. url: https://aaai.org/ojs/index.php/AAAI/article/view/4365.

[75] Tetsuro Morimura et al. “Parametric Return Density Estimation for Reinforcement Learning”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (2010), p. 8.

[76] Marc G. Bellemare et al. “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[77] Carlo Filippi et al. “Conditional value-at-risk beyond finance: a survey”. In: *International Transactions in Operational Research* 27.3 (2020). Publisher: Wiley Online Library, pp. 1277–1319.

[78] Cheng Liu et al. “Adaptive Risk Tendency: Nano Drone Navigation in Cluttered Environments with Distributional Reinforcement Learning”. In: (2022). url: https://arxiv.org/abs/2203.14749.

[79] Antonio Loquercio et al. “A General Framework for Uncertainty Estimation in Deep Learning”. In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 3153–3160. doi: 10.1109/LRA.2020.2974682. url: https://ieeexplore.ieee.org/document/9001195/.

[80] Borislav Mavrin et al. “Distributional Reinforcement Learning for Efficient Exploration”. In: *International Conference on Machine Learning* (2019), pp. 4424–4434. url: http://proceedings.mlr.press/v97/mavrin19a.html.

[81] Derek Yang et al. “Fully Parameterized Quantile Function for Distributional Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. url: https://proceedings.neurips.cc/paper/2019/file/f471223d1a1614b58a7dc45c9d01df19-Paper.pdf.

[82] Thibaut Théate et al. “Distributional Reinforcement Learning with Unconstrained Monotonic Neural Networks”. In: (June 2021). url: http://arxiv.org/abs/2106.03228.

[83] Fan Zhou et al. “Non-decreasing Quantile Function Network with Efficient Exploration for Distributional Reinforcement Learning”. In: *arXiv:2105.06696 [cs]* (May 2021). url: http://arxiv.org/abs/2105.06696.

[84] Abbas Abdolmaleki et al. “A Distributional View on Multi-Objective Policy Optimization”. In: *Proceedings of the 37th International Conference on Machine Learning* 119 (2020), pp. 11–22. url: https://proceedings.mlr.press/v119/abdolmaleki20a.html.

[85] Gabriel Dulac-Arnold et al. “An empirical investigation of the challenges of real-world reinforcement learning”. In: *arXiv:2003.11881 [cs]* (Mar. 2021). url: http://arxiv.org/abs/2003.11881.

[86] Killian Dally. *Deep Reinforcement Learning for Flight Control*. Master Thesis. Delft: Technical University of Delft, 2021. url: http://resolver.tudelft.nl/uuid:fcef2325-4c90-4276-8bfc-1e230724c68a.

[87] Javier Garcıa et al. “A comprehensive survey on safe reinforcement learning”. In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[88] Tommaso Mannucci et al. “SHERPA: a safe exploration algorithm for Reinforcement Learning controllers”. In: *AIAA Guidance, Navigation, and Control Conference*. 2015, p. 1757.

[89] Tommaso Mannucci et al. “Safe exploration algorithms for reinforcement learning controllers”. In: *IEEE transactions on neural networks and learning systems* 29.4 (2017). Publisher: IEEE, pp. 1069–1081.

[90] Tijmen Pollack et al. *Safe Curriculum Learning for Primary Flight Control*. Master Thesis. Delft: Technical University of Delft, 2019. url: http://resolver.tudelft.nl/uuid:1b2becfd-c2db-43fc-a273-a3ff6a9ba50a.

[91] Matthias Hutsebaut-Buysse et al. “Hierarchical Reinforcement Learning: A Survey and Open Research Challenges”. In: *Machine Learning and Knowledge Extraction* 4.1 (Feb. 2022), pp. 172–221. doi: 10.3390/make4010009. url: https://www.mdpi.com/2504-4990/4/1/9.

[92] Richard S. Sutton et al. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial Intelligence* 112.1-2 (Aug. 1999), pp. 181–211. doi: 10.1016/S0004-3702(99)00052-1. url: https://linkinghub.elsevier.com/retrieve/pii/S0004370299000521.

[93] Ronald Edward Parr. *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley, 1998.

[94] Thomas G. Dietterich. *Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition*. Tech. rep. arXiv:cs/9905014. arXiv:cs/9905014 type: article. arXiv, May 1999. doi: 10.48550/arXiv.cs/9905014. url: http://arxiv.org/abs/cs/9905014.

[95] J M Hoogvliet. *Hierarchical Reinforcement Learning for Model-Free Flight Control*. Master Thesis. Technical University of Delft, 2019. url: http://resolver.tudelft.nl/uuid:d66efdb7-d7c7-4c44-9b50-64678ffdf60d.

[96] Tengyang Xie et al. "Policy finetuning: Bridging sample-efficient offline and online reinforcement learning". In: *Advances in neural information processing systems* 34 (2021).

[97] 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). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems, pp. 142–160. doi: 10.1109/TSMC.2020.3042876.

[98] Y. Zhou. "Online reinforcement learning control for aerospace systems". PhD thesis. Delft University of Technology, 2018. doi: 10.4233/UUID:5B875915-2518-4EC8-A1A0-07AD057EDAB4. url: http://resolver.tudelft.nl/uuid:5b875915-2518-4ec8-a1a0-07ad057edab4.

[99] Hangxu Li et al. "Incremental Dual Heuristic Dynamic Programming Based Hybrid Approach for Multi-Channel Control of Unstable Tailless Aircraft". In: *IEEE Journals & Magazine* (2022). url: https://ieeexplore.ieee.org/abstract/document/9734032.

[100] S. Heyer. *Reinforcement Learning for Flight Control*. Master Thesis. Delft: Technical University of Delft, 2019. url: http://resolver.tudelft.nl/uuid:dc63cae7-4289-47c7-889e-253f7abd7c72.

[101] Cristian Bodnar et al. "Quantile QT-Opt for Risk-Aware Vision-Based Robotic Grasping". In: *Robotics: Science and Systems XVI* (July 2020). doi: 10.15607/RSS.2020.XVI.075. url: http://arxiv.org/abs/1910.02787.

[102] Jinyoung Choi et al. "Risk-Conditioned Distributional Soft Actor-Critic for Risk-Sensitive Navigation". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, May 2021, pp. 8337–8344. doi: 10.1109/ICRA48506.2021.9560962. url: https://ieeexplore.ieee.org/document/9560962/.

[103] Danial Kamran et al. "Minimizing Safety Interference for Safe and Comfortable Automated Driving with Distributional Reinforcement Learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Prague, Czech Republic: IEEE, Sept. 2021, pp. 1236–1243. doi: 10.1109/IROS51168.2021.9636847. url: https://ieeexplore.ieee.org/document/9636847/.

[104] Yizhang Dong et al. "Self-learned suppression of roll oscillations based on model-free reinforcement learning". 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.

[105] *Pendulum - Gym Documentation*. 2022. url: https://www.gymlibrary.ml/environments/classic_control/pendulum/.

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

[107] *Citation PH-LAB - Delft University of Technology*. 2022. url: https://cs.lr.tudelft.nl/citation.

[108] Siddharth Mysore et al. "Regularizing Action Policies for Smooth Control with Reinforcement Learning". In: 2021.

[109] Fabian Grondman et al. "Design and flight testing of incremental nonlinear dynamic inversion-based control laws for a passenger aircraft". In: *2018 AIAA Guidance, Navigation, and Control Conference*. 2018, p. 0385.

[110] CAAM Van Der Linden. "DASMAT-Delft University aircraft simulation model and analysis tool: A Matlab/Simulink environment for flight dynamics and control analysis". In: *Series 03: Control and Simulation 03* (1998).

[111] Ye Zhou et al. “Incremental model based online dual heuristic programming for nonlinear adaptive control”. In: *Control Engineering Practice* 73 (Apr. 2018), pp. 13–25. url: https://linkinghub.elsevier.com/retrieve/pii/S096706611730285X.

[112] Casper Teirlinck et al. *Reinforcement Learning for Flight Control: Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance*. Master Thesis. Technical University of Delft, 2022. url: http://resolver.tudelft.nl/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85.