Reinforcement Learning for Flight Control

Sim-to-Real Transfer of a Soft Actor-Critic Flight Control System with Long Short-Term Memory Deep Neural Networks

Matan Neumark

TU Delft

Reinforcement Learning for Flight Control

Sim-to-Real Transfer of a Soft Actor-Critic Flight Control System with Long Short-Term Memory Deep Neural Networks

Thesis report

by

Matan Neumark

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on August 26, 2025 at 13:00

Thesis committee:

Chair:	Dr. F. Oliviero
Supervisors:	Dr. C. Varriale
	Dr. ir. E. van Kampen
	Ir. R. Konatala
External examiner:	Dr. X. Wang
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	November 2024 - August 2025
Student number:	5920078

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

Faculty of Aerospace Engineering · Delft University of Technology

Preface

Toen ik naar Nederland kwam, net vóór de coronacrisis, had ik nooit kunnen geloven dat ik in vijf jaar tijd zoveel zou bereiken. Van een BSc in Werktuigbouwkunde aan de TU Eindhoven, waar ik ook heb leren zweefvliegen en Nederlands heb geleerd, tot een MSc in Luchtvaart- en Ruimtevaarttechniek aan de TU Delft, iets waar ik al jaren van droomde. Ik kwam hier met een missie, en nu kan ik eindelijk zeggen: missie geslaagd. Ik ben dankbaar voor de kansen die ik als student in Nederland heb gekregen, en voor de mensen die mij onderweg hebben geholpen.

Be it a design that gets to be manufactured, an assembly that comes together, or a reinforcement learning agent that *finally* learns, what makes engineering exciting for me are the 'it's alive!' moments where everything comes together to fulfil the intended purpose. More often than not, reaching these moments requires persevering through various challenges, setbacks, and much hard work, making them all the more rewarding.

This thesis has certainly presented me with plenty of challenges and setbacks, but also with appropriately rewarding moments. And so, I would like to thank my supervisors, Erik-Jan van Kampen, Ramesh Konatala, and Carmine Varriale, for the opportunity to work on this topic and for the guidance throughout the project's duration. My utmost gratitude goes to my parents; the confidence to pursue my aspirations stems from your unconditional support and faith in me, and for that, I am grateful. My gratitude extends to my sister, who has been and continues to be my role model, and to the rest of my family for their encouragement and love.

Matan Neumark
Rotterdam, August 2025

Contents

Nomenclature

List of Abbreviations

ACD Adaptive Critic Designs

ADDHP Action Dependent Dual Heuristic Programming

ADGDHP Action Dependent Global Dual Heuristic Programming

ADHDP Action Dependent Heuristic Dynamic Programming

ADP Approximate Dynamic Programming

AI Artificial Intelligence

ANDI Adaptive Nonlinear Dynamic Inversion

ANN Artificial Neural Network

BPTT Backpropagation Through Time

CA Control Activity

CoG Centre of Gravity

DASMAT Delft University of Technology Aircraft Simulation Model and Analysis Tool

DDPG Deep Deterministic Policy Gradient

DHP Dual Heuristic Programming

DNN Deep Neural Network

DoF Degree of freedom

DP Dynamic Programming

DPG Deterministic Policy Gradient

DQN Deep Q Network

DRL Deep Reinforcement Learning

EASA European Union Aviation Safety Agency

EOM Equations of Motion

ESA European Space Agency

FAA Federal Aviation Administration

FCL Flight Control Law

FCS Flight Control System

FF Feedforward

GDHP Global Dual Heuristic Programming

GPI Generalized Policy Iteration

HDP Heuristic Dynamic Programming

IADP Incremental Approximate Dynamic Programming

IDHP Incremental Dual Heuristic Programming

IGDHP Incremental Global Dual Heuristic Programming

IHDP Incremental Heuristic Dynamic Programming

INDI Incremental Nonlinear Dynamic Inversion

KL Kullback-Leibler

LC Linear Controller

LOC-I Loss of Control In-flight

LSTM Long Short-Term Memory

LTI Linear Time Invariant

MC Monte Carlo

MSBE Mean Square Bellman Error

MTV Mean Total Variation

NDI Nonlinear Dynamic Inversion

nMAE Normalized Mean Absolute Error

PID Proportional-Integral-Derivative

PPO Proximal Policy Optimisation

ReLU Rectified Linear Unit

RL Rate Limit

RL Reinforcement Learning

RNN Recurrent Neural Network

SAC Soft Actor-Critic

SGD Stochastic Gradient Descent

SOB Simulation Optimisation Bias

TC Test Case

TD Temporal Difference

TD3 Twin-Delayed Deep Deterministic Policy Gradient

TF Transfer Function

TRPO Trust Region Policy Optimisation

UAS Undesired Aircraft State

UAV Unmanned Aerial Vehicle

VPG Vanilla Policy Gradient

VTOL Vertical Take-off and Landing

ZSG Zero-Shot Generalisation

List of Symbols

δ Temporal difference error

α Angle of attack [rad]

α Learning rate

α Temperature coefficient

$\bar{\mathcal{H}}$ Target Entropy

β Angle of sideslip [rad]

Δt Sample step [s]

δ_a Aileron deflection angle [rad]

δ_e Elevator deflection angle [rad]

δ_r Rudder deflection angle [rad]

γ Discount factor

$\hat{s}$ Observable state

λ Learning rate

u Input vector

w Weights of an artificial neural network layer

x State vector

$\mathcal{A}$ Action space

$\mathcal{B}$ Number of samples in a minibatch

$\mathcal{D}$ Experience buffer capacity

$\mathcal{H}$ Entropy

$\mathcal{M}$ A minibatch

$\mathcal{S}$ State Space

μ Mean

ϕ Roll angle in the vehicle-carried normal-earth reference frame [rad]

π Policy

π^* Optimal Policy

π_ϕ	Parametric approximation of the policy
ψ	Heading angle in the vehicle-carried normal-earth reference frame $[\text{rad}]$
σ	Sample standard deviation
τ	Polyak smoothing factor
τ	Time constant $[\text{s}]$
θ	Pitch angle in the vehicle-carried normal-earth reference frame $[\text{rad}]$
a	An action
d	Binary episode termination signal
G	Return
J_π	Policy loss function
J_Q	State-action value function loss function
J_α	Temperature coefficient loss function
p	Roll rate in the body-fixed reference frame $[\text{rad}/\text{s}]$
q	Pitch rate in the body-fixed reference frame $[\text{rad}/\text{s}]$
$q(s, a)$	State-action value function
$q_*(s, a)$	Optimal state-action value function
Q_θ	Parametric approximation of the state-action value function
r	A reward
r	Yaw rate in the body-fixed reference frame $[\text{rad}/\text{s}]$
$r(s, a)$	Reward function
s	A state
t	Time step
$v(s)$	State value function
$v_*(s)$	Optimal state value function
V_{TAS}	True airspeed $[\text{m}/\text{s}]$

List of Figures

List of Tables

1

Introduction

1.1. Background

Commercial aviation is considered to be one of the safest modes of transport and has been continually improving over the last fifty years, as illustrated in Figure 1.1. This can be attributed to improvements in technology, operations, training and safety management [1, 2]. In the technological front, increased safety can be attributed in part to improvements in the automation of flight operations. Automatic flight control systems (FCS) reduce pilot workload and assist in navigation and in maintaining control of otherwise unstable aircraft. Yet, loss of control in-flight (LOC-I) remains a prominent cause of fatal accidents. LOC-I is defined as an unrecoverable loss of control of the aircraft and can result from various factors. These may include pilot error, atmospheric conditions such as icing or turbulence, and technical failures such as reduced control authority or loss of power. According to a safety and accident prevention report by the IATA, new technologies that can assist pilots to recover from an undesired aircraft state (UAS) can improve safety when it comes to LOC-I related accidents [3]. The need for more robust FCS is further emphasised when considering the emergence of new types of aircraft for urban air mobility, such as VTOLs and other (electric) configurations with distributed propulsion. Their unique designs and operational use cases introduce new risks when it comes to loss of control [4].

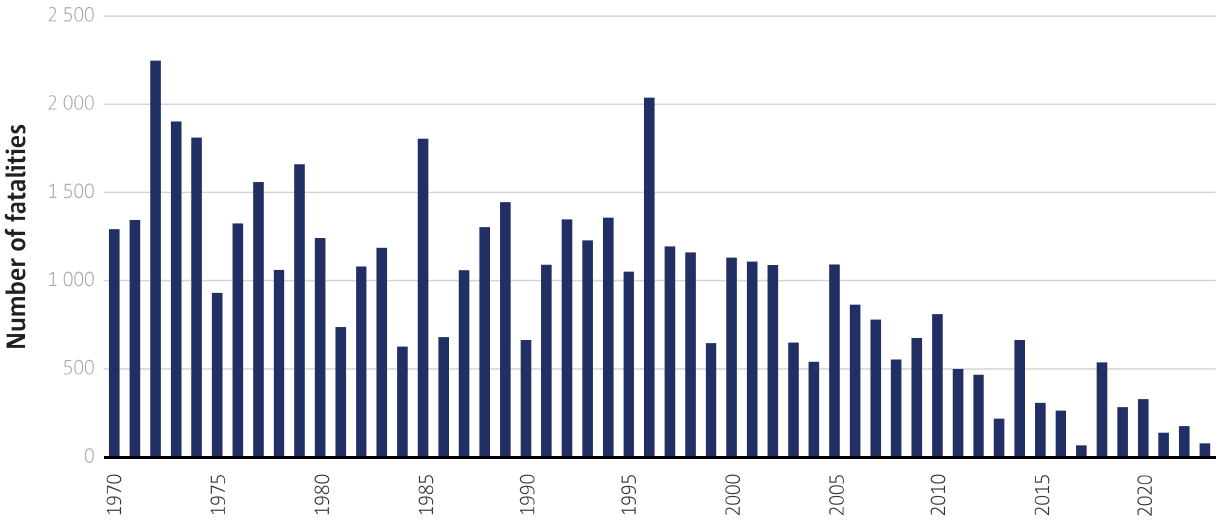


Figure 1.1: Fatalities involving large passenger and cargo aircraft worldwide. Image adapted from EASA's annual safety report 2024 [2].

Traditional FCS are designed by tuning multiple sets of gains, each one optimising the performance of a linear controller (LC) for a given operational condition. Together, the resulting sets of gains span the aircraft's entire flight envelope [5]. During flight, the flight computer interpolates between the pre-tuned gains based on the current flight conditions. While the technique, known as gains scheduling, is reliable, it has substantial limitations. The main limitation when it comes to robustness is that the performance of

linear controllers significantly degrades when the system dynamics change with respect to the dynamics the controller was tuned for. So, any anomaly in the dynamics, such as damaged control surfaces, icing or (partial) loss of lifting surfaces, would reduce the performance of the FCS to unsafe levels, thus requiring human intervention. While it is possible in principle to tune a LC such that it can handle various faults in advance, the possibilities for what can go wrong are endless. So, this method is inherently not fault-tolerant. There exist nonlinear control methods which do not require gain scheduling and have the potential to solve the deficiencies of LC. For example, Nonlinear dynamic inversion (NDI) is a model-based method in which the dynamics of the plant are linearised globally, instead of at discrete points [6]. Adaptive NDI (ANDI) can instead adapt online by updating the parameters of a known model of the plant. Lastly, there is the incremental NDI variant (INDI), which reduces the model dependency of the aforementioned methods. In turn, it requires sensors that can measure accelerations as well as knowledge of the aircraft's control effectiveness coefficients. The challenge here is that during a fault, their value is likely to change, thus not solving the issue of fault tolerance.

Reinforcement learning (RL) comes in as a technique with the potential to produce robust, fault-tolerant control laws while being entirely model-free. RL is a subfield of machine learning in which an agent learns a desired behaviour through interaction with an environment. In contrast to supervised-learning techniques, where existing data is used for training an artificial neural network (ANN), in RL, the data is produced by the agent as it explores the environment. During exploration, the agent receives a reward when its behaviour achieves a desired goal. And so, "learning" means finding a behaviour that maximises the expected sum of rewards (the return) over time. The learnt behaviour is represented by a *policy*, which can be a simple table with instructions based on the agent's state, or it can be embodied in a deep neural network (DNN) when the state and actions are continuous. RL is used in various fields and has demonstrated success in robotic and aeronautical applications such as autonomous drone racing [7], real-time locomotion learning [8] and has even been used to learn a flight control law online [9]. Despite its success, there are several challenges in the application of RL to flight control. First, the resultant behaviour is unpredictable. Learning online means handing over control to a controller whose performance was not verified a priori. On the other hand, learning offline enables an extensive validation and verification process to be carried out, but comes with additional challenges. Training an RL agent offline requires a simulation model of the environment, which is never identical to the real system. So, the agent has to be insensitive to the differences between simulation and reality.

Previous work on robust, fault-tolerant FCS [10] has shown that RL can indeed produce robust policies. There, it was shown in simulation that a soft actor-critic (SAC) agent can control the altitude and attitude of a full degrees-of-freedom model of the Cessna Citation II, TU Delft's research aircraft, shown in Figure 1.2. The FCS proved to be robust against adverse weather conditions such as gusts and icing, and demonstrated the ability to maintain control while facing various types of failures, including a jammed rudder, reduced control effectiveness, and a reduction in the control surface range of motion. In [11], it was shown that an agent can be trained to maintain handling qualities requirements when controlling the pitch rate of the Citation. Other work proposed a hybrid online-offline method in which the sim-to-real gap can be bridged by adapting the controller online [12, 13]. The work done so far, while promising, offers limited solutions when it comes to the prospect of sim-to-real transfer, which is essential for the next step in research: flight testing.

Figure 1.2: TU Delft's Cessna Citation II (PH-LAB) research aircraft. Image by Andre Pronk, licensed under CC BY-SA 2.0.

The objective of this thesis, as formalised below, is broken down into several intermediate objectives which are realised through the literature review, preliminary analysis, and the scientific article. Following the objective are three research questions which define the scope of the study. The various research activities and the project's timeline can be found in Appendix B.

1.2. Research Formulation

Research Objective

To enable the implementation and flight testing of an offline reinforcement learning (RL) flight control system (FCS) in TU Delft's Cessna Citation II research aircraft by bridging the simulation-reality gap, thereby contributing to the development of robust and fault-tolerant FCS meant to improve safety, performance and autonomy.

RO1 Identify the most suitable offline RL framework for a FCS for the Cessna Citation II by reviewing literature and the current state-of-the-art applications.
RO2 Identify the main challenges and solutions relating to systems and control in transferring an offline synthesised controller to a real system.
RO3 Design and develop a pitch rate controller utilising the chosen RL framework and test in simulation against a baseline controller.
RO4 Extend the framework to enable pitch attitude control, test in a simulation environment that emulates sim-to-real transfer, and compare against a baseline controller.

Research Question 1

Which RL algorithm is the most appropriate for the development of a FCS for the Cessna Citation II?

1.1 What are the state-of-the-art RL algorithms and applications for flight control?
1.2 Which requirements must the FCS meet?
1.3 What are the assumptions, and how do they influence the solution space?

Research Question 2

What are the challenges in transferring from simulation to reality, and what are the solutions that can enable implementation in the Cessna Citation II?

2.1 What are, in general, the challenges with sim-to-real transfer and which are unique to RL-based controllers?
2.2 Which aspects of the real aircraft's dynamics are not accounted for in the available simulation model, but are relevant in enabling sim-to-real transfer?
2.3 How should the control architecture be designed?
2.4 How can the real system be emulated in simulation to facilitate testing of the sim-to-real transfer?

Research Question 3

How does the proposed RL-based controller perform?

3.1 What are the most appropriate performance indicators and how should they be measured?
3.2 How well does the controller generalise across different tracking tasks?
3.3 How sensitive is it to the various sensor and actuator dynamics?
3.4 How does it compare to other controllers in terms of performance and sim-to-real transferability?

1.3. Report Outline

This report begins with a scientific article in Part I in which the main results, sim-to-real transfer of a pitch attitude SAC-based FCS, are presented. Part II contains the literature study, found in Chapter 3, where theoretical concepts are laid out and synthesised into a set of assumptions and requirements for the preliminary analysis in Chapter 4. There, the SAC algorithm is developed, and its applicability to a pitch rate controller is validated. Lastly, Part III includes the conclusions, found in Chapter 5, and a set of recommendations in Chapter 6.

Part I

Scientific Article

Sim-to-Real Transfer of a Soft Actor-Critic Reinforcement Learning Flight Control System

Matan Neumark*
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Advancements in deep reinforcement learning (RL) open the door to the development of robust flight control systems (FCS) that have the potential to improve both safety and performance during off-nominal flight conditions. Simulation-based work on offline-RL FCS has already demonstrated robustness to adverse weather conditions, mechanical failures, and a wide range of operational conditions. However, it has neglected important dynamical phenomena that limit its applicability to reality. In anticipation of a future flight testing campaign of similar RL-based FCS, this research emulates the transition from simulation to reality by modelling prevalent sensor and actuator dynamics, and introduces a method to incorporate a long short-term memory (LSTM) artificial neural network (ANN) into the policy of a Soft Actor-Critic (SAC) agent. The approach is found to largely diminish the sensitivity of the controller to sensor noise and actuator dynamics, while increasing its robustness to delays in comparison with the ubiquitous feedforward deep neural network (DNN) and a traditional linear controller.

Nomenclature

$p,\ q,\ r$	=	Roll, pitch and yaw rates in the body-fixed reference frame [rad/s]
$\phi,\ \theta,\ \psi$	=	Roll, pitch and yaw angles in the vehicle-carried normal-earth reference frame [rad]
$\alpha,\ \beta$	=	Angle-of-attack and angle of sideslip [rad]
$\delta_e,\ \delta_a,\ \delta_r$	=	Deflection angle of the elevator, aileron, and rudder [rad]
V_{TAS}	=	True airspeed [m/s]
$\mathbf{x},\ \mathbf{u}$	=	State and input vectors
$\mu,\ \sigma$	=	Mean and sample standard deviation
γ	=	Discount factor
λ	=	Learning rate
τ	=	Polyak smoothing factor
$\mathcal{S},\ \mathcal{A}$	=	State and action spaces
$\lvert\mathcal{S}\rvert,\ \lvert\mathcal{A}\rvert$	=	Number of states and actions
$\mathbf{s},\ \mathbf{a}$	=	A state and an action in $\mathcal{S}$ and $\mathcal{A}$, respectively
$\hat{\mathbf{s}}$	=	Observable state
$r(\mathbf{s})$	=	Reward function
$\pi,\ \pi^*$	=	Policy and optimal policy
π_ϕ	=	Parametric approximation of the policy
$Q_\theta,\ Q_{\bar{\theta}}$	=	Parametric approximation of the state-action value function and its target
α	=	Temperature coefficient
$J_\pi,\ J_Q,\ J_\alpha$	=	Loss function for the policy, state-action value function and temperature coefficient
$\lvert\phi\rvert,\ \lvert\theta\rvert,\ \lvert\bar{\theta}\rvert$	=	Number of hidden units in the deep neural networks of π, Q and $\bar{Q}$,
$\mathcal{H},\ \bar{\mathcal{H}}$	=	Entropy and target entropy
$\mathcal{D},\ \mathcal{B}$	=	Size of the experience buffer and minibatch
$\mathcal{M}$	=	A minibatch of experience samples
$K_P,\ K_I,\ K_D$	=	Proportional, integral, and derivative gains of a linear controller
N_D	=	Derivative filter coefficient
$t,\ \Delta t$	=	Time step [−] and sample time [s]

*M.Sc. Student in the Flight Performance and Propulsion Group, Faculty of Aerospace Engineering, Delft University of Technology.

I. Introduction

Aircraft today rely heavily on automated control and navigation systems to improve safety and optimise operational efficiency. With the emergence of new aircraft configurations, such as UAVs and manned (e)VTOL for logistical operations and urban transportation, it is expected that reliance on automated, safety-critical systems will only increase. Still, loss of control in-flight (LOC-I) is one of the leading causes of fatal accidents in commercial aviation [1, 2]. LOC-I refers to a loss of control resulting in unrecoverable veering of the aircraft from its intended course [1]. It can be caused by environmental conditions such as turbulence or icing, technical failure such as decreased control authority or loss of power, or crew error. Ideally, an aircraft's flight control systems (FCS) would be robust enough to maintain control during adverse conditions or technical faults, assuming controllability can be ascertained despite the aircraft's altered dynamics.

Traditional FCS are designed using gain scheduling, a technique in which the nonlinear dynamics are approximated by a series of linear time-invariant (LTI) models, linearised at different operational conditions which span the entire operational envelope. Then, for each LTI model, the gains of a linear controller are tuned [3]. During flight, the FCS interpolates between sets of gains based on the current operational conditions, which can be measured or approximated by the aircraft's sensors. The main limitation of this control strategy is that a linear controller is only effective if the model used for gain tuning is similar to the actual system, which is not the case during off-nominal flight conditions. Alternative model-based approaches such as nonlinear dynamic inversion (NDI) avoid gain scheduling by globally linearising the nonlinear dynamics [4], whereas its adaptive counterpart (ANDI) can adapt online to changes in the system's dynamics. Then, the incremental NDI (INDI), while reducing model dependency, still requires knowledge of the control effectiveness matrix and specialised instruments to measure angular accelerations.

Reinforcement learning (RL), a branch of machine learning, is a model-free approach that can be used to produce robust controllers and has been successfully employed in various robotic and aeronautical applications [5–7]. The idea of RL is to train an agent to learn a desired behaviour through interaction with an environment. The behaviour is embodied in a *policy* which, in the continuous domain, is often represented by deep neural networks (DNN) because they can be trained to fit nonlinear functions implicitly found in datasets. In RL, unlike in supervised learning techniques, the data required to train the network is generated by the agent through exploration of the environment. RL agents are often trained offline (i.e in simulation) to reduce the costs and risks associated with learning on the real system.

Several offline, RL-based FCS have already been designed specifically for the Cessna Citation II (PH-LAB), TU Delft's research aircraft, shown in Figure 2. In [8], an incremental, coupled-dynamics, cascaded altitude-attitude controller was developed using the soft actor-critic (SAC) RL algorithm. The FCS, embodied in a feedforward DNN, demonstrated robustness to various types of failures and flight conditions, highlighting the potential RL-based controllers have in improving aviation safety. It was also tested with biased sensor noise and a low-pass filter as a model of the elevator. The FCS exhibited almost no degradation in tracking performance, albeit some noise in the control commands. In [9], an SAC-based, incremental pitch rate controller was designed for the Citation with the goal of meeting handling qualities criteria. There, the FCS was able to handle actuator rate limits despite not being trained with them. To further progress the research field, it is essential to test and validate this type of controllers in flight. However, the transition from simulation to reality is not trivial because the policy is optimised on a model of the system's dynamics, which can differ significantly from reality, giving rise to the so-called simulation-reality gap. Both [8] and [9] made considerable simplifications in the modelling of sensors and actuators, which limit their applicability to simulation. One approach may be to use high-fidelity models which capture higher-order dynamics, but these are often unavailable or impractical to train with due to their high computational cost. So, ideally, an agent trained in a lean simulation environment would learn a robust policy that generalises to the dynamics of the real system.

The contribution of this paper is in the development of a model-free, offline RL-based FCS that is robust against sensor and actuator dynamics such as measurement noise, bias, and delay, along with a realistic actuator model in the form of a rate-limited first-order transfer function. To achieve this, the Soft Actor-Critic algorithm [10] is modified to facilitate a long short-term memory (LSTM) layer in both the actor and critic. The agent is trained in a longitudinal, linearised-dynamics environment, simulating the Cessna Citation II CS-25 aircraft with the task of learning to control the aircraft's pitch attitude. The agent's performance is evaluated with a high-fidelity, nonlinear dynamics model and compared with the performance of both a linear controller and an additional SAC policy with a strictly feedforward actor-critic. The proposed training and testing framework is meant to emulate the sim-to-real transition, demonstrating what it takes to bridge the simulation-reality gap and pave the road towards flight testing.

The paper begins by introducing the fundamentals of RL and the core challenges related to the simulation-reality gap in section II. The same section also explains the Soft Actor-Critic algorithm and its implementation with LSTM layers. The design of the SAC controller and a baseline linear controller are discussed in section III along with a description of

the simulation environment and the experimental setup. The results and discussion, found in section IV, first treat the learning performance of the various agents. Then, a general performance overview, obtained from various simulations, is presented. These are analysed further in a sensitivity analysis in subsection IV.C, leading to the main results concerning the emulated sim-to-real transfer as discussed in subsection IV.D. Finally, the conclusions are presented in section V.

II. Preliminaries

A. Reinforcement Learning

RL problems are framed with the Markov Decision Process (MDP), according to which the agent's current state contains all necessary information for choosing the optimal action [11]. An MDP consists of the tuple $\langle \mathcal{S},\ \mathcal{A},\ \mathcal{P}_{\mathcal{A}},\ \mathbf{r},\ \gamma \rangle$ in which $\mathcal{S}$ and $\mathcal{A}$ are the continuos state and action spaces, $\mathcal{P}_{\mathcal{A}}$ represents the probability of transitioning from state $\mathbf{s}$ to the next state $\mathbf{s}'$ having taken the action $\mathbf{a}$, and $\mathbf{r}$ is the reward earned by the agent for this transition. The cumulative reward, known as the return, is bound to a finite number by discounting it with a factor $0 < \gamma < 1$.

In reality, environments are rarely Markovian due to unmodelled dynamics and delays in the states and actions, causing the state $\mathbf{s}$ to be *partially* observable, which is symbolised with $\hat{\mathbf{s}}$. Problems of this kind are referred to as partially observable MDP (POMDP) and can be denoted by $\langle \mathcal{S},\ \mathcal{A},\ \mathcal{P}_{\mathcal{A}},\ \mathbf{r},\ \gamma,\ d_s,\ d_a \rangle$ where d_s and d_a represent the delays in the observed states and the applied actions, respectively. According to [12] a delay in the action is equivalent to a delay in the state and therefore a POMDP with constant delays can be reduced to an MDP by augmenting the state $\mathbf{s}_t$ with actions up to time $t - d_s - d_a$ such that the state-space becomes $\mathcal{S} \times \mathcal{A}^{d_s+d_a}$ with the augmented state $[\mathbf{s}_{t-d_s},\ \mathbf{a}_{t-1},\ \dots\ \mathbf{a}_{t-d_s-d_a}]$. Augmenting the state increases its size by $\mathcal{O}(d \cdot |\mathcal{A}|)$, which can become prohibitively expensive for systems with substantial delays and many actuators. An alternative to augmenting the state with actions up to $t - d_s - d_a$ is to equip the agent with some form of memory, enabling it to hold onto previous interactions and learn their hidden time-dependencies. One way to achieve this is to incorporate Long Short-Term Memory (LSTM) layers in the agent's neural networks. LSTMs are an evolution of the Recurrent Neural Network (RNN) and can hold on to information spanning many time steps while solving the vanishing gradient problem of the former [13]. With LSTM layers incorporated, it is sufficient to augment the state with $\mathbf{a}_{t-1}$, irrespective of the number of delayed time steps.

B. Soft Actor Critic

Soft Actor Critic (SAC) is an off-policy RL algorithm that, in addition to the objective of maximising the expected return, has the objective of maximising the policy's entropy according to Equation 1 [10]. The entropy, as given by Equation 2 is a measure of the likelihood of an outcome and is high for unlikely events. Including the entropy as an optimisation objective ensures that the policy remains stochastic and exploratory, thus lowering the chance of stagnating at a local optimum. In Equation 1, the entropy is balanced against the return through the temperature coefficient α. This hyperparameter can be implemented as a constant or, better yet, as a self-regulating variable as done here. To facilitate off-policy learning, the experience at each timestep, represented by $[\mathbf{s}_t,\ \mathbf{a}_t,\ \mathbf{r}_t,\ \mathbf{s}_{t+1},\ \mathbf{d}_t]$, is normalised and stored in an experience buffer with capacity $\mathcal{D}$. When it is time to update the actor and critic, a minibatch $\mathcal{M}$ of $\mathcal{B}$ experiences is sampled from the buffer at random. To train an agent with LSTM layers in its networks, the minibatch consists of a randomly chosen sequence of samples, preserving their time-dependent dynamics. Figure 1 illustrates the implementation of SAC with an LSTM actor-critic.

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot \mid \mathbf{s}_t)) \right], \tag{1}$$

$$\mathcal{H}(\pi(\cdot \mid \mathbf{s})) = \mathbb{E}_{\mathbf{a} \sim \pi} \left[-\ln \pi(\mathbf{a} \mid \mathbf{s}) \right] \tag{2}$$

1. Actor

The actor in SAC consists of a parametrised policy π_ϕ where ϕ represents the learnable weights and biases of the DNN. The policy takes a state as input and outputs a mean action μ and standard deviation σ, which, during training, are used to sample a random action with mean μ and Gaussian noise from the distribution $\mathcal{N}(0, \sigma)$. The added noise facilitates the exploration mechanism in SAC and is regulated through the entropy; an action sampled from a distribution with high variance will have a high entropy, which is exactly the objective. Consequently, the sampled action is squashed with a hyperbolic tangent function, bounding it in the range $[-1, 1]$, and the associated entropy is calculated. Note that during inference, the action is sampled deterministically such that $\mathbf{a_t} = \mu_t$.

$$J_\pi = \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \left[\alpha \cdot \ln \pi_\phi(\mathbf{a}_i \mid \mathbf{s}_i) - \min_{j=1,2} (Q_{\theta_j}(\mathbf{s}_i, \mathbf{a}_i)) \right] \tag{3}$$

2. Critic

The critic uses two Q-functions, an approach originally proposed in [14] and applied in the TD3 algorithm. The use of double Q-functions is found to reduce overestimation, resulting in increased learning speed and stability [10, 15]. Similar to the policy, the Q-functions are represented by DNNs with learnable parameters θ_1 and θ_2. During actor updates, both Q-functions are evaluated, but only the minimum-valued one is used to calculate the policy loss, as shown in Equation 3. In addition, the critic consists of two target Q-functions parametrised by $\bar{\theta}_1$ and $\bar{\theta}_2$, which are updated via Polyak averaging as per Equation 4 rather than through gradient descent. This soft update mechanism causes the target value functions to lag behind the primary ones, making them a stabilising reference. The smoothing factor τ is a hyperparameter determining how much the target functions lag behind the primary ones. The target state-action values are used in the Bellman backup in Equation 5, which is modified to include the entropy. Finally, the primary state-action values and the backup values are used in the Mean Square Bellman Error (MSBE) loss function shown in Equation 6.

$$\bar{\theta}_j = (1 - \tau) \cdot \bar{\theta}_j + \tau \cdot \theta_j \qquad \text{for } j \in \{1, 2\} \tag{4}$$

$$y_i = \mathbf{r}_i + \gamma \cdot (1 - \mathbf{d}_i) \cdot \left(\min_{j=1,2} (Q_{\bar{\theta}_j}(\mathbf{s}'_i, \mathbf{a}'_i)) - \alpha \cdot \ln \pi_\phi(\mathbf{a}'_i \mid \mathbf{s}'_i) \right) \qquad \text{with: } \mathbf{a}'_i = \pi_\phi(\cdot \mid \mathbf{s}'_i) \tag{5}$$

$$J_Q = \sum_{j=1}^{2} \left[\frac{1}{2\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \left[Q_{\theta_j}(\mathbf{s}_i, \mathbf{a}_i) - y_i \right]^2 \right] \tag{6}$$

3. Automatic Temperature coefficient

As the agent learns, the expected return changes, requiring the weight of the entropy to be adjusted accordingly. as found in [10], the optimal temperature coefficient α is given by Equation 7 which in practice is found through gradient descent using the loss function shown in Equation 8. As suggested in [10], the target entropy $\overline{\mathcal{H}}$ is equal to the negative of the action space's size, such that $\overline{\mathcal{H}} = -|\mathcal{A}|$.

$$\alpha_t^* = \arg\min_{\alpha_t} \mathbb{E}_{\mathbf{a}_t \sim \pi_t^*} \left[-\alpha_t \ln \pi_t^* (\mathbf{a}_t \mid \mathbf{s}_t; \alpha_t) - \alpha_t \overline{\mathcal{H}} \right] \tag{7}$$

$$J_\alpha = -\frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \left[\alpha \cdot \left(\ln \pi_\phi(\mathbf{a}_i \mid \mathbf{s}_i) + \bar{\mathcal{H}} \right) \right] \tag{8}$$

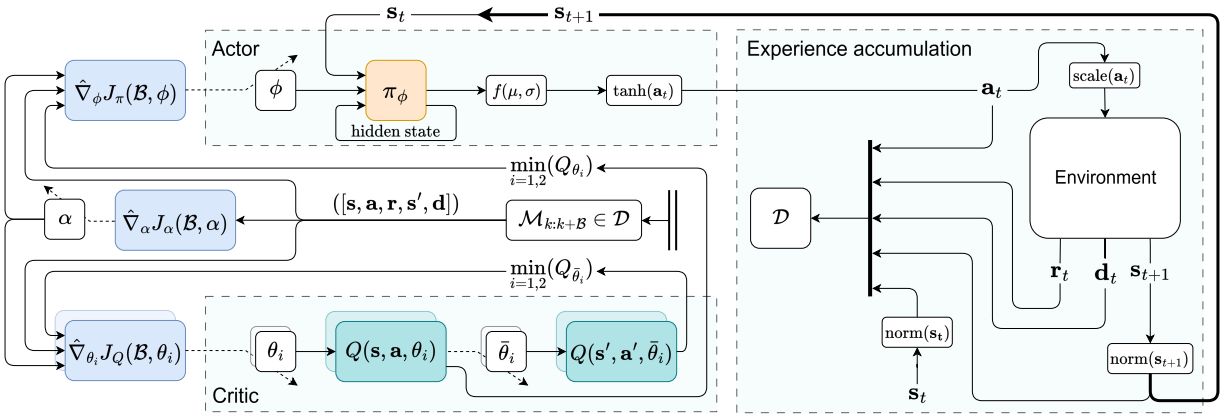


Fig. 1 Diagram of the SAC-LSTM implementation.

Algorithm 1: Soft Actor-Critic with Long Short-Term Memory layers. Adapted from [10]

Initialize: Policy DNN with weights ϕ, Q-function DNNs with weights θ_1, θ_2 and target DNNs with weights $\bar{\theta}_1 = \theta_1, \bar{\theta}_2 = \theta_2$

Hyperparameters: learning rates $\lambda_\phi, \lambda_\theta, \lambda_\alpha$, Polyak update factor τ, buffer size $\mathcal{D}$, minibatch size $\mathcal{B}$, discount factor γ, initial temperature coefficient α_0, target entropy $\bar{\mathcal{H}}$

Get initial state $\mathbf{s}_0$ and $\mathbf{a}_0$ from the environment

for $t = 1$: *number of time steps* **do**

  Sample action $\mathbf{a}_t$ from policy $\pi_\phi(\mathbf{s}_t^{norm},\ \mathbf{a}_{t-1})$

  Update the hidden state and cell state of the LSTM layer

  Execute $\mathbf{a}_t^{scale}$ in the environment and get a new state $\mathbf{s}_{t+1}$, reward $\mathbf{r}_t$, and termination signal d

  Store the experience $\left[\mathbf{s}_t^{norm},\ \mathbf{a}_t,\ \mathbf{r}_t,\ \mathbf{s}_{t+1}^{norm},\ \mathbf{d}\right]$ in the experience buffer

  if *Time to update* **then**

    for *iteration* $= 1 : \lfloor \mathcal{D}/\mathcal{B} \rfloor$ **do**

      Sample a random minibatch $\mathcal{M}$ with sequential samples $k : k + \mathcal{B}$ from the buffer

      Compute the loss of the Critic, Actor, and temperature coefficient

      Calculate the gradients and update the weights:

$$\theta_{1,2} \leftarrow \theta_{1,2} - \lambda_\theta \hat{\nabla}_{\theta_{1,2}} J_Q$$

$$\bar{\theta}_{1,2} \leftarrow (1-\tau) \cdot \bar{\theta}_{1,2} + \tau \cdot \theta_{1,2}$$

$$\phi \leftarrow \phi - \lambda_\phi \hat{\nabla}_\phi J_\pi$$

$$\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J_\alpha$$

    end

    Reset the buffer's *is sampled* flag to allow old experiences to be used again in the next epoch

  end

end

III. Flight Control System Design

A. Simulation Environment

Fig. 2 Image by Andre Pronk, licensed under CC BY-SA 2.0.

The Citation Analysis and Simulation Toolkit (CitAST) offers a high-fidelity simulation package with a nonlinear-dynamics Simulink model of the Citation, shown in Figure 2. It is based on the Delft University Aircraft Simulation Model and Analysis Tool (DASMAT), which has been validated against flight data in [16]. To facilitate training, the nonlinear model is trimmed and linearised at nominal flight conditions with $V_{TAS} = 90\,\text{m/s}$, $m = 4500\,\text{kg}$, and density altitude $h = 2000\,\text{m}$. Additionally, the full degrees-of-freedom model is reduced to longitudinal motions with the state vector including only the pitch rate q, the angle of attack α, and the pitch attitude θ as in Equation 9. The input vector

contains only the elevator angle δ_e as per Equation 10. Several considerations justify the use of such a simplified model. First, being that the focus of the research is on overcoming the sim-to-real gap, longitudinal motions are sufficient to demonstrate the performance that can be obtained with the suggested approach. Second, since a flight testing campaign is not part of this research, training the agent on a linearised model leaves the high-fidelity simulation environment as a proxy of the real aircraft, allowing emulation of sim-to-real transfer. Lastly, the applicability of the results presented in this paper would be limited if a high-fidelity model were a prerequisite, since such models are scarce and computationally expensive to train with.

$$\mathbf{x} = [q,\ \alpha,\ \theta] \tag{9}$$

$$\mathbf{u} = \delta_e \tag{10}$$

1. Sensor and Actuator Dynamics

Information about the aircraft's state comes from a suite of sensors, each with its own characteristics, as summarised in Table 1. The variance of the noise and the magnitude of the bias are modelled as constants, even though in reality, these dynamics are influenced by factors such as temperature. Environmental conditions also affect latency and may add stochasticity to the delay. Despite this, the asynchronous delays are considered constant. Furthermore, the sampling rate is assumed to match the simulation's temporal discretisation of 100Hz for all state variables.

Table 1 Sensor characteristics PH-LAB. Values based on [7, 17].

Signal	**Noise** (σ^2)	**Bias**	**Delay** [ms]	**Sampling Rate** [Hz]	**Signal Unit**
$p,\ q,\ r$	$4.0 \cdot 10^{-7}$	$3.0 \cdot 10^{-5}$	15	1000	[rad/s]
$\theta,\ \phi,\ \psi$	$1.0 \cdot 10^{-9}$	$4.0 \cdot 10^{-3}$	90	52	[rad]
V_{TAS}	$8.5 \cdot 10^{-4}$	2.5	300	8	[m/s]
h	$4.5 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	300	20	[m]
$\delta_a, \delta_e, \delta_r$	$5.5 \cdot 10^{-7}$	$2.4 \cdot 10^{-3}$	10	100	[rad]
$\alpha_{boom},\ \beta_{boom}$	$7.5 \cdot 10^{-8}$	$1.8 \cdot 10^{-3}$	100	100	[rad]

The elevator is modelled by a first-order transfer function with time constant, saturation limits and rate limits as described in Table 2. During flight, the reaction forces acting on the control surfaces depend on factors such as the dynamic pressure and the deflection angle δ, making the actuator's rate a dependent variable. However, as a simplification, the control forces are assumed to be constant, implying that the servo's rate limit remains constant throughout the elevator's range of motion.

Table 2 Actuator characteristics PH-LAB. Values based on [7, 17].

Actuator	**Time Constant** τ [s]	**Deflection Upper Limit** [deg]	**Deflection Lower Limit** [deg]	**Rate Upper Limit** [deg/s]	**Rate Lower Limit** [deg/s]
Aileron	$\frac{1}{13}$	22	−22	20	−20
Elevator	$\frac{1}{13}$	15	−17	20	−20
Rudder	$\frac{1}{13}$	34	−34	20	−20

The agent's environment is set up with the option to independently enable or disable each of the four dynamical properties of the sensors and actuator: measurement noise and bias, measurement and elevator deflection delay, elevator transfer function, and servo rate limit. This setup allows individual effects to be examined, as well as up to sixteen combinations, later denoted as test cases (TC).

B. Reinforcement Learning Controller Implementation

Pitch attitude control with an RL policy is implemented as illustrated in the control loop diagram in Figure 3. The policy's input state shown in Equation 11 contains the system's state vector **x** from Equation 9 along with the reference pitch attitude θ_{ref}, the measured elevator deflection angle δ_e, and the commanded deflection δ_e^{com}. The inclusion of

δ_e^{com} is motivated by the need to inform the agent of its previous actions $\mathbf{a}_{t-1}$ when dealing with delays, as explained in subsection II.A. One of the challenges with the addition of actuator dynamics to the system's model during training is that it restricts exploration because the elevator's transfer function, and the servo rate limit in particular, add a significant lag. Given the specs in Table 2, it takes more than two seconds (200 time steps) to traverse the elevator's entire range of motion. This is contradictory to SAC's stochastic nature, which can sample actions from either end of the range in a single consecutive step. And, as confirmed through experimentation, the agent achieves poor performance when trained with a rate-limited actuator because it fails to discover the full action space. The solution is to train the agent with only the elevator's transfer function because it doesn't have a devastating impact on exploration. Then, by adding δ_e to the policy's input state, the agent has information about the elevator's true position, enabling it to handle the effect of the actuator rate limit despite not experiencing it in training.

$$\mathbf{s} = \left[q,\ \alpha,\ \theta,\ \theta_{ref},\ \delta_e,\ \delta_e^{com}\right] \tag{11}$$

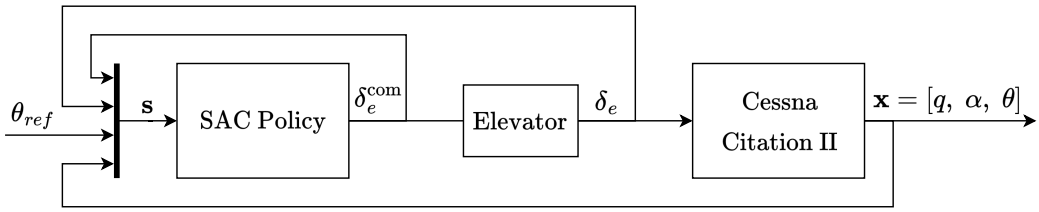


Fig. 3 Pitch attitude control loop with a SAC policy as the controller.

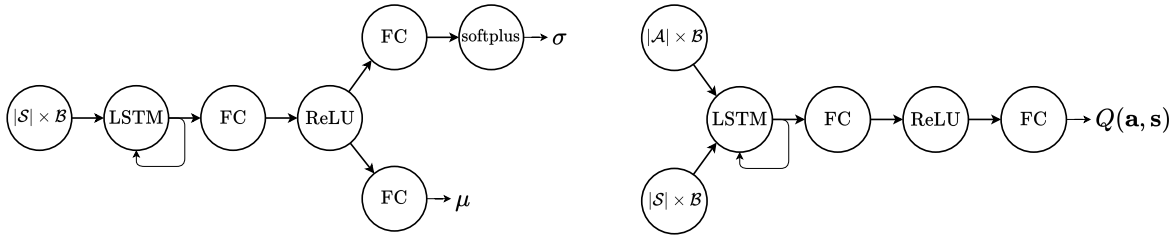
In RL, the agent's objective is defined through a reward function. If poorly designed, the agent may not learn anything of significance, or, more commonly, it will learn to perform the task, but it will do so while employing an undesired behaviour. A typical occurrence in robotic applications of RL is the agent learning a highly active and oscillatory policy, resulting in jerky motions, vibrations, and accelerated mechanical wear [18]. In the case of the Citation, the issue is amplified when controlling the pitch angle θ instead of the pitch rate q because the dynamics of the former are slower than those of the latter. This leads to a situation where the pitch rate can oscillate violently while the error in the pitch angle remains small, thus only technically satisfying the learning goal. To prevent the agent from converging to such a policy, the reward function in Equation 12 penalises the agent for exceeding predefined state variable thresholds as shown in Equation 13. The threshold of 10 deg/s for the pitch rate in Equation 13 is chosen because it is found to be restrictive enough so the agent can't achieve optimal tracking performance by employing an oscillatory behaviour, while allowing sufficiently fast response to tracking errors. An additional method for promoting learning a smooth policy is to square the error term $\theta - \theta_{ref}$ because it reduces the sensitivity of the reward to the error around zero. However, it may come at the cost of an increased steady-state error [19]. The sensitivity away from zero is tuned via a scaling factor which, through experimentation, is chosen to be $180/\pi$. Finally, to mitigate the issue of exploding gradients, the reward is clipped if its value is less than -1.

$$r(\mathbf{s}) = \begin{cases} \text{clip}\left[-\frac{180}{\pi}(\theta - \theta_{\text{ref}})^2,\ -1,\ 0\right] & \text{if } -\mathbf{x}_{\text{lim}} < \mathbf{x} < \mathbf{x}_{\text{lim}} \\ -1 & \text{otherwise} \end{cases} \tag{12}$$

$$\mathbf{x}_{lim} = [10\,\text{deg/s},\ 10\,\text{deg},\ 10\,\text{deg}] \tag{13}$$

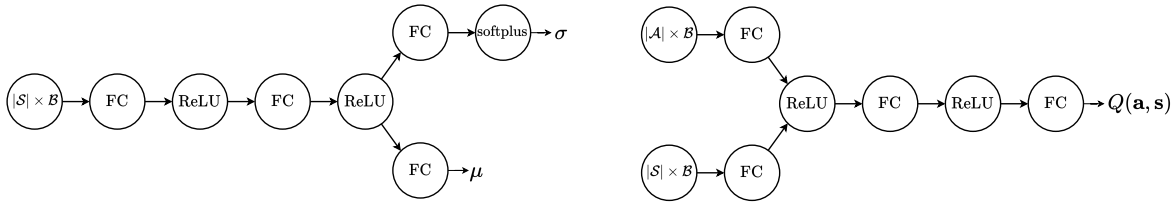
1. Deep Neural Network Architecture

An LSTM layer is incorporated in both the actor and critic, as illustrated in Figure 4. A series of tests, in which several network structures were tested, showed that placing the LSTM layer right after the input works better than placing it after a fully connected (FC) layer. For this reason, the LSTM critic has a single branch, whereas the feedforward (FF) critic shown in Figure 5b has separate FC layers for the state and action inputs before concatenation down to the main branch. The weights of the FC layers in both topologies are initialised with the He method [20] which, compared to the alternatives, increases learning speed and stability when paired with ReLU activation functions [21]. The weights are updated with the Adam optimiser [22] with MATLAB's default hyperparameter values. The number of hidden units and learning rates can be found in Table 4.



(a) Recurrent Actor. **(b) Recurrent critic.**

Fig. 4 DNN architecture for the SAC-LSTM implementation.



(a) Feedforward actor. **(b) Feedforward critic.**

Fig. 5 DNN architecture for the SAC-FF implementation.

C. Linear Controller

The performance of the RL controllers is compared against a baseline linear controller (LC), serving as a means of validation in two facets: Tracking performance and sensitivity to sensor and actuator dynamics. The LC is cascaded with pitch rate in the inner loop and pitch attitude in the outer loop, as shown in Figure 6. The gains for the controller of each loop, detailed in Table 3, were found through a manual tuning process at the nominal flight conditions detailed in subsection III.A with the goal of achieving a balance between tracking performance and sensitivity to disturbances. The tuning process began with tuning the inner loop, followed by the outer loop for the disturbance-free plant. Once the tracking performance was deemed satisfactory (nMAE $< 10\%$) for both a step and a sinusoidal pitch angle reference signal, the controller was fine-tuned to reduce its sensitivity to noise, bias, delay, and the presence of the elevator's transfer function. While the resulting controller can deal with each of these dynamical effects separately, it cannot maintain the system's stability for several of their combinations, as shown later in section IV. The servo's rate limitation is intentionally excluded as an objective of the tuning procedure because it is found to have too great an effect on the system's response, requiring an entirely different set of gains. This emphasises the limitation of LC when it comes to handling changes in the dynamical system and their dependence on accurate models for offline tuning.

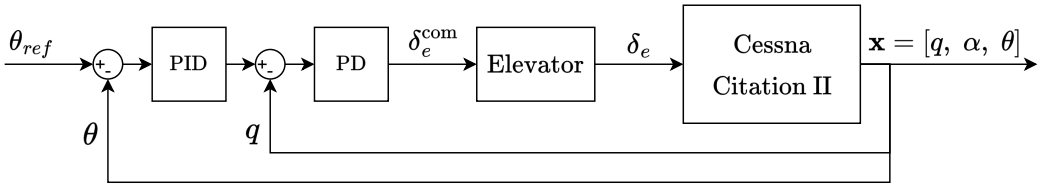
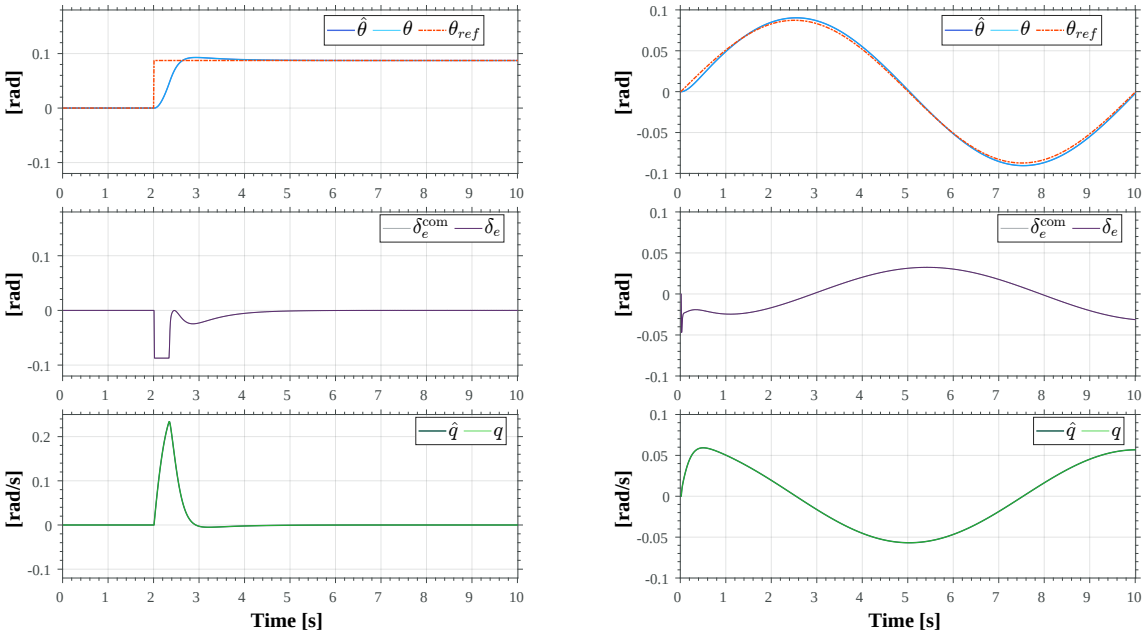


Fig. 6 Pitch attitude control loop with a cascaded linear controller.

Table 3 Gains for the cascaded linear controller.

Controlled variable	K_P	K_I	K_D	N_D	Saturation
θ (Outer loop)	8	12	0.2	100	$\pm 20\,\text{deg/s}$
q (Inner loop)	1.8	0	0.05	25	$\pm 5\,\text{deg}$

In the time response in Figure 7a, the LC demonstrates an overshoot of 6.4% of the reference value and a settling time of less than 1.9 s given a 2% tolerance. The steady state error at the end of the simulation (10 s) is in the order of $1 \cdot 10^{-7}$ rad. Figure 7b demonstrates the LC's ability to track a smoothly varying signal. At the peaks (at 2.5 seconds and 7.5 seconds), the overshoot is approximately 3.5%. These figures are intended to give an impression of the baseline controller's performance and provide context when interpreting the time response plots of the SAC controllers. Maximising tracking performance is not the sole objective when designing a FCS. Factors such as passenger comfort and structural loads must also be considered. Therefore, in the discussion of the results in section IV, the characteristics of the time responses of the SAC controllers are not quantified to the same level of detail as done here for the baseline LC.



(a) LC, TC-16, Linear Env, nMAE= 5.0%, CA= $2.5 \cdot 10^{-2}$ rad/s. **(b) LC, TC-16, Linear Env, nMAE= 4.6%, CA= $2.1 \cdot 10^{-2}$ rad/s.**

Fig. 7 Time response of the linear controller. TC-16: No sensor and actuator dynamics.

D. Experimental Setup

1. Training Method

Each agent is trained in the linear-dynamics environment for 320,000 time steps, divided into episodes of 1,000 steps (10 seconds with $\Delta t = 0.01$ s), unless the episode terminates early because the agent has exceeded the state limits in Equation 13. The availability of a six-core processor enables the acceleration of experience accumulation through parallel simulation of a batch of six episodes. After every such batch, one epoch of actor-critic updates takes place, according to algorithm 1. At the beginning of every episode, a sinusoidal pitch attitude reference signal θ_{ref} is generated. The signal's frequency and maximal amplitude are sampled randomly from a uniform distribution where $f \in [0,\ 0.2 \cdot \pi]$ Hz and $\max(|\theta_{ref}|) \in [1,\ 5]$ deg, respectively. Additionally, exploring starts is implemented by resetting the system's state variables to a non-zero value, such that q_0, α_0 and θ_0 are sampled from the interval $[-0.1, 0.1]$[deg/s, deg, deg]. Both of these techniques are meant to reduce overfitting, which may occur when the agent is faced with the same task through the course of training.

Based on a series of tests in which an agent was trained with various combinations of the sensor and actuator dynamics described in subsubsection III.A.1, it was found that the highest performance is achieved when training with the elevator's transfer function activated, while noise and bias, delay, and servo rate limit are turned off. This configuration, named Test Case 1 (TC-1) is used to train all agents across both types of DNNs (LSTM and FF).

The hyperparameters in Table 4 are also identical across both types, and were manually tuned, starting with values found in [8] for a similar application through a tree-structured Parzen optimisation routine.

Table 4 Hyperparameters for the SAC implementation described in algorithm 1

Parameter	Value	Description
$\lambda_{0,\phi}, \lambda_{0,\theta}$	$1 \cdot 10^{-2}$	Initial actor-critic learning rate
$\lambda_{T,\phi}, \lambda_{T,\theta}$	$1 \cdot 10^{-5}$	Terminal actor-critic learning rate (exponential decay)
λ_α	$3 \cdot 10^{-4}$	Temperature coefficient learning rate
α_0	1	Initial temperature coefficient
$\lvert\phi\rvert, \lvert\theta_{1,2}\rvert, \lvert\bar{\theta}_{1,2}\rvert$	64	Number of hidden units in each layer
τ	$5 \cdot 10^{-3}$	Polyak smoothing factor
$\mathcal{D}$	$3.6 \cdot 10^4$	Experience buffer size
$\mathcal{B}$	64	Minibatch size
γ	0.99	Discount factor
$\bar{\mathcal{H}}$	$-\lvert\mathcal{A}\rvert$	Target entropy

2. Evaluation criteria

Tracking performance is evaluated with the normalised mean absolute error (nMAE) shown in Equation 14 in which N is the number of time steps in an episode. Furthermore, it is measured using the observable state variable $\hat{\theta}$ (rather than the true value of the state variable θ), because this is the signal the controller sees and tries to match to the reference. The nMAE is a useful measure because it gives a task-independent measure of how well the system tracks the reference signal. The control activity (CA) in Equation 15 quantifies how smooth the controller's commanded actions are. So appropriately, it is measured with δ_e^{com} instead of with the elevator's true position δ_e. The success thresholds, under which a controller's performance is deemed adequate, are nMAE $< 10\%$ and CA < 0.1 rad/s. All controllers are evaluated for two types of reference signals: sinusoidal and a step, in two environments: linear and nonlinear-dynamics (CitAST), and sixteen test cases of various combinations of sensor and actuator dynamics.

$$\text{nMAE} = \frac{\sum_{t=0}^{N} |\theta_{ref,t} - \hat{\theta}_t|}{\sum_{t=0}^{N} |\theta_{ref,t}|} \tag{14}$$

$$\text{CA} = \frac{1}{N} \sum_{t=1}^{N} \frac{|\delta_{e,t}^{\text{com}} - \delta_{e,t-1}^{\text{com}}|}{\Delta t} \tag{15}$$

IV. Results and Discussion

A. Learning Performance

During training, the weights of an agent's policy are logged after every batch of six parallel episodes, resulting in around 55 policies per agent. The evolution of an agent's performance over the course of training is then mapped by using each policy to simulate a single episode with deterministic action sampling. In contrast to the usual return-based learning curves, this performance mapping approach allows the best policy for each type of DNN to be identified based on more relevant indicators, and under the desired dynamical configuration. Figure 8 shows both the nMAE and CA of the agents for TC-1 (training configuration) and TC-2 (all sensor and actuator dynamics are activated), which are simulated in the linear environment. The bold curves indicate the agents whose performance surpasses both thresholds at some point during training. The first observation when looking at the nMAE for TC-1 is that there are significant differences between agents whose only differentiating factor is the weights with which their DNNs were initialised. Furthermore, considering the thresholds, the success rate of the SAC-LSTM agents is 40%, which is in line with the results found in [8, 9] for a similar application and thresholds.

TC-2 shows some loss in performance compared to TC-1 in the form of an upwards offset in the curves, but two agents still remain under the thresholds. Interestingly, none of the agents shows a significant, order-of-magnitude degradation in nMAE or CA, suggesting that the LSTM policies are generally indifferent to the addition of noise and bias, delay, and servo rate limit, despite never experiencing these dynamics during training.

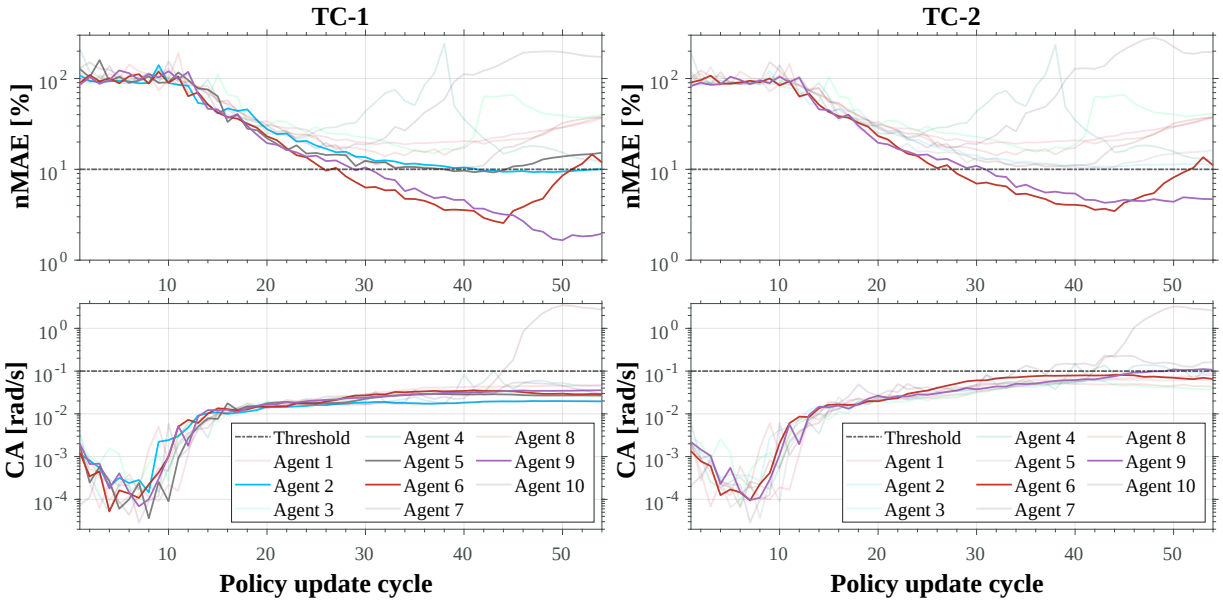


Fig. 8 Policy sweep for the SAC-LSTM agents. Done in the linear-dynamics environment with a sinusoidal reference signal. TC-1 (training configuration) and TC-2 are defined in Table 5.

According to Figure 9, the SAC-FF agents have a 50% success rate and are more consistent in terms of nMAE in TC-1 compared with the SAC-LSTM agents. However, the policies of eight of the FF agents converge to a behaviour with high-frequency actuations, evident by the high CA. The tendency of the SAC-FF agents to resort to high CA behaviour is further emphasised when looking at TC-2. There, the nMAE is hardly degraded with respect to TC-1, whereas the CA across all agents increases substantially, causing all of them to fail to stay beneath the thresholds. The root cause for the difference in CA between TC-1 and TC-2 is attributed to the sensitivity of FF policies to noisy inputs, as identified in subsection IV.C.

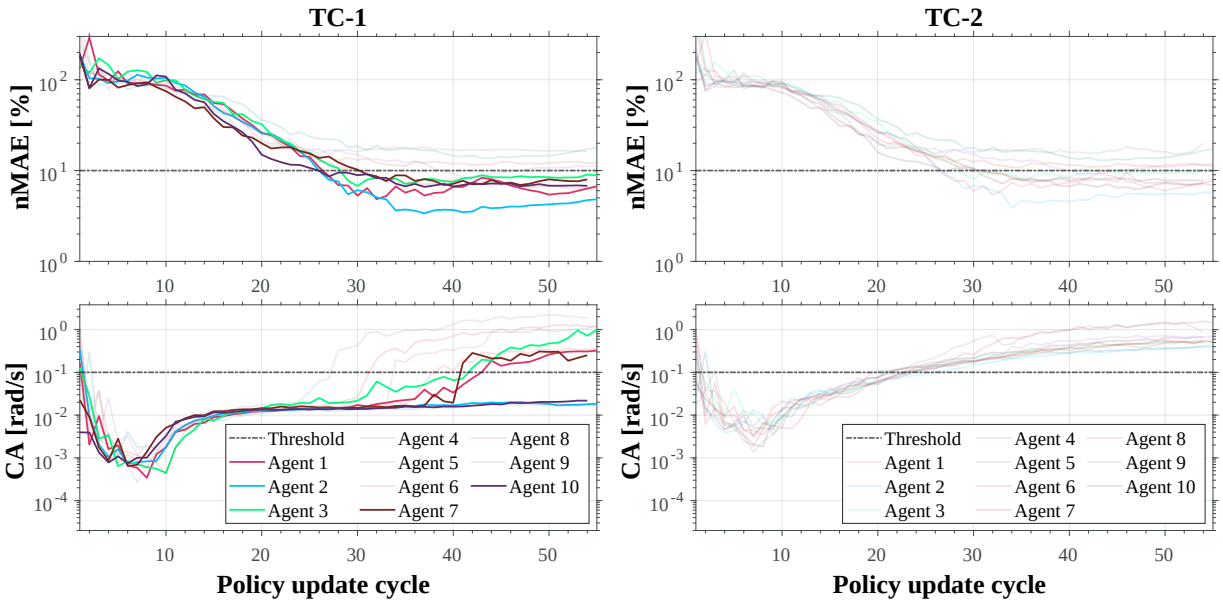


Fig. 9 Policy sweep for the SAC-FF agents. Done in the linear-dynamics environment with a sinusoidal reference signal. TC-1 (training configuration) and TC-2 are defined in Table 5.

Irrespective of how RL-based FCS end up performing, their low training consistency serves as an example of the many challenges that limit their adoption, as outlined by EASA's AI roadmap [23]. While the focus of this paper is on the technical challenges of sim-to-real transfer, it is essential to recognise that appropriate validation and verification processes are necessary for the implementation of AI in safety-critical systems. For example, the black-box nature of DNNs makes it challenging to trace how decisions are made and predict their behaviour under unfamiliar conditions [24].

B. Simulation Results For All Test Cases

To further analyse the performance and sensitivities of the three controllers, each one is used to simulate an episode in two different environments, tracking two types of reference signals, while facing sixteen different combinations of actuator and sensor dynamics. The policies for this evaluation are chosen with the aim of achieving balanced performance in TC-1 with both a sinusoidal and a step reference signal while striving to adhere to the thresholds. The two chosen policies are the 32nd LSTM policy of agent 10, visible in Figure 8, and the 37th FF policy of agent 6 from Figure 9. The results from all simulations can be found in Table 5. The overall picture painted by the data in Table 5 is that the LSTM policy is more robust against various actuator and sensor dynamics, as illustrated by the fact that its performance indicators remain below the thresholds in all test cases. Most importantly, they remain below the thresholds for TC-2 in the CitAST environment, which serves as an emulation of the real system, highlighting the controller's potential in transitioning from simulation to reality.

Table 5 Simulation results overview. In red: values above the threshold of nMAE< 10% and CA< 0.1rad/s

						SAC-LSTM				SAC-FF				LC			
						Sine		Step		Sine		Step		Sine		Step	
Env	Test Case	Noise and Bias	Delay	Actuator RL	Elevator TF	nMAE	CA [rad/s]	nMAE	CA [rad/s]	nMAE	CA [rad/s]	nMAE	CA [rad/s]	nMAE	CA [rad/s]	nMAE	CA [rad/s]
Linear	TC-1	0	0	0	1	5.9%	$2.9 \cdot 10^{-2}$	7.1%	$3.8 \cdot 10^{-2}$	6.6%	$1.6 \cdot 10^{-2}$	8.1%	$2.6 \cdot 10^{-2}$	4.6%	$2.6 \cdot 10^{-2}$	5.6%	$4.5 \cdot 10^{-2}$
	TC-2	1	1	1	1	6.7%	$6.4 \cdot 10^{-2}$	9.5%	$8.7 \cdot 10^{-2}$	7.7%	$3.8 \cdot 10^{-1}$	10.6%	$4.1 \cdot 10^{-1}$	74.7%	$2.8 \cdot 10^{-1}$	55.5%	$2.9 \cdot 10^{-1}$
	TC-3	1	1	1	0	6.6%	$6.8 \cdot 10^{-2}$	9.3%	$8.7 \cdot 10^{-2}$	7.6%	$6.1 \cdot 10^{-1}$	10.4%	$8.2 \cdot 10^{-1}$	65.1%	$3.1 \cdot 10^{-1}$	51.6%	$3.3 \cdot 10^{-1}$
	TC-4	1	1	0	1	6.7%	$6.4 \cdot 10^{-2}$	8.8%	$8.5 \cdot 10^{-2}$	7.7%	$3.7 \cdot 10^{-1}$	10.0%	$4.3 \cdot 10^{-1}$	4.7%	$5.7 \cdot 10^{-1}$	8.0%	$5.5 \cdot 10^{-1}$
	TC-5	1	1	0	0	6.5%	$6.5 \cdot 10^{-2}$	8.0%	$8.8 \cdot 10^{-2}$	7.1%	3.7	10.7%	6.1	4.7%	$5.7 \cdot 10^{-1}$	6.2%	$6.0 \cdot 10^{-1}$
	TC-6	1	0	1	1	5.9%	$6.8 \cdot 10^{-2}$	7.7%	$8.6 \cdot 10^{-2}$	7.3%	$4.4 \cdot 10^{-1}$	8.5%	$4.6 \cdot 10^{-1}$	4.7%	$5.5 \cdot 10^{-1}$	33.5%	$3.6 \cdot 10^{-1}$
	TC-7	1	0	1	0	5.7%	$6.7 \cdot 10^{-2}$	7.7%	$8.1 \cdot 10^{-2}$	7.3%	$8.1 \cdot 10^{-1}$	8.3%	$9.4 \cdot 10^{-1}$	4.7%	$5.9 \cdot 10^{-1}$	30.4%	$3.9 \cdot 10^{-1}$
	TC-8	1	0	0	1	5.8%	$6.7 \cdot 10^{-2}$	7.0%	$8.6 \cdot 10^{-2}$	7.3%	$4.5 \cdot 10^{-1}$	7.9%	$4.6 \cdot 10^{-1}$	4.7%	$5.3 \cdot 10^{-1}$	5.9%	$5.7 \cdot 10^{-1}$
	TC-9	1	0	0	0	5.7%	$6.3 \cdot 10^{-2}$	6.3%	$8.0 \cdot 10^{-2}$	6.6%	9.2	8.7%	$1.0 \cdot 10^{1}$	4.6%	$6.6 \cdot 10^{-1}$	5.2%	$6.4 \cdot 10^{-1}$
	TC-10	0	1	1	1	6.7%	$3.0 \cdot 10^{-2}$	9.4%	$4.4 \cdot 10^{-2}$	6.9%	$1.9 \cdot 10^{-2}$	10.9%	$3.1 \cdot 10^{-2}$	75.8%	$2.6 \cdot 10^{-1}$	52.4%	$2.0 \cdot 10^{-1}$
	TC-11	0	1	1	0	6.6%	$2.9 \cdot 10^{-2}$	9.4%	$3.6 \cdot 10^{-2}$	6.9%	$4.8 \cdot 10^{-1}$	10.6%	$8.4 \cdot 10^{-1}$	69.6%	$2.7 \cdot 10^{-1}$	52.0%	$2.0 \cdot 10^{-1}$
	TC-12	0	1	0	1	6.7%	$3.0 \cdot 10^{-2}$	8.8%	$4.3 \cdot 10^{-2}$	6.9%	$1.9 \cdot 10^{-2}$	10.3%	$3.1 \cdot 10^{-2}$	4.7%	$6.8 \cdot 10^{-2}$	7.5%	$1.5 \cdot 10^{-1}$
	TC-13	0	1	0	0	6.6%	$2.9 \cdot 10^{-2}$	8.0%	$3.8 \cdot 10^{-2}$	6.6%	3.6	10.8%	6.2	4.7%	$3.8 \cdot 10^{-2}$	5.9%	$5.0 \cdot 10^{-2}$
	TC-14	0	0	1	1	5.9%	$2.8 \cdot 10^{-2}$	7.7%	$3.9 \cdot 10^{-2}$	6.6%	$1.6 \cdot 10^{-2}$	8.7%	$2.6 \cdot 10^{-2}$	4.7%	$3.3 \cdot 10^{-2}$	33.8%	$2.4 \cdot 10^{-1}$
	TC-15	0	0	1	0	5.8%	$2.8 \cdot 10^{-2}$	7.8%	$3.3 \cdot 10^{-2}$	6.7%	$6.9 \cdot 10^{-1}$	8.5%	$9.5 \cdot 10^{-1}$	4.6%	$2.7 \cdot 10^{-2}$	32.6%	$2.5 \cdot 10^{-1}$
	TC-16	0	0	0	0	5.7%	$2.4 \cdot 10^{-2}$	6.4%	$3.1 \cdot 10^{-2}$	6.2%	9.2	8.9%	$1.0 \cdot 10^{1}$	4.6%	$2.1 \cdot 10^{-2}$	5.0%	$2.5 \cdot 10^{-2}$
Nonlinear (CitAST)	TC-1	0	0	0	1	6.0%	$2.8 \cdot 10^{-2}$	6.4%	$4.0 \cdot 10^{-2}$	7.9%	$1.6 \cdot 10^{-2}$	8.2%	$2.7 \cdot 10^{-2}$	4.6%	$3.1 \cdot 10^{-2}$	5.7%	$5.6 \cdot 10^{-2}$
	TC-2	1	1	1	1	6.9%	$6.7 \cdot 10^{-2}$	8.9%	$9.8 \cdot 10^{-2}$	8.8%	$3.8 \cdot 10^{-1}$	11.5%	$3.9 \cdot 10^{-1}$	71.3%	$2.7 \cdot 10^{-1}$	55.6%	$2.9 \cdot 10^{-1}$
	TC-3	1	1	1	0	6.6%	$7.1 \cdot 10^{-2}$	9.0%	$9.1 \cdot 10^{-2}$	8.8%	$6.6 \cdot 10^{-1}$	11.4%	$7.8 \cdot 10^{-1}$	68.9%	$2.9 \cdot 10^{-1}$	63.6%	$2.8 \cdot 10^{-1}$
	TC-4	1	1	0	1	6.9%	$6.6 \cdot 10^{-2}$	8.2%	$9.4 \cdot 10^{-2}$	8.8%	$3.8 \cdot 10^{-1}$	10.9%	$3.9 \cdot 10^{-1}$	5.1%	$6.9 \cdot 10^{-1}$	11.4%	$7.1 \cdot 10^{-1}$
	TC-5	1	1	0	0	6.7%	$6.8 \cdot 10^{-2}$	7.4%	$8.8 \cdot 10^{-2}$	8.3%	4.4	9.4%	4.4	4.6%	$5.6 \cdot 10^{-1}$	6.3%	$5.8 \cdot 10^{-1}$
	TC-6	1	0	1	1	6.1%	$6.4 \cdot 10^{-2}$	7.1%	$8.7 \cdot 10^{-2}$	8.4%	$4.2 \cdot 10^{-1}$	9.4%	$4.5 \cdot 10^{-1}$	40.6%	$3.6 \cdot 10^{-1}$	41.3%	$3.4 \cdot 10^{-1}$
	TC-7	1	0	1	0	5.8%	$6.5 \cdot 10^{-2}$	7.2%	$8.3 \cdot 10^{-2}$	8.4%	$7.2 \cdot 10^{-1}$	9.3%	$7.9 \cdot 10^{-1}$	4.6%	$5.1 \cdot 10^{-1}$	34.2%	$3.6 \cdot 10^{-1}$
	TC-8	1	0	0	1	6.0%	$6.4 \cdot 10^{-2}$	6.5%	$8.5 \cdot 10^{-2}$	8.5%	$4.1 \cdot 10^{-1}$	8.8%	$4.2 \cdot 10^{-1}$	4.6%	$5.4 \cdot 10^{-1}$	6.0%	$5.7 \cdot 10^{-1}$
	TC-9	1	0	0	0	5.9%	$6.2 \cdot 10^{-2}$	5.8%	$7.8 \cdot 10^{-2}$	7.8%	7.8	7.7%	9.1	4.6%	$5.6 \cdot 10^{-1}$	5.3%	$5.4 \cdot 10^{-1}$
	TC-10	0	1	1	1	6.8%	$3.0 \cdot 10^{-2}$	8.7%	$5.5 \cdot 10^{-2}$	8.2%	$2.0 \cdot 10^{-2}$	11.0%	$3.3 \cdot 10^{-2}$	72.4%	$2.5 \cdot 10^{-1}$	54.2%	$1.8 \cdot 10^{-1}$
	TC-11	0	1	1	0	6.6%	$2.7 \cdot 10^{-2}$	8.8%	$4.1 \cdot 10^{-2}$	8.1%	$5.8 \cdot 10^{-1}$	10.7%	$7.9 \cdot 10^{-1}$	71.4%	$2.5 \cdot 10^{-1}$	53.7%	$1.8 \cdot 10^{-1}$
	TC-12	0	1	0	1	6.8%	$3.0 \cdot 10^{-2}$	8.1%	$4.7 \cdot 10^{-2}$	8.2%	$2.0 \cdot 10^{-2}$	10.4%	$3.4 \cdot 10^{-2}$	5.1%	$4.7 \cdot 10^{-1}$	10.4%	$4.9 \cdot 10^{-1}$
	TC-13	0	1	0	0	6.7%	$2.8 \cdot 10^{-2}$	7.3%	$3.9 \cdot 10^{-2}$	8.0%	4.4	9.5%	4.5	4.6%	$3.3 \cdot 10^{-2}$	6.0%	$4.3 \cdot 10^{-2}$
	TC-14	0	0	1	1	6.0%	$2.8 \cdot 10^{-2}$	7.0%	$4.3 \cdot 10^{-2}$	7.9%	$1.6 \cdot 10^{-2}$	8.8%	$2.7 \cdot 10^{-2}$	38.2%	$3.4 \cdot 10^{-1}$	35.9%	$2.3 \cdot 10^{-1}$
	TC-15	0	0	1	0	5.8%	$2.6 \cdot 10^{-2}$	7.2%	$3.6 \cdot 10^{-2}$	7.9%	$5.2 \cdot 10^{-1}$	8.5%	$7.9 \cdot 10^{-1}$	4.6%	$3.2 \cdot 10^{-2}$	34.4%	$2.4 \cdot 10^{-1}$
	TC-16	0	0	0	0	5.9%	$2.4 \cdot 10^{-2}$	5.7%	$3.3 \cdot 10^{-2}$	7.4%	7.8	7.6%	9.1	4.6%	$2.4 \cdot 10^{-2}$	5.1%	$2.7 \cdot 10^{-2}$

Even though actions are sampled deterministically from the policies, there is uncertainty in the results for test cases in which noise is added to the state (TC-2 up to and including TC-9). Noise introduces uncertainty in the measurement of performance indicators directly and through its influence on the time response of controllers that amplify noisy

inputs. The uncertainty introduced by the addition of noise is quantified by the coefficient of variation (CV). This is done by simulating 100 episodes with each controller under TC-8, which is identical to the training configuration, with the exception that noise and bias are included. It is apparent in Table 6 that the CA of the FF policy is most affected by noise, followed by the LC and the LSTM policy. Yet, for consistency, the results are presented with a uniform number of decimal places across all test cases.

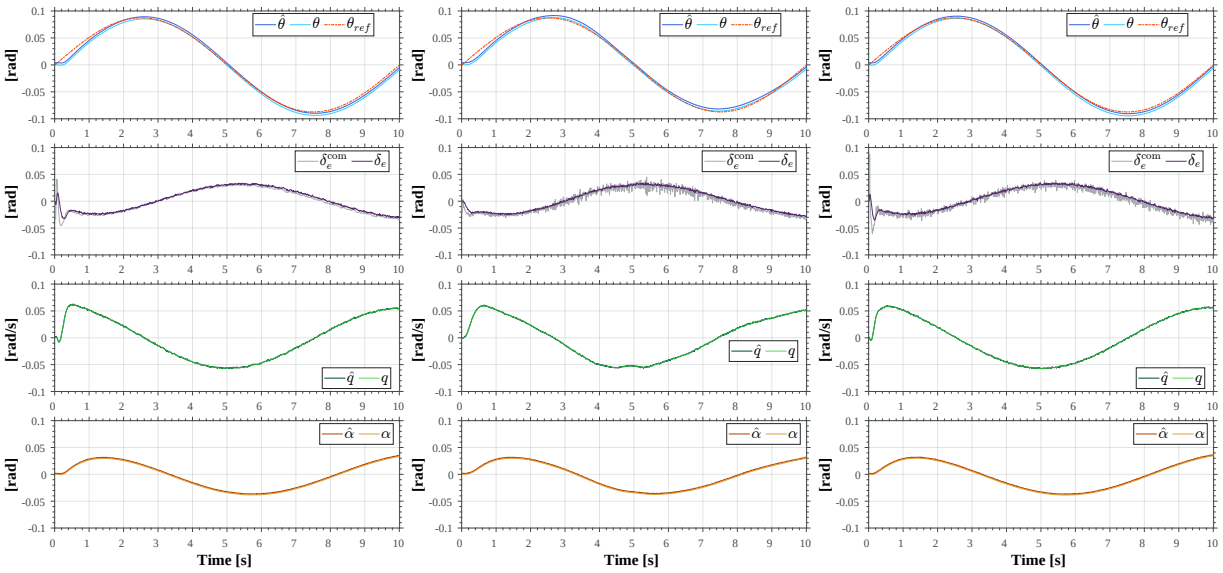
Table 6 nMAE and CA Coefficient of Variation (CV) for each controller. Based on 100 simulations of TC-8 in the linear environment with a sinusoidal reference signal.

CV [-]	SAC-LSTM	SAC-FF	LC
nMAE	$1.4 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$	$6.6 \cdot 10^{-4}$
CA	$1.8 \cdot 10^{-2}$	$3.0 \cdot 10^{-2}$	$2.9 \cdot 10^{-2}$

C. Sensitivity to Various Sensor and Actuator Dynamics

1. Noise and Bias

Figure 10 shows the time response of the LC and the selected LSTM and FF policies for TC-8, with which their sensitivity to noise and bias can be examined. The elevator deflection plot in Figure 10b shows that the SAC-FF controller outputs noisy action commands (δ_e^{com}). Despite the elevator (δ_e), attenuating the high-amplitude commands to some degree, it is still an undesirable response because it causes vibrations and increases the wear of mechanical components. A similar behaviour by FF policies to noisy states is also observed in [8, 9]. In contrast, the SAC-LSTM controller does not seem to amplify noise beyond its baseline level, as can be seen in Figure 10a. These differences can be attributed to the FF policy's lack of memory, causing it to treat any fluctuation in the input as a true change in the state, which requires a corrective action. Thus, it is rather sensitive to noise. The LSTM net, on the other hand, can "see through" the noise because it retains information from many previous time steps. As for the LC, the noisy response in Figure 10c may be attributed to the derivative action in the inner loop, which amplifies noise in $\hat{q}$, and could benefit from additional filtering. None of the controllers can compensate for bias because they have no access to information that can be used to estimate it. So, it remains as a constant offset between the true and observable state.



(a) SAC-LSTM, TC-8, Linear Env, nMAE= 5.8%, CA= $6.7 \cdot 10^{-2}$ rad/s.

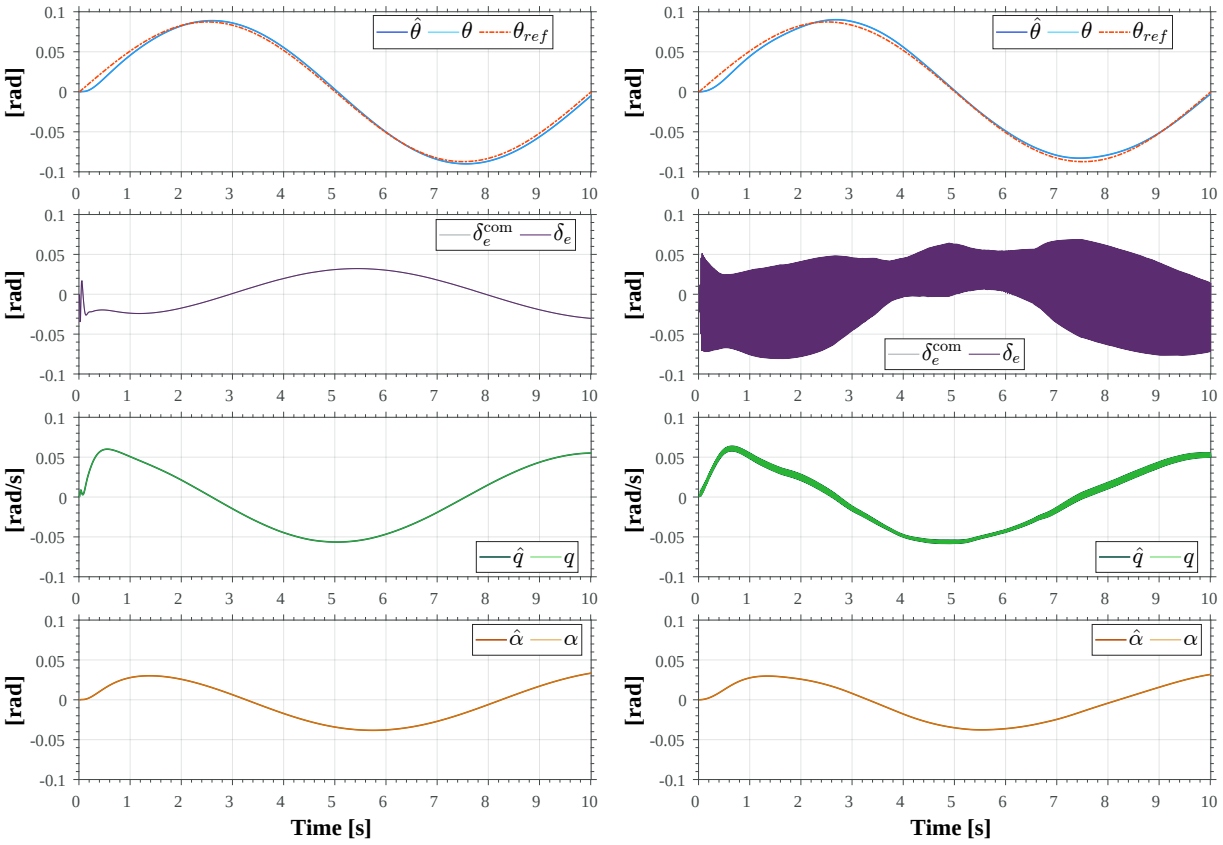
(b) SAC-FF, TC-8, Linear Env, nMAE= 7.3%, CA= $4.5 \cdot 10^{-1}$ rad/s.

(c) LC, TC-8, Linear Env, nMAE= 4.7%, CA= $5.3 \cdot 10^{-1}$ rad/s.

Fig. 10 Time responses of the three controllers. TC-8: Measurement noise and bias, and elevator transfer function.

2. Actuator Dynamics

The sensitivity of the controllers to the dynamics of the elevator is first examined by isolating the influence of its transfer function on their response. TC-16 represents the limit for $\tau \rightarrow 0$, such that $u(s) = y(s)$. Although unrealistic in and of itself, it demonstrates an edge case of a continuous trend in the response of the controllers, and is thus useful to analyse. Figure 11a shows that the LSTM policy has no issues with controlling the elevator in spite of its altered dynamics, as indicated by the smooth elevator deflections. The FF policy in comparison, whose time response is visible in Figure 11b, behaves like a bang-bang controller with extremely oscillatory elevator deflections. As evident by the acceptable nMAE, the effect of the high-rate deflections does not propagate to the pitch angle, but it does cause the pitch rate to oscillate. The difference can again be attributed to the memory of the LSTM network, enabling it to infer the elevator's rate $\dot{\delta}_e$ and thus predict its response. Meanwhile, the FF controller, which only has information about the elevator's position, fails to smoothly control it when it responds faster than it did during training.



(a) SAC-LSTM, TC-16, Linear Env, nMAE= 5.7%, **CA**= $2.4 \cdot 10^{-2}$ **rad/s.**

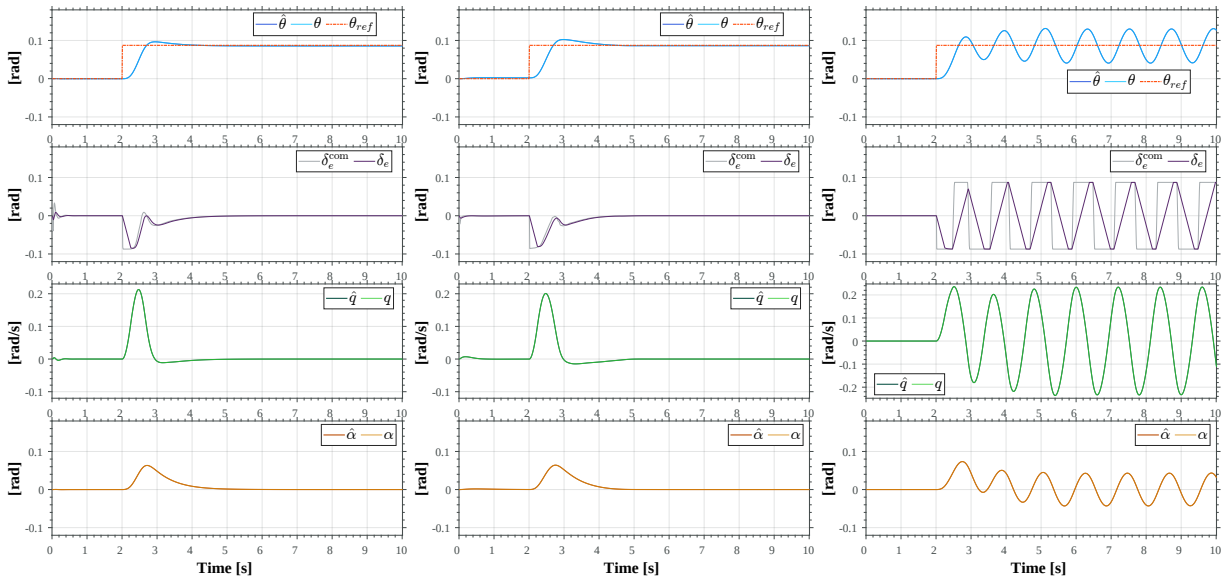
(b) SAC-FF, TC-16, Linear Env, nMAE= 6.2%, **CA**= $9.2 \cdot 10^{0}$ **rad/s.**

Fig. 11 TC-16: No sensor and actuator dynamics.

The effect of the inclusion of a servo rate limit, in addition to the transfer function, is additional lag in the elevator's response. At this point, the LC, which was shown to handle the dynamics of the transfer function in Figure 16c, can no longer bring the Citation to steady state once it's perturbed. It can, of course, be tuned to handle the altered actuator dynamics, but it goes to show the limitations of linear controllers. Meanwhile, both the LSTM and FF controllers have no issues handling it, as can be seen in Figure 12a and Figure 12b, respectively. One problematic aspect in the response of the LSTM policy is found around $t = 0$ in the elevator deflection curve in Figure 12a. It can be observed that the policy flickers the elevator despite all states, including the reference, being initialised to zero. This is caused by the hidden state of the LSTM layer, which requires some time to build up context. Deeper inspection of Table 5 reveals that in TC-15, in which only the dynamics of a rate-limited servo are applied, the FF controller suddenly has a very high CA. Upon investigation of the behaviour of the elevator under various test cases, it turns out that the elevator response

lag caused by the transfer function alone is very close to being constant, thus independent of the magnitude of the commanded deflection δ_e^{com}. However, the lag resulting from a rate-limited servo is directly related to the magnitude of $\Delta\delta_e^{\text{com}}$. And so, in TC-15, the elevator responds faster with a servo rate limit than with a transfer function when $|\Delta\delta_e^{\text{com}}| \lessapprox 2°$, causing the FF controller to rapidly oscillate, just like it did in Figure 11b.

The takeaway is that when it comes to actuator dynamics, the LSTM controller appears to be far more robust than both its counterparts. Robustness is favourable not just because it enables sim-to-real transfer, but also because it allows training in an environment that maximises exploration without compromising performance during deployment.



(a) SAC-LSTM, TC-14, Linear Env, nMAE= 7.7%**, CA**= $3.9 \cdot 10^{-2}$ **rad/s.**

(b) SAC-FF, TC-14, Linear Env, nMAE= 8.7%**, CA**= $2.6 \cdot 10^{-2}$ **rad/s.**

(c) LC, TC-14, Linear Env, nMAE= 33.8%**, CA**= $2.4 \cdot 10^{-1}$ **rad/s.**

Fig. 12 Time responses of the SAC-LSTM and SAC-FF policies. TC-14: Elevator transfer function and actuator rate limit.

3. Observation Delay

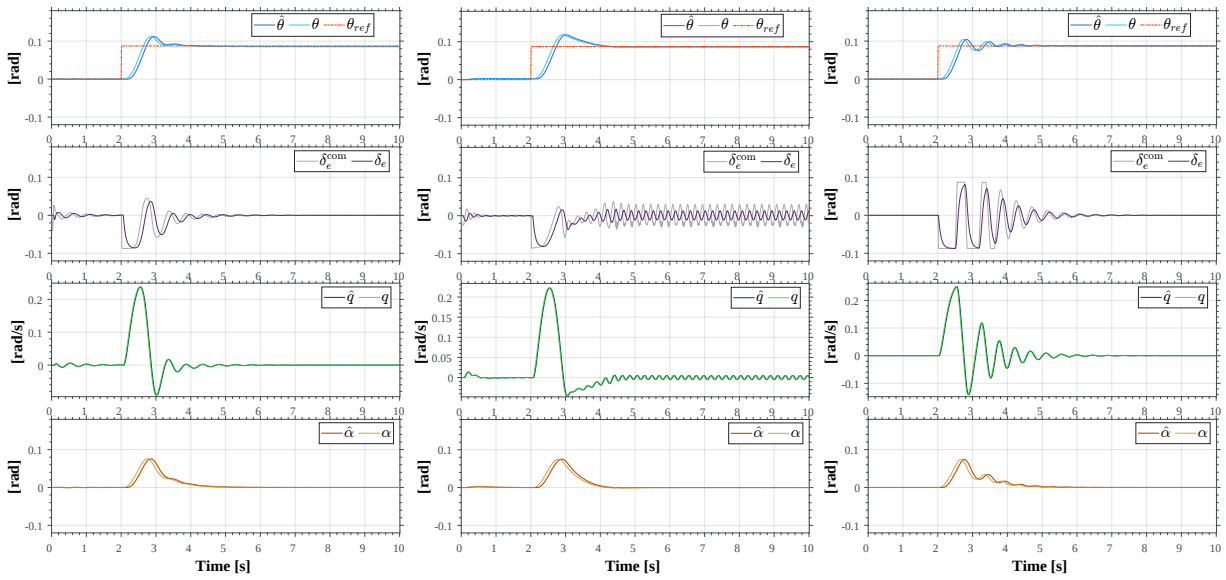
The results in Table 5 for TC-12, in which delays are added to the training configuration (TC-1), show that the sensitivity of both types of policies to delays is minimal. Specifically, the asynchronous delays degrade the nMAE of both the LSTM and FF policies by less than 1% for the sinusoidal reference and around 2% for the step. The differences in nMAE between the two types of reference signals can be explained by the differences in the magnitude of the error during the delayed period $d_\theta + d_{\delta_e}$, which is much larger for the step than for the sine wave. Because the effect of the standard delays on the time response for both DNN controllers is minimal, their figures are excluded.

The influence of delays on the controllability of the system is related to a state's temporal difference. In other words, when the rate of change of a state variable is high, the more sensitive the controlled system is to a delay in its observation. Conveniently, even though the delay in the controlled variable θ is larger than the delay in q, it is slower to react, so its observable value is still close to the value of the true state. Furthermore, thanks to a latency of only 15 ms (Rounded up to two time-steps) in q, the difference between the measured and real value is also small enough for the RL controllers to deal with despite its high rate of change. A similar combination of circumstances also applies to δ_e and to α such that their temporal differences are small enough for the LSTM and FF controllers to handle.

To test this hypothesis and find the limits of the two DNN controllers, the delay in the action is increased from one time-step to five, as shown in the time responses in Figure 13a and Figure 13b. The increased action delay causes oscillatory actions, which mostly affect q and hardly propagate through to θ. The LSTM controller is able to dampen this out within about four seconds after the step input. In contrast, the FF controller reaches an oscillatory steady state. For the sake of completeness, it should be noted that when performing the same simulation in the nonlinear-dynamics environment, the response of the FF controller does eventually dampen out after approximately twelve seconds.

The response of the linear controller to the standard delays can be seen in Figure 13c. Evidently, it is rather sensitive to delays as shown by its under-damped response. But, it is only fair to point out the fact that it outperforms its DNN counterparts for TC-1, which makes it more sensitive due to a trade-off between tracking performance and disturbance rejection. Additional tests confirm that decreasing the proportional gain in the inner loop reduces its sensitivity to delays, which in turn shortens the settling time and the amplitude of the overshoot. With that, the step response of the LC becomes similar to that of the LSTM controller in Figure 13a.

Based on the findings above, it may be reasonable to conclude that memoryless DNN-based controllers, such as the FF policy, are not inherently more robust to delays than a linear controller of equivalent performance, unless it has access to additional information that can improve its perception of the true state. Having access to previous time steps is an example of such additional information and is exactly what the LSTM layer takes advantage of.



(a) SAC-LSTM, TC-12, Linear Env, nMAE= 9.3%, CA= $7.1 \cdot 10^{-2}$ rad/s, 50ms action delay.

(b) SAC-FF, TC-12, Linear Env, nMAE= 10.8%, CA= $4.7 \cdot 10^{-1}$ rad/s, 50ms action delay.

(c) LC, TC-12, Linear Env, nMAE= 7.5%, CA= $1.5 \cdot 10^{-1}$ rad/s, 10ms action delay (original value).

Fig. 13 Time responses of the three controllers. TC-12: Measurement and action delay.

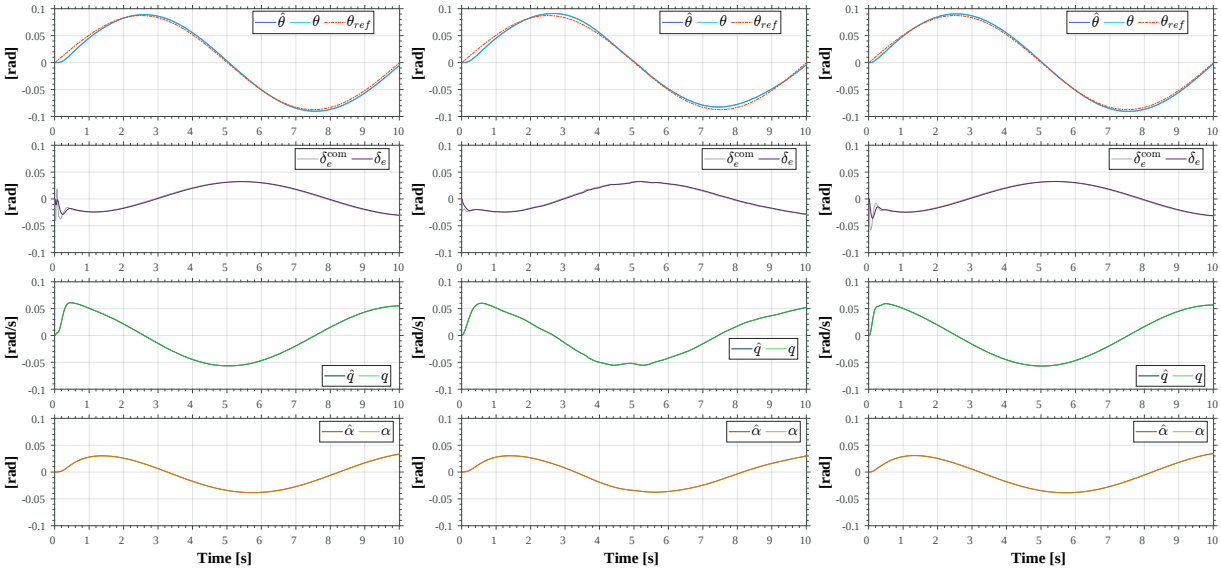
D. Emulated Sim-to-Real Transfer

Sim-to-real transfer is emulated with the nonlinear-dynamics environment with all sensor and actuator dynamics enabled, as specified by TC-2: noise and bias, delay, elevator transfer function, and actuator rate limit. Additionally, time responses are generated for each controller in the training configuration (TC-1) in the linear-dynamics environment. This allows direct comparison and evaluation of the ability of each controller to make the transition.

Of the three controllers, the performance indicators of the LSTM policy degrade the least when making the transition, for both types of reference signals. This is not surprising considering that it proved to be the most insensitive to the individual sensor and actuator dynamics. The FF controller is also able to maintain control once transitioning to the realistic environment, but suffers a greater penalty in its performance, especially in terms of CA, causing it to exceed the thresholds. Lastly, the LC fails to stabilise the system when making the transition.

Interestingly, the observed differences in the time responses below cannot be attributed to variations in the dynamics between the two environments. This is further supported by the data in Table 5, where no consistent trend is observed in nMAE and CA when comparing identical test cases across the two environments. This likely stems from the fact that the dynamics of the Citation remain approximately linear within its flight envelope [16]. Moreover, there are no pronounced adverse coupling effects when enabling all sensor and actuator dynamics. The only exception can be observed in Figure 17a, where the LSTM controller aggressively tries to correct for the bias. Signs of its aggressiveness can also be observed in the response of the pitch rate q in Figure 13c, where the effect of delay was investigated, though to a lesser extent. Other characteristics can be tied back to individual sensitivities. For example, the amplified action

noise seen in Figure 15b and Figure 17b is attributed to the memorylessness of the FF controller when facing noisy observations. Lastly, the inability of the LC to stabilise the system in Figure 15c and Figure 17c was shown to be caused by the combination of a servo rate limit with the elevator transfer function.

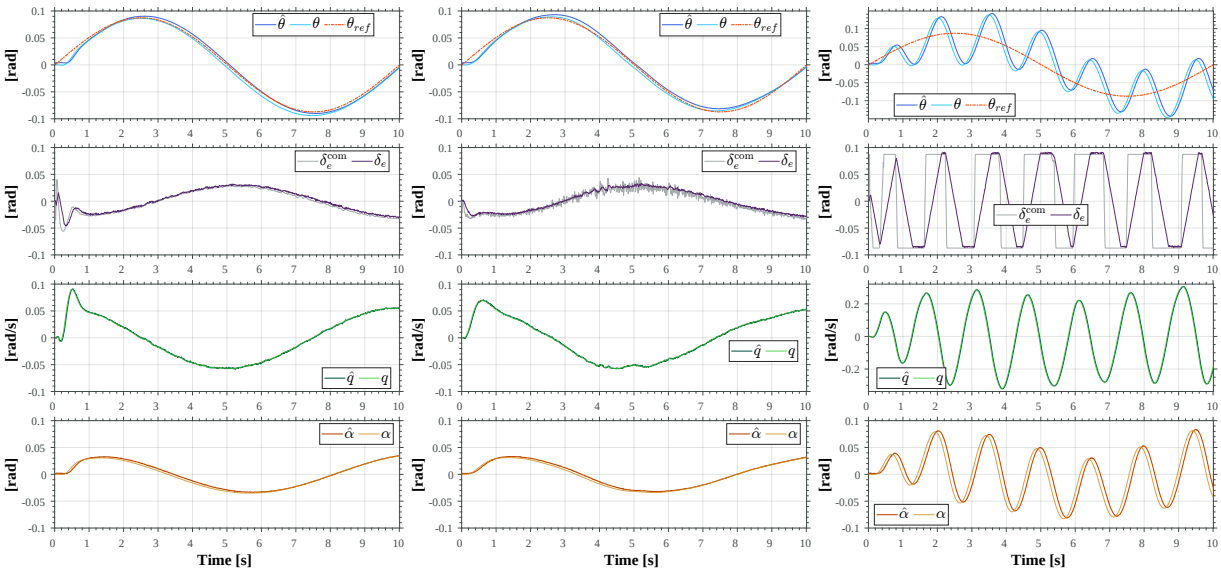


(a) SAC-LSTM, TC-1, Linear Env, nMAE= 5.9%, CA= $2.9 \cdot 10^{-2}$ rad/s.

(b) SAC-FF, TC-1, Linear Env, nMAE= 6.6%, CA= $1.6 \cdot 10^{-2}$ rad/s.

(c) LC, TC-1, Linear Env, nMAE= 4.6%, CA= $2.6 \cdot 10^{-2}$ rad/s.

Fig. 14 Time responses of the three controllers. TC-1 (training configuration): Elevator transfer function.

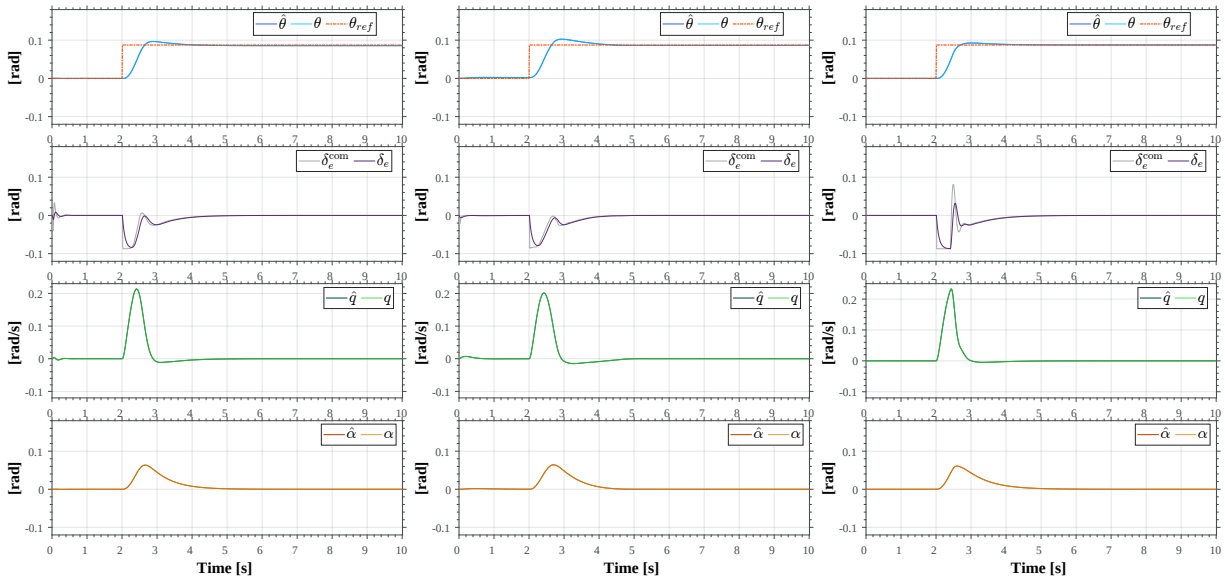


(a) SAC-LSTM, TC-2, Nonlinear Env, nMAE= 6.9%, CA= $6.7 \cdot 10^{-2}$ rad/s.

(b) SAC-FF, TC-2, Nonlinear Env, nMAE= 8.8%, CA= $3.8 \cdot 10^{-1}$ rad/s.

(c) LC, TC-2, Nonlinear Env, nMAE= 71.3%, CA= $2.7 \cdot 10^{-1}$ rad/s.

Fig. 15 Time responses of the three controllers. TC-2: Measurement noise and bias, delay, elevator transfer function, and actuator rate limit.

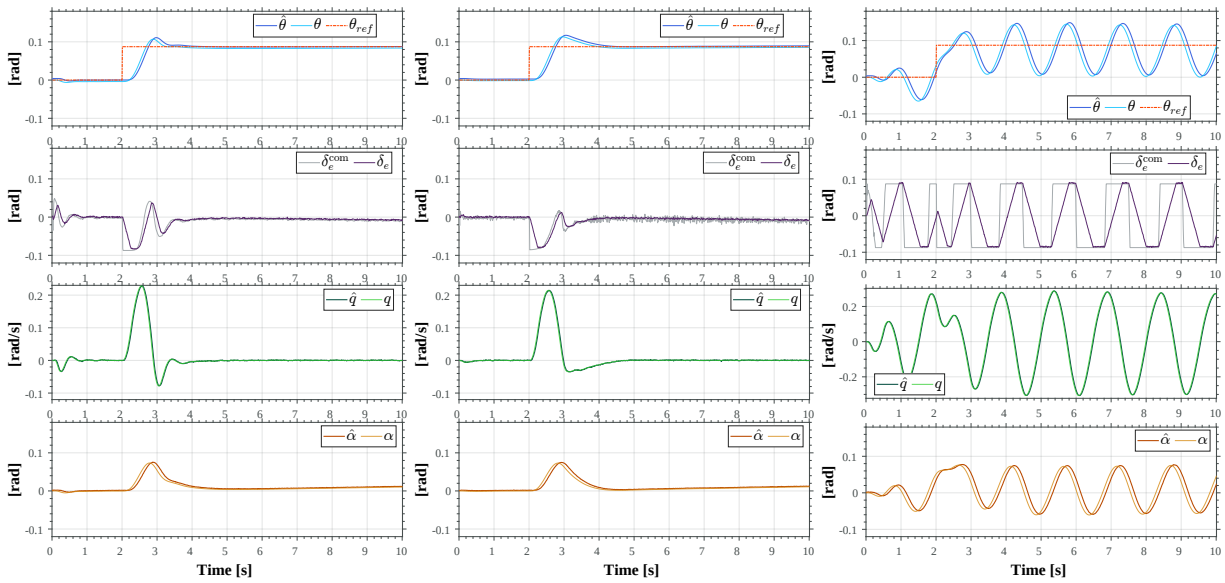


(a) SAC-LSTM, TC-1, Linear Env, nMAE= 7.1%, CA= $3.8 \cdot 10^{-2}$ rad/s.

(b) SAC-FF, TC-1, Linear Env, nMAE= 8.1%, CA= $2.6 \cdot 10^{-2}$ rad/s.

(c) LC, TC-1, Linear Env, nMAE= 5.6%, CA= $4.5 \cdot 10^{-2}$ rad/s.

Fig. 16 Time responses of the three controllers. TC-1 (training configuration): Elevator transfer function.



(a) SAC-LSTM, TC-2, Nonlinear Env, nMAE= 8.9%, CA= $9.8 \cdot 10^{-2}$ rad/s.

(b) SAC-FF, TC-2, Nonlinear Env, nMAE= 11.5%, CA= $3.9 \cdot 10^{-1}$ rad/s.

(c) LC, TC-2, Nonlinear Env, nMAE= 55.6%, CA= $2.9 \cdot 10^{-1}$ rad/s.

Fig. 17 Time responses of the three controllers. TC-2: Measurement noise and bias, delay, elevator transfer function, and actuator rate limit.

Based on the above, it is apparent that in the case of the Cessna Citation II, a low-fidelity, linear model is entirely sufficient for training an offline SAC controller. Furthermore, the results suggest that incorporating LSTM layers in the policy is a promising method for reducing the controller's sensitivity to the effects of realistic sensor and actuator dynamics, enabling it to bridge the sim-to-real gap. Yet, there are limitations which must be taken into account. First, the controllers were only tested at nominal operating conditions with $V_{TAS} = 90\,\mathrm{m/s}$, $m = 4500\,\mathrm{kg}$, and density altitude

$h = 2000\,\text{m}$. While flight tests can be conducted at (approximately) specified conditions, it is essential to test the controller at other operating points to verify that its behaviour is acceptable. Specifically, testing at near-stall flight conditions is recommended because the dynamics are considerably less linear in that regime. Second, the robustness of the controllers to atmospheric disturbances also remains untested. Being that the weather is not a controlled variable, it is another important phenomenon to verify that a controller is robust against. The last point to consider is the computational resources on board the PH-LAB, which may limit the type of controller that can be tested. Evaluating LSTM layers requires additional matrix operations, which may be difficult to perform at the desired sampling rate.

V. Conclusion

This paper presents the design of an SAC-based pitch attitude FCS meant to enable the transition from simulation to reality. A control policy with an LSTM layer trained in a simplified linear-dynamics environment proves to be robust against prevalent sensor and actuator dynamics, despite not experiencing them during training. A sensitivity analysis reveals that in comparison with a strictly feedforward policy and a linear controller, the LSTM controller does not amplify the noise of the input state. Furthermore, it is shown to be highly robust against changes in the dynamics of the elevator, enabling training in a configuration that maximises exploration without causing substantial degradation in performance when evaluated under realistic constraints such as a servo rate limit. While it can handle deterministic and asynchronous delays in both the state and action, the system's response demonstrates reduced damping when the action delay is extended. Due to the approximately linear dynamics of the Citation, the transition from the training environment to the nonlinear-dynamics model does not cause meaningful differences in the characteristics of the time response. Visible differences are mostly attributed to individual sensor and actuator dynamics, with little coupling effects. Despite demonstrating robustness, its performance under various flight conditions and atmospheric disturbances is untested, and is a point for further work. Additionally, evaluating a DNN policy, and an LSTM one in particular, may be too computationally intensive for the installed hardware in the PH-LAB. So, as a future step, it is recommended to test the controller with hardware-in-the-loop. Lastly, the low training reliability of the SAC agents, and of the LSTM policies in particular, serves as an example of the challenges associated with certifying RL-based FCS, a topic requiring further attention.

References

[1] "IATA Annual Safety Report-2024 Executive Summary and Safety Overview," Tech. rep., IATA, 2024.

[2] "Annual safety review 2024," Tech. rep., European Union Aviation Safety Agency, 7 2024. https://doi.org/10.2822/49362.

[3] Slotine, J.-J. E., and Li, W., *Applied nonlinear control*, Prentice Hall, 1991.

[4] Stein, G., Bugajski, D., Hendrick, R., and Stein, G., "Dynamic inversion: An evolving methodology for flight control design," *International Journal of Control*, Vol. 59, No. 1, 1994, pp. 71–91. https://doi.org/10.1080/00207179408923070.

[5] Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., and Scaramuzza, D., "Champion-level drone racing using deep reinforcement learning," *Nature*, Vol. 620, No. 7976, 2023, pp. 982–987. https://doi.org/10.1038/s41586-023-06419-4.

[6] Smith, L., Kostrikov, I., and Levine, S., "A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning," 2022. URL http://arxiv.org/abs/2208.07860.

[7] R. Konatala, D. Milz, C. Weiser, G. Looye, and E. van Kampen, "Flight Testing Reinforcement Learning based Online Adaptive Flight Control Laws on CS-25 Class Aircraft," *AIAA SciTech Forum and Exposition, 2024*, American Institute of Aeronautics and Astronautics Inc, AIAA, 2024. https://doi.org/10.2514/6.2024-2402.

[8] Dally, K., and van Kampen, E., "Soft Actor-Critic Deep Reinforcement Learning for Fault-Tolerant Flight Control," *AIAA Science and Technology Forum and Exposition, AIAA SciTech Forum 2022*, American Institute of Aeronautics and Astronautics Inc, AIAA, 2022. https://doi.org/10.2514/6.2022-2078.

[9] Jansen, H., and Van Kampen, E.-J., "Longitudinal Handling Qualities Evaluation for Soft Actor-Critic Deep Reinforcement Learning Flight Control," *AIAA SCITECH 2025 Forum*, American Institute of Aeronautics and Astronautics, Reston, Virginia, 2025. https://doi.org/10.2514/6.2025-2794, URL https://arc.aiaa.org/doi/10.2514/6.2025-2794.

[10] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S., "Soft Actor-Critic Algorithms and Applications," 2018. URL http://arxiv.org/abs/1812.05905.

[11] R. Sutton, and A. Barto, *Reinforcement learning : an introduction*, The MIT Press, 2020.

[12] Nath, S., Baranwal, M., and Khadilkar, H., "Revisiting State Augmentation methods for Reinforcement Learning with Stochastic Delays; Revisiting State Augmentation methods for Reinforcement Learning with Stochastic Delays," 2021. https://doi.org/10.1145/3459637.3482386, URL https://doi.org/10.1145/3459637.3482386.

[13] Sepp Hochreiter, and Jurgen Schmidhuber, "Long Short-term Memory," *Neural Computation*, 1997.

[14] Fujimoto, S., van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," 2018. URL http://arxiv.org/abs/1802.09477.

[15] Zhang, S., Yao, H., and Whiteson, S., "Breaking the Deadly Triad with a Target Network," 2021. URL http://arxiv.org/abs/2101.08862.

[16] De Visser, M. C., and Pool, C. M., "Identification of a Cessna Citation II Model Based on Flight Test Data," Tech. rep., MSc thesis, Delft University of Technology, 2017.

[17] Grondman, F., Looye, G. H., Kuchar, R. O., Chu, Q. P., and van Kampen, E. J., "Design and flight testing of incremental nonlinear dynamic inversion based control laws for a passenger aircraft," *AIAA Guidance, Navigation, and Control Conference, 2018*, American Institute of Aeronautics and Astronautics Inc, AIAA, 2018. https://doi.org/10.2514/6.2018-0385.

[18] Raffin, A., Kober, J., and Stulp, F., "Smooth Exploration for Robotic Reinforcement Learning," *Conference on Robot Learning*, 2021.

[19] Wang, L., Zheng, Z., and Lin, Y., "Steady-State Error Compensation for Reinforcement Learning with Quadratic Rewards," 2024. URL http://arxiv.org/abs/2402.09075.

[20] He, K., Zhang, X., Ren, S., and Sun, J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015. URL https://arxiv.org/abs/1502.01852.

[21] Krishna Kumar, S., "On weight initialization in deep neural networks," 2017. URL https://arxiv.org/abs/1704.08863.

[22] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," 2014. URL http://arxiv.org/abs/1412.6980.

[23] "Artificial Intelligence Roadmap 2.0 Human-centric approach to AI in aviation Human-centric approach to AI in aviation," Tech. rep., EASA, 5 2023.

[24] Lemos, A., "Explainable Reinforcement Learning in Flight Control through Reward Decomposition," Tech. rep., MSc thesis, Delft University of Technology, 2020.

Part II

Preliminary Research

3

Literature Review

3.1. Reinforcement Learning Fundamentals

The following chapter introduces the building blocks of reinforcement learning (RL), a branch of machine learning with roots dating back to the 50's [14]. Much of this chapter was written in consultation with the invaluable book "Reinforcement Learning; an Introduction" by Richard S. Sutton and Andrew G. Barto [14].

3.1.1. Markov Decision Process

The Markov Decision Process (MDP) is a framework for modelling sequential decision-making processes and has applications in various fields. In reinforcement learning, it is used to characterise a problem in terms of the environment's features and the action space within it, under an explicit goal to be achieved [14]. The framework is an extension of the Markov Chains, a concept devised by Andrey Andreyevich Markov (1856-1922), a Russian mathematician known for his work on stochastic processes [15].

Agent-Environment Interaction

An MDP consists of an *agent* who interacts with an *environment* by taking an *action*, leading him to a new *state* for which he receives a *reward*. The process is illustrated in Figure 3.1. Formally, the agent-environment interaction takes place in sequential, discrete time steps where the agent evolves from state $s_t \in \mathcal{S}$ to s_{t+1} by taking action $a_t \in \mathcal{A}(s)$, for which a reward $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$ is granted. Over time, this process forms a trajectory as shown in Equation 3.1.

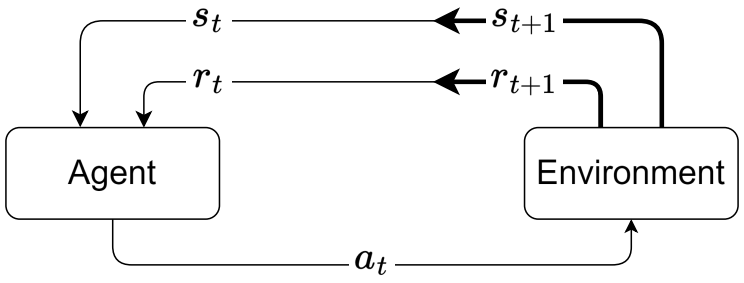


Figure 3.1: The agent-environment Interaction model according to a Markov Decision Process. Adapted from [14]

$$s_t,\ a_t,\ r_{t+1} \rightarrow s_{t+1},\ a_{t+1},\ r_{t+2}, \rightarrow s_{t+2},\ a_{t+2},\ r_{t+3}... \tag{3.1}$$

The dynamics of the decision process in a finite MDP is determined by a function describing the probability of transitioning to the next state $s' \in \mathcal{S}$ and receiving a reward $r \in \mathcal{R}$ by taking action $a \in \mathcal{A}(s)$ while in state $s \in \mathcal{S}$ Equation 3.2. Being a probability function, the sum of all transition probabilities

given the current state and action is unity, for all states and actions in the finite trajectory. As shown by Equation 3.2, the transition probability distribution is a function of only the current state and action, and is thus independent of earlier state-action pairs. This property, known as the *Markov property*, requires the state to carry all information representing the outcome of past agent-environment interactions that influence the future.

$$p(s', r \mid s, a) \doteq Pr\{s_{t+1} = s',\ r_{t+1} = r \mid s_t = s,\ a_t = a\} \tag{3.2}$$

Reward and Return

The reward defines the agent's task in the environment by telling it what to do, but not *how* to do it. In the MDP framework, the reward serves as a signal to tell the agent to what degree the previous state and the action it took there are encouraged according to the function in Equation 3.3. His goal is maximisation of the cumulative reward, rather than the immediate reward given after a single state transition. The balance between pursuing immediate rewards versus acting sub-optimally at present in order to maximise future rewards is embodied in the *discounted return* G_t shown by Equation 3.4 in which the factor $\gamma \in [0, 1]$, called the *discount rate* is a hyperparameter that can be tuned to adjust how 'farsighted' the agent is. Note the recursive nature of the discounted return, making current returns a function of future ones Equation 3.5.

$$r(s, a) \doteq \mathbb{E}[r_t \mid s_{t-1} = s,\ a_{t-1} = a] \tag{3.3}$$

$$G_t \doteq r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} \tag{3.4}$$

$$\begin{aligned} G_t &\doteq r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + ...) \\ &= r_{t+1} + \gamma \cdot (r_{t+2} + \gamma \cdot r_{t+3} + ...) \\ &= r_{t+1} + \gamma \cdot G_{t+1} \end{aligned} \tag{3.5}$$

3.1.2. Policies and Value Functions

The agent's experience and knowledge of which action to take at each state such that it transitions to a desirable next state, is represented by a *policy* π. The policy contains the probability of choosing action $a \in \mathcal{A}(s)$ according to $Pr(a \mid s)$, usually written as $\pi(a \mid s)$. To express how desirable different states are, *value functions* are used, in which the value of the state or state-action pair under the policy π is given by the *expected return* as shown in Equation 3.6 for the *state-value function* and Equation 3.7 for the *action-value function*.

$$v_\pi(s) \doteq \mathbb{E}[G_t \mid s_t = s] \tag{3.6}$$

$$q_\pi(s, a) \doteq \mathbb{E}[G_t \mid s_t = s,\ a_t = a] \tag{3.7}$$

Bellman Equations

Similarly to the return in Equation 3.5, value functions can also be written in a recursive form. borrowing the derivation from [14], the state value function can be written recursively as in Equation 3.8, conforming to a special type of equation, called *Bellman equations*. The recursivity of the value function can be interpreted as the property that allows information about the value of future states to trickle back in time, and influence the value of preceding states. Similarly, the state-action value function can be written as a Bellman equation as shown in Equation 3.9.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma \cdot G_{t+1} \mid s_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s',\ r \mid s,\ a) \big[r + \gamma \cdot \mathbb{E}_\pi[G_{t+1} \mid s_{t+1} = s']\big] \\ &= \sum_a \pi(a \mid s) \sum_{s',\, r} p(s',\ r \mid s,\ a)\, [r + \gamma \cdot v_\pi(s')] \qquad\qquad \forall\ s \in \mathcal{S} \end{aligned} \tag{3.8}$$

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}[r_{t+1} + \gamma \cdot \max_{a'} q_\pi(s_{t+1}, a') \mid s_t = s,\ a_t = a] \\ &= \sum_{s',\, r} p(s',\ r \mid s,\ a) \Big[r + \gamma \cdot \max_{a'} q_\pi(s', a')\Big] \qquad \forall\ s \in \mathcal{S},\ a \in \mathcal{A} \end{aligned} \tag{3.9}$$

Optimal Policy and Value Functions
As stated previously, the Reinforcement Learning goal is to maximise the reward over time by following an optimal sequence of states and actions according to a policy π. Mathematically, π is better than π' if its expected return is higher for all states, which is the case if and only if $v(s)_\pi \geq v(s)_{\pi'}\ \forall\ s \in \mathcal{S}$. An optimal policy π_* isn't unique, and they all share the optimal value functions $v_*(s)$ and $q_*(s,a)$ which are defined according to Equation 3.10 and Equation 3.11, respectively. Finally, the Bellman optimality equations for the optimal value functions which are as derived in [14] are given by Equation 3.12 and Equation 3.13. These show that under an optimal policy, the value of a state and state-action pair must be equal to the expected return for taking the best action from that state.

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{3.10}$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \tag{3.11}$$

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid s_t = s,\ a_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[r_{t+1} + \gamma \cdot G_{t+1} \mid s_t = s,\ a_t = a] \\ &= \max_a \mathbb{E}[r_{t+1} + \gamma \cdot v_*(s_{t+1}) \mid s_t = s,\ a_t = a] \\ &= \max_a \sum_{s',\, r} p(s',\ r \mid s,\ a)\, [r + \gamma \cdot v_*(s')] \end{aligned} \tag{3.12}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[r_{t+1} + \gamma \cdot \max_{a'} q_*(s_{t+1}, a') \mid s_t = s,\ a_t = a] \\ &= \sum_{s',\, r} p(s',\ r \mid s,\ a) \Big[r + \gamma \cdot \max_{a'} q_*(s', a')\Big] \end{aligned} \tag{3.13}$$

3.1.3. On-Policy Versus Off-Policy

If the agent acts based on the policy it is learning then it can be said to be an *on-policy* learning process. If, on the other hand, the agent chooses actions based on a *behavioural policy* while a different, *target policy*, is learnt, it can be said to be an *off-policy* process.

On-policy can lead to faster learning because decisions are made based on the most recent knowledge about the state-action values, allowing the agent to exploit what has been learnt so far. Exploitation means choosing a *greedy* action, which is the action that has the highest action value at the current state. However, this comes at the expense of convergence to the optimal policy due to limited exploration of the environment by the agent, who could have discovered more optimal behaviours otherwise. One way in

which the behavioural policy can encourage exploration is by using ε-greedy decision-making, where ε is the probability that a random action is chosen, instead of the greedy one.

Off-policy methods solve the inherently exploitative nature of on-policy methods by separating the learnt and the behavioural policies. This allows a balance to be struck between exploration and exploitation. The behavioural policy is designed for exploration by having a stochastic component, while the learnt policy can be exploited based on knowledge gained from the former. This approach leads to slower learning with larger variance, but it has the potential to find a well-generalised, near-optimal target policy [14].

3.1.4. Online Versus Offline Learning

The learning process can be classified as *online* or *offline*, depending on the environment in which learning takes place. If training is done on the real system, which in the context of this report is the Cessna Citation II in flight, then the process is classified as online. If training is instead done on a model of the real environment, i.e. a simulation, then the process is classified as offline. Both online and offline learning present unique sets of challenges for robotics and control purposes.

Online learning is advantageous for systems whose dynamics are difficult to model such as robotic manipulators and aircraft because the agent can directly learn from interaction with the real system without ever having to interact with a simulation. Online methods open the door for adaptive policies that can react to changes in the plant dynamics, such as faults or wildly varying environmental conditions. Thus. An adaptive policy is appealing for applications in which robust or fault-tolerant control is desired. However, training on physical systems is time-consuming, expensive, and unsafe. An RL agent often takes thousands of episodes to learn a task and requires exploration of the state-action space which could be catastrophic for a physical system. Online learning has been explored by many, to various degrees of success, such as done by [9], [16], [17], [18], [7] and [19].

Offline learning offers the opportunity to explore the system's dynamics much faster, cheaper and without risk, but it requires a simulation of sufficiently high fidelity, a metric that is difficult to determine a priori. Deploying the simulation-trained agent in the real system is the source of several challenges stemming from discrepancies between the dynamics in simulation and reality. These include the simulation-reality gap, and the simulation optimisation bias, which will be discussed later. Offline learning has also been investigated by many researchers in diverse applications such as in [10], [20], [21], and [22].

Hybrid methods combine an offline simulation-trained agent with an online system identification algorithm in an attempt to gain the best of both worlds. The approach aims to restrict the expensive and risky exploration phase to simulation and use some form of system identification during deployment as a way to overcome the discrepancies between the dynamics in simulation and reality. However, in comparison to strict offline or online methods, Hybrid methods are more complex to design and implement. Examples of hybrid approaches and implementations include [12], [23], [13].

3.1.5. Model Free Versus Model Based

When referring to the model dependence of reinforcement learning methods, the intention is whether a probability distribution model Equation 3.2 or a sample model is available and can be used by the agent to predict the outcome of its actions [14]. Generally speaking, Model-based methods such as Dynamic Programming rely on planning [24], whereas model-free methods learn an approximation of a function describing the relationship between states and actions. [14]. However, in practice, probability distributions are rarely available; thus, modern RL methods are, for the most part, model-free despite the fact that a model, i.e. simulator, is used during training.

3.1.6. Classification of Reinforcement Learning Algorithms

In the following chapters, various RL algorithms will be discussed. First are tabular methods in Section 3.2 and then methods for continuous problems in Section 3.3. Each algorithm can be classified according to the characteristics introduced earlier, such as whether it is model-based or model-free Section 3.1.5, if it is on or off-policy Section 3.1.3, or if it is meant to solve RL problems with continuous or discrete state and action spaces. Classifying the different methods as such helps keep track of the bigger picture and can be consulted with when it's time to choose an approach for developing an RL controller for the Cessna Citation II Figure 3.2.

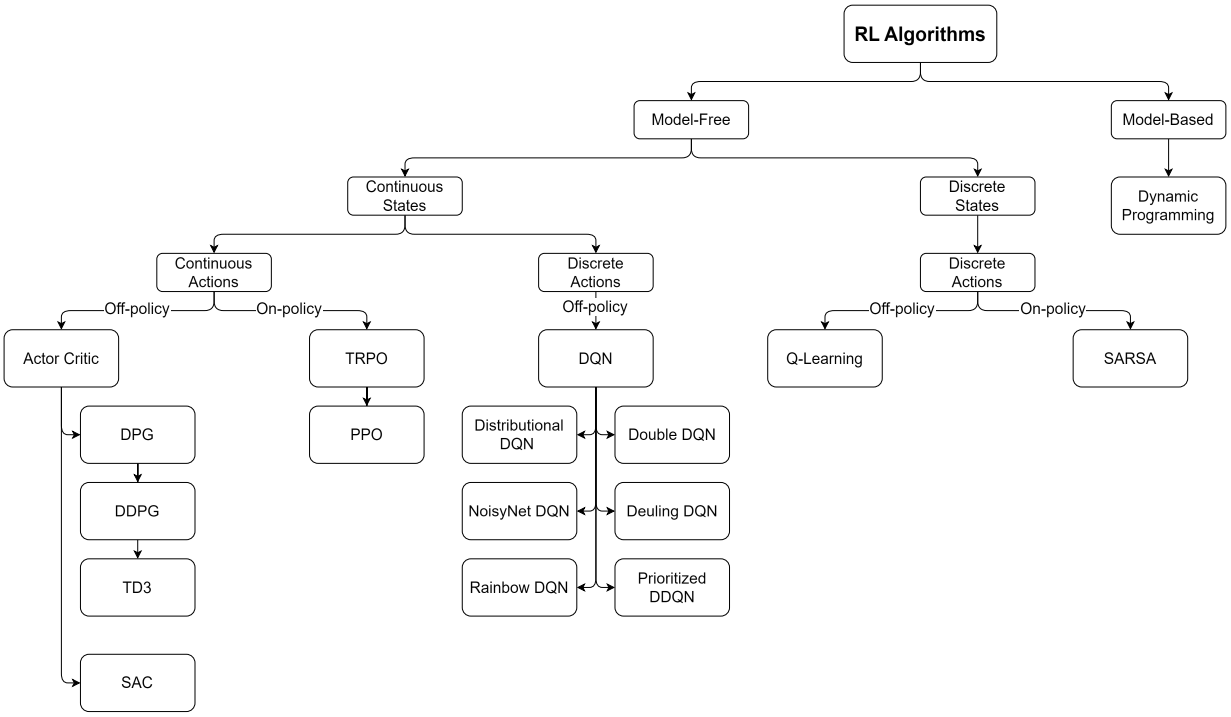


Figure 3.2: Classification of the RL algorithms treated in Section 3.2 and Section 3.4. Adapted from [25]

3.2. Common Tabular Solution Methods

Tabular solution methods are particularly well suited for discrete problems with finite state and action spaces. Tabular methods, as the name suggests, represent value functions as real-valued entries in tables. Thus, the more states and actions, the larger the tables must be. This leads to an exponential increase in the time it takes to fill and the memory required to store all the values as explained in Section 3.6.1. It is immediately clear that tabular methods are not at all suited for the task of controlling an aircraft with multi-dimensional, continuous state and action spaces. However, the algorithms described below are the basis of other more advanced RL solutions.

3.2.1. Dynamic Programming

Dynamic Programming (DP) is a term used to describe a family of algorithms meant for computing optimal policies in finite MDPs, assuming full knowledge of the environment's dynamics. Hence, DP is a model-based method. In most realistic cases, the assumption that the probability function $p(s', r \mid s, a)$ is known doesn't hold, making DP infeasible. Yet, it forms the foundations for other techniques such as adaptive DP (ADP) as introduced in Section 3.5, and is therefore important.

Policy Iteration Through Evaluation and Improvement

DP algorithms are based on turning the Bellman optimality equations for $v_*(s)$ and $q_*(s, a)$ into update rules that iteratively improve the approximation of the value functions, allowing better policies to be found [14]. The first step towards finding an optimal policy is to compute the state-value function as defined in Equation 3.8 under an arbitrary policy π. This is called *policy evaluation* and it is done using the Bellman equation for $v_\pi(s)$. Once the state value function is known, it can be used to compute the state-action value function according to Equation 3.14, which is a required element in the proof of the *policy improvement theorem*. The proof can be found in [14] and it shows that, using $q_\pi(s, a = \pi'(s))$, a policy π' is better than another policy, if and only if $v_\pi(s) \leq v_{\pi'}(s)$. This leads to the next step of *policy improvement*, an operation in which a better policy is found by acting greedy with respect to the value function of the old policy Equation 3.15. The process of repeatedly evaluating policies and improving them is called *policy iteration* as illustrated in Equation 3.16. Since each new policy is guaranteed to be at least as good as the last one, iteration leads to convergence towards the optimal policy.

$$q_\pi(s,a) \doteq \mathbb{E}[r_{t+1} + \gamma \cdot v_\pi(s_{t+1}) \mid s_t = s,\ a_t = a] \tag{3.14}$$

$$\pi'(s) \doteq \underset{a}{argmax} \sum_{s',\, r} p(s',\ r \mid s,\ a)\,[r + \gamma \cdot v_\pi(s')] \tag{3.15}$$

$$\pi_0 \xrightarrow{eval} v_{\pi_0} \xrightarrow{impr} \pi_1 \xrightarrow{eval} v_{\pi_1} \xrightarrow{impr} \pi_2 \xrightarrow{eval} ... \xrightarrow{impr} \pi_* \xrightarrow{eval} v_* \tag{3.16}$$

Value Iteration
A pitfall of policy iteration is that evaluating the policy requires iterative sweeps through the state space to reach a convergent approximation of the value function, which is computationally expensive. Fortunately, it turns out that evaluating the policy until the value function converges is unnecessary, and instead, the process can be truncated after a single sweep of the state space without giving up on the converging behaviour of the policy itself. This approach is called *value iteration* and it allows faster, more efficient approximation of the optimal policy π_* Equation 3.17.

$$\begin{aligned} v_{k+1}(s) &\doteq \underset{a}{max}\ \mathbb{E}[r_{t+1} + \gamma \cdot v_k(s_{t+1}) \mid s_t = s,\ a_t = a] \\ &= \underset{a}{max} \sum_{s',\, r} p(s',\ r \mid s,\ a)\,[r + \gamma \cdot v_k(s')] \end{aligned} \tag{3.17}$$

General Policy Iteration
Regardless of the exact method through which policy evaluation and policy improvement interact, as long as they do, and that leads to the policy converging, then the principle of policy iteration is maintained. This is reflected by the term *general policy iteration* as illustrated in Figure 3.3 and it is the basis for the vast majority of RL methods.

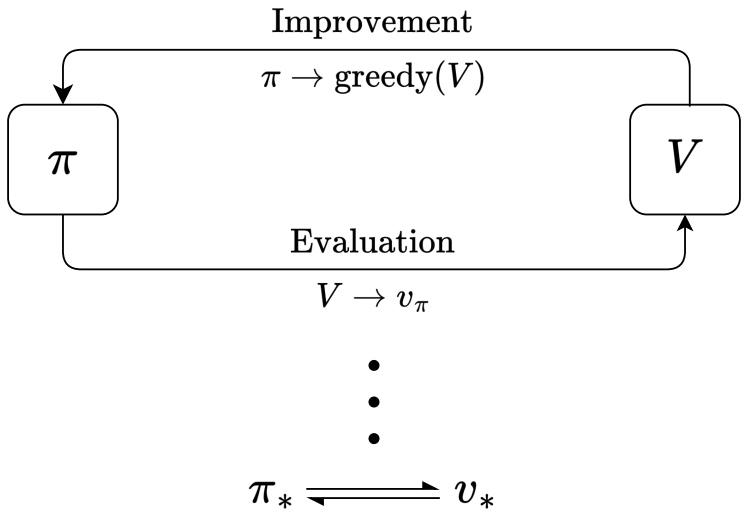


Figure 3.3: The general policy iteration process involves policy evaluation and improvement until convergence. Adapted from [14]

3.2.2. Monte Carlo

Monte Carlo (MC) is an RL method which doesn't require any knowledge of the environment's dynamics, making it model-free. This characteristic makes MC, in contrast with DP, applicable to real-world problems where the probability function is unknown. Instead of relying on $p(s', r \mid s, a)$, the state value function is

estimated (rather than computed as in DP) by averaging the returns after multiple visits to state s. In the limit, the average becomes the expected value. Iteration in MC takes place on an episodic basis, where average returns are computed for the *first visit* to a particular state, or for *every visit* to a state in the episode. Another notable difference from DP is that since the estimate of a state value $v(s)$ is independent of other state values $v(s')$, MC does not *bootstrap*. An advantageous consequence of not bootstrapping is that not all states must be visited to gain estimates for other states, making the GPI more efficient than in DP.

In model-free methods, the value function isn't sufficient to find a policy, so an estimate of the action value function $q_*(s, a)$ is required. It can be found iteratively using GPI as shown in Equation 3.18. The problem is that when a state-action pair s, a is visited and its value estimated, under a deterministic policy, a will become the default action in the state s Equation 3.19, leaving the rest of the actions $\mathcal{A}(s)$ and following states $\mathcal{S}'(s)$ unexplored. One way of addressing it is using the on-policy method of *exploratory starts* (ES), where the environment is initialised randomly in each episode, enabling some degree of exploration. However, this method is not applicable to many realistic scenarios where the initial state s_0 is constant. The alternative solution (also on-policy) is to use ε-greedy policy as was described in Section 3.1.3.

$$\pi_0 \xrightarrow{eval} q_{\pi_0} \xrightarrow{impr} \pi_1 \xrightarrow{eval} q_{\pi_1} \xrightarrow{impr} \pi_2 \xrightarrow{eval} ... \xrightarrow{impr} \pi_* \xrightarrow{eval} q_* \tag{3.18}$$

$$\pi(s) \doteq \underset{a}{argmax}\ q(s, a) \tag{3.19}$$

Policy Evaluation in off-policy MC
Off-policy MC is meant to overcome the limitations of on-policy MC by separating the policy into two entities: an exploratory behavioural policy b and a target policy. The latter is represented by a greedy evaluation of an action-value function $q(s, a)$. i.e choosing the highest value action at a given state Equation 3.19. The update rule for the estimate of $q(s, a)$, written as $Q(s, a)$ in Equation 3.20, deviates from the definition for DP where the Bellman equation was used. Here, the state-action values are tied spatially not through the next state s' and action a' as seen before, but through the return G in Equation 3.21. The factor α, which in this context is called the step-size parameter, has either a dynamic or a constant value depending on the definition, but its role is to discount the error $G - Q(s_t, a_t)$ over time, thus accelerating the convergence of $q(s, a)$ towards $q_*(s, a)$

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha \cdot [G - Q(s_t, a_t)] \tag{3.20}$$

$$G \longleftarrow \gamma \cdot G + r_{t+1} \tag{3.21}$$

3.2.3. Temporal Difference

Temporal Difference (TD) leverages elements from MC and DP, resulting in a generally more efficient and widely applicable algorithm. Like MC, it is model-free but rather than being episodic it updates on a time-step basis which increases the learning rate and sample efficiency. What it borrows from DP is bootstrapping, enabling learning from acquired knowledge, thus alleviating the amount of exploration required and further increasing learning efficiency. Equation 3.22 shows the update rule for the one-step TD method, where one-step refers to the number of time steps the target is ahead of the current state. α is again some step-size parameter.

$$V(s_t) \longleftarrow V(s_t) + \alpha \cdot [r_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t)] \tag{3.22}$$

SARSA: On-policy TD
SARSA uses the quintuple $s_t, a_t, r_t, s_{t+1}, a_{t+1}$ (hence the name) to estimate the value of the current state-action value $Q(s_t, a_t)$ according to Equation 3.23. It is considered on-policy because state-action values are assigned based on the next state s_{t+1} and the action a_{t+1} taken there, making the policy and the updates to the action value function coupled.

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.23}$$

Q-learning: Off-policy TD
In contrast to SARSA, Q-learning updates the value of $Q(s_t, a_t)$ using the maximum valued action $a \in \mathcal{A}(s_{t+1})$, regardless of whether this action was taken or not as can be seen in the update rule in Equation 3.24. It is this decoupled update behaviour that makes Q-learning an off-policy TD method. Being that it is an off-policy method, the Q-function represents the target policy, so it isn't used during exploration.

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3.24}$$

A classic example that illustrates the difference between SARSA and Q-learning in terms of the resulting policy is the cliff walking example from [14] in which an agent needs to walk along a cliff to reach its target. When using SARSA, the agent develops a conservative policy which avoids risk by walking at a safe distance from the cliff's edge. While reducing risk during exploration may be desired in some cases, the end policy doesn't represent the optimal solution. With Q-learning on the other hand, the learnt policy will result in the agent walking right on the edge because during exploration it found this path to have the highest value, despite it having led to early termination.

3.3. Reinforcement Learning in Continuous Space

Until now, the methods considered relied on the assumption that the learning problem can be represented by finite, discrete state and action spaces. While many RL problems can be framed as such to align with this assumption, problems in robotics and control, in general, cannot. In the state representation of the Cessna Citation [10], more than nine variables are needed for a complete description of the state and action spaces. Each variable can have infinitely many values and can span several orders of magnitude. Some have a small range, such as attitude angles, and some span a much larger range such as flight altitude. This is elaborated on in Section 3.6.1 where the curse of dimensionality is treated. The implication of having infinitely many states is that most of them will never be seen by the agent, requiring it to estimate their value based on knowledge from similar states. It is therefore instrumental that the policy can generalise through value approximation. Due to this incompatibility of continuous-space problems with tabular RL methods, new methods are needed for calculating state and action values, and for representing their functions.

3.3.1. Function Approximation Methods

Parametrisation
The first function approximation method to consider is approximation via parameterisation. The idea is to represent the function with a vector of weights $\mathbf{w} \in \mathbb{R}^d$ such that $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$ [14]. It is assumed that d is much smaller than the number of states, meaning that a change in a weight's value influences the value of many states. Finding the values of weights that lead to the best approximation requires optimisation with the objective of minimising the value error $\overline{\mathbf{VE}}$. It is the sum over all states of the squared error between the real value function and its approximation Equation 3.25. $\mu(s)$ is a distribution function used to weigh the relative importance of states, often based on the time spent in or the number of visits to a state. Looking back at the assumption of infinitely large state space and an unknown value function $v_\pi(s)$, it is unclear at this point how Equation 3.25 can even be calculated. Well, in the next section, several optimisation techniques show how this can be done.

$$\overline{\mathbf{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w})\right]^2 \tag{3.25}$$

Linear Features
Another function approximation technique is using a feature vector $\mathbf{x}$ as shown in Equation 3.26. There are multiple methods to construct linear features such as polynomials, Fourier basis, tile coding and more. The benefits of linear methods are that they are efficient and have guaranteed convergence. However, their main limitation is that features are decoupled, meaning that interactions between state variables are not taken into account unless extra features are used.

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s) \tag{3.26}$$

Artificial Neural Networks

The third and by far the most popular way to approximate functions is artificial neural networks (ANN) whose structure is inspired by neural networks in the brains of animals [14]. The basic structure consists of an input layer, a hidden layer (often multiple), and an output layer. The working principle is that all neurons in a layer are connected to all neurons in neighbouring layers by weights w. The weights represent the relative importance of the output of each neuron in the preceding layer. Consider neuron $h_{1,1}$ in Figure 3.4. Its inputs are the outputs of neurons I_1 to I_i multiplied by their respective weights $w_{1,1}$ to $w_{i,1}$. Then, the weighted inputs of $h_{1,1}$ are summed and processed through an *activation function* before passing onto the next layer of neurons. The role of the activation function is to introduce non-linearities, which is the key to learning non-linear relationships between inputs and outputs. There are many types of activation functions, but among the common ones are the hyperbolic tangent (tanh) Equation 3.27, the rectified linear unit (ReLU) Equation 3.29 and the softplus in Equation 3.28. The choice of which activation function to use as well as how many hidden layers and how many neurons are in each layer depend on the application. Generally, when there are multiple hidden layers, the net is considered to be a deep ANN (DNN), but there are far more complex structures such as transformers [26] and convolutional ANNs (CNN).

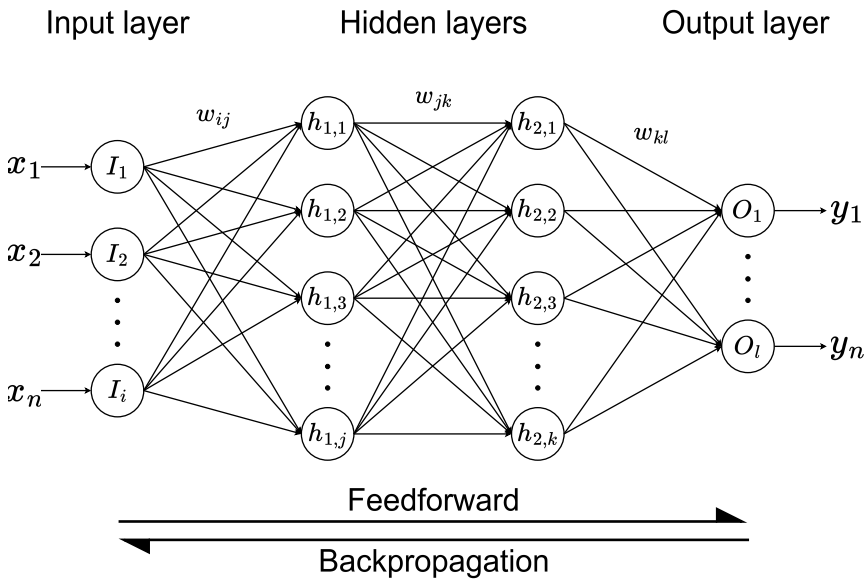


Figure 3.4: Illustration of a generic artificial neural network

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.27}$$

$$f(x) = \ln(1 + e^x) \tag{3.28}$$

$$f(x) = 0 \text{ if } x \leq 0, \text{ else } x \tag{3.29}$$

Learning in ANN amounts to adjusting the weights until the desired relationship between the inputs and outputs is achieved. Training is done in a supervised manner by passing labelled data through the net and comparing its output with what the label says about a piece of data. If, for example, a net is fed a labelled picture of a dog and at the output it claims that the picture is of a cat, then that discrepancy is measured with a cost function in a similar manner to what was shown before. To adjust the weights based on the cost function, its gradient with respect to the weights is calculated for each layer, starting from the last layer and moving backwards through the net until the first one. This method is called backpropagation. Still, it doesn't say much about how the gradient is used to adjust the weights. A common optimisation method for adjusting the weights using the cost function gradient is Adam [27], but other methods, as introduced in Section 3.3.2, can also be used, such as stochastic gradient descent (SGD) [28].

3.3.2. Value Error Optimisation

Stochastic Gradient Descent
Stochastic Gradient Descent accomplishes the goal of minimising the value error by finding a locally optimal weight vector $\mathbf{w}_*$. Assuming $\hat{v}(s, \mathbf{w})$ is differentiable, its gradient is used to find the direction that minimises the error the fastest. The unknown target value $v_\pi(s_t)$, is substituted with an estimate U_t which can be a noisy estimate, obtained after several training rounds. If U_t is unbiased, which means that $\mathbb{E}[U_t \mid s_t = s] = v_\pi(s)$, then $\mathbf{w}$ will converge to a local minimum [14]. Equation 3.30 shows the SGD update rule for the weights.

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t + \alpha \cdot [U_t - \hat{v}(s_t, \mathbf{w}_t)] \cdot \nabla \hat{v}(s_t, \mathbf{w}_t) \tag{3.30}$$

Gradient Monte Carlo
A slight modification to SGD is to use $U_t = G_t$. This substitution is motivated by considering that if U_t is indeed the expected value, then it is equivalent to what is shown in the definition of $v_\pi(s)$ Equation 3.6. This results in a method called Gradient Monte Carlo (GMC) whose update rule is shown in Equation 3.31. The downside of GMC is that similar to MC, the episode has to end before an update takes place, therefore slowing convergence down.

$$\mathbf{w} \longleftarrow \mathbf{w} + \alpha \cdot [G_t - \hat{v}(s_t, \mathbf{w})] \cdot \nabla \hat{v}(s_t, \mathbf{w}) \tag{3.31}$$

Semi-gradient Descent
The last optimisation algorithm considered here is Semi-Gradient Descent Equation 3.32. Here the target is bootstrapped such that $U_t \doteq r_{t+1} + \gamma \cdot \hat{v}(s_{t+1}, \mathbf{w})$. Because the target uses an estimate at the next time step, it is considered biased so that $\mathbb{E}[U_t \mid s_t = s] \neq v_\pi(s)$, meaning convergence is not guaranteed. Moreover, because U_t is now a function of $\mathbf{w}$, this method doesn't perform a true gradient descent, hence the name Semi-gradient Descent. Despite these drawbacks, it is considered reliable for optimisation in linear approximation methods as discussed in Section 3.3.1.

$$\mathbf{w} \longleftarrow \mathbf{w} + \alpha \cdot [R_t + \gamma \cdot \hat{v}(s_{t+1}, \mathbf{w}) - \hat{v}(s_t, \mathbf{w})] \cdot \nabla \hat{v}(s_t, \mathbf{w}) \tag{3.32}$$

3.3.3. Objects to Approximate

Now that the tools to approximate functions are known, it is worth discussing what it is that should even be approximated. Previously, three objects were introduced: the probability function, the value function and the policy. For model-based methods, it can be useful to approximate the model, i.e. the probability function. Then, for Model-free algorithms, three agent structures exist. Those that learn to approximate the value function are called *critic* methods, while those that approximate the policy are regarded to as *actor* methods. Then there are hybrid methods which learn both and are called *actor-critic* methods.

Model Approximation
Approximating the model means learning an implicit function of the probability distribution Equation 3.2 used in DP. Once the model is mapped, it can be used in the DP policy iteration process to find a policy and state value function. In control problems, the learnt model doesn't represent the probabilities as was introduced in the definition of DP, but rather the dynamics of the system through the input-output relationship of states and actions. A survey of various model learning methods in robotic applications is presented in [29]. The motivation for learning a model is that it makes DP applicable to RL problems for which the model is unknown a priori. However, since the policy is iterated with a surrogate model of the environment, the optimal policy and value function are only optimal for the surrogate, not the real environment.

Critic: Value Function Approximation
In critic-only methods, the only learnt object is the approximated value function $\hat{q}(a, s, \mathbf{w})$ or $\hat{v}(s, \mathbf{w})$. Therefore, the policy is not an explicit function or an approximation thereof, but rather a greedy evaluation of $\hat{q}(a, s, \mathbf{w})$ where the highest valued action at a state is chosen (or, for an ε-greedy policy, a random action with probability ε). The main limitation of Critic-only methods is that they only apply to discrete action spaces because choosing a greedy action requires evaluation of the values of all actions at a state, which cannot be done when there are infinitely many actions.

Actor: Policy approximation
Actor-only methods learn a policy function, i.e. a mapping from states to actions, and use it to directly select an action based on the current state. Since the focus here is on continuous state spaces which require function approximation, the policy is approximated with parameterisation vector θ, giving that $\pi(a \mid s, \theta) = Pr\{a_t = a, \mid s_t = s,\ \theta_t = \theta\}$. In contrast to critic-only methods in which $\hat{q}(a, s, \mathbf{w})$ can only be evaluated for the values of discrete actions, here the policy is a continuous probability function; thus, it is stochastic and differentiable, enabling evaluation for any action. These properties make it well-suited for infinite action spaces.

Optimising the parameter vector θ can be done using *policy gradient methods* [28], which rely on a different theory than that shown for value functions in Section 3.3.1. According to the policy gradient theorem, if a policy performance indicator $J(\pi_\theta)$ is given, then the change in θ should be approximately proportional to the change of performance with respect to θ. This is shown in Equation 3.33 where α is a step size parameter. This method is called *gradient ascent* and the local optimum is achieved when Equation 3.33 is zero. A more practical definition is given by the REINFORCE MC policy gradient update rule from [14] as shown in Equation 3.34. The intuition is that the policy parameters change proportional to the discounted return $\gamma^t \cdot G_t$ and the gradient of the policy $\nabla\pi(a_t \mid s_t, \theta_t)$, and inversely proportional to the probability of choosing a_t at s_t which would otherwise cause the updates to be in favour of previously chosen actions, causing bias. Like other MC methods, the REINFORCE update rule is episodic so it is rather slow to converge and it suffers from large variance due to dependence on the returns, which vary a lot. To improve on this weakness, REINFORCE with baseline adds the state-value function into the update rule for θ Equation 3.35. Note that the step size parameter α may have different values in the policy's update rule and in that of the value function in Section 3.3.2.

$$\Delta\theta \approx \alpha \cdot \frac{\partial J(\pi_\theta)}{\partial \theta} \tag{3.33}$$

$$\begin{aligned} \theta_{t+1} &= \theta_t + \Delta\theta \\ &\doteq \theta_t + \alpha \cdot \nabla_\theta J(\pi_{\theta_t}) \\ &= \theta_t + \alpha \cdot \gamma^t \cdot G_t \cdot \frac{\nabla_\theta \pi(a_t \mid s_t, \theta_t)}{\pi(a_t \mid s_t, \theta_t)} \\ &= \theta_t + \alpha \cdot \gamma^t \cdot G_t \cdot \nabla_\theta \ln \pi(a_t \mid s_t, \theta_t) \end{aligned} \tag{3.34}$$

$$\theta_{t+1} \doteq \theta_t + \alpha \cdot \gamma^t \cdot (G_t - \hat{v}(s_t, \mathbf{w})) \cdot \nabla \ln \pi(a_t \mid s_t, \theta_t) \tag{3.35}$$

Actor-Critic
The last agent structure consists of both an actor- the policy, and a critic- the value function. Similarly to REINFORCE with baseline, a value function (can be either state or action-value function, depending on the method) is used in the policy's update rule to modulate the increment of θ, but it differs in that it bootstraps by using the one-step return $G_{t:t+1}$ as means of assessing the action. This is also called the TD error $\delta_t = r_{t+1} + \gamma \cdot \hat{v}(s_{t+1}, \mathbf{w}) - \hat{v}(s_t, \mathbf{w})$. Equation 3.36 presents a generic one-step TD update rule for θ [14]. The advantage of using TD instead of MC for the policy gradient estimation is that bootstrapping reduces variance, while learning efficiency is increased because updates occur every time step rather than once per episode. The actor-critic agent structure is illustrated in Figure 3.5, showing how both the critic and the actor use the TD error to optimise the value function and policy parameter vectors $\mathbf{w}$ and θ, respectively.

$$\begin{aligned} \theta_{t+1} &\doteq \theta_t + \alpha \cdot \gamma^t \cdot (G_{t:t+1} - \hat{v}(s_t, \mathbf{w})) \cdot \nabla \ln \pi(a_t \mid s_t, \theta_t) \\ &= \theta_t + \alpha \cdot \gamma^t \cdot (r_{t+1} + \gamma \cdot \hat{v}(s_{t+1}, \mathbf{w}) - \hat{v}(s_t, \mathbf{w})) \cdot \nabla \ln \pi(a_t \mid s_t, \theta_t) \end{aligned} \tag{3.36}$$

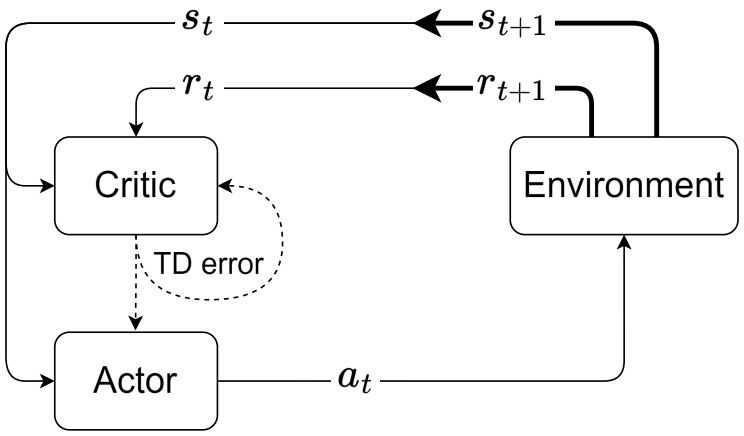


Figure 3.5: Actor-critic agent structure and its interaction with the environment. Adapted from [30]

3.4. Deep Reinforcement Learning

3.4.1. Deep Q-Networks

Deep Q-Networks (DQN) is a Deep Reinforcement Learning (DRL) variant of Q-learning, the tabular, off-policy method presented in Section 3.2.3. The main distinction is that the action-value function $q(s, a)$ is approximated by a Deep Neural Net (DNN), called a Q-network, such that $\hat{q}(a, s, \theta) \approx Q_*(s, a)$ where θ is the Q-network's weights. These can be optimised by minimising a loss function using, for example, SGD. Action sampling in DQN works similarly to the tabular approach- the current state is used as the input to the Q-network whose output are the action values. Then, the decision of which action to take depends on the policy. Due to this architecture, the action space must be discrete.

Another distinct feature of DQN, besides using a DNN, is the use of *experience replay* which stores $e_t = [s_t, a_t, r_t, s_{t+1}]$ at each time step in the *replay memory* $\mathcal{D} = [e_1, ...e_N] \in \mathbb{R}^N$ where N is the buffer size. During an action-value function update, an experience sequence e_t is chosen at random from $\mathcal{D}$ and is used to make a gradient descent step. The motivation behind using random experiences instead of consecutive ones is that it reduces sample correlation, which in turn, reduces update variance, leading to smoother, more stable learning. DQN is model-free because instead of using experience from interaction to learn the environment's model, it directly learns the action value function. It is also off-policy because it uses a behaviour policy such as ε-greedy to explore the environment while using the strictly greedy evaluation to train the Q-network. While the results of using DQN for Atari games in [31] are impressive and show good convergence behaviour (although, without a theoretical proof and under risk of the deadly triad Section 3.6.2 [32]), the method is limited to low dimensional, discrete action spaces [33] making it unsuitable for tasks such as controlling an aircraft.

Since the conception of DQN, many algorithms have been developed to improve its weaknesses and adapt it for various applications. Double Q-Learning (DDQN) [34] is designed to reduce the overestimation tendencies of DQN. Prioritised experience replay [35] learns from experiences from which there is more to learn, thus increasing sample efficiency. Duelling network [36] offers a new ANN architecture which represents the state value and action-advantage separately for improved generalisation. Distributional Q-Learning [37] aims to learn value distribution, rather than the value function. NoisyNet [38] adds stochasticity to the weights of the policy's network to increase exploration. Finally, Rainbow [39] harnesses the benefits of all aforementioned methods, plus A3C [40] in one algorithm to achieve state-of-the-art performance.

3.4.2. Deterministic Policy Gradient

The function and policy optimisation methods treated up to this point- SGD, gradient-MC, semi-gradient and gradient ascent all assume the objective functions to be stochastic, continuous functions and hence differentiable. Specifically, they were assumed to be probability distributions with variance $\sigma > 0$. The Deterministic Policy Gradient (DPG) theorem [41] suggests a different approach where the gradient of the

deterministic policy μ_θ with $\sigma = 0$ is given by the expected gradient of the stochastic action-value function. In the theory's derivation, it is shown that a stochastic policy is equivalent to a deterministic policy in the limit of $\sigma \to 0$. The primary motivation for using DPG is the reduced computational cost due to only having to integrate over the state space when calculating $J(\mu_\theta)$ versus integrating over both states and actions for $J(\pi_\theta)$. However, deterministic policies don't allow exploration because $a(s)$ is always the same, so an off-policy method can be employed where actions are sampled using a separate stochastic behaviour policy.

Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a model-free, actor-critic RL algorithm which extends DQN into continuous action spaces through the use of DPG [33]. In DQN, actions had to be discrete because the action-value function was used directly for action selection. In DDPG, it is used for calculating the gradient of the objective function Equation 3.37 so the weights of the deterministic policy μ_θ can be optimised. Due to the combination of off-policy learning and function approximation, learning tends to be unstable. To stabilise it, a copy of the actor and critic networks $\mu'(s, \theta^{\mu'})$ and $\hat{q}'(s, a, \theta^{q'})$ respectively, are updated via *soft* updates with a tunable parameter $\tau \ll 1$ as in Equation 3.38 and Equation 3.39. Since the actor is deterministic, and the Q-network critic can only output discrete actions, a mechanism to promote exploration is still missing. Instead of creating yet another network, a rather simple solution is to use noise $\mathcal{N}$ to add stochasticity to $\mu(s, \theta^\mu)$ according to Equation 3.40.

$$\nabla_{\theta^\mu} J(\mu_\theta) = \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_a \hat{q}(s_t, \mu_\theta(s_t), \theta^q) \cdot \nabla_{\theta^\mu} \mu(s, \theta^\mu) \right] \tag{3.37}$$

$$\theta^{\mu'} \longleftarrow \tau \cdot \theta^\mu + (1 - \tau) \cdot \theta^{\mu'} \tag{3.38}$$

$$\theta^{q'} \longleftarrow \tau \cdot \theta^q + (1 - \tau) \cdot \theta^{q'} \tag{3.39}$$

$$\mu^{\mathcal{N}}(s_t) = \mu(s_t, \theta^\mu) + \mathcal{N} \tag{3.40}$$

The last aspect of DDPG to treat is Batch Normalisation (BN). When state variables of different orders of magnitude and increment ranges are directly fed into the DNN, it may become insensitive to those variables that don't significantly vary in comparison to others. The solution is to maintain a mean value for each state variable and normalise new samples with respect to it before feeding them into the network [42], [33].

In the Atari environment, DDPG was able to achieve DQN performance but with a factor 20 fewer time steps [33]. While this is promising, DDPG, like other off-policy, function approximating methods, is prone to instabilities and divergence thanks to the deadly triad Section 3.6.2, requiring methods like soft updates to stabilise learning. This adds implementation complexity, computational cost and memory requirements. Moreover, according to [33], DDPG also doesn't excel in sample efficiency, making it a non-ideal solution for online learning. Lastly, the algorithm is sensitive to the choice of hyperparameters, of which there are plenty [43]. A common vulnerability of Q-learning methods is the overestimation of Q-values and error accumulation, leading to suboptimal policies [44].

Twin Delayed DDPG

Twin Delayed DDPG (abbreviated TD3) is a derivative of DDPG and is also an actor-critic algorithm meant for continuous state and action spaces [44]. TD3 aims to address the issues DDPG has with the overestimation of Q-values. The first modification of TD3 with respect to DDPG is the use of target policy smoothing which is meant to prevent the policy from exploiting overestimated Q-values. It does so by adding clipped noise to the action when sampled from the deterministic policy. The idea behind it is that similar actions should have similar values, so if action a is chosen and has an abnormally high value compared to neighbouring actions, then adding noise would mean that a similar, yet differently valued action is chosen, enabling the overestimated action values to be avoided. The second difference is that TD3 learns two Q-functions concurrently and uses the lower-valued one for the target in the loss function. This approach is also meant to battle the overestimation of action values. The third differentiating aspect between the two algorithms is that TD3 updates the policy at a lower frequency than the Q-networks. The goal is to minimise the estimation error in the action value function before using it to update the policy.

3.4.3. Trust Region Policy Optimisation

Trust Region Policy Optimisation (TRPO) is an on-policy algorithm designed to guarantee monotonic policy improvement [45]. It does so by taking the largest update step possible when optimising the policy parameters θ while adhering to a KL-divergence constraint to prevent steps that cause drastic variations between consecutive policies. This method solves the issue of policy divergence when taking too large a step in normal policy gradient methods. The update rule in Equation 3.41 shows mathematically what has been explained above- the policy parameters are updated by maximising over the surrogate advantage Equation 3.42 in which $A^{\pi_{\theta_t}}(s,a) = \hat{q}(s,a) - \hat{v}(s)$, bounded by an average KL-divergence Equation 3.43 which defines a trust region for the update. In practice, these quantities are not easy to obtain, so approximations thereof are used instead.

The Kullback–Leibler (KL) divergence is used as a measure of similarity of two different probability distributions [46]. In the context of RL and TRPO in particular, it is used to evaluate if the policy π_θ is very different from the old policy π_{θ_t}. The update then takes place under the condition that the difference is smaller than the permissible margin δ.

$$\begin{aligned} \theta_{t+1} &\longleftarrow \underset{\theta}{argmax}\ L_{\theta_t}(\theta) \\ &\text{subject to } \overline{D}_{KL}(\theta \mid\mid \theta_t) \leq \delta \end{aligned} \tag{3.41}$$

$$L_{\theta_t}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_t}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)} \cdot A^{\pi_{\theta_t}}(s,a)\right] \tag{3.42}$$

$$\overline{D}_{KL}(\theta \mid\mid \theta_t) = \mathbb{E}_{s\sim\pi_{\theta_t}}\left[D_{KL}(\pi_\theta(\cdot \mid s) \mid\mid \pi_{\theta_t}(\cdot \mid s))\right] \tag{3.43}$$

According to [45], TRPO delivers on the promise to stabilise and accelerate learning by using a trust region for policy updates as evident by the high performance achieved in a wide range of robotic locomotion and game-playing tasks. However, because TRPO is on-policy, exploration and exploitation are mutually exclusive. As the policy converges, it explores less and exploits more by choosing actions that lead to high-reward trajectories. The limitation of this approach is that the policy is likely to find only a local optimum and not explore beyond it. Furthermore, TRPO is considered to be complicated to implement, making its successor, PPO, an attractive alternative [47].

3.4.4. Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is an evolution of TRPO and is based on the same idea of taking large steps during policy updates while maintaining similarity between consecutive policies [47]. However, both versions of PPO; PPO-Clip and PPO-Penalty are much simpler to implement than TRPO. Here, the focus lies on PPO-Clip which will be referred to as PPO for convenience. In contrast to TRPO which uses the KL-divergence to form a trust region, PPO simply clips the value of the policy ratio in the surrogate advantage function Equation 3.45 if it deviates from unity by more than a margin $\pm\epsilon$. Where ϵ is a tunable hyperparameter that influences how far the current policy is allowed to be from the old one. The clipping function is defined in Equation 3.46.

$$\theta_{t+1} \longleftarrow \underset{\theta}{argmax}\ \mathbb{E}_{s,a\sim\pi_{\theta_t}}\left[L_{\theta_t}(s,a,\theta)\right] \tag{3.44}$$

$$L^{clip}_{\theta_t}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_t}}\left[\text{min}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)} \cdot A^{\pi_{\theta_t}}(s,a),\ g(\epsilon, A^{\pi_{\theta_t}}(s,a))\right)\right] \tag{3.45}$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & \text{if } A \geq 0 \\ (1-\epsilon)A & \text{otherwise} \end{cases} \tag{3.46}$$

PPO has a similar exploration-exploitation mechanism as TRPO, where, due to its on-policy nature, it explores less as the policy improves, which can lead to it being stranded in a local optimum. Despite this,

in terms of sample efficiency and obtained returns, it was able to outperform TRPO [45], Cross-entropy Method (CEM) [48], Vanilla Policy Gradient (VPG) [28], Advantage Actor-critic (A2C) [40] and A2C with trust region [49]. Of course, different papers reach different conclusions based on the specific implementation and task, so universal statements regarding an algorithm's performance should be taken with a grain of salt.

3.4.5. Soft Actor-Critic

Soft Actor-Critic (SAC) is an off-policy algorithm that learns a stochastic policy and two action-value functions, a feature akin to that of TD3. SAC is unique among the algorithms seen so far in that it uses entropy regularisation to keep the policy as stochastic as possible [50]. In statistics, the entropy $\mathcal{H}$ quantifies how random a random process is according to Equation 3.47. A highly random process has high entropy and vise versa. Balancing between high returns and high entropy allows exploration to be maintained, preventing the policy from stagnating in a low-performing local optimum- a problem earlier identified in on-policy algorithms such as TRPO and PPO. The definition of the optimal policy in SAC is shown in Equation 3.48, which is taken from [19]. Here, α is a new hyperparameter called the temperature coefficient and it is used to weigh the entropy's significance relative to the return. SAC can be implemented with either a constant temperature coefficient or with an entropy target, requiring α to vary over time according to the gradient of $J(\alpha)$ in Equation 3.49. The nonparametrised form of the value function $V(s)$ in Equation 3.50 is defined using $Q(s, a)$, making it a redundant object in and of itself, thus the parametrised action-value function $\hat{q}(s, a, \theta^q)$ is computed directly from the gradient of $J(\theta^q)$ in Equation 3.51. Lastly, the policy parameters are computed according to a soft update Equation 3.38 using the gradient of $J(\theta^\mu)$ Equation 3.52.

$$\mathcal{H}(\pi(\cdot \mid s)) = \mathbb{E}_{a\sim\pi}\left[-\log \pi(a \mid s)\right] \tag{3.47}$$

$$\pi_* = \underset{\theta}{argmax} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot \mid s_t))\right] \tag{3.48}$$

$$J(\alpha) = \mathbb{E}_{a_t\sim\mu(s_t,\theta^\mu)}\left[-\alpha \cdot \log \mu(a_t \mid s, \theta^\mu) - \alpha \cdot \mathcal{H}'\right] \tag{3.49}$$

$$V(s_t) = \mathbb{E}_{a_t\sim\pi}\left[Q(s_t, a_t) - \alpha \cdot \log \pi(a_t \mid s_t)\right] \tag{3.50}$$

$$J(\theta^q) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(\hat{q}(s_t, a_t, \theta^q) - \left(r(s_t, a_t) + \gamma \cdot \mathbb{E}_{s_{t+1}\sim p}\left[\hat{v}(s_{t+1}, \theta^{v'})\right]\right)\right)^2\right] \tag{3.51}$$

$$J(\theta^\mu) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\mu(\theta^\mu)}\left[\alpha \cdot \log \mu(a_t \mid s, \theta^\mu) - \hat{q}(s_t, a_t, \theta^q)\right]\right] \tag{3.52}$$

The characteristics that make SAC an attractive algorithm for high-dimensional, continuous problems is that exploration is maintained throughout the training phase, allowing the agent to discover near-optimal behaviours. For control tasks, SAC proved to be robust to modelling discrepancies and is able to generalise to situations the agent hasn't encountered during training [16]. Furthermore, while thorough exploration does come at the expense of learning stability and seed dependence, in comparison to TRPO and PPO, sample efficiency is higher, thus shortening the training duration. Also mentioned in [50] is that at deployment, choosing actions deterministically from the stochastic policy leads to higher returns. Like every method, SAC also has limitations which are important to consider. First, it is sensitive to the scale of the reward signal. Since the reward is weighed against the entropy, their magnitudes should be compatible, otherwise poor balance between exploration and exploitation is obtained. Secondly, it is sensitive to hyperparameter tuning, specifically to the temperature coefficient, necessitating the addition of a dynamic value which complicates implementation.

3.4.6. State-of-the-Art DRL Applications for Flight Control

Interchangeable Reinforcement-Learning Flight Controller for Fixed-Wing UASs | M. Chowdhury and S. Keshmiri [51]

With the idea of creating an airframe-invariant controller, M. Chowdhury et al [51] use PPO to train an agent to control the attitude (roll, pitch and yaw) and velocity. Training is done offline in an engineering-level dynamics simulation and is later flight-tested on a fixed-wing unmanned aircraft system (UAS) that is different from the model used in training. To achieve an interchangeable controller, which is synonymous with a high degree of generalisation, domain randomisation is applied to the model on several levels; randomisation of flight conditions, stability and control derivatives, sensor noise, and control delay. To solve the issue of high-frequency control actuation, the reward function regulates both the Euler rates and the control deflection rates by penalising their (clipped) weighted contributions as can be seen in Equation 3.53. w_x symbolises the weight each parameter is given while γ_x and ϵ_x determine the clipping intervals.

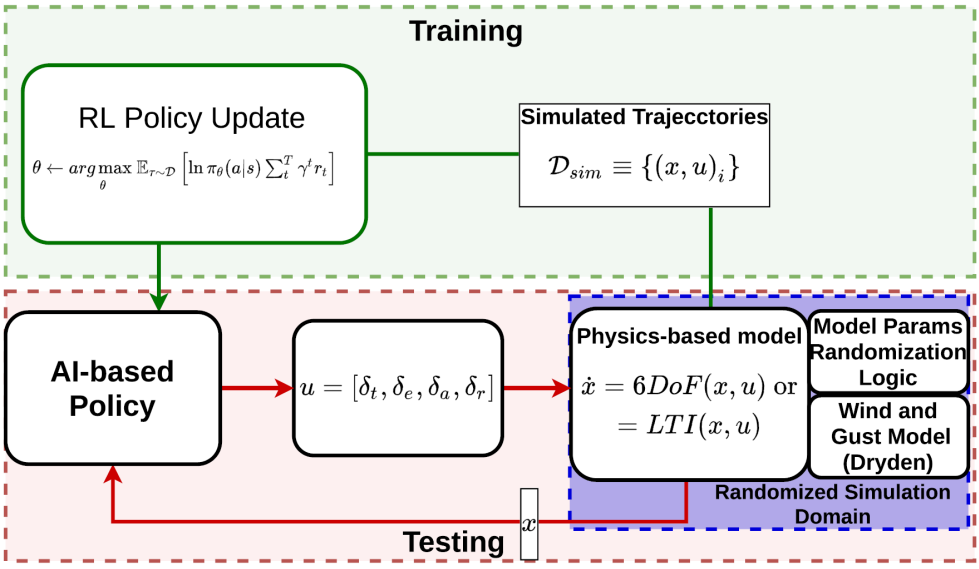


Figure 3.6: Policy training in the domain randomised environment. Taken from [51]

$$r(s,a) = \begin{cases} \text{longitudinal: } w_v|V_{a_e}| + w_\theta|\theta_e| + w_q|Q| \\ + \text{clip}(w_{\dot{\delta}_t}|\dot{\delta}_t|, 0, \epsilon_{\dot{\delta}_t}) + \text{clip}(w_{\dot{\delta}_e}|\dot{\delta}_e|, 0, \epsilon_{\dot{\delta}_e}) \\ + \text{clip}(w_{\dot{Q}}|\dot{Q}|, 0, \epsilon_{\dot{Q}}) \\ \\ \text{lateral: } w_\phi|\phi_e| + w_p|P| + w_r|R| \\ + \text{clip}(w_{\dot{\delta}_a}|\dot{\delta}_a|, 0, \gamma_{\dot{\delta}_a}) + \text{clip}(w_{\dot{\delta}_r}|\dot{\delta}_r|, 0, \gamma_{\dot{\delta}_r}) \\ + \text{clip}(w_{\dot{P}}|\dot{P}|, 0, \epsilon_{\dot{P}}) + \text{clip}(w_{\dot{R}}|\dot{R}|, 0, \epsilon_{\dot{R}}) \end{cases} \tag{3.53}$$

Testing was performed with three different UAS platforms to validate the algorithm's interchangeability. To evaluate performance, the PPO controller was compared against a Pixhawk, a conventional UAS FCS utilising a linear quadratic regulator (LQR) control law for attitude hold and PID for airspeed tracking. The performance metrics included the airspeed and altitude root mean square (RSM) of the tracking error, while attitude tracking was measured indirectly through the examination of flight trajectories relative to a rectangular reference path. The results show that the RL controller had a significant edge over the Pixhawk. In airspeed tracking, the RL autopilot outperformed its counterpart by an order of magnitude and was five times better at altitude tracking. While the authors claim that their RL controller outperforms the Pixhawk in trajectory tracking, the results leave a lot to be desired, especially considering how much it deviates from the results obtained in simulation in which excellent tracking capabilities were demonstrated.

Soft Actor-Critic Deep Reinforcement Learning for Fault-Tolerant Flight Control | K. Dally and E.J van Kampen [10]
In his paper, K. Dally developed a fault-tolerant controller using SAC to control a 6 DoF simulation model of the Cessna Citation II. His cascaded, coupled-dynamics architecture controls reference attitude angles in the inner loop and a reference altitude on the outer loop Figure 3.7. Each SAC controller's agent is trained on the high-fidelity Delft University Aircraft Simulation Model and Analysis Tool (DASMAT) simulation environment which models the Citation as a non-linear system. Actuator dynamics were modelled with a low-pass filter and saturation limits, while ideal sensors were assumed. Because the PH-LAB doesn't have an auto-throttle, airspeed is not included in the agent's action space and instead is controlled with a separate PID controller.

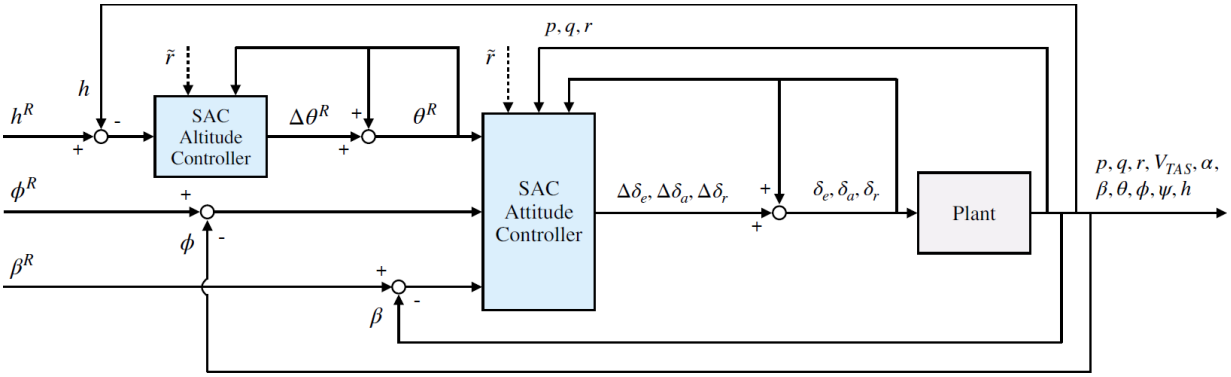


Figure 3.7: Cascaded SAC controller architecture from [10]

The SAC attitude and altitude controllers were trained consecutively using the standard dynamical model available in DASMAT as well as on six failure modes, although these were not evaluated further because the performance of the agent trained on the nominal plant was satisfactory. It is noted that training reliability was overall low, as evident by the sensitivity to the seed and high reward variance between episodes. The controller's performance was then evaluated through its ability to track a reference signal while experiencing flight conditions unseen in training, sensor noise and selected failure modes. The tested scenarios included:

- Default plant dynamics
- Jammed rudder $(\delta_r = 15°)$
- Reduced aileron effectiveness $(70\%$ reduction)
- Reduced elevator range (from $[-20.05°, 14.90°]$ to $[-2.50°, 2.50°]$)
- Partial loss of horizontal tail $(70\%$ reduction in $C_{(L/D/m)_{\delta_e}}$ and $C_{m_q})$
- Icing $(30\%$ reduction in $C_{L_{max}}$ and an increase of C_D by 0.06$)$
- Center-of-gravity shift (aft shift of $0.25m$)
- Sensor bias and noise (applied to the rates p, q, r, the Euler angles θ, ϕ, β and altitude h)
- Upwards vertical gust $(15ft/s)$

The results demonstrated high disturbance rejection, the ability to perform highly coupled manoeuvres, tolerance to various nominal initial flight conditions and robustness when faced with a significant reduction in control authority. These results are attributed to the highly exploratory and stochastic nature of SAC, enhanced by the generalisation power of DNN-based policies. The author recommends improving the agent's training stability, optimising the hyperparameters, and for implementation in the PH-LAB, expanding the simulation model to include sensor dynamics and actuator transport delays. Lastly, it is recommended to evaluate the controller's run time performance to ensure it is within the limitations of the on-board avionics.

3.5. Approximate Dynamic Programming

Approximate dynamic programming (ADP) is, like the name suggests, an extension to DP, and its aim is to find an optimal policy by approximating a cost function (cost-to-go). ADP methods can take several forms, which can be discerned based on whether they rely on the state or action value functions and or on their derivatives. The common goal of all ADP methods is to overcome the curse of dimensionality, a problem that makes running true dynamic programming computationally prohibitive. This is achieved through function approximation by using, for example, a neural network [24], [52]. The range of methods discussed below falls under the Adaptive Critic Designs (ACDs), which is a subclass of ADP in which the policy and the Bellman form of the cost function are optimised within an actor-critic structure [53]. In [54] two ACD methods were proposed: Heuristic Dynamic Programming (HDP) and Dual Heuristic Programming (DHP). Additionally, in [55] a third variation called Global Dual Heuristic Programming (GDHP) is presented. Each of these designs can be modified to employ action-dependent critics (AD), resulting in three more variations: ADHDP, ADDHP and ADGDHP [54]. Lastly is the incremental ADP (iADP), which can be combined with the aforementioned non-AD ACD methods, give rise to iHDP, iDHP and iGDHP. Table 3.1 shows a taxonomy of the different methods to help in understanding the solution space before exploring them in detail in the coming subsections.

3.5.1. Adaptive Critic Designs

ACDs can learn an optimal policy either on or offline, in which case a model of the environment is required for interaction. ACDs can also be adapted to take advantage of a known probability model, but they are not all inherently model-based. Moreover, since the cost function improvement cycle occurs every time step, the actor and actor can quickly adapt to changes in the system's dynamics, making it particularly useful for online applications.

Table 3.1: Taxonomy of the approximate dynamic programming methods. *model-free. adapted from [10]

Adaptive Critic Designs								
Heuristic			Dual Heuristic			Global Dual Heuristic		
Name	Input	Output	Name	Input	Output	Name	Input	Output
HDP	s	$v(s)$	**DHP**	s	$\frac{\partial v(s)}{\partial s}$	**GDHP**	s	$v(s)$, $\frac{\partial v(s)}{\partial s}$
ADHDP*	s, a	$q(s,a)$	**ADDHP**	s, a	$\frac{\partial q(s,a)}{\partial s}$, $\frac{\partial q(s,a)}{\partial a}$	**ADGDHP**	s, a	$q(s,a)$, $\frac{\partial q(s,a)}{\partial s}$, $\frac{\partial q(s,a)}{\partial a}$
iHDP*	s	$v(s)$	**iDHP***	s	$\frac{\partial v(s)}{\partial s}$	**iGDHP***	s	$v(s)$, $\frac{\partial v(s)}{\partial s}$

Heuristic Dynamic Programming

HDP is a common and simple adaptive critic structure for approximating $v(s)$ via a parametrised estimate $\hat{v}(s,\theta)$. As shown in Figure 3.8, the value function estimation error e_v in Equation 3.54 is computed using the reward, the value estimate at the current time step, and the discounted value estimate at the next one. The critic improves the value function approximation by minimisation e_v through optimisation of the parameter vector θ. The actor's iteration process is more convoluted because it requires satisfying the optimality condition in Equation 3.55 [56] for which the dynamical model $f(s_t, a_t)$ is needed, meaning that HDP is model-based. When using ANN for function approximation, the iteration process can be done with backpropagation. HDP's limitation, besides being model-based, is that convergence is rather slow because the parameter vector updates are based on the state value function [56].

$$e_v = \hat{v}(s_t,\theta) - [r_{t+1} + \gamma \cdot \hat{v}(s_{t+1},\theta)] \tag{3.54}$$

$$\frac{\partial \hat{v}(s_t,\theta)}{\partial a_t}\bigg|_{a_t=a_t^D} = \left[\frac{\partial r_t}{\partial a_t} + \left(\frac{\partial f(s_t,a_t)}{\partial a_t}\right)^T \frac{\partial \hat{v}(s_{t+1},\theta)}{\partial s_{t+1}}\right]\bigg|_{a_t=a_t^D} = 0 \tag{3.55}$$

The action-dependent variation of HDP (ADHDP) retains the same design as HDP but replaces the state value function $v(s)$ with the action value function $q(s, a)$. Now, the derivative with respect to the action in Equation 3.55 can be calculated directly, without requiring $f(s_t, a_t)$. Hence, ADHDP is model-free and its critic is in fact equivalent to Q-learning Section 3.2.3.

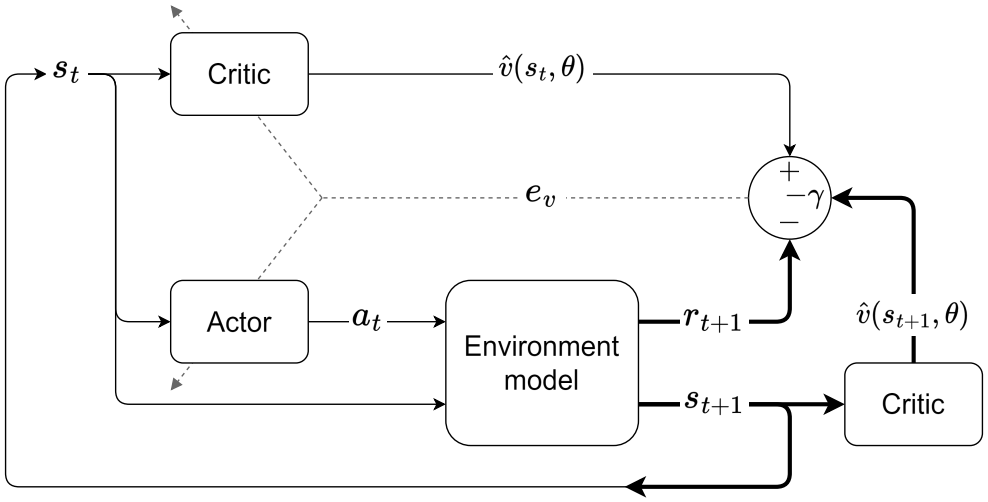


Figure 3.8: Heuristic dynamic programming structure and information flow. Adapted from [52]

Dual Heuristic Programming
In DHP, the critic estimates the gradient of the value function rather than the value function itself and is updated using the gradient error in Equation 3.56. Actor updates work in basically the same way as in HDP where the value function derivative $\partial\hat{v}(s_t, \theta)/\partial a_t$ was used. This method is also model-based because iterating on both the actor and the critic requires $f(s_t, a_t)$ which appears in the calculation of the gradient of $\hat{v}(s_{t+1}, \theta)$. According to [56], DHP is in general faster than HDP in terms of the number of iterations required to reach convergence while its limitation is the lack of internal consistency stemming from approximating the derivatives, leading to larger approximation errors compared with HDP [55].

$$e_v = \frac{\partial\hat{v}(s_t, \theta)}{\partial s_t} - \left[\frac{\partial r_{t+1}}{\partial s_t} + \gamma \cdot \frac{\partial\hat{v}(s_{t+1}, \theta)}{\partial s_{t+1}}\right] \tag{3.56}$$

ADDHP, the action-dependent version of DHP, uses the state derivatives of the Q-function instead of the value function. Consequently, while the actor updates are now model-free, the critic does still depend on the action derivative of $f(s_t, a_t)$.

Global Dual Heuristic Programming
GDHP can be viewed as a blend of DHP and HDP because its critic estimates both the value function and its gradients using the same tools discussed above. To do so, two critics are maintained; one for the estimate of the value function and one for the derivative. Its limitations are that similarly to its counterparts, it is also a model-based method. Additionally, it is more computationally intensive because it minimises the error of both critics simultaneously [57]. Despite this, GDHP's advantage is that it reduces the approximation errors of DHP, contributing to it being faster to learn than HDP [58].

Lastly, ADGDHP, to no one's surprise, estimates the action value function and its derivative instead of the value function. Just like ADDHP, it relies on the model function $f(s_t, a_t)$ to estimate $\partial\hat{q}(s_t, a_t)/\partial s_t$, making it a model-based method.

Incremental Approximate Dynamic Programming
iADP is a family of algorithms that can, in contrast to most ACDs, learn to control a system without any prior knowledge of the dynamical model through incremental system identification. While non-incremental ADP might use a third ANN to learn the global, non-linear system, many samples are needed before reaching satisfactory convergences, thus requiring an offline training phase because lengthy online exploration is

highly undesirable in Aerospace applications [59]. The challenge with offline training is that a simulation model is required a priori, which contradicts the point of learning to identify the system from scratch.

The following derivation, leading to a general incremental model and the technique to identify the dynamics online is taken from [60] and [59]. The non-linear state-space representation of a dynamical system Equation 3.57 can be linearised by taking a first order Taylor expansion of $\dot{\mathbf{x}}(t)$ around t_0, resulting in Equation 3.58 where the partial derivatives with respect to the state input vectors are called the system matrix $\mathbf{F}[\mathbf{x}(t_0), \mathbf{u}(t_0)]$ and control effectiveness matrix $\mathbf{G}[\mathbf{x}(t_0), \mathbf{u}(t_0)]$. Then, assuming a high sampling frequency, $\dot{\mathbf{x}}(t)$ can be approximated by $(\mathbf{x}_{t+1} - \mathbf{x}_t)/\Delta t$, leading to Equation 3.59 where $\mathbf{F}_{t-1}^T$ and $\mathbf{G}_{t-1}^T$ are the system transition matrix and input distribution matrix, respectively. Finally, the incremental form in Equation 3.60 is the pathway for identifying the system's dynamics. To do so, the least squares (LS) or recursive least squares (RSL) techniques can be applied by first calculating the error between the system's actual output and the output predicted by the incremental module. This is called the prediction error or *innovation* $\epsilon = \Delta\mathbf{x}_{t+1} - \Delta\hat{\mathbf{x}}_{t+1}$. Then, the covariance matrix C_{ov_t} and a forgetting factor γ_{RLS} are used to calculate $\hat{\Theta}_{t-1}$. The curious reader can find C_{ov_t} in the referenced derivation.

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{3.57}$$

$$\dot{\mathbf{x}}(t) \approx \dot{\mathbf{x}}(t_0) + \left.\frac{\partial f(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{x}(t)}\right|_{x_0, u_0} \cdot \Delta\mathbf{x}(t) + \left.\frac{\partial f(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)}\right|_{x_0, u_0} \cdot \Delta\mathbf{u}(t) \tag{3.58}$$

$$\Delta\mathbf{x}_{t+1} \approx \begin{bmatrix} \Delta\mathbf{x}_t^T & \Delta\mathbf{u}_t^T \end{bmatrix} \cdot \begin{bmatrix} \mathbf{F}_{t-1}^T \\ \mathbf{G}_{t-1}^T \end{bmatrix} \cdot \Delta t \tag{3.59}$$

$$\Delta\hat{\mathbf{x}}_{t+1} = \mathbf{X}_t^T \cdot \hat{\Theta}_{t-1} \cdot \Delta t \tag{3.60}$$

This general procedure can be applied to any of the ACDs, turning HDP, DHP and GDHP into the model-free methods iHDP, iDHP and iGDHP. Although, the incremental approach cannot be applied to the action-dependent ADP methods. iHDP was used in [61] to create an adaptive controller for a satellite model whose internal states are partially observable, resulting in significantly faster system identification than ANN-based methods. Then, [62] applied iDHP to test its ability to track a reference angle of attack and compared its performance with DHP. According to the results, iDHP learnt the dynamics much faster and achieved fault tolerance while DHP diverged. Finally, the results in [59], in which iGDHP was applied to a simulation of an F-16 Fighting Falcon, show that it can deal with various initial states and performs better than its non-incremental counterpart. It was however sensitive to disturbance and sensor noise because they corrupted the online-learnt model. This demonstrates the challenge in achieving a reactive and fast-to-learn controller that is also robust. One of the obstacles for using adaptive controllers for flight control, and especially in manned aircraft, is that the controller cannot be verified and validated in advance because its performance depends on various instantaneous, stochastic factors.

3.5.2. Sate-of-the-Art ADP Applications for Flight Control

Flight Testing Reinforcement Learning based Online Adaptive Flight Control Laws on CS-25 Class Aircraft | R. Konatala et al [9]

Traditional offline flight controllers are designed using gain scheduling- a method in which the non-linear dynamics are linearised around several operating points and a set of control gains is tuned to achieve satisfactory control at each point. However, gain-scheduled controllers struggle in situations where the dynamics can no longer be approximated by the linearised dynamics, such as during a fault. To overcome this limitation and develop a more robust, fault-tolerant controller, R. Konatala et al develop an incremental Approximate Dynamic Programming (iADP) controller. iADP is a model-free adaptive control method which identifies an incremental model online and uses Reinforcement Learning to optimise a quadratic cost function. In the control architecture in Figure 3.9, two decoupled iADP controllers track reference pitch and roll rates by computing an incremental control input. The control increment is then added to the current state of the controls and passed onto the plant. Because attitude angles are not controlled, over time, they are prone to drifting.

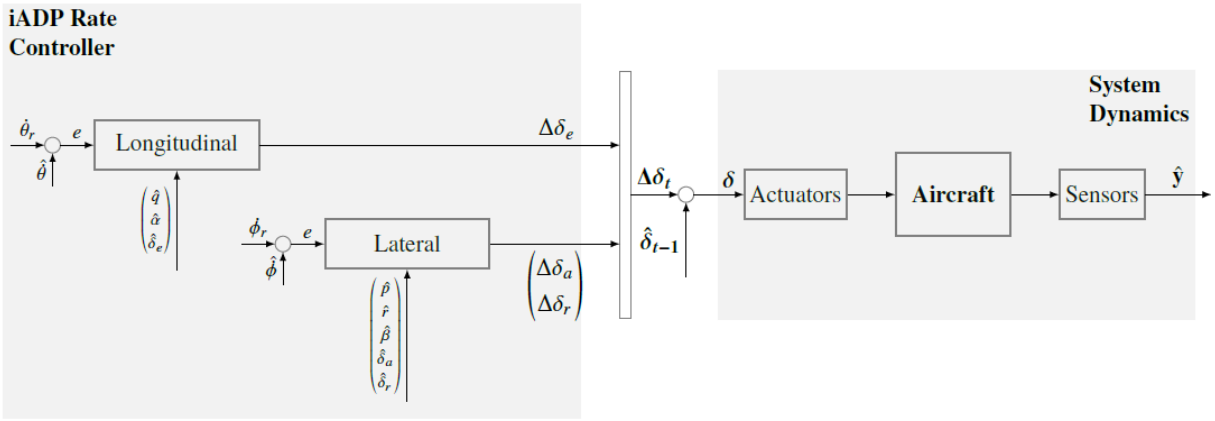


Figure 3.9: Decoupled control architecture of an iADP control law for tracking reference attitude rates. Taken from [9]

The process through which each controller learns its respective axis and outputs a control increment is illustrated in Figure 3.10. During model learning, RLS is used to improve the dynamics prediction. Then, in training, the value function is evaluated using the incremental model and one-step cost. Finally, in the third phase, the policy is improved and an incremental action is taken.

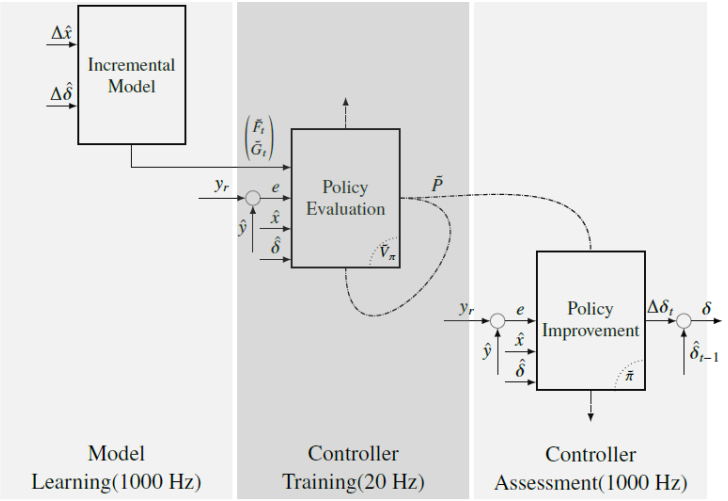


Figure 3.10: Structure of the iADP agent. Taken from [9]

After a verification and validation process which included testing the controller with both software-in-the-loop (SIL) and hardware-in-the-loop (HIL), flight tests were conducted on PH-LAB, TU Delft's Cessna Citation II experimental aircraft. The flight test's objectives were to demonstrate the minimisation of tracking errors, low sensitivity to varying flight conditions, reproducibility, adaptability and flexibility when facing different flight configurations.

When it comes to tracking performance, results are not unequivocal. The controller has demonstrated its ability to minimise the tracking error in both the stable longitudinal and unstable lateral motions, although when performing sequential learning, the unstable spiral mode restricted the time that could be allocated to model identification due to drift in the lateral direction. Moreover, The controller couldn't handle large deviations in operating conditions. The conclusion regarding sensitivity is that there's a trade-off between sensitivity and curiosity. On the one hand, the controller has to explore in order to identify the system properly. On the other hand, it must maintain stable flight and reject stochastic disturbances. So a balance must be struck to between these contradictory requirements. Testing how iADP handles non-linearities proved to be challenging because the PH-LAB behaves rather linearly in most of its flight envelope.

3.6. Challenges and Limitations of Reinforcement Learning

3.6.1. Curse of Dimensionality

The required computational power and memory grow exponentially as the number of variables in the state and action spaces increases [24], [14]. The curse of dimensionality is prominent in tabular methods when used to solve continuous-time RL problems where the approximation of the value function requires discretisation of the state and action spaces into sufficiently small elements of size C^{disc} resulting in k elements according to Equation 3.63. The problem is not only a function of the number of variables n in the state and action spaces but also of their value range. If, for control purposes, a certain discretisation resolution is needed to achieve adequate performance, then the number of elements per state or action variable k would increase proportionally to the variable's range Equation 3.63 and discretisation resolution, and then exponentially with the state space dimension n as shown in Equation 3.61 and Equation 3.62.

$$|\mathcal{S}| = \prod_{i=1}^{n_s} k_i \tag{3.61}$$

$$|\mathcal{A}| = \prod_{i=1}^{n_a} k_i \tag{3.62}$$

$$k_i = \frac{|s_{i,max} - s_{i,min}|}{C_i^{disc}} \tag{3.63}$$

3.6.2. The Deadly Triad

The deadly triad refers to the combination of three common elements in RL methods, which causes instabilities and divergence [14]. These are:

1. **Bootstrapping**: Using estimated values in update rules.
2. **Function approximation**: Approximation of policies and or value functions when solving an MDP in continuous state and action spaces. Often performed using an ANN.
3. **Off-policy training** Rather than using the learnt policy to select actions, actions are being selected by a behavioural policy while a target policy is learnt separately.

Of the three, function approximation is the most difficult element to give up on. Continuous spaces are a characteristic of many real-life RL applications, making function approximation and ANN in particular an essential component. Bootstrapping can be substituted with MC, but it would come with a substantial penalty to learning performance. Lastly, the choice between off-policy and on-policy is not obvious and depends on the specific problem. So, having to choose one or the other is not as limiting as having to give up on function approximation, but it can lead to a solution that is far from ideal.

Studies about the deadly triad found several approaches and tricks to improve convergence. These include modifying the update rule of a target network to use Polyak-averaging [63], bootstrapping on a separate network and reducing overestimation bias [64], entropy regularisation [50], and more [32]. So, while the deadly triad is an important phenomenon to consider, ample methods have been developed to improve convergence behaviour.

3.6.3. Sample Efficiency

Sample efficiency is a measure of the agent's ability to learn from each interaction with the environment in order to maximise the reward, be it on a time-step basis or episodic. This is an important performance metric because getting samples in the real world is prohibitively expensive and time-consuming, so maximising sample efficiency is essential. It is also a useful metric for performance comparison of different RL algorithms. Sample efficiency can be measured in two ways: if there is a performance threshold to reach, then it can be represented by the number of iterations it takes, where fewer samples mean higher efficiency. Otherwise, it can be represented by the area under the *learning curve* within a bounded time range in a return-per-episode plot. The greater the area, the higher the sample efficiency [65]. Often, it is evaluated qualitatively by comparing several overlapped learning curves rather than numerically. Despite general progress in sample efficiency over the years, comparing results remains a challenge due to the dependency of many algorithms on hyperparameter tuning and the specific application in which they are implemented. Even comparison using common benchmarks [43] is not trivial because performance in one type of task may not be indicative of the performance in other types of tasks.

3.6.4. Simulation-Reality Gap

The simulation-reality gap, or sim-to-real, is a term used to describe various challenges relating to the implementation of offline synthesised controllers in real systems. Sim-to-real emerges from the model dependence of offline control design processes. Since a model of a system is never identical to it in its dynamics (otherwise it wouldn't have been a model, but the real system), there exists a gap between the behaviour of the model and that of the real plant when subjected to the same input. Another facet of sim-to-real is related to the assumptions made during the design phase. For example, previous work on RL controllers for the PH-LAB [18] neglected sensor dynamics. While this assumption is valid when testing the feasibility of novel controllers, it restricts applicability in reality. So it is essential that assumptions align with reality and that the controller is designed with the constraints of the actual system in mind. Figure 3.11 illustrates sim-to-real by representing the variation in tasks and dynamics between training and testing as separate regions in the state and action spaces.

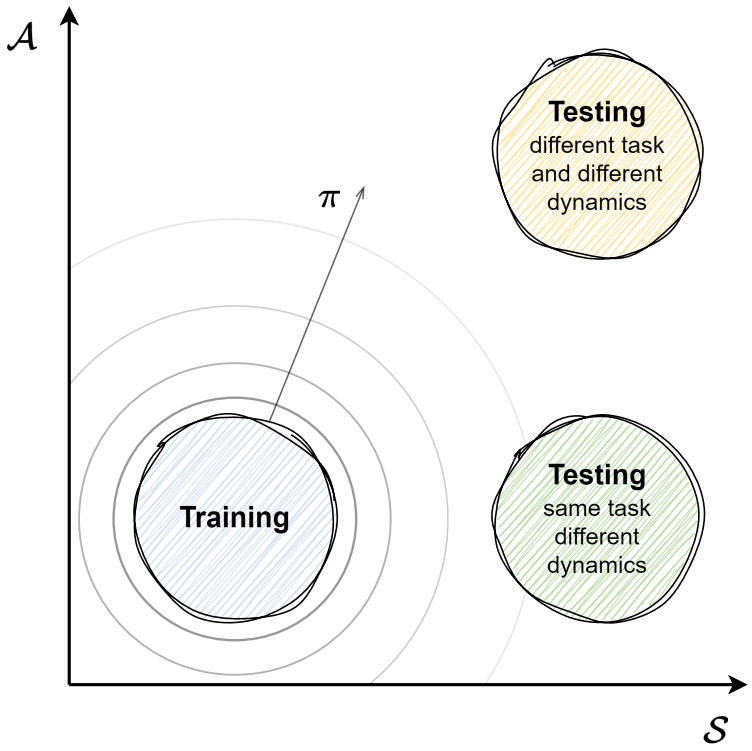


Figure 3.11: Illustration of the simulation reality gap and how a policy should generalise to capture the true dynamics and achieve robustness

Simulation Optimisation Bias

The Simulation Optimisation Bias (SOB) describes a situation where an agent trained in simulation learns behaviours that apply exclusively to the simulated environment, causing a degradation in performance or even damage to the real system once it is rolled out [66]. For instance, if the simulator does not model control surface travel limits, the agent might learn to deflect them to large angles that would cause physical damage in reality. Another example to demonstrate the agent's simulation-exploitation tendencies is how it might be sensitive to the centre-of-gravity (CoG) location because, in simulation, it is assumed to be constant, while in reality, it changes during flight. These two examples demonstrate why SOB makes it challenging to transfer control policies from simulation directly to the real system, requiring several techniques to overcome the agent's tendency to over-fit to simulation. In Section 3.6.5, common methods used to tackle the SOB and other sources for the sim-to-real gap are presented.

Partially Observable MDP
The MDP, as introduced in Section 3.1.1, assumes the Markov property, which says that the current state carries all required information and fully represents the system's actual state. In reality, however, the state is often not fully observable. Phenomena such as aeroelasticity and hysteresis mean that the information given by various sensors doesn't tell the whole story regarding the system's response and its dependence on historical states and actions. For example, at time t the ailerons may be deflected an amount $\Delta\delta_a$, causing a change in the state such that $s_{t+1} = [\phi + \Delta\phi,\ p + \Delta p,\ \delta_a + \Delta\delta_a]$. However, aerodynamic reaction forces cause the wings to heave, which isn't represented in the state, even though it has an effect on the plant's dynamics in future time steps, thus the next state is not known with complete certainty. This example illustrates why the problem of controlling an aircraft can only be assumed to be a partially observable MDP (POMDP) [67]. Formally, the MDP tuple $[s,\ a,\ Pr,\ r]$ can be transformed into the POMDP tuple $[s,\ a,\ Pr,\ r,\ \Omega,\ O]$ where Ω is a set of observations, and O is the probability of experiencing a certain observation in Ω [68]. A POMDP can be formulated as an MDP by using a ***belief state*** b, which represents the probability of reaching a certain next state s' having taken action a. Alternately, the agent can implicitly learn about the internal dynamics by feeding it a series of past states instead of only the most recent one.

generalisation
Generalisation is an adjacent concept to sim-to-real, and is about the ability of an agent to act optimally at deployment when encountering situations it hadn't seen during training. Since real environments are dynamic and uncertain, the agent's ability to ignore variables and effects irrelevant to the task is crucial to its success. Methods such as domain adaptation and meta-learning rely on using data from the real environment to pre-train the agent so it can transfer from simulation to reality. In contrast, zero-shot generalisation (ZSG) aims to produce a policy that doesn't overfit and can thus generalise to the real environment zero-shot; so without any additional training or knowledge of the environment [69]. The challenge of generalisation spans several subproblems. First is the policy's ability to perform the same task whilst controlling a system with different dynamical behaviour than that of the simulated environment. Second, it extends the agent's utility by having it perform different tasks than those performed in training. As shown in Figure 3.11, generalisation is akin to a radial expansion in space, such that regions of experience that fall outside of the simulated environment are included in the policy.

Quantifying generalisation
The policy's ability to cope with dynamical differences between sim and real when executing the same task can be quantified by the difference in returns according to Equation 3.64 [69]. This metric is particularly useful for algorithm comparison. A similar example to the one given by R. Kirk [69] is, if algorithm A and B perform similarly in testing, but A has a larger generalisation gap, (suggesting it performed better than B in training), then it may be wiser to choose algorithm B because it proved to be more robust than A. Of course, the gap itself says nothing about absolute performance, so further indicators are required.

$$GenGap(\pi) \doteq G(\pi \mid \mathcal{S}_{sim}) - G(\pi \mid \mathcal{S}_{real}) \tag{3.64}$$

Another indicator for generalisation can be measured in game-based benchmarks, where the agent is trained on a training set of levels and then tested on another. Consequently, the fraction of completed test levels is indicative of its ability to generalise between tasks [70], [71], [72]. The limitation of this approach is that there have to be levels and a completion signal, which are both discrete in nature. This doesn't translate well for use in a FCS where tasks are continuous and have no binary measure of success. Lastly, generalisation can be estimated indirectly by measuring how much the agent explored the environment [73].

3.6.5. Solutions to the Simulation Reality Gap

Several techniques are recognised for their effectiveness in reducing the gap between the dynamics in simulation and in reality. These are domain adaptation, domain randomisation and online system identification, as elaborated below. In Section 3.7.2, assumptions are made based on a synopsis of the literature study. These are meant to facilitate the design phase such that the end product, an RL controller, is reality-oriented.

Domain Adaptation
The most obvious way to minimise the sim-to-real gap is to improve the simulation model. The simulation's accuracy can be improved by expanding its scope and adding models of phenomena that were previously neglected. Then, precision can be improved by adding higher-order terms, leading to higher confidence in the predictions. Of course, this is easier said than done. In practice, the development and computational cost make it prohibitively expensive. Another method to bring simulation closer to reality is to use domain adaptation (DA), a method in which invariant features are learnt in a source domain and injected into a target domain [20]. For example, the values of various aerodynamic coefficients can be learnt from flight data and then used in simulation. In [74], DA is used to transfer a learnt policy between two quadcopters with different dynamical properties.

Domain Randomisation
As elegantly explained by X.B. Peng, "With domain randomisation, discrepancies between the source and target domains are modelled as variability in the source domain" [20]. Domain randomisation (DR) can be applied in several ways. In robotic applications with vision-based state estimation, DR has been applied to the appearance of the scene. In [21], the lighting, texture, colours, camera position and perspective change every episode. These variations make the policy insensitive to task-independent factors in the environment. A similar approach was used in [22] for drone racing, resulting in a policy that can control the drone while facing varying lighting conditions and visual disturbances. DR can also be applied to the dynamical properties of the simulation model. [20] used DR while training a policy for a robotic manipulator whose task is to push an object towards a target zone. DR was applied at each episode by varying the masses of the robot's links as well as the friction and damping coefficients for both the robot's joints and of the manipulated object. The results demonstrate that DR is a powerful method to train policies that are invariant to environmental and dynamical disturbances.

The choice of which coefficients to vary and by how much is highly correlated with the policy's performance. [75] found that using variance that is too high can lead to unrealistic behaviours and break the policy. On the other hand, to successfully capture the dynamics of the real system the variance should not be too small.

Online System Identification
Instead of trying to match simulation to reality, an online learning method could be employed, making the training environment redundant. Online methods can either learn a dynamical model of the environment and use that to learn a policy, or they can learn the policy directly. Methods of the first kind include the incremental derivatives of ADP as introduced in Section 3.5. Then, to directly learn a policy online, both on-policy and off-policy methods can be used. The main consideration is achieving high sample efficiency to drive down the training time, and the costs and risks associated with exploring online with the real system. In [8], a quadruped robot was able to learn how to walk within just 20 minutes of training using a custom derivative of SAC.

3.6.6. Safety in Reinforcement Learning

Reward Hacking
A fascinating phenomenon in RL is that of agents who develop unexpected behaviours which technically achieve high reward, but do not fulfil the task as intended. For example, if an agent's task is to drive a car around a racetrack and it is rewarded for crossing the finish line, it might end up learning to drive over it-first forwards, then in reverse, and repeat. This behaviour would achieve a high reward, but not align with the intent of completing laps around the track. This is recognised in literature as *reward hacking* [76]. In an offline learning environment, it is rather trivial to identify when the agent exploits the reward function and to have the opportunity to correct it without harm. However, if an agent trains online on a physical system such as a robot, a car, or an aircraft, it can have catastrophic consequences for the system's structural integrity and to people's safety.

Safe Exploration
The definition of exploration is taking non-optimal actions. Often, these lead to the termination of an episode in simulated environments, where the termination state may be non-recoverable with the intention of extending the agent's learning opportunities as much as possible. For online training, however, where non-recoverable is equivalent to causing damage and harm, safe exploration is particularly important

[76]. To reduce risk during online training, the search space has to be contained within conservative bounds. To prevent safe exploration from being at the expense of learning rate and generalisation, the (safe) termination states can be used as the initial state when resetting the environment before the next episode, such that the agent learns to recover from every position in the allocated space. This is called reset-free learning [77].

Verification & Validation (V&V) of Reinforcement Learning-based Online Adaptive Flight Control Laws on CS-25 Class Aircraft | R. Konatala et al [78]

One of the challenges in incorporating innovative controllers in manned aircraft is the strict regulations regarding safety, requiring a thorough verification and validation process. In [78], a V&V process is designed to prove the controller is safe and guarantee performance within the flight envelope it was designed for. During validation, the architecture and performance of the FCL are compared against the design requirements to ensure compliance. in the context of FCS, this requires flight testing to confirm that criteria such as ride comfort are met. Verification is about ensuring that everything works upon integration into the subsystems and then into the complete system. More concretely, it involves testing the FCL in software-in-the-loop and later hardware-in-the-loop environments to make sure it works correctly on the intended systems.

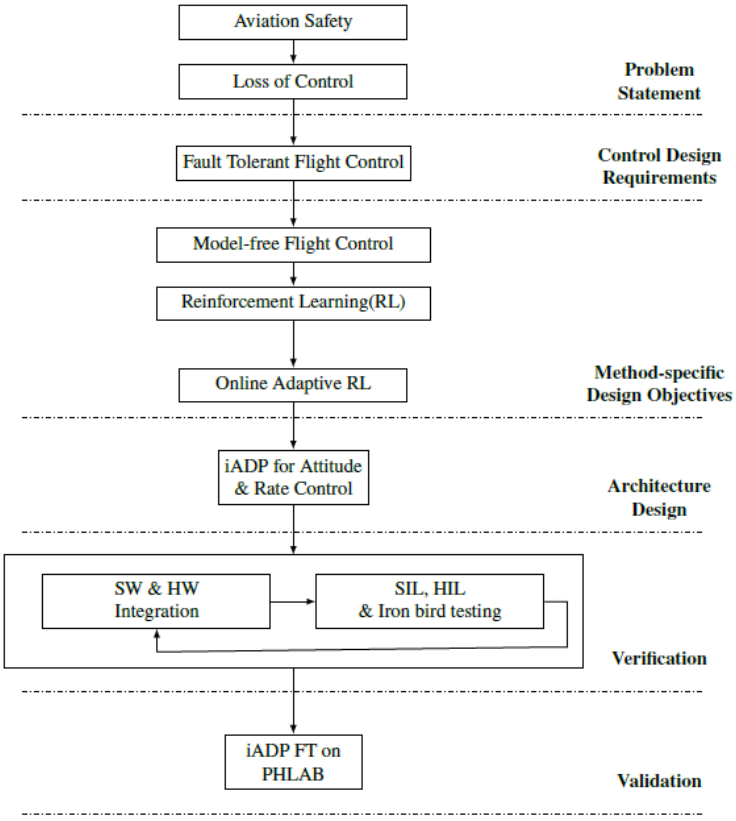


Figure 3.12: V&V framework for iADP FCS Design for Fault Tolerant Flight Control. Taken from [78]

To verify the robustness of the controller, an anti-optimisation approach is taken, where the uncertainties of several critical aerodynamic coefficients are used to create a worst-case model. Then, the hyperparameters are tuned using a Multi-Objective Parameter Synthesis to optimise them for the worst-case model with the hope that rate-tracking requirements can still be met.

In the conclusions, the author states that using the V&V framework shown in Figure 3.12 led to expedited clearance for flight-testing the iADP FCL, demonstrating the importance of machine-learning oriented V&V frameworks. These can aid in accelerating the development of RL-based FCS and lead to a future where aviation authorities such as the FAA and EASA approve their implementation and ultimately improve aviation safety.

3.7. Conclusion Literature Study

This literature study explored the reinforcement learning solution space with the goal of finding the most appropriate algorithm and accompanying tools and methods for developing a FCS which has the potential to increase performance, fault tolerance and safety. Backed by the motivation to enable future flight testing of the proposed RL FCS in the PH-LAB, the focus lied on those tools and methods that help in bridging the simulation-reality gap, not only through policy generalisation but also by considering the limitations of and compatibility with the aircraft in question and the systems installed in it.

After introducing the basic building blocks of RL in Section 3.1 and common tabular solution methods in Section 3.2, solutions that are applicable to continuous space were discussed in Section 3.3. There, in Section 3.3.1, function approximation methods were identified as the pathway for solving RL problems with continuous state and action spaces. Among those, artificial neural networks struck as the most prominent method due to their modularity and ability to capture non-linearities. In Section 3.4, the four deep RL algorithm families, DQN, DPG, TRPO and SAC, were reviewed, and two examples from literature showcasing their use for FCSs were presented. The section that followed, Section 3.5 treated approximate dynamic programming methods and illustrated their utility in online, adaptive controllers while highlighting the challenges in validating them. The last section, Section 3.6, touched on the challenges and limitations RL faces, followed by several approaches to mitigate them. Of particular relevance are the challenges concerning policy generalisation, safety in real-world applications, and the need for a V&V process to facilitate flight testing.

To aid in selecting the most appropriate method, high-level requirements are defined, followed by a set of assumptions, both of which help in navigating through the range of options.

3.7.1. Qualitative RL Controller Requirements

1. **Offline**: While online learning and system identification methods such as (i)ADP or online PPO have been shown to perform well in a range of applications, the low sample efficiency of online RL and the limited computational resources on board the aircraft limit applicability to FCS. Another factor in support of offline methods is the risk stemming from the inherent unpredictability of online algorithms.
2. **Attitude and altitude hold** To demonstrate autopilot capabilities, the control task is to track reference attitude and altitude signals set by the pilot.
3. **sim-to-real transfer**: Achieve similar tracking performance in the real system as is achieved in the training environment, despite differences in the physical properties and coefficient values between the two.
4. **Fault tolerant**: Maintain acceptable tracking performance when faced with a (partial) in-flight loss of control.
5. **Generalise to never-before-seen flight conditions** Training cannot cover all possible flight conditions, so the policy should be able to generalise to situations it hadn't encountered before. These may include atmospheric disturbances, unfamiliar manoeuvres and varying aircraft mass, and airspeed.
6. **Model free**: Stemming from the desire to design an offline, fault-tolerant controller, a model-based method is unsuitable because if it cannot adapt its model online, it would not be fault-tolerant.
7. **Continuous actions and states**: The chosen framework has to be compatible with continuous state and action spaces.
8. **Run time performance**: Computational power onboard the aircraft is limited, hence the controller must be designed to work on existing hardware with sufficient margins.
9. **Smooth control surface deflections**: To prevent passenger discomfort, excessive structural loads and accelerated servo wear, the magnitude and frequency of control surface deflections should be restricted. Passenger comfort can be gauged by the load factor n while frantic servo actuation can be quantified by the magnitude or rate of the of incremental deflections. By including these measures in the reward function, the policy can be trained to restrain its control outputs.

3.7.2. Assumptions

Imperfect Sensors
In contrast to a simulated environment in which information about the state is deterministic and instant, in reality, the aircraft's sensors have a finite resolution, sample rate, noise, bias and delays. The effect of these phenomena is that the state following an action is stochastic. To some degree, these can be modelled for use during the training phase. Despite this, the sensors are assumed to be in operational condition and report representative data for state estimation. While sensor failure is a realistic scenario, and it is interesting to investigate how to deal with a majorly obscured state estimation, it falls outside the scope of this research.

Imperfect Actuators
The Servos are also imperfect in that actuation isn't instant; there exist transport delays. They are also not infinitely powerful, and at high speeds, it can become a limiting factor with respect to the maximum allowable deflection angle. Lastly, the control chain between the servo and the control surface has slop and is not infinitely rigid. All of these factors lead to uncertainty in both the application of the chosen action and the following state.

Partially Observable MDP
As seen in Section 3.6.4, the real aircraft cannot be assumed to adhere to the MDP model because not all phenomena are modelled, and state history *does* influence the system's response in future states. The consequence of this assumption on the solution is that instead of using the state as given by the environment, it can be made into a belief state by adding stochasticity through a probability distribution with mean given by a sensor's direct measurement and variance according to the uncertainty of each sensor based on data found in [79]. Moreover, a buffer of historical states $\mathcal{D}$ can be used, as is done in many of the DRL algorithms presented earlier in this chapter.

Independent Controls
In the routine operation of fly-by-wire aircraft, the control over the control surfaces is often coupled by means of pitch and yaw dampers. These are meant to automatically compensate for undesired changes in the angle of attack and side slip, respectively. Such changes may be caused by external disturbances like gusts and turbulence, but only when a single degree of freedom is actuated during coupled motions. A coordinated turn, for example, requires the actuation of both the ailerons and rudder, but with a yaw damper, it is sufficient for the pilot to only actuate the ailerons because the rudder would automatically move to maintain a zero slide slip angle. While control surface dampers can improve stability and reduce pilot fatigue when flying the aircraft manually, they 'hide' a layer of control, thus contradicting the goal of training an RL agent to *fully* control the aircraft.

Separate Throttle Control
Since PH-LAB doesn't have an auto-throttle, this degree of freedom is excluded from the RL controller. Instead, it will be controlled manually by the pilot who will try to track a desired throttle setting given by an independent controller.

3.7.3. Reinforcement Learning Framework
Based on the requirements that the RL framework should be model-free and compatible with continuous spaces, attention can be drawn on the left-most branch in the RL framework classification tree Figure 3.2. Between the on-policy methods, PPO has the upper hand in terms of performance and simplicity while amongst the off-policy methods, SAC appears to be the best candidate thanks to its highly exploratory nature. The requirements of a general and robust policy suggest that exploration is highly desired and is an inherent quality of off-policy methods.

Before making a decision based purely on technical notions, it is also worth considering the context of this research and where it lies in the overall field. Previous work has already explored adaptive methods both in simulation and in-flight testing [9], [12], [13], [18]. PPO has also been explored extensively in simulation and on fixed-wing UAVs in flight [51] and is the most common algorithm of choice in similar applications [80]. SAC in comparison has not been used as much despite the fact that the papers in which it is utilised indicate that it achieves high performance both in simulation and in deployment in UAVs [10],

[81], [82]. The authors in those references cite the exploratory behaviour of SAC and its ability to deal with POMDP as their motivation for choosing it. Unfortunately, there exist no benchmarks that allow direct comparison between different papers, and no research has been done to directly compare SAC and PPO in the context of flight control. Since the choice of algorithm and its performance are highly dependent on the problem and the implementation, comparisons in other contexts, such as in [83], [84] and [85], are of limited relevance. With that in mind, the fact that SAC has been used in simulation to create a FCS for the PH-LAB is a significant contribution that serves as a stepping stone towards flight testing a SAC-based controller.

Considering the above, SAC appears to be a promising algorithm that can answer all the requirements while adhering to the assumptions. On top of that, this research avenue has not been exhausted, making it an interesting subject while taking advantage of the few papers that do treat it, namely [10] offers a good starting point. To conclude, SAC is chosen as the algorithm to be used to develop a FCS for the PH-LAB.

4

Preliminary Analysis

This chapter is a continuation of Chapter 3 and aims to realise Research Objective 3 through the development of a pitch rate RL-based controller for a linear model of the Cessna Citation and compare its performance with a baseline controller, as done in Section 4.1 and Section 4.2. Following that, in Section 4.3, Section 4.4, and Section 4.5 are a series of experiments meant to answer Research Question 3.1 and 3.2.

4.1. Simulation Environment

To simplify the learning problem and accelerate experience accumulation, the non-linear, full DoF Simulink model of the Cessna Citation is linearised around a nominal trim point as specified in Table 4.1 and reduced to a linear time-invariant (LTI), short-period model. Linearisation returns the first-order coefficients of the state-space representation of the system in Equation 4.1. These are z_α, z_q, m_α, m_q for the state matrix A, and z_{δ_e} and $m_{\delta e}$ for the input matrix B. Their values at the chosen trim conditions are specified in Table 4.1. The expanded terms and values can be found in Table A.1 and Table A.2, which are taken from [86]. To simulate how the system's state develops over time, the state space representation in Equation 4.2 is integrated with the Euler method to obtain the new state from the current state and action as shown in Equation 4.3. The state-space bounds are set to $\pm 20 \cdot \frac{\pi}{180}$ $[rad], [rad/s]$ for α and q respectively. And, while the trimmed angle of attack is in principle non-zero, the environment resets to an initial state where $\alpha = 0$ $[rad]$ and $q = 0$ $[rad/s]$ (hence without exploring starts).

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} z_\alpha & z_q \\ m_\alpha & m_q \end{bmatrix} \begin{bmatrix} \alpha \\ q \end{bmatrix} + \begin{bmatrix} z_{\delta_e} \\ m_{\delta_e} \end{bmatrix} \delta_e \tag{4.1}$$

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{4.2}$$

$$\mathbf{x}(t+1) = (A \cdot \Delta t + I)\mathbf{x}(t)^T + B \cdot \Delta t \cdot \delta_e \tag{4.3}$$

Table 4.1: Nominal trim conditions used for linearisation of the non-linear dynamics Citation model.

Trim point	V_{TAS} $\left[\frac{m}{s}\right]$	m $[kg]$	h $[m]$	p_d $[Pa]$
Nominal $(p_{d,0})$	90	4500	2000	4076.286

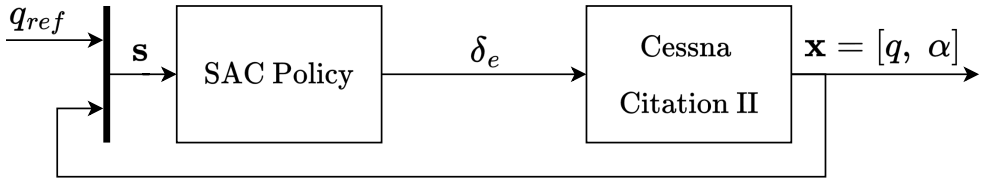
$$A_0 = \begin{bmatrix} -1.4175 & 9.6924 \cdot 10^{-1} \\ -8.1181 & -2.0298 \end{bmatrix} \tag{4.4}$$

$$B_0 = \begin{bmatrix} -1.4589 \cdot 10^{-1} \\ -1.2662 \cdot 10^{1} \end{bmatrix} \tag{4.5}$$

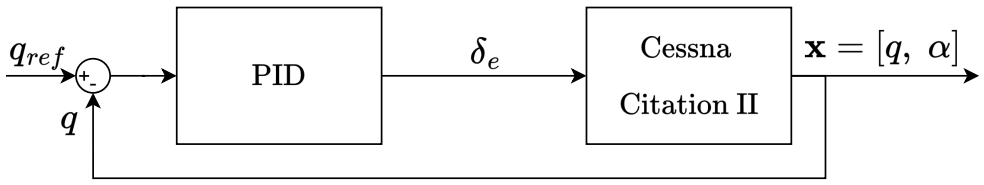
While using a linear system simplifies the learning problem and is computationally efficient, it is essential to be aware of its limitations. For one, the dynamics are only valid around the trimmed condition, and certainly not at the upper and lowermost bounds. This is a compromise meant to allow the agent to have the opportunity to explore the environment without immediately exceeding the state bounds and terminating the episode. Beyond that, several assumptions were made to simplify the problem further, such as that the system's state is assumed to be fully known and deterministic; so actuator and sensor dynamics are excluded from the environment's model. Furthermore, external disturbances such as turbulences are not modelled so their effect on the system's dynamics is not considered. The only actuator characteristic that is implemented is the elevator's saturation limit of $\pm 0.5\ [rad]$, or approximately $\pm 28.65°$.

4.2. SAC Implementation

Having concluded in the Literature Study that Soft Actor-Critic is the most promising algorithm for controlling the Cessna Citation, a custom implementation is developed in MATLAB in consultation with various sources [19], [87], [88], [10], [89]. While the focus in this part of the report is on creating a pitch-rate controller as shown in Figure 4.1a, the code is written according to the Object Oriented Programming paradigm so that the end product is flexible enough to be used with a variety of environments. Specifically, with the full Dof, non-linear, Simulink Citation model. Figure 4.1b shows the equivalent pitch-rate control loop for a PID controller used in the following sections as a baseline for performance comparison.



(a) Pitch-rate control loop designed for the SAC agent.



(b) Pitch-rate control loop with a traditional PID linear controller.

Figure 4.1: Controller design for both types of controllers- a trained DNN and a PID linear controller.

4.2.1. SAC Architecture

Figure 4.2 visualises the flow of information and the role of the different components that make up the SAC algorithm. The actor block encapsulates the DNN representing the policy and several functions to select an action based on its mean μ and standard deviation σ and to bound it with the hyperbolic tangent function. Then, in the experience accumulation block, the action a_t is scaled before being passed to the environment, which in turn outputs the reward r_t, termination signal d_t and the next state s_{t+1}. The new state is then normalised before being fed to the policy for further agent-environment interactions. The experience collected at each interaction, $[s,\ a,\ r,\ s',\ d]$ is saved in the in the experience buffer $\mathcal{D}$.

When it is time to train the networks, a minibatch $\mathcal{M}$ containing $\mathcal{B}$ unique, randomly chosen experiences is sampled from the buffer and used to calculate the loss functions and their gradients with respect to the learnables of each network or parameter. Once those are calculated, a gradient step is performed using the Adam optimiser [27], thus completing a single learning iteration.

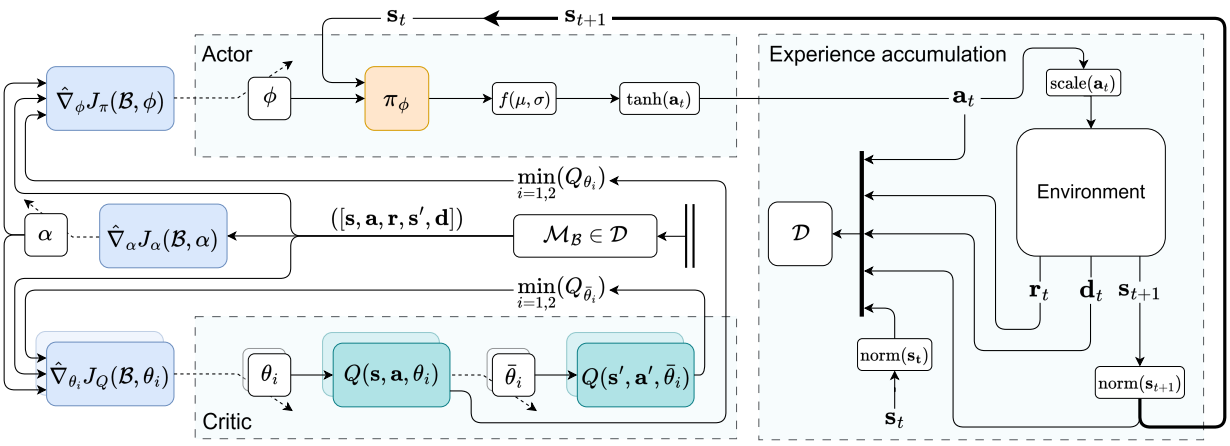
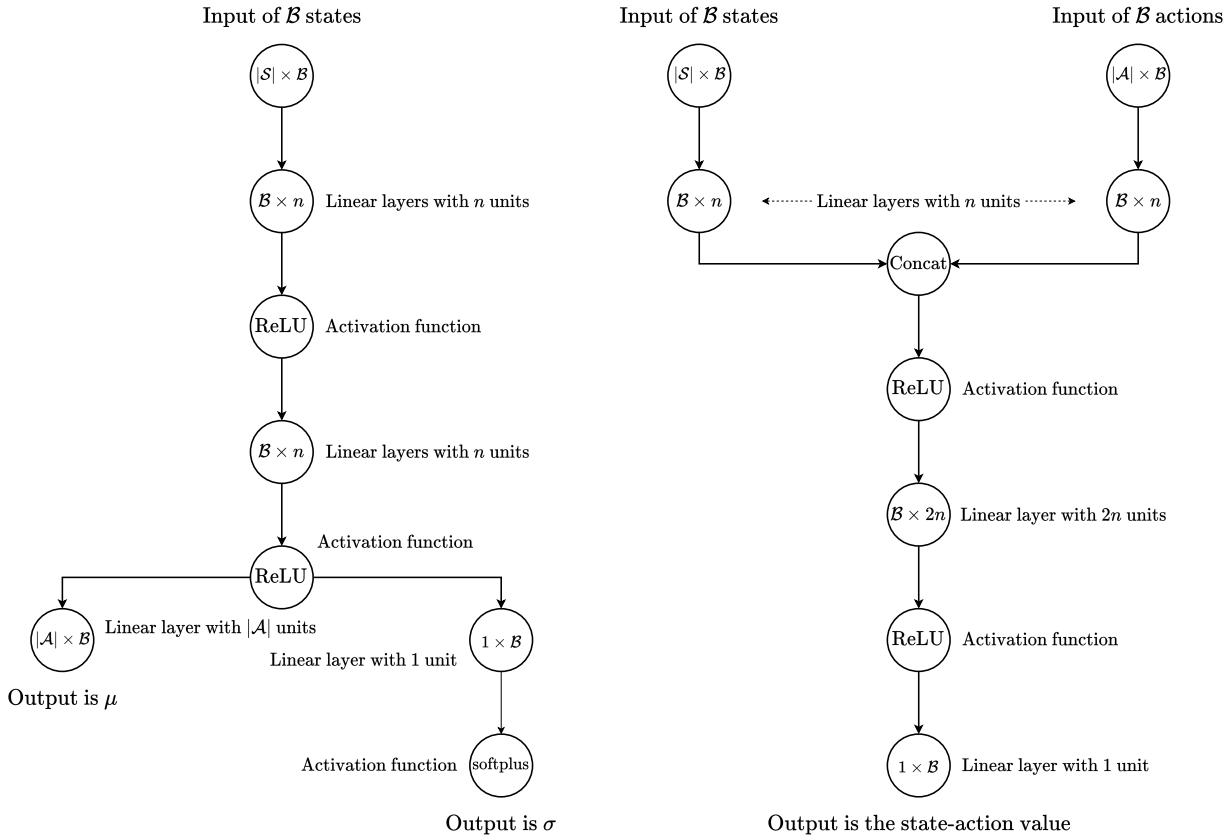


Figure 4.2: Architecture of the custom SAC implementation.

4.2.2. DNN Architecture

Learning is facilitated by deep neural networks (DNN) that approximate the policy and Q-functions as depicted in Figure 4.3a and Figure 4.3b, respectively. The policy's net takes either a minibatch or a single state vector as input, depending on whether the net is used for interaction with the environment or for loss calculation. After feeding through two linear layers and their respective activation functions, the data is split into two branches. The first learns to approximate the mean μ of the actions, while the other branch learns to output the natural logarithm of the standard deviation σ. With these two outputs, an action is sampled (stochastically) from a normal distribution. However, when deploying the trained network, actions are sampled deterministically, thus $a = \mu$. The DNNs of the double Q-functions and their targets are more straightforward- they take in minibatches of state and action vectors, concatenate, and pass them on through two more layers. The output is unbounded and represents the value of the state-action pair. The architecture in Figure 4.3 is similar to the structure used in MATLAB's RL toolbox (RLTb) SAC agent [89] but differs in the number of hidden units.



(a) Policy net. **(b)** Q function net.

Figure 4.3: Architecture of the DNNs of the actor and critic.

4.2.3. Training Algorithm

The algorithm for collecting agent-environment interactions and for optimising the policy, Q-functions and temperature coefficient is presented in algorithm 1. In contrast to other implementations, such as in the RLTb, updates to the actor and critics occur in regular intervals and are independent of the episode's length. Thus, if an episode is terminated before the maximal number of steps has been reached, it will not trigger a series of updates but rather reset the environment and continue collecting experiences. The consideration for this logic was to increase learning consistency and reduce bias between the number of new experiences being collected and the occurrence of actor-critic updates.

Algorithm 1: Soft Actor-Critic. Adapted from [19].

Initialize: Policy DNN with weights ϕ, Q-function DNNs with weights θ_1, θ_2 and target DNNs with weights $\bar{\theta}_1 = \theta_1$, $\bar{\theta}_2 = \theta_2$

Hyperparameters: learning rates λ_ϕ, λ_θ, λ_α, Polyak update factor τ, buffer size $\mathcal{D}$, minibatch size $\mathcal{B}$, discount factor γ, target entropy $\bar{\mathcal{H}}$, and number of epochs $\mathcal{E}$

Get the initial state s_0 from the environment

for $t = 1 :$ *number of time steps* **do**

Normalise the state s_t

Sample action a_t from policy $\pi_\phi(s_t^{norm})$

Scale a_t according to the system's limits

Execute a_t^{scale} in the environment and get a new state s_{t+1}, reward r_t, and termination signal d

Store the experience $\left[s_t^{norm},\ a_t,\ r_t,\ s_{t+1}^{norm},\ d\right]$ in the experience buffer

if *Time to update* **then**

for *epoch* $= 1 : \mathcal{E}$ **do**

for *iteration* $= 1 : \mathcal{D}/\mathcal{B}$ **do**

Sample a minibatch $\mathcal{M}$ with $\mathcal{B}$ randomly chosen experiences from the buffer

Compute the loss of the log of the temperature coefficient α:

$$J_\alpha = -\frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \left[\ln \alpha \cdot \left(\ln \pi_\phi(s_i) + \bar{\mathcal{H}}\right)\right]$$

Compute the Critic loss:

$$J_Q = \sum_{j=1}^{2} \left[\frac{1}{2\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \left[Q_{\theta_j}(a_i, s_i) - y_i\right]^2\right]$$

Where

$$y_i = r_i + \gamma \cdot (1 - d_i) \cdot \left(\min_{j=1,2}(Q_{\bar{\theta}_j}(a_i', s_i')) - \alpha \cdot \ln \pi_\phi(s_i')\right)$$

And

$$a_i' = \pi_\phi(s_i')$$

Compute the Actor loss:

$$J_\pi = \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \left[\alpha \cdot \ln \pi_\phi(s_i) - \min_{j=1,2}(Q_{\theta_j}(a_i, s_i))\right]$$

Perform a gradient step and update the weights:

$$\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J_\alpha$$

$$\theta_{1,2} \leftarrow \theta_{1,2} - \lambda_\theta \hat{\nabla}_{\theta_{1,2}} J_Q$$

$$\bar{\theta}_{1,2} \leftarrow (1 - \tau) \cdot \bar{\theta}_{1,2} + \tau \cdot \theta_{1,2}$$

$$\phi \leftarrow \phi - \lambda_\phi \hat{\nabla}_\phi J_\pi$$

end

Reset the buffer's *is sampled* flag to allow old experiences to be used again in the next epoch

end

end

end

4.2.4. Hyperparameters

Hyperparameters can make or break a reinforcement learning algorithm and often need to be tuned for the application. The initial values of the hyperparameters detailed in algorithm 1 were based on a mix between the findings of K. Dally [10] and the default values in MATLAB's RL Toolbox [89]. Then, based on a series of tests, the hyperparameters were tuned to stabilise learning, increase learning rate and convergence, and reduce training time. Ideally, an optimisation routine such as a Parzen estimator [90] is employed to find the hyperparameter values that maximise the objectives mentioned. However, this is left outside the scope of the research. Fortunately, the learning performance achieved with the manually tuned hyperparameters is satisfactory and enables the agent to learn the task successfully, as shown in the following sections. Table 4.2 shows the values of the hyperparameters used with the custom SAC algorithm.

Table 4.2: Hyperparameters used in the SAC in algorithm 1.

Parameter	Value	Description
$\lambda_\phi, \lambda_\theta$	$1 \cdot 10^{-2}$	DNNs Learning rate
λ_α	$3 \cdot 10^{-4}$	Temperature coefficient learning rate
$\lvert\phi\rvert, \lvert\theta_{1,2}\rvert, \lvert\bar{\theta}_{1,2}\rvert$	32	Number of hidden units in each layer
τ	$5 \cdot 10^{-3}$	Polyak smoothing factor
$\mathcal{D}$	$1 \cdot 10^{4}$	Experience buffer size
$\mathcal{B}$	64	Minibatch size
γ	0.99	Discount factor
$\bar{\mathcal{H}}$	$-\lvert\mathcal{A}\rvert$	Target entropy
$\mathcal{E}$	2	Number of epochs

4.3. Experiment I: Reward Functions

4.3.1. Objective and Motivation

The objective of experiment I is to find a reward function that maximises performance in terms of learning efficiency, tracking performance and control smoothness. While the last requirement might be trivial for traditional controllers, it is not uncommon for RL agents to learn a policy that outputs high-frequency actions or acts like a bang-bang-type controller [91]. Such behaviour might work in simulation but cause damage to the real system, as was explained in Section 3.6.4.

With these requirements in mind, four reward functions were chosen. Their mathematical definition can be found in Table 4.3 where $e = s_t^{ref} - s_t$. For the pitch rate tracking task dealt with here, $e = q_t^{ref} - q_t$. The motivation for the chosen reward functions is as follows:

- **RF1**: Being a quadratic function, the reward's value becomes less and less sensitive to the magnitude of the error as it approaches zero. This suggests that as the error gets smaller, it will have less influence on the value function, resulting in a 'lazy' policy. On the one hand, it might lead to worse tracking performance due to the policy's indifference to small errors. On the other hand, the policy might be smoother and output less erratic elevator deflections, especially during deployment at flight conditions with high dynamic pressure.
- **RF2**: To increase the reward's sensitivity to the error, it is multiplied by a gain to increase its magnitude. Although, near zero, the function approaches the value of $RF1$.
- **RF3**: Because it is a linear function, its sensitivity to the error is constant. Thus, it has the potential to achieve better nMAE performance than the former functions. The risk is that the elevator deflections would be of high frequency and amplitude, which is undesired.
- **RF4**: To prevent high frequency actuations, $\Delta\delta_e$ is included as a form of penalty. The challenge is in determining the ideal scaling factor weighing the error and the control increment. After some experiments, the scaling factor was chosen to have the value of 0.1. Larger values cause the policy to become indifferent to the error because it was cheaper for the agent to simply not move the elevator at all. An identical value was chosen in [13].

Table 4.3: The chosen reward functions.

Designation	Reward function
$RF1$	$-e^2$
$RF2$	$-\frac{180}{\pi} \cdot e^2$
$RF3$	$-\lvert e \rvert$
$RF4$	$-\lvert e \rvert - 0.1 \cdot \lvert \Delta\delta_e \rvert$

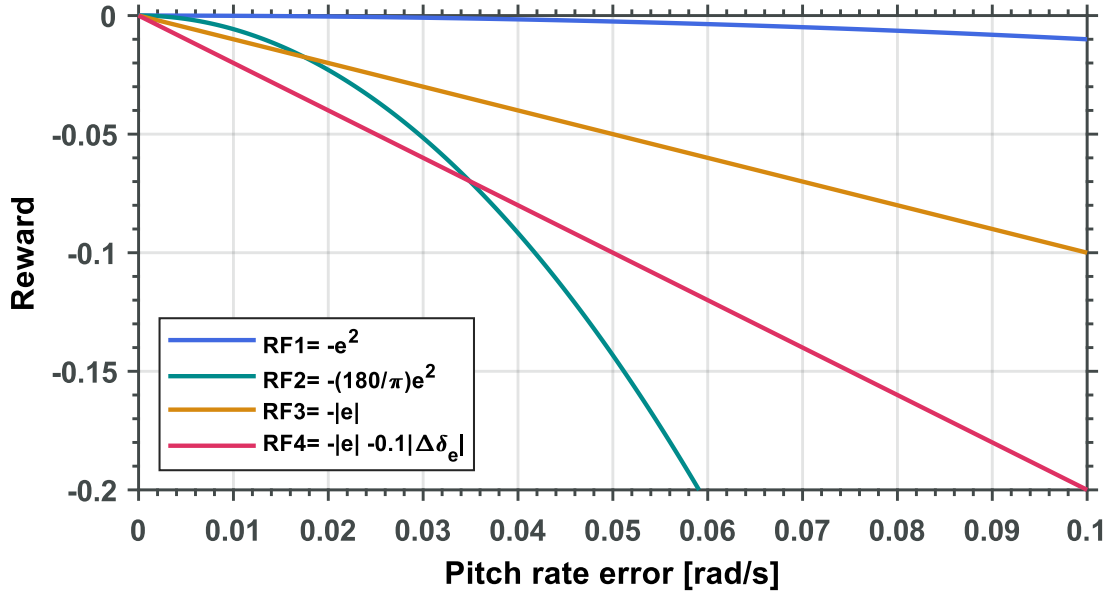


Figure 4.4: Comparison of the various reward functions. $RF4$ is plotted with $\Delta\delta_e = 1\ [rad]$ which is the maximal value possible with saturation limits of $\pm 0.5\ [rad]$.

4.3.2. Method and Setup

Tracking performance is measured quantitatively by the normalised mean absolute error (nMAE) as shown in Equation 4.6, where N is the total number of time steps per episode. Learning performance is evaluated comparatively by examining the learning curves obtained with each reward function, providing insight into the agent's exploration/exploitation behaviour and the degree to which the task is successfully learnt. Lastly, control smoothness can be evaluated through the mean total variation of the elevator deflection Equation 4.7. Each reward function is used to train two agents, each initialised with different seeds. A summary of relevant training parameters is shown in Table 4.4.

$$\text{nMAE} = \frac{\sum_{t=1}^{N} \left| s_t^{ref} - s_t \right|}{\sum_{t=1}^{N} \left| s_t^{ref} \right|} \tag{4.6}$$

$$\text{MTV} = \frac{1}{N} \sum_{t=1}^{N} |a_t - a_{t-1}| \tag{4.7}$$

Table 4.4: High-level parameters used during Experiment I.

Parameter	Value
Environment	Linear Citation
State vector	$[\alpha, q, q_{ref}]$
Action vector	$[\delta_e]$
Steps per episode	1,000
Step size	0.01 $[s]$
Simulation time	10 $[s]$
Total number of steps	100,000
Update frequency (in steps)	1000
Task	Pitch rate tracking
q_{ref}	$\text{deg2rad}(5) \cdot \sin\left(\frac{2\pi}{\frac{1}{2}\text{simulation time}} \cdot t\right)$ $[\frac{rad}{s}]$

To validate the performance of the custom SAC, the same setup and parameters are used for training with the RLTb SAC implementation. The only difference is that instead of training the RLTb agents with the reward functions shown in Table 4.3, the reward is defined according to Equation 4.8. The reason is that MATLAB's proprietary SAC implementation handles the reward scale differently, resulting in poor performance when using the reward functions in Table 4.3. Still, it serves as a useful baseline to compare against.

$$\text{RF}_{RLTb} = -\left(\frac{180}{\pi} \cdot e\right)^2 \tag{4.8}$$

4.3.3. Results

Learning Performance

To produce the learning curves in Figure 4.5, first, the mean is taken between the two agents that were trained with each reward function. Then, the data is normalised with respect to the return with the maximum absolute value. Finally, a moving average with a window size of 5 is used to smooth out the curves. As can be seen in the figure, the RLTb agents converge the fastest, after around $2.5 \cdot 10^4$ time steps, whereas the best-performing SAC agents, according to the learning curves, are the ones that were trained using $RF2$, requiring about $7 \cdot 10^4$ steps to converge. While $RF2$ converges more slowly than the RLTb, it manages to surpass it in terms of return. Of course, learning curves do not tell the whole story because the return is collected through stochastic evaluation of the policy. While this promotes exploration, it isn't representative of the agent's performance once it's deployed.

Another detail to consider is that the policy at the end of training isn't necessarily the best one. Over-training can lead to oscillatory, high-gain-like behaviours and should be avoided. Therefore, during training, the actor's DNN (the policy) is logged every 1000 time steps, resulting in a total of 100 policies per agent. Once training is done, each policy is used to deterministically sample actions in the simulation of a single episode. The policy leading to the highest return is then used in subsequent experiments. The dashed vertical lines in Figure 4.5 represent the moment during training resulting in the best-performing policy. The fact that the best policy for $RF4$ is also the last one suggests that it could benefit from further training, even though it seems to have reached convergence at $8 \cdot 10^4$ steps.

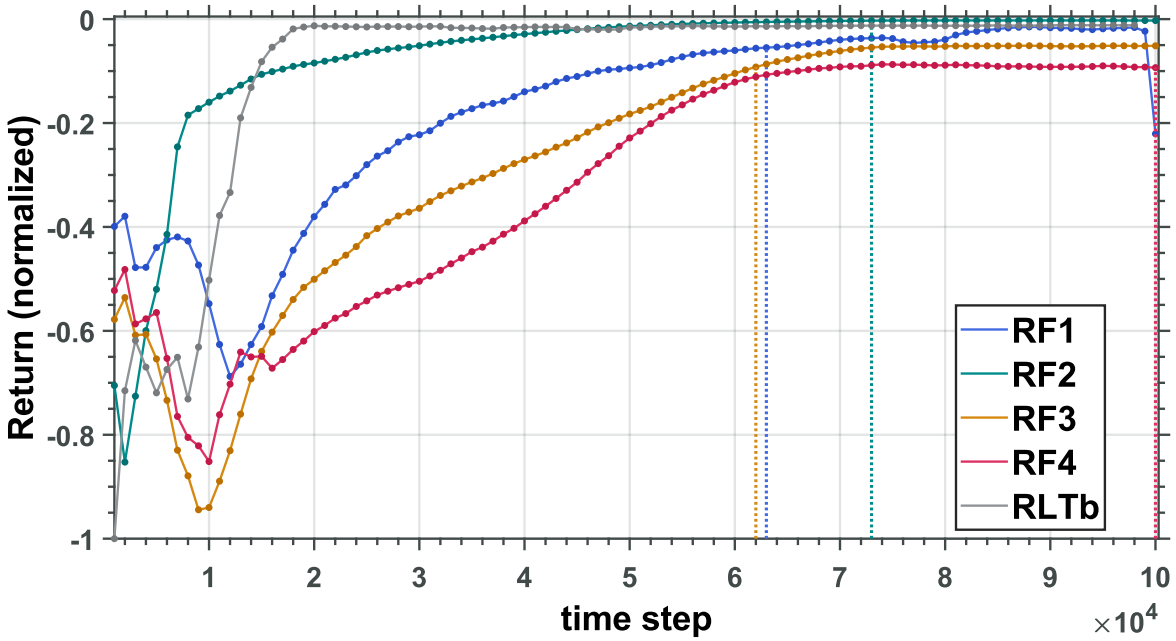


Figure 4.5: Learning curves for $RF1$, $RF2$, $RF3$, $RF4$ and the RLTb. The curves shown are the Window-averaged, normalised, mean values of the raw returns.

Tracking Performance

Figure 4.6 shows the tracking performance of the best-performing policy per reward function, versus the RLTb policy and a manually tuned PID controller. The symbol $\pi_{x/100}$ in the title of each sub-figure indicates that the policy used for simulation is taken from the xth update cycle. In line with the findings in Figure 4.5, $RF2$ leads to the best-performing policy, followed by $RF3$, $RF4$. In contrast to what is suggested in Figure 4.5, the worst-performing policy is given by the $RF1$ agents, which was to be expected considering how insensitive this reward function is. Noteworthy is that all controllers (except the $RF1$ policy) attain an nMAE in the same order of magnitude, suggesting that they all achieve near-optimal control.

Table 4.5 summarises the findings and shows the mean nMAE and MTV ($\overline{\text{nMAE}}$ and $\overline{\text{MTV}}$, respectively) between the two agents of each reward function. Despite the expectation that $RF4$ would have the lowest MTV, and thus control the elevator most smoothly, it is, in fact, the worst-performing reward function in that regard. Lastly, the RLTb has the highest nMAE variance among all RL agents, suggesting its learning consistency is not on par with the custom SAC implementation.

Table 4.5: Result overview for Experiment I comparing the performance achieved with each reward function, SAC implementation and PID controller. In bold is the best result per criterion. *Result is based on a single value, not a mean.

	Reward Function	$\overline{\text{nMAE}}$	σ^2_{nMAE}	$\overline{\text{MTV}}$	σ^2_{MTV}
Custom SAC	$RF1$	1.706%	$5.559 \cdot 10^{-5}\%$	$\mathbf{2.947 \cdot 10^{-4}}$	$4.421 \cdot 10^{-6}\%$
	$RF2$	$\mathbf{0.300}\%$	$2.855 \cdot 10^{-2}\%$	$2.979 \cdot 10^{-4}$	$3.735 \cdot 10^{-8}\%$
	$RF3$	0.337%	$7.885 \cdot 10^{-3}\%$	$2.988 \cdot 10^{-4}$	$1.023 \cdot 10^{-6}\%$
	$RF4$	0.575%	$9.615 \cdot 10^{-2}\%$	$2.992 \cdot 10^{-4}$	$1.062 \cdot 10^{-6}\%$
RLTb SAC	RF_{RLTb}	1.949%	1.802%	$3.000 \cdot 10^{-4}$	$1.479 \cdot 10^{-6}\%$
PID	–	0.549% *	–	$2.977 \cdot 10^{-4}$ *	–

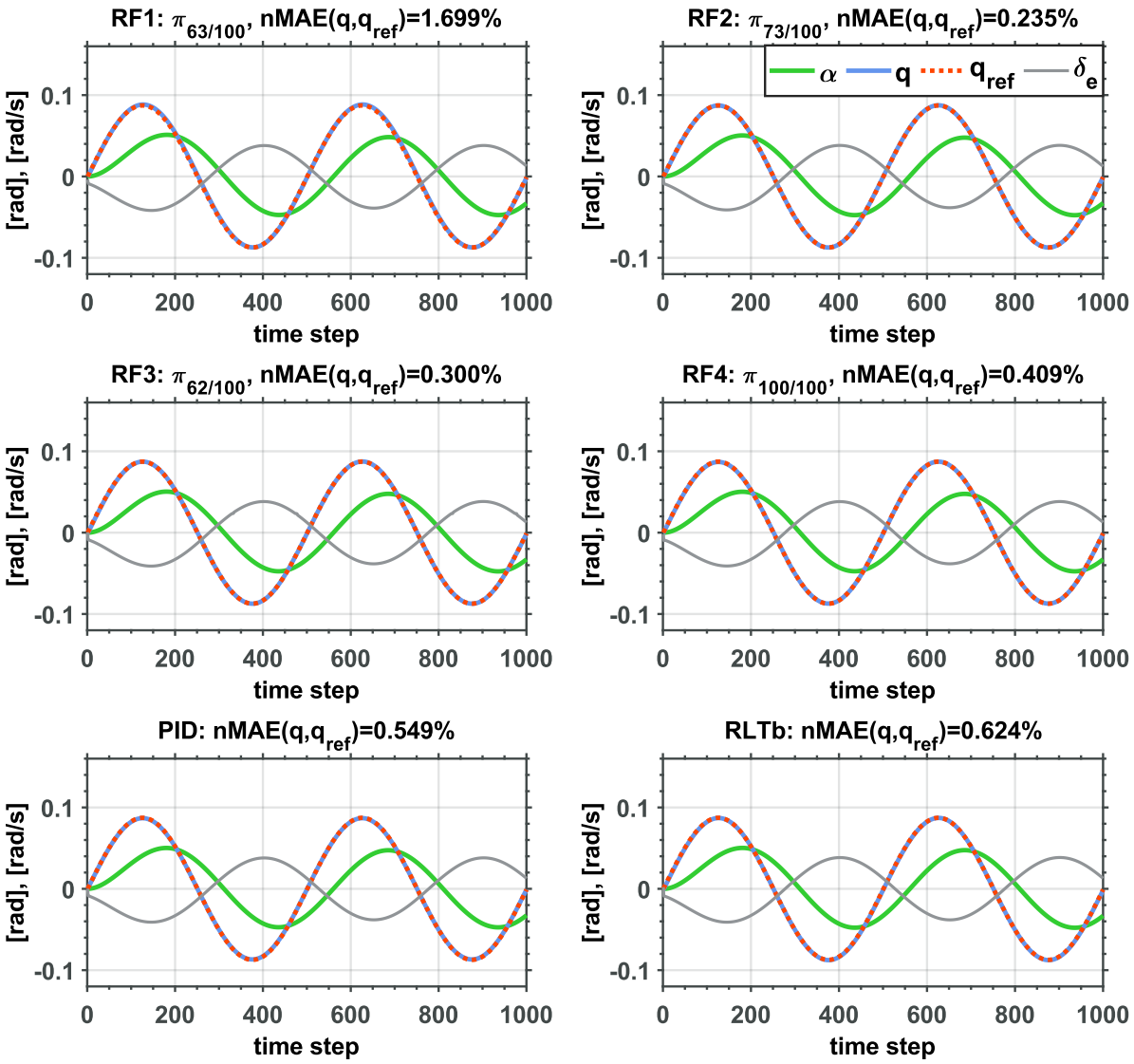


Figure 4.6: Time responses demonstrating the tracking performance of each controller. The best-performing policy per reward function is used in the simulation.

4.4. Experiment II: Tracking Performance in Various Tasks

4.4.1. Objective and Motivation

Till now, the agents were tested with the same reference signal they were trained on. But, in reality, tasks are diverse and are never identical to the training task. So, it is highly relevant to test the agent's ability to generalise and perform in states not previously encountered. To do so, each agent is tested on two different pitch rate reference signals.

1. **Step**: With a magnitude of $10 \cdot \frac{\pi}{180}$ $[\frac{rad}{s}]$, which is twice the magnitude of the reference signal used in training. A step is useful for testing a controller's transient and steady-state response in terms of rise time, stability and error.
2. **3211**: Where each step has a constant area of $15 \cdot \frac{\pi}{180}$ $[rad]$. It is meant to test the system's response to signals of varying duration and magnitude.

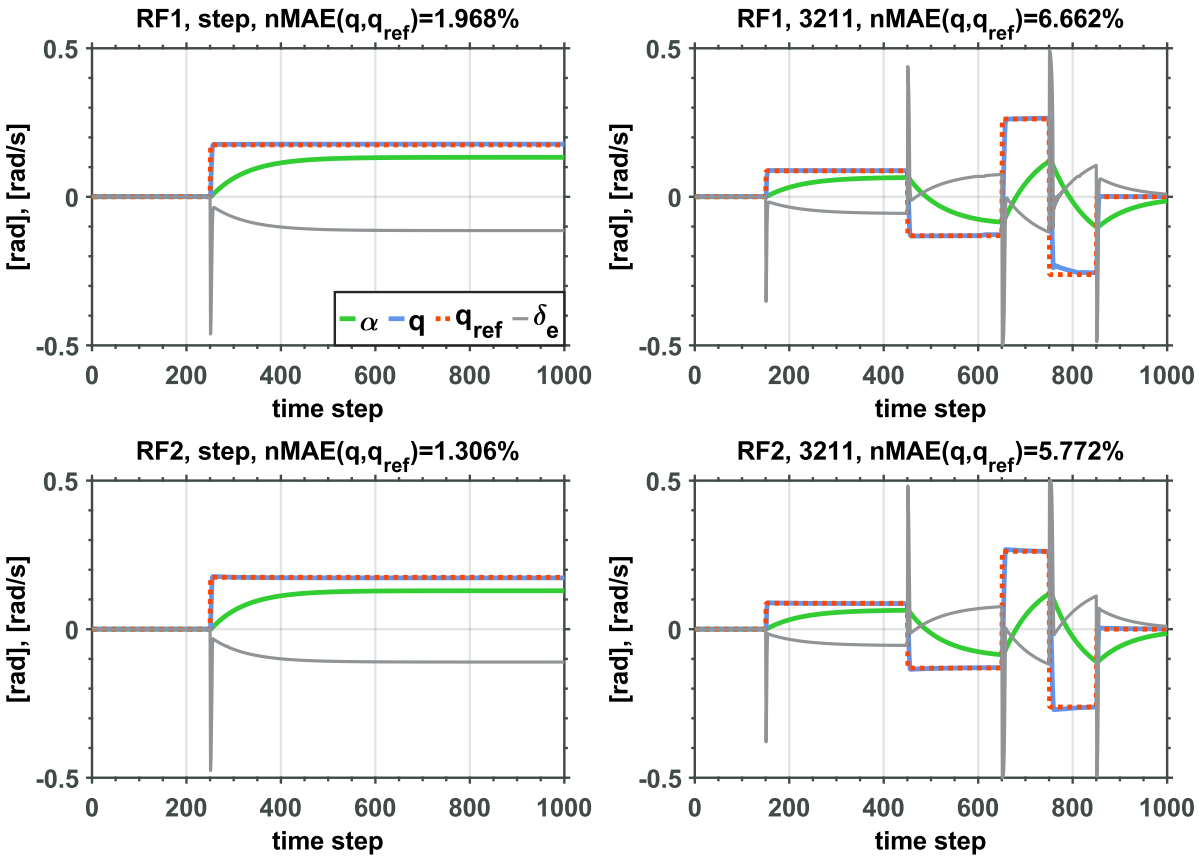
4.4.2. Method and Setup

The policies identified in Experiment I as those who perform the best are used to simulate a single episode with the step and 3211 reference signals. The PID values also remain the same. While the nMAE can be

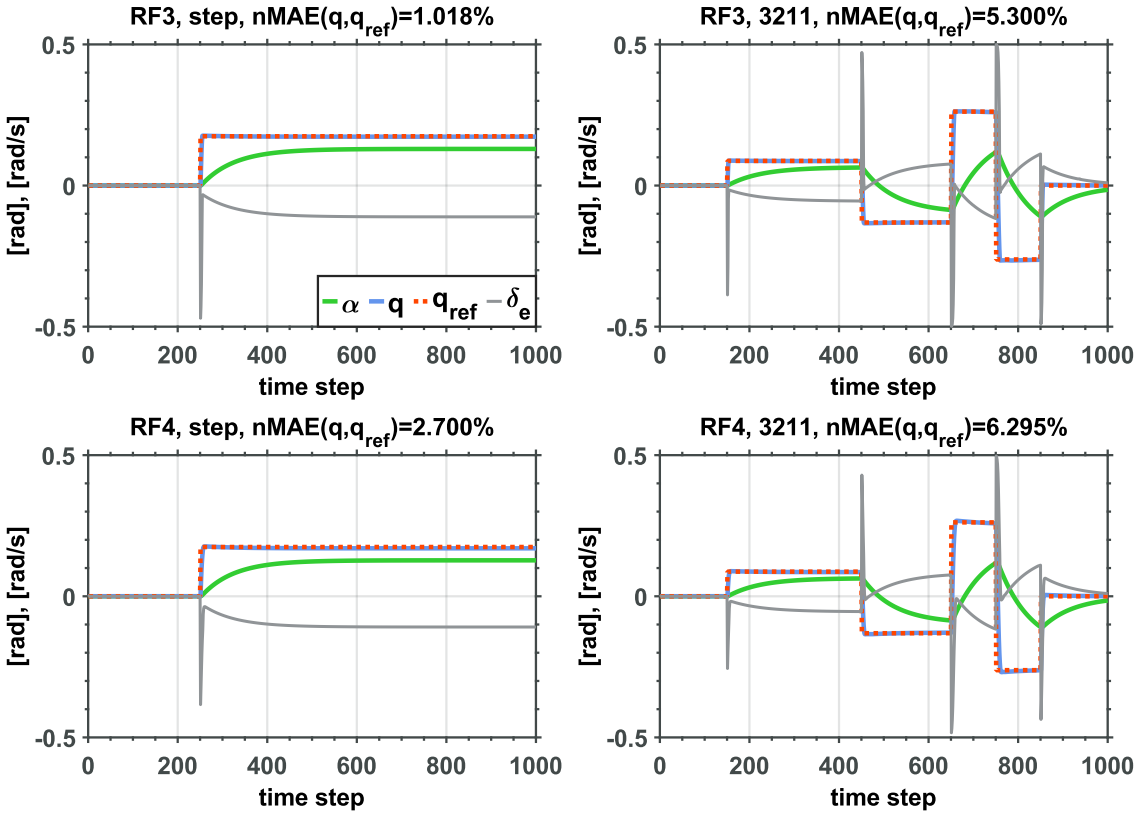
used to compare tracking performance across different signals, the MTV cannot, and should only be used to compare the performance of the various controllers at the same reference signal.

4.4.3. Results

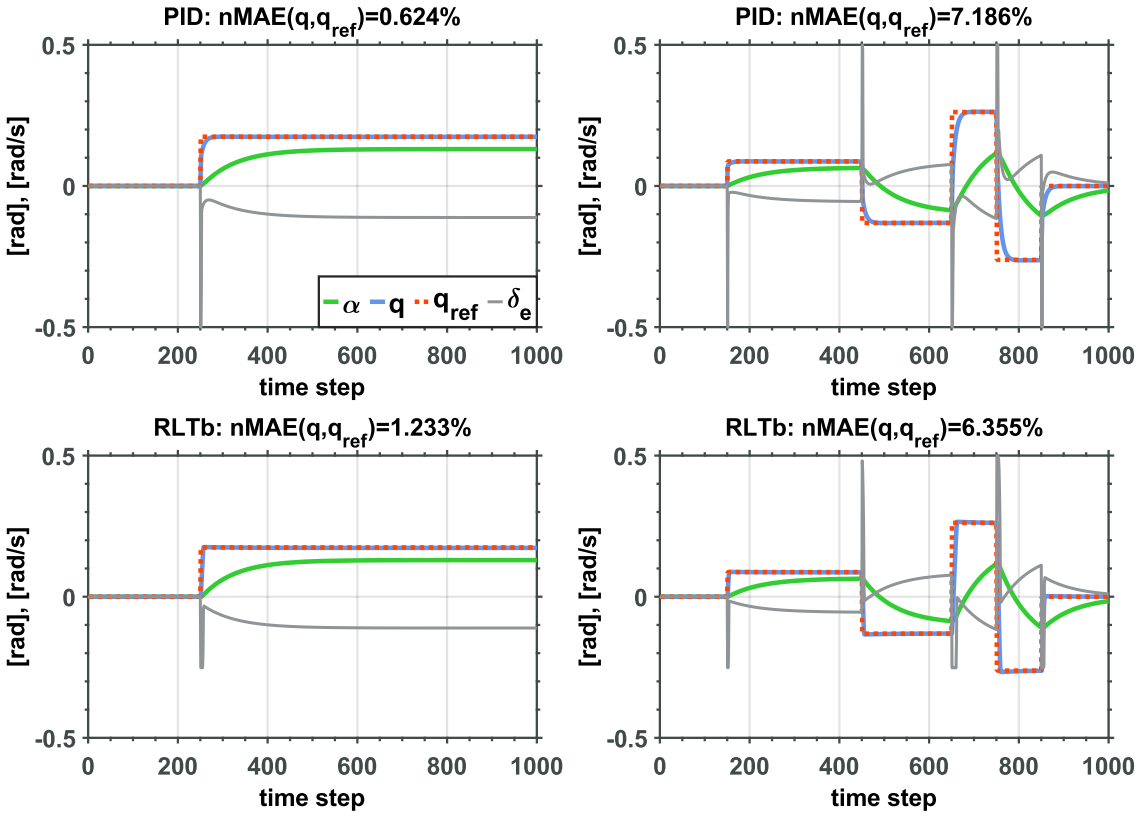
Figure 4.7a, Figure 4.7b, and Figure 4.7c show the resulting time responses for the custom SAC agents, PID and RLTb, respectively. The first notable observation is that while the PID controller outperforms the RL policies in terms of nMAE for the step signal, it is the worst-performing controller for the 3211 signal. When inspecting the time responses, it appears that all of the RL controllers reach a steady-state faster than the PID, but have a larger steady-state error. So, for the step signal whose steady state populates most of the simulation time, the accumulated error is larger for the RL controllers, whereas the 3211 signal requires the system to go through five transient phases. In those, the PID is slower to respond and thus has a higher error. The second observation is that the RLTb policy tracks the 3211 signal asymmetrically- the magnitude of the positive elevator deflections is greater than the magnitude of the negative ones. To compensate, the policy holds the elevator's angle for a few more time steps, as can be seen in the bottom right sub-figure in Figure 4.7c.



(a) Time response of the $RF1$ and $RF2$ custom SAC controllers.



(b) Time response of the $RF3$ and $RF4$ custom SAC controllers.



(c) Time response of the PID and $RLTb$ controllers.

Figure 4.7: Time responses demonstrating the tracking performance of the various controllers on a step (left column) and 3211 (right column) signals.

Table 4.6: Results overview for Experiment II comparing the performance achieved with each reward function, SAC implementation and a PID controller for the step and 3211 reference signals. In bold is the best result per criterion.

	Step		**3211**	
	nMAE	MTV	nMAE	MTV
$RF1$	1.968%	$9.701 \cdot 10^{-4}$	6.662%	$58.011 \cdot 10^{-4}$
$RF2$	1.306%	$10.008 \cdot 10^{-4}$	5.772%	$56.130 \cdot 10^{-4}$
$RF3$	1.018%	$9.876 \cdot 10^{-4}$	$\mathbf{5.300}\%$	$56.023 \cdot 10^{-4}$
$RF4$	2.700%	$8.080 \cdot 10^{-4}$	6.295%	$50.442 \cdot 10^{-4}$
RLTb	1.233%	$\mathbf{5.513 \cdot 10^{-4}}$	6.355%	$\mathbf{43.566 \cdot 10^{-4}}$
PID	$\mathbf{0.624}\%$	$10.131 \cdot 10^{-4}$	7.186%	$57.668 \cdot 10^{-4}$

4.5. Experiment III: Tracking Performance with Modified Dynamics

4.5.1. Motivation and Objective

The differences between simulation and reality manifest not only in variations between the tasks in training and testing but also in the system's dynamics, be it due to differences in operational conditions or due to modelling discrepancies. Here, operational conditions, i.e. airspeed, altitude and mass, affect the dynamics by changing the dynamic pressure p_d and moments of inertia. The objective is therefore to test the controllers' limits with respect to the operational conditions and compare their performance.

4.5.2. Method and Setup

To find their limits, each controller has been tested in ten environments, each linearised about a trim point with a different dynamic pressure value, ranging between $0.35 \cdot p_{d,0}$ to $5.4 \cdot p_{d,0}$. Eventually, two operational limits have been identified and tested on all controllers to provide uniform grounds for comparison. These are described in Table 4.7 and their respective state and input matrices are in Equation 4.9, Equation 4.10, Equation 4.11, and Equation 4.12.

Table 4.7: Edge-case trim conditions used for linearisation of the non-linear dynamics Citation model.

Trim point	$V_{TAS}\left[\frac{m}{s}\right]$	$m\ [kg]$	$h\ [m]$	$p_d\,[Pa]$
low p_d	60	5175	3050	$0.38 \cdot p_{d,0}$
Nominal p_d	90	4500	2000	$p_{d,0}$
high p_d	135	4500	1020	$2.3 \cdot p_{d,0}$

$$A_{low} = \begin{bmatrix} 9.0217 \cdot 10^{-1} & 9.7620 \cdot 10^{-1} \\ -1.6177 \cdot 10^{1} & -1.1253 \end{bmatrix} \quad (4.9) \qquad B_{low} = \begin{bmatrix} -1.8962 \cdot 10^{-2} \\ -4.2468 \end{bmatrix} \quad (4.10)$$

$$A_{high} = \begin{bmatrix} -2.3758 & 9.6442 \cdot 10^{-1} \\ -2.0143 \cdot 10^{1} & -3.3351 \end{bmatrix} \quad (4.11) \qquad B_{high} = \begin{bmatrix} -2.3228 \cdot 10^{-1} \\ -2.5734 \cdot 10^{1} \end{bmatrix} \quad (4.12)$$

The evaluation criteria remain the same as before, and focus on the nMAE and MTV to quantitatively compare the degree to which each controller is able to track the reference input. Since the controllers are tested at the edge of their ability, the time response plots will reveal the failure mode of each controller and offer insight into how their operational envelope may be extended.

4.5.3. Results

In Figure 4.9 are the resulting time responses at the low dynamic pressure operating point for each one of the four custom SAC controllers and the RLTb and PID controllers. Clearly, all controllers suffer from significant degradation in performance. Among the custom SAC agents, $RF4$ is the best performing one, which could be attributed to its conservative elevator use, as evident by its low MTV in Table 4.8. As for the PID controller, it hits the elevator's negative saturation limit after about 140 time steps, after which the aircraft appears to oscillate around a stable equilibrium.

The RLTb agent also fails to track q_{ref} while demonstrating a peculiar failure mode. It seems (and can also be observed in Figure 4.7c) that the agent has learnt a false saturation limit for negative elevator deflections. To verify this isn't a problem with the training setup, the other RLTb agent who was trained with a different seed was tested in the same manner. The test showed that the second agent did not learn such behaviour, suggesting this is an isolated case. This issue is investigated further in Section 4.6.

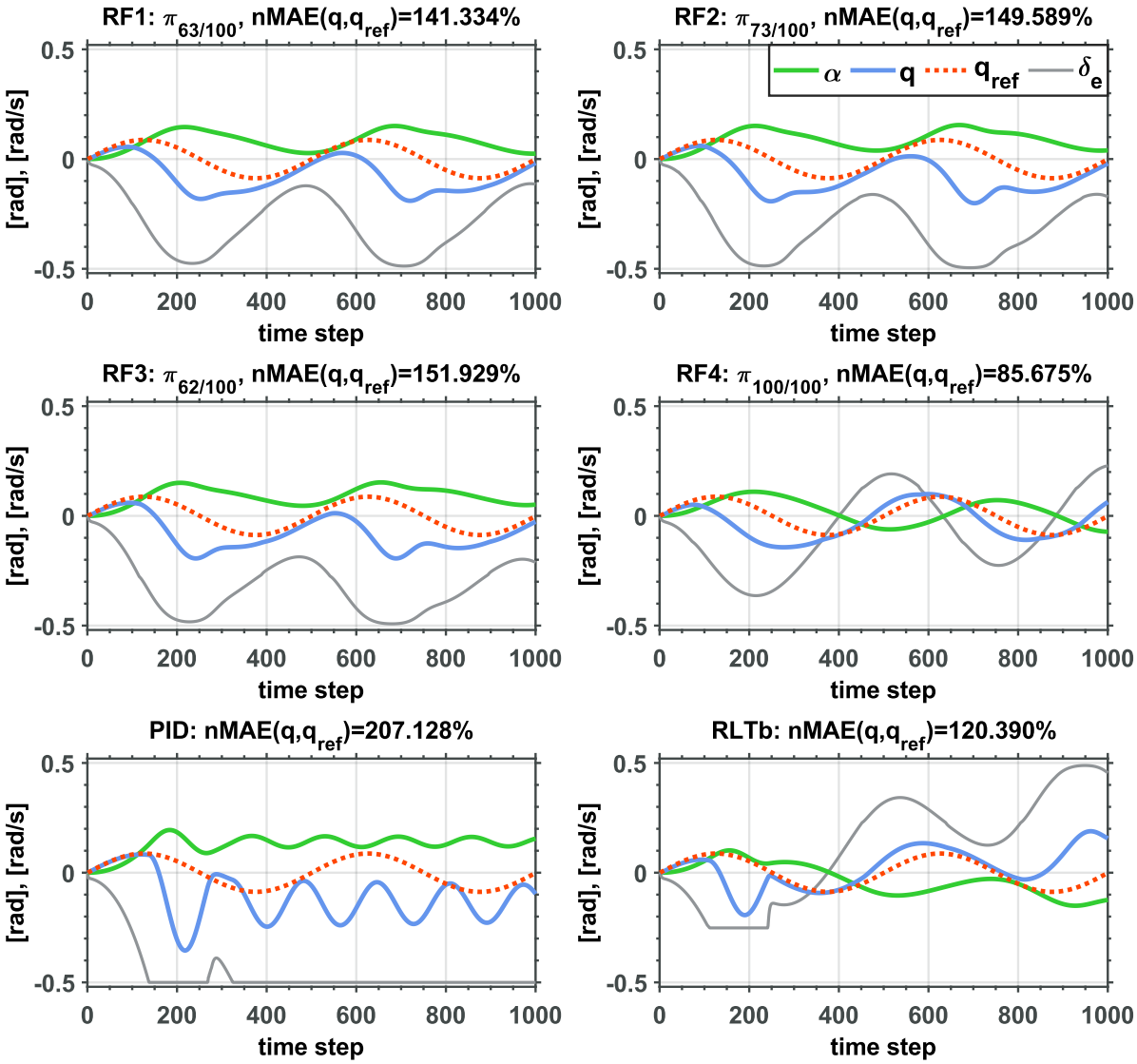


Figure 4.8: Time responses demonstrating the tracking performance of each controller while operating at 38% of the dynamic pressure at linearisation of the system used during training.

When examining the results at the other end, where the model is trimmed and linearised at a dynamic pressure that's higher than nominal, the trend is the opposite- the PID controller outperforms the rest, and even itself when operating at the nominal trim condition. This can be attributed to the increased elevator

effectiveness, which has a similar effect as increasing the proportional gain. On the other hand, the SAC agents who performed well in Experiments I and II (The $RF2$, $RF3$, and RLTb agents) now exhibit strong oscillations, while the $RF1$ agent controls the elevator smoothly. The oscillations are most likely caused by the high error sensitivity of the aforementioned policies, which, when coupled with a higher elevator effectiveness, lead to oscillations. Because the $RF1$ policy is, in comparison, a lower-gain policy to begin with, it can tolerate the new dynamics without outputting high-frequency actions.

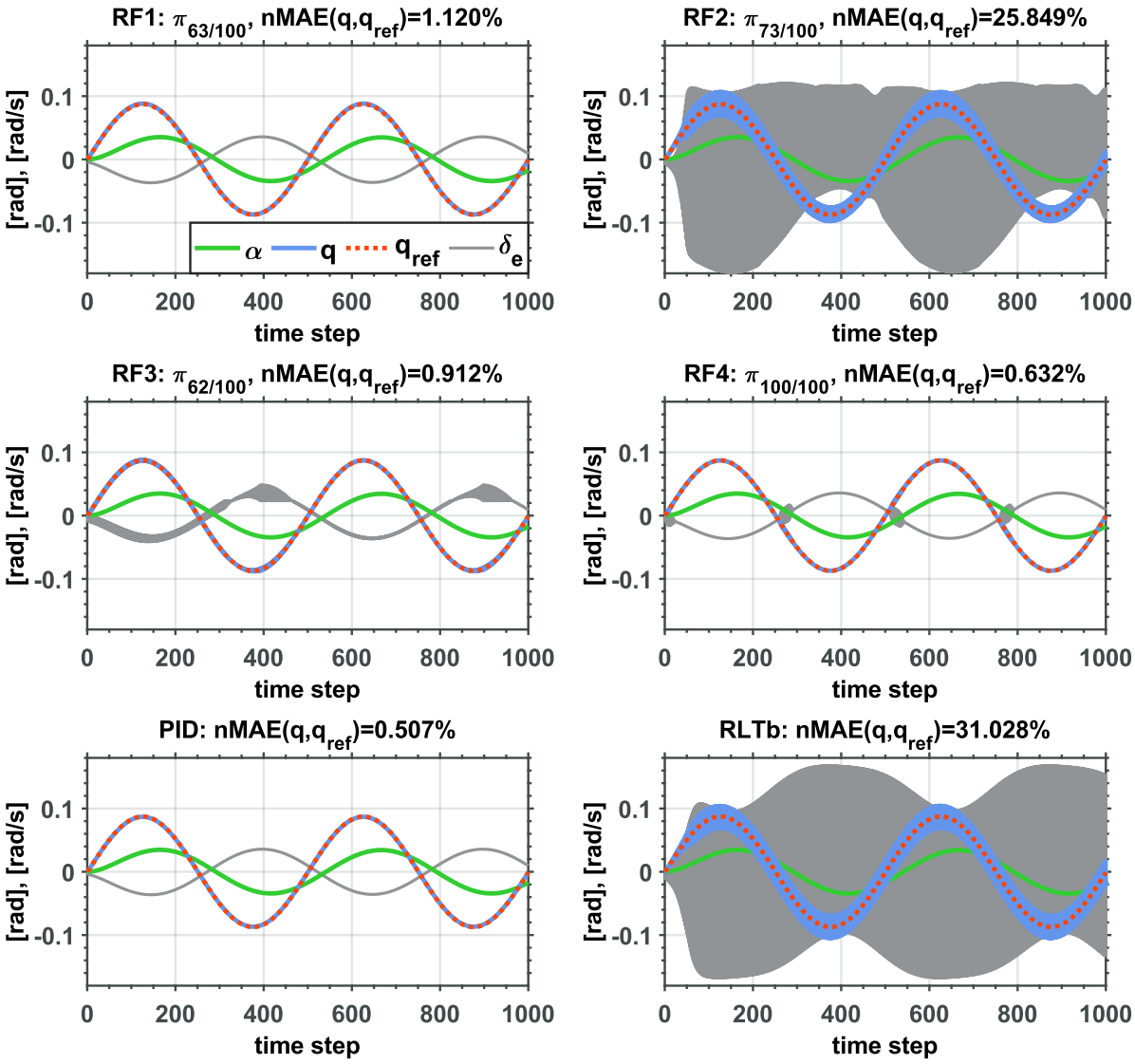


Figure 4.9: Time responses demonstrating the tracking performance of each controller while operating at 230% of the dynamic pressure at linearisation of the system used during training.

Table 4.8: Results overview for Experiment III comparing the performance achieved with each reward function, SAC implementation and a PID controller for two different operating conditions- a dynamic pressure that's below and above the nominal value. In bold is the best result per criterion.

	$\mathbf{p_{d,low} = 0.38 \cdot p_{d,0}}$		$\mathbf{p_{d,high} = 2.3 \cdot p_{d,0}}$	
	nMAE	MTV	nMAE	MTV
$RF1$	141.334%	$1.571 \cdot 10^{-3}$	1.120%	$\mathbf{2.769 \cdot 10^{-4}}$
$RF2$	149.589%	$1.492 \cdot 10^{-3}$	25.849%	$2.194 \cdot 10^{-1}$
$RF3$	151.929%	$1.395 \cdot 10^{-3}$	0.912%	$7.2001613 \cdot 10^{-3}$
$RF4$	**85.675%**	$1.790 \cdot 10^{-3}$	0.632%	$2.333 \cdot 10^{-3}$
RLTb	120.390%	$1.478 \cdot 10^{-3}$	31.028%	$2.633 \cdot 10^{-1}$
PID	207.128%	$\mathbf{7.225 \cdot 10^{-4}}$	**0.507%**	$2.777 \cdot 10^{-4}$

4.6. Policy Response Mapping

The results of experiments I, II, and III highlighted the importance of reward function design and demonstrated the effect each function has on the performance of the various agents. Now, the policies are examined further to explain *why* the different agents behave the way they do. This is done by mapping the response of each policy to every point in its two-dimensional state-space, resulting in a surface. One horizontal dimension for varying α, another horizontal dimension for varying the pitch rate q, and the vertical dimension for the output δ_e. In Equation 4.13 is the range of states the policy was fed. This range is equal to the environment's state limits. Since the reference state $q_{ref} = 0$, the response of the policy to the range of q values represents the action needed to reduce the error, while its response to α values represents the action required to maintain that angle of attack.

$$\begin{bmatrix} \alpha \\ q \\ q_{ref} \end{bmatrix} = \frac{\pi}{180} \begin{bmatrix} -20 & \cdots & 20 \\ -20 & \cdots & 20 \\ 0 & \cdots & 0 \end{bmatrix} \tag{4.13}$$

The left columns in Figure 4.10 and Figure 4.11 show the response of the policies while varying α at different pitch rates. The right columns show the response for varying q at different angles of attack.

The most glaring observation can be found in the response of the $RLTb$ agent with $\pi_{seed=91}$. As mentioned in the previous section, it has learnt a false saturation limit of approximately $-0.25\ [rad]$ while the agent of the second seed, $\pi_{seed=13}$ did not learn such a behaviour. This finding emphasises the vulnerability of RL agents to the seed, even if they seem to have found a near-optimal policy, requiring thorough validation and verification to ensure the trained control policy responds appropriately and predictably.

Two characteristics of the curves in the right columns can be examined in order to explain the performance difference between the agents. The first is the slope of the curves around $q \approx 0$. The $RF2$, $RF3$ and the $RLTb$ agents are a lot steeper, and thus more sensitive to the error in pitch rate than $RF4$ and $RF2$, which is why they performed better in the various tracking tasks in experiments I and II. To strengthen this claim, the q-response curves can be compared to the response of a proportional controller, which can be found in Figure 4.12. The $RF2$, $RF3$ and $RLTb$ agents better approximate the response of the proportional controller than $RF4$ and $RF1$ do. The second characteristic is the width of the q-response curves, which is influenced by a policy's sensitivity to the angle of attack. when comparing $RF1$ and $RF2$, it can be seen that at $q = 0$, the $RF1$ policy deflects the elevator within a wider range than the $RF2$ policy for the same range of angles of attack. Since the model is linear and the agents are trained to track a reference pitch rate, the response should not be a function of the angle of attack, just as the PID controller is independent of it.

Lastly, it is also worth considering the smoothness and continuity of the policies. The curves in the α

response plot of the $RF1$ policy are jaggered, inconsistent and asymmetric compared to those of $RF2$, which can explain why it performs poorly relative to the other policies.

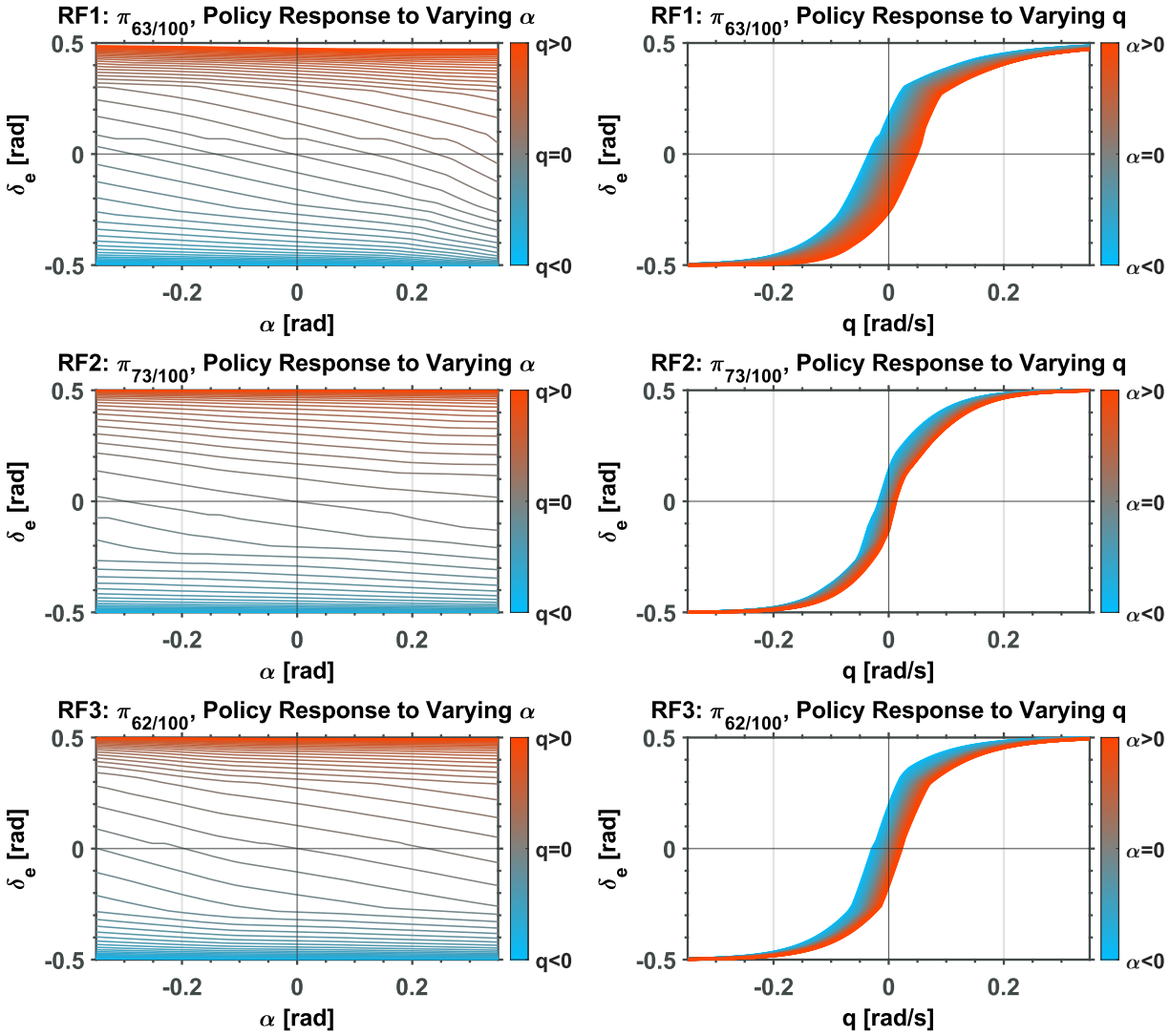


Figure 4.10: Policy response of the $RF1$, $RF2$ and $RF3$ agents for α and q in the range $\frac{\pi}{180} \cdot [-20, 20]\ [rad], [\frac{rad}{s}]$ while $q_{ref} = 0\ [\frac{rad}{s}]$.

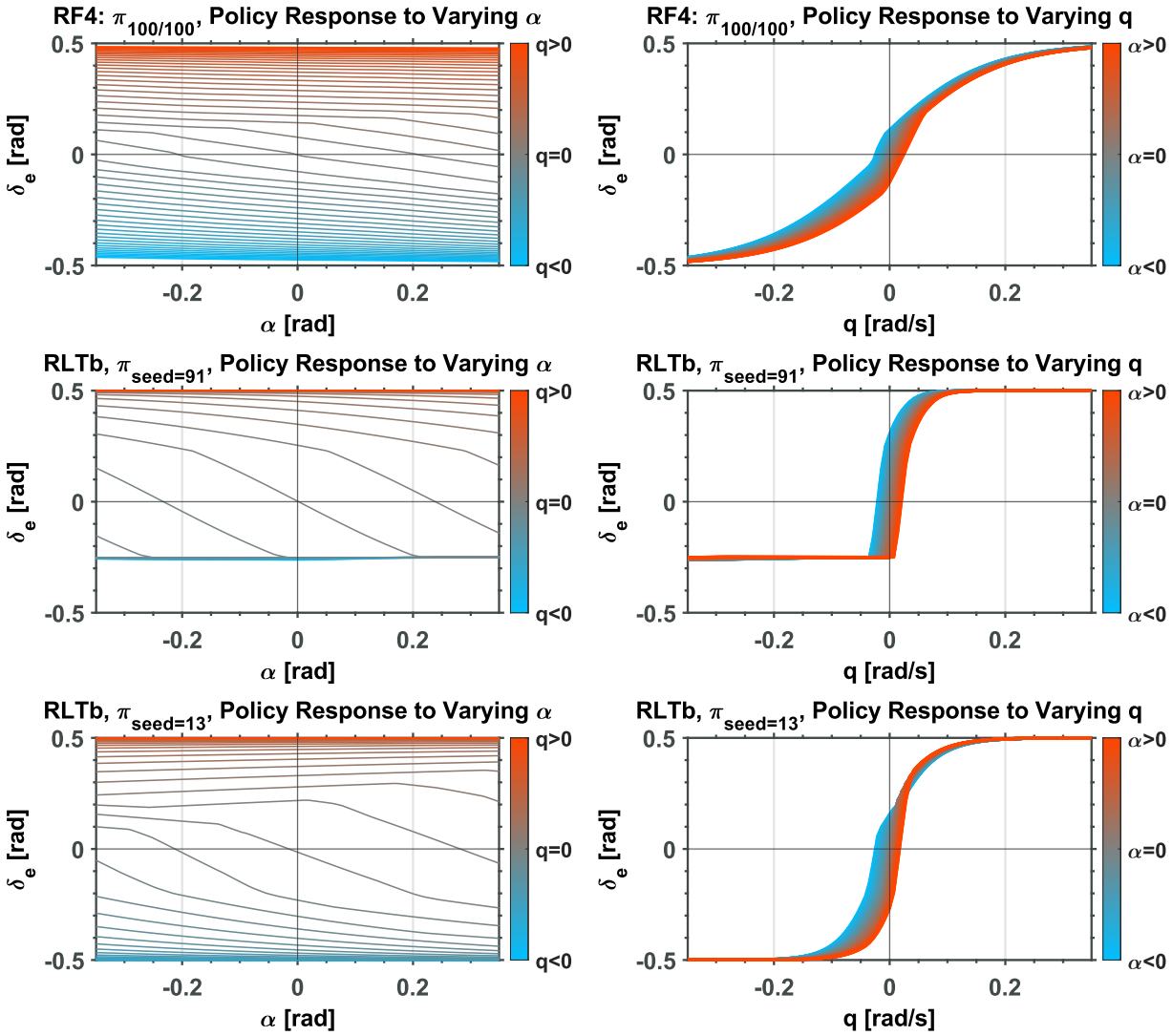


Figure 4.11: Policy response of the $RF4$, and the $RLTb$ agents for α and q in the range $\frac{\pi}{180} \cdot [-20, 20]\ [rad], [\frac{rad}{s}]$ while $q_{ref} = 0\ [\frac{rad}{s}]$.

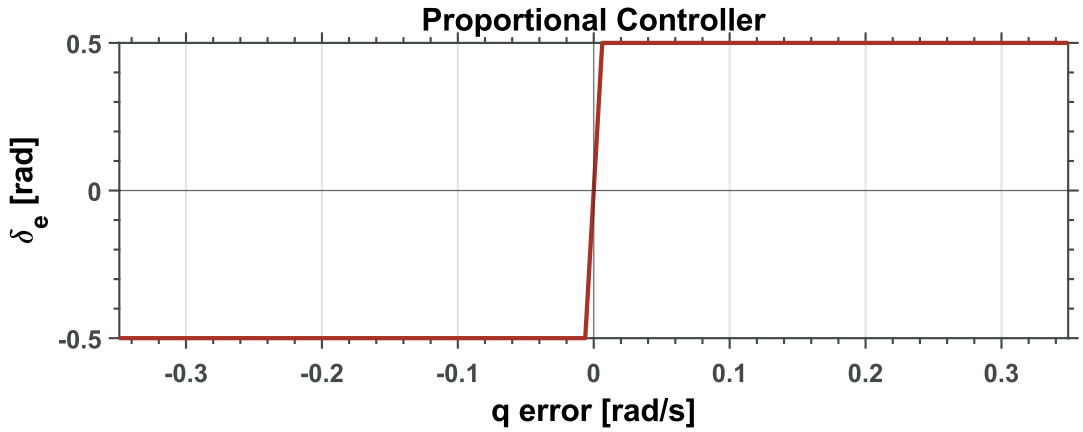


Figure 4.12: Feed-forward response of a proportional controller for a q input signal in the range $\frac{\pi}{180} \cdot [-20, 20]\ [\frac{rad}{s}]$.

4.7. Summary Preliminary Analysis

In this chapter, Research Objective 2 is realised by developing a custom implementation of the Soft Actor-Critic RL algorithm and along with a linear, short-period simulation environment of the Cessna Citation, in which agents are trained to track a reference pitch rate.

To determine which reward function leads to the best performance in terms of learning, tracking error minimisation, and control smoothness, three experiments are performed as described below. Tracking performance and control smoothness are evaluated numerically by the nMAE and the MTV, respectively, while learning is evaluated by comparing learning curves. Having established these performance indicators, Research Question 3.1 is answered. The secondary aim of these experiments is to compare the performance of the policies resulting from the custom SAC implementation to those resulting from MATLAB's RL toolbox and the PID controller. A comparison against such external baselines is valuable for the validation and verification of the custom SAC implementation.

In Experiment I in Section 4.3, eight agents were trained across two seeds and four reward functions with the custom SAC implementation. Using the RLTb, two more agents were trained with two seeds on a single reward function. Results showed that $RF2$ came out on top in terms of learning rate and stability and maximal mean-normalised return amongst the four reward functions. The RLTb agents converged the fastest but had the highest variance between seeds. As for tracking performance, here to, $RF2$ shined and was one of only two agents to outperform the baseline PID controller (the other was $RF3$). All agents demonstrated smooth elevator control, and the MTV differences between the agents and the PID controller were minimal.

In Experiment II in Section 4.4, the agents' ability to generalise to new tasks was tested by simulating an episode using the highest performing policy per reward function on a step and a 3211 pitch rate reference signals. This time $RF3$ had the lowest nMAE among RL agents for both reference signals, but the PID controller was able to track the step with about 60% of $RF3$'s error.

In Experiment III in Section 4.5, the environment's dynamics were modified to simulate flight at both a higher and lower dynamic pressure than the trim point the agent was trained on. This experiment is meant to test the agent's flexibility and to demonstrate the potential of RL in the field of non-linear control. This time, $RF4$ had the lowest error for both flight conditions between the RL agents. However, at high dynamic pressure, the PID performed the overall best when considering both the nMAE and MTV. Besides $RF1$, all RL agents struggled with oscillatory elevator deflections. This experiment highlighted the challenges in creating robust controllers and showed that the agents could potentially benefit from training with domain randomisation and other techniques meant to extend their generalisation skills.

Lastly, in Section 4.6, the responses of the policies were mapped by sweeping over the entire state space. The goal here was to identify why the agents performed the way they did and to validate the observed behaviour by examining key features. The figures revealed that one of the RLTb agents learnt a false saturation limit, emphasising the need to test and verify DNN-based controllers and not take them at face value. Moreover, the policy response plot showed that the best-performing agents, $RF2$ and $RF3$, more closely approximated the response of a proportional controller.

4.8. Conclusions Preliminary Analysis

To conclude, based on the results and observations from the three experiments and the policy response plots, $RF2$ is the overall best reward function in terms of learning, tracking and control smoothness. At the same time, there is still room for improvement, especially in terms of generalisation. One of the motivations for using RL in a Flight Control System was its potential to serve as a non-linear controller and replace traditional gain-scheduled PID controllers. So, it is suggested to implement domain randomisation and test if it can expand the controller's flight envelope beyond what was demonstrated in Experiment III. Then, to enable sim2real transfer, actuator and sensor dynamics should be added, and training with the non-linear model should be considered, to further minimise the gap.

Part III

Closure

5

Conclusion

Robust, fault-tolerant flight control systems (FCS) are key in improving aviation safety, where loss of control in flight (LOC-I) is still a prominent cause of fatal accidents. Reinforcement learning (RL) is a promising technique for the development of robust, model-free flight control laws and has demonstrated impressive performance in simulation when applied to a model of TU Delft's Cessna Citation II research aircraft (PH-LAB). To advance the research field towards flight testing this type of FCS, it is essential to first test them in simulation under realistic conditions that emulate the transition from simulation to reality. This research investigates what it meak the leap from simulation to reality and develops a method to overcome prevalent sensor and actuator dynamics.

The research questions and objectives below break down the end goal into smaller components that can be answered individually, helping to guide the research process.

Research Question 1

Which RL algorithm is the most appropriate for the development of a FCS for the Cessna Citation II?

1.1 What are the state-of-the-art RL algorithms and applications for flight control?
1.2 Which requirements must the FCS meet?
1.3 What are the assumptions, and how do they influence the solution space?

- **1.1:** In Chapter 3 PPO and SAC are identified as the two most relevant state-of-the-art offline algorithms for the application. SAC has already been used several times to develop a FCS for the Citation in simulation and demonstrated promising results. PPO has yet to be applied as a FCS for the PH-LAB, but it has been successfully used in other robotic applications. Additional state-of-the-art methods include approximate dynamic programming (ADP) and its derivatives. Incremental ADP has already been applied as an online adaptive FCS and test flown in the PH-LAB [9].
- **1.2:** Per the requirements defined in Section 3.7.1, the algorithm with which the FCS is developed has to support continuous state and action spaces, be model-free, and offline-trained. It should maintain adequate performance, as defined later, when transitioning from simulation to reality and be capable of generalising to tasks that were not encountered during training.
- **1.3:** According to the set of assumptions as laid out in the literature review in Section 3.7.2, under realistic assumptions, sensors are not perfect and their measurements, which may be noisy, biased and delayed, do not represent the true state of the system. Thus, the naive application of the Markov decision process framework for RL does not hold. Instead, the aircraft's state is assumed to only be partially observable, requiring the agent to have access to additional information such as a history of states and or actions. In addition, control surfaces have inertia, and their actuators are force and rate-limited. These effects are often simplified but are important to consider not only due to their effect on the dynamical response of the system, but also due to their influence on the agent's ability to explore the environment.

Based on the requirements and assumptions above, SAC is the algorithm of choice. Beyond its technical compatibility, it has a proven track record in similar applications, which this research builds on

top of. The partial observability of the state is solved by adding an LSTM layer to both the actor and critic instead of augmenting the state with a series of past actions. This concludes the answer to question 1.

Research Question 2

What are the challenges in transferring from simulation to reality, and what are the solutions that can enable implementation in the Cessna Citation II?

2.1 What are, in general, the challenges with sim-to-real transfer and which are unique to RL-based controllers?
2.2 Which aspects of the real aircraft's dynamics are not accounted for in the available simulation model, but are relevant in enabling sim-to-real transfer?
2.3 How should the control architecture be designed?
2.4 How can the real system be emulated in simulation to facilitate testing of the sim-to-real transfer?

- **2.1:** InSection 3.6.4, several challenges in transferring from simulation to reality are identified. First, there are discrepancies between the simulation model and reality. The choice of assumptions and which dynamics to simplify has a major impact on the performance of offline synthesised systems when deployed. Challenges unique to RL are reward exploitation and loss of generalisation, which occur when the reward function and environment are not set up correctly. Often, it is difficult to identify such issues in advance, requiring experimentation.
- **2.2:** The CitAST simulation package offers a nonlinear-dynamics model of the Citation, which has been validated against flight data [92]. But, it models the control surfaces with only a first-order transfer function, thus neglecting the actuator's rate limit. Additionally, it lacks sensor models. To mitigate these discrepancies, sensors are modelled by adding noise, bias and delay to the clean state measurement. As for the actuators, a rate limit is applied after the existing transfer function.
- **2.3:** In contrast to traditional linear controllers (LC), where it is common to cascade the control for dynamics with significantly different bandwidths, RL agents can often learn to handle this, thus not requiring cascaded architectures. The control loop presented in Part I takes advantage of this by employing a single policy to control the pitch attitude directly. Another important difference with respect to the control loop of the baseline LC is that the commanded action and the true elevator position are fed back to the policy at every time step to provide the LSTM layer with information about the history of actions and the response of the elevator, respectively. Together, these design choices enable the sim-to-real transition as demonstrated in Part I.
- **2.4:** Sim-to-real transfer is emulated by training the SAC agents in a linear-dynamics environment in which the elevator is modelled with only a first-order transfer function. Then, testing is performed using the nonlinear-dynamics CitAST model, to which the various sensor and actuator dynamics are added: noise, bias, delay, and a full elevator model as described above. Ideally, the robustness of the FCS is tested under various operational conditions and with simulated atmospheric disturbances. Due to time limitations, this remains untested.

Having answered questions 2.1-2.4, it can be concluded that the proposed FCS design and approach are suitable for overcoming the challenges associated with sim-to-real transfer and have the potential to enable implementation and flight testing in the PH-LAB.

Research Question 3

How does the proposed RL controller perform?

3.1 What are the most appropriate performance indicators and how should they be measured?
3.2 How well does the controller generalise across different tracking tasks?
3.3 How sensitive is it to the various sensor and actuator dynamics?
3.4 How does it compare to other controllers in terms of performance and sim-to-real transferability?

- **3.1:** The most appropriate performance indicators are identified to be the normalised mean absolute error (nMAE) and control activity (CA). nMAE is useful in quantifying tracking performance independent of the type of reference signal. For it to be useful in quantifying the effect of the various sensor and actuator dynamics, it is measured with the observable state rather than the true state. The CA measures the cumulative commanded elevator deflections and is used to evaluate the smoothness of the output of each controller. Time response plots, while not providing quantitative data, are indispensable in the assessment of a controller's performance. This is because nMAE and CA may produce the same values under different circumstances, thus not telling the whole story.
- **3.2:** The SAC-LSTM controller proved to be the most indifferent to the type of pitch attitude reference signal in comparison with both the SAC-FF and the LC when tested at the training configuration (and tuning configuration, respectively). Of course, this conclusion is limited in applicability because only two types of reference signals were tested- a sinusoidal and a reference, both with a maximum amplitude of $5°$.
- **3.3:** A sensitivity analysis in Part I points at the SAC-LSTM controller as the least sensitive to noise and bias, as indicated by the smooth elevator deflections. In contrast, both the SAC-FF policy and the LC amplify the noise. When it comes to elevator dynamics, here too, the SAC-LSTM policy is largely indifferent to the addition or absence of the first-order transfer function and servo rate limit, whereas the SAC-FF controller struggles to control the elevator if it is allowed to react faster than it did during training. The LC is unable to maintain control when faced with the servo rate limit unless a different set of gains is used. As for delays, while the SAC-LSTM controller is again the most robust, its margins are not as large in comparison with the SAC-FF controller. Delay significantly degrades the performance of the LC, but this too can be mitigated by using a different, less aggressive set of gains.
- **3.4:** All in all, the SAC-LSTM controller performs better than the SAC-FF and LC. This is based on both a quantitative assessment, in which it clears all performance thresholds in all test cases, and based on comparison of the time response plots, in which it can be seen to track aggressivity, but smoothly.

Based on the answers above, it can be concluded that the SAC-LSTM agent demonstrates the greatest potential to successfully transition from simulation to reality, one-shot.

Research Objective

To enable the implementation and flight testing of an offline reinforcement learning (RL) flight control system (FCS) in TU Delft's Cessna Citation II research aircraft by bridging the simulation-reality gap, thereby contributing to the development of robust and fault-tolerant FCS meant to improve safety, performance and autonomy.

RO1 Identify the most suitable offline RL framework for a FCS for the Cessna Citation II by reviewing literature and the current state-of-the-art applications.
RO2 Identify the main challenges and solutions relating to systems and control in transferring an offline synthesised controller to a real system.
RO3 Design and develop a pitch rate controller utilising the chosen RL framework and test in simulation against a baseline controller.
RO4 Extend the framework to enable pitch attitude control, test in a simulation environment that emulates sim-to-real transfer, and compare against a baseline controller.

To conclude, this research presents the development of a RL-based FCS with the potential to successfully transfer from simulation to reality without additional online training. The flight control law is realised by training an SAC agent offline. The key to success is to incorporate an LSTM layer in both the actor and critic, enabling the policy to capture time dependencies which only become significant when the effects of sensor and actuator dynamics are considered. The findings in this report contribute to the development of robust and fault-tolerant FCS by providing solutions that can enable flight testing of similar controllers, hence contributing to advancements in aviation safety.

6

Recommendations

Below is a set of recommendations for follow-up research based on the conclusions and on adjacent topics that did not fit in the scope of this research, but are worthy of further investigation.

Comprehensive Robustness Testing
The SAC-LSTM FCS has demonstrated resilience to various sensor and actuator dynamics not previously considered. However, its performance across a range of operational and environmental conditions was not tested. Specifically:

1. **Operational conditions:** Factors such as the aircraft's mass, centre of gravity, true airspeed and altitude were treated as constants in both training and testing. The recommendation is to test the FCS at several combinations of weight and dynamic pressure, covering the Citation's flight envelope. Additionally, considering that within the flight envelope, the behaviour of the Citation is rather linear, it would also be valuable to test the controller's performance while flying at a high angle of attack, where the dynamics become significantly more nonlinear.
2. **Atmospheric disturbances:** In contrast to operational conditions, atmospheric disturbances cannot be controlled to the same degree. So, it is perhaps even more important to test the FCS response when facing disturbances such as gusts, turbulence and ice formation. Their effects on control authority, lift, and stall may be substantial, so testing first in simulation is essential.
3. **Technical failures:** Being that the motivation to use RL is to gain fault tolerance, it is of course relevant to test how the proposed FCS behaves when facing failures such as reduced control authority, jammed control surfaces, reduced stability margins and damaged horizontal and vertical stabilisers.

Training Reliability
A major challenge in RL, and for SAC in particular, is the low training reliability and variance in the resultant behaviour. The first recommendation is to perform hyperparameter optimisation, which can improve learning stability. Additionally, techniques such as state-dependent exploration [93] can prevent the policy from converging to an oscillatory behaviour, a phenomenon that was observed in multiple agents.

Verification and validation of RL-based FCS
The black-box nature of DNN makes it difficult to map the performance of the controller across all possible states, of which there are, in the continuous case, infinitely many. So, it is not straightforward to guarantee performance and safety. The recommendation is to consider this aspect in future work because it will influence the controller implementation method, and may be the differentiating factor between receiving a flight clearance or not.

Flight Testing
Simulations alone can only take the field so far; therefore, it is recommended to experiment with the proposed FCS through flight tests in the PH-LAB.

References

[1] *IATA Annual Safety Report-2024 Executive Summary and Safety Overview*. Tech. rep. IATA, 2024.

[2] *Annual safety review 2024*. Tech. rep. European Union Aviation Safety Agency, July 2024, p. 140. DOI: 10.2822/49362.

[3] *IATA Annual Safety Report Recommendations for Accident Prevention*. Tech. rep. IATA, 2024.

[4] L. Tauszig. *Special Condition VTOL: Airworthiness Requirements As A First Building Block For VTOL Safety*. Tech. rep. EASA, 2024.

[5] J.-J. E.. Slotine et al. *Applied nonlinear control*. Prentice Hall, 1991, p. 461.

[6] Gunter Stein et al. "Dynamic inversion: An evolving methodology for flight control design". In: *International Journal of Control* 59.1 (1994), pp. 71–91. DOI: 10.1080/00207179408923070.

[7] Elia Kaufmann et al. "Champion-level drone racing using deep reinforcement learning". In: *Nature* 620.7976 (Aug. 2023), pp. 982–987. DOI: 10.1038/s41586-023-06419-4.

[8] Laura Smith et al. "A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning". In: (Aug. 2022). URL: http://arxiv.org/abs/2208.07860.

[9] R. Konatala et al. "Flight Testing Reinforcement Learning based Online Adaptive Flight Control Laws on CS-25 Class Aircraft". In: *AIAA SciTech Forum and Exposition, 2024*. American Institute of Aeronautics and Astronautics Inc, AIAA, 2024. DOI: 10.2514/6.2024-2402.

[10] Killian Dally. *Deep Reinforcement Learning for Flight Control*. Tech. rep. MSc thesis, Delft University of Technology, 2021. URL: http://repository.tudelft.nl/..

[11] Hidde Jansen et al. "Longitudinal Handling Qualities Evaluation for Soft Actor-Critic Deep Reinforcement Learning Flight Control". In: *AIAA SCITECH 2025 Forum*. Reston, Virginia: American Institute of Aeronautics and Astronautics, Jan. 2025. DOI: 10.2514/6.2025-2794. URL: https://arc.aiaa.org/doi/10.2514/6.2025-2794.

[12] Casper Teirlinck. *Reinforcement Learning for Flight Control Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance*. Tech. rep. MSc thesis, Delft University of Technology, 2022. URL: http://repository.tudelft.nl/..

[13] Lucas Vieira dos Santos. *Safe & Intelligent Control Fault-tolerant Flight Control with Distributional and Hybrid Reinforcement Learning using DSAC and IDHP*. Tech. rep. MSc thesis, Delft University of Technology, 2023.

[14] R. Sutton et al. *Reinforcement learning : an introduction*. The MIT Press, 2020.

[15] Gely P. Basharin et al. "The life and work of A.A. Markov". In: *Linear Algebra and Its Applications*. Vol. 386. 1-3 SUPPL. July 2004, pp. 3–26. DOI: 10.1016/j.laa.2003.12.041.

[16] Tuomas Haarnoja et al. "Learning to Walk via Deep Reinforcement Learning". In: (Dec. 2018). URL: http://arxiv.org/abs/1812.11103.

[17] Bart Helder. *Reinforcement Learning for Helicopter Flight Control*. Tech. rep. MSc thesis, Delft University of Technology, 2020.

[18] S Heyer. *Reinforcement Learning for Flight Control Learning to Fly the PH-LAB*. Tech. rep. MSc thesis, Delft University of Technology, 2019.

[19] Tuomas Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: (Dec. 2018). URL: http://arxiv.org/abs/1812.05905.

[20] Xue Bin Peng et al. “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”. In: (Oct. 2017). DOI: 10.1109/ICRA.2018.8460528. URL: http://arxiv.org/abs/1710.06537%20http://dx.doi.org/10.1109/ICRA.2018.8460528.

[21] Josh Tobin et al. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: (Mar. 2017). URL: http://arxiv.org/abs/1703.06907.

[22] Antonio Loquercio et al. “Deep Drone Racing: from Simulation to Reality with Domain Randomization”. In: *IEEE Transactions on Robotics* vol 36 (Feb. 2020). DOI: 10.1109.

[23] Robin Ferede et al. “End-to-end Reinforcement Learning for Time-Optimal Quadcopter Flight”. In: (Nov. 2023). URL: http://arxiv.org/abs/2311.16948.

[24] Richard. Bellman. *Dynamic programming*. Princeton University Press, 1972.

[25] Fadi AlMahamid et al. “Reinforcement Learning Algorithms: An Overview and Classification”. In: (Sept. 2022). DOI: 10.1109/CCECE53047.2021.9569056. URL: http://arxiv.org/abs/2209.14940%20http://dx.doi.org/10.1109/CCECE53047.2021.9569056.

[26] Ashish Vaswani et al. “Attention Is All You Need”. In: *NIPS’17: Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017.

[27] Diederik P. Kingma et al. “Adam: A Method for Stochastic Optimization”. In: (Dec. 2014). URL: http://arxiv.org/abs/1412.6980.

[28] R. Sutton et al. *Policy Gradient Methods for Reinforcement Learning with Function Approximation*. Tech. rep.

[29] D. Nguyen-Tuong et al. “Model Learning for Robot Control: A Survey”. In: *Cognitive Processing* (2011). DOI: 10.1007/s10339-011-0404-1.

[30] C. Szepesvári. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[31] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. Tech. rep. 2015.

[32] Joshua Achiam et al. “Towards Characterizing Divergence in Deep Q-Learning”. In: (Mar. 2019). URL: http://arxiv.org/abs/1903.08894.

[33] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: (Sept. 2015). URL: http://arxiv.org/abs/1509.02971.

[34] Hado van Hasselt et al. “Deep Reinforcement Learning with Double Q-learning”. In: (Sept. 2015). URL: http://arxiv.org/abs/1509.06461.

[35] T. Schaul et al. “Prioritized Experience Replay”. In: *ICLR* (2016). URL: https://arxiv.org/abs/1511.05952.

[36] Ziyu Wang et al. “Dueling Network Architectures for Deep Reinforcement Learning”. In: (Nov. 2015). URL: http://arxiv.org/abs/1511.06581.

[37] Will Dabney et al. “Distributional Reinforcement Learning with Quantile Regression”. In: (Oct. 2017). URL: http://arxiv.org/abs/1710.10044.

[38] Meire Fortunato et al. “Noisy Networks for Exploration”. In: (June 2017). URL: http://arxiv.org/abs/1706.10295.

[39] Matteo Hessel et al. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: (Oct. 2017). URL: http://arxiv.org/abs/1710.02298.

[40] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: (Feb. 2016). URL: http://arxiv.org/abs/1602.01783.

[41] D. Silver et al. “Deterministic Policy Gradient Algorithms”. In: *Proceedings of the 31st International Conference on Machine Learning*. 2014.

[42] Sergey Ioffe et al. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: (Feb. 2015). URL: http://arxiv.org/abs/1502.03167.

[43] Yan Duan et al. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: (Apr. 2016). URL: http://arxiv.org/abs/1604.06778.

[44] Scott Fujimoto et al. “Addressing Function Approximation Error in Actor-Critic Methods”. In: (Feb. 2018). URL: http://arxiv.org/abs/1802.09477.

[45] John Schulman et al. “Trust Region Policy Optimization”. In: (Feb. 2015). URL: http://arxiv.org/abs/1502.05477.

[46] S. Kullback et al. *On Information and Sufficiency*. Tech. rep. The George Washington University and Washington D.C., 1951.

[47] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: (July 2017). URL: http://arxiv.org/abs/1707.06347.

[48] István Szita et al. *Communicated by Andrew Barto Learning Tetris Using the Noisy Cross-Entropy Method András Lo rincz*. Tech. rep. URL: http://direct.mit.edu/neco/article-pdf/18/12/2936/816651/neco.2006.18.12.2936.pdf.

[49] Ziyu Wang et al. “Sample Efficient Actor-Critic with Experience Replay”. In: (Nov. 2016). URL: http://arxiv.org/abs/1611.01224.

[50] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: (Jan. 2018). URL: http://arxiv.org/abs/1801.01290.

[51] Mozammal Chowdhury et al. “Interchangeable Reinforcement-Learning Flight Controller for Fixed-Wing UASs”. In: *IEEE Transactions on Aerospace and Electronic Systems* 60.2 (Apr. 2024), pp. 2305–2318. DOI: 10.1109/TAES.2024.3351608.

[52] Fei Yue Wang et al. “Adaptive dynamic programming: An introduction”. In: *IEEE Computational Intelligence Magazine* 4.2 (May 2009), pp. 39–47. DOI: 10.1109/MCI.2009.932261.

[53] P. J. Werbos. *Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence*. Vol. XXII. Society For General Systems Research, 1977.

[54] P. Werbos. “Approximate dynamic programming for real-time control and neural modeling”. In: *HANDBOOK OF INTELLIGENT CONTROL NEURAL, FUZZY, AND ADAPTIVE APPROACHES*. Ed. by D. A. White et al. New York: Van Nostrand, 1992. Chap. 13.

[55] P. Werbos. “A menu of designs for reinforcement learning over time”. In: *Neural Networks for Control*. Ed. by W. Thomas Miller et al. Cambridge: The MIT Press, 1990. DOI: 10.7551/mitpress/4939.001.0001. URL: https://direct.mit.edu/books/book/3977/Neural-Networks-for-Control.

[56] F. Silvia et al. “Model-Based Adaptive Critic Designs”. In: *Handbook of Learning and Approximate Dynamic Programming*. Ed. by Jenni Si et al. Wiley-IEEE Press, 2004. Chap. 3, pp. 65–95.

[57] D. V. Prokhorov et al. “Adaptive Critic Designs”. In: *IEEE Transaction On Neural Networks* 8.5 (1997), pp. 997–1007.

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

[59] Bo Sun et al. “Incremental model-based global dual heuristic programming with explicit analytical calculations applied to flight control”. In: *Engineering Applications of Artificial Intelligence* 89 (Mar. 2020). DOI: 10.1016/j.engappai.2019.103425.

[60] Ye Zhou et al. “Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback”. In: *Journal of Guidance, Control, and Dynamics*. Vol. 40. 2. American Institute of Aeronautics and Astronautics Inc., 2017, pp. 489–496. DOI: 10.2514/1.G001762.

[61] Bo Sun et al. “Incremental model-based heuristic dynamic programming with output feedback applied to aerospace system identification and control”. In: *CCTA 2020 - 4th IEEE Conference on Control Technology and Applications*. Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 366–371. DOI: 10.1109/CCTA41146.2020.9206261.

[62] 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. DOI: 10.1016/j.conengprac.2017.12.011.

[63] Shangtong Zhang et al. “Breaking the Deadly Triad with a Target Network”. In: (Jan. 2021). URL: http://arxiv.org/abs/2101.08862.

[64] Hado van Hasselt et al. “Deep Reinforcement Learning and the Deadly Triad”. In: (Dec. 2018). URL: http://arxiv.org/abs/1812.02648.

[65] Florian E. Dorner. “Measuring Progress in Deep Reinforcement Learning Sample Efficiency”. In: (Feb. 2021). URL: http://arxiv.org/abs/2102.04881.

[66] Fabio Muratore et al. “Assessing Transferability from Simulation to Reality for Reinforcement Learning”. In: (July 2019). DOI: 10.1109/TPAMI.2019.2952353. URL: http://arxiv.org/abs/1907.04685%20http://dx.doi.org/10.1109/TPAMI.2019.2952353.

[67] Scott A. Miller et al. “A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking”. In: *Eurasip Journal on Advances in Signal Processing* 2009 (2009). DOI: 10.1155/2009/724597.

[68] Leslie Pack Kaelbling et al. “Planning and acting in partially observable stochastic domains Kaelbling 1998”. In: *Artificial Intellegence* 101 (1998).

[69] Robert Kirk et al. “A Survey of Zero-shot Generalisation in Deep Reinforcement Learning”. In: (2023). DOI: 10.1613/jair.1.14174. URL: http://arxiv.org/abs/2111.09794%20http://dx.doi.org/10.1613/jair.1.14174.

[70] Karl Cobbe et al. “Quantifying Generalization in Reinforcement Learning”. In: (Dec. 2018). URL: http://arxiv.org/abs/1812.02341.

[71] Alex Nichol et al. “Gotta Learn Fast: A New Benchmark for Generalization in RL”. In: (Apr. 2018). URL: http://arxiv.org/abs/1804.03720.

[72] Karl Cobbe et al. “Leveraging Procedural Generation to Benchmark Reinforcement Learning”. In: (Dec. 2019). URL: http://arxiv.org/abs/1912.01588.

[73] Yiding Jiang et al. “On the Importance of Exploration for Generalization in Reinforcement Learning”. In: (June 2023). URL: http://arxiv.org/abs/2306.05483.

[74] Shreyansh Daftry et al. “Learning Transferable Policies for Monocular Reactive MAV Control”. In: (Aug. 2016). URL: http://arxiv.org/abs/1608.00627.

[75] Yevgen Chebotar et al. “Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience”. In: (Oct. 2018). URL: http://arxiv.org/abs/1810.05687.

[76] Dario Amodei et al. “Concrete Problems in AI Safety”. In: (June 2016). URL: http://arxiv.org/abs/1606.06565.

[77] Abhishek Gupta et al. “Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention”. In: (Apr. 2021). URL: http://arxiv.org/abs/2104.11203.

[78] R. Konatala et al. “Verification & Validation (V&V) of Reinforcement Learning based Online Adaptive Flight Control Laws on CS-25 Class Aircraft”. In: *EuroGNC*. Bristol, June 2024.

[79] C. A. A. M. Van Der Linden. *DASMAT-Delft University Aircraft Simulation Model and Analysis Tooi A Matlab/Simulink Environment for Flight Dynamics and Control Analysis*. Delft University Press, 1998.

[80] David J. Richter et al. *A Review of Reinforcement Learning for Fixed-Wing Aircraft Control Tasks*. 2024. DOI: 10.1109/ACCESS.2024.3433540.

[81] Eivind Bohn et al. “Data-Efficient Deep Reinforcement Learning for Attitude Control of Fixed-Wing UAVs: Field Experiments”. In: *IEEE Transactions on Neural Networks and Learning Systems* 35.3 (Mar. 2024), pp. 3168–3180. DOI: 10.1109/TNNLS.2023.3263430.

[82] Jung Ho Bae et al. “Deep Reinforcement Learning-Based Air-to-Air Combat Maneuver Generation in a Realistic Environment”. In: *IEEE Access* 11 (2023), pp. 26427–26440. DOI: 10.1109/ACCESS.2023.3257849.

[83] Chengyuan Xu et al. “Karting racing: A revisit to PPO and SAC algorithm”. In: *Proceedings - 2021 International Conference on Computer Information Science and Artificial Intelligence, CISAI 2021*. Institute of Electrical and Electronics Engineers Inc., 2021, pp. 310–316. DOI: 10.1109/CISAI54367.2021.00066.

[84] Abu Jafar Md Muzahid et al. “Comparison of PPO and SAC Algorithms towards Decision Making Strategies for Collision Avoidance among Multiple Autonomous Vehicles”. In: *Proceedings - 2021 International Conference on Software Engineering and Computer Systems and 4th International Conference on Computational Science and Information Management, ICSECS-ICOCSIM 2021*. Institute of Electrical and Electronics Engineers Inc., Aug. 2021, pp. 200–205. DOI: 10.1109/ICSECS52883.2021.00043.

[85] Supriya Seshagiri et al. “An Empirical Study of On-Policy and Off-Policy Actor-Critic Algorithms in the Context of Exploration-Exploitation Dilemma”. In: *Proceedings of the 2023 International Conference on Emerging Techniques in Computational Intelligence, ICETCI 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 238–243. DOI: 10.1109/ICETCI58599.2023.10331400.

[86] J A Mulder et al. *Flight Dynamics Lecture Notes AE3202*. Control and Simulation Division, Delft University of Technology, 2013.

[87] Josh Achiam. *OpenAI Spinning Up Documentation*. 2025. URL: https://spinningup.openai.com/en/latest/index.html.

[88] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22 (2021), pp. 1–8.

[89] MathWorks. *Reinforcement Learning Toolbox™ User’s Guide R2024b*. MathWorks, 2024. URL: www.mathworks.com.

[90] J. Bergstra et al. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems 24 (NIPS 2011)*. 2011.

[91] Steven Bohez et al. “Value constrained model-free continuous control”. In: (Feb. 2019). URL: http://arxiv.org/abs/1902.04623.

[92] M C De Visser et al. *Identification of a Cessna Citation II Model Based on Flight Test Data*. Tech. rep. MSc thesis, Delft University of Technology, 2017.

[93] Antonin Raffin et al. “Smooth Exploration for Robotic Reinforcement Learning”. In: *Conference on Robot Learning*. 2021.

A

Aircraft Parameters

Table A.1: The terms behind the symbols in the state-space representation of the equations of motions. Taken from [86].

	$x\ldots$	$z\ldots$	$m\ldots$
u	$\frac{V}{\bar{c}}\frac{C_{X_u}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_u}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_u}+C_{Z_u}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
α	$\frac{V}{\bar{c}}\frac{C_{X_\alpha}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_\alpha}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_\alpha}+C_{Z_\alpha}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
θ	$\frac{V}{\bar{c}}\frac{C_{Z_0}}{2\mu_c}$	$-\frac{V}{\bar{c}}\frac{C_{X_0}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$-\frac{V}{\bar{c}}\frac{C_{X_0}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
q	$\frac{V}{\bar{c}}\frac{C_{X_q}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_q}+C_{m_{\dot{\alpha}}}\frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
δ_e	$\frac{V}{\bar{c}}\frac{C_{X_{\delta_e}}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_{\delta_e}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_{\delta_e}}+C_{Z_{\delta_e}}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$
δ_t	$\frac{V}{\bar{c}}\frac{C_{X_{\delta_t}}}{2\mu_c}$	$\frac{V}{\bar{c}}\frac{C_{Z_{\delta_t}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}$	$\frac{V}{\bar{c}}\frac{C_{m_{\delta_t}}+C_{Z_{\delta_t}}\frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2}$

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

Table A.2: Symmetric and asymmetric stability and control derivatives for the Cessna Ce500 'Citation', at Cruise. Taken from [86].

B

Project Plan

Matan Neumark

Project Start: Tue, 19/11/2024

Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END
Literature Review				
Kick-off meeting	Matan + supervisors	100%	19/11/24	19/11/24
Initial research questions	Matan	100%	20/11/24	22/11/24
Literature Review	Matan	100%	25/11/24	20/12/24
Write Literature Review chapter	Matan	100%	12/12/24	5/1/25
Submit proposal	Matan	100%	6/1/25	6/1/25
Planning for research phase 1 and 2	Matan	100%	8/1/25	9/1/25
Research proposal review	Matan + supervisors	100%	13/1/25	13/1/25
Research Phase 1				
SAC implementation (longitudinal, pitch rate)	Matan	100%	14/1/25	28/2/25
SAC architecture design	Matan	100%	14/1/25	17/1/25
SAC implementation (w/ CitAST)	Matan	100%	20/1/25	14/2/25
MATLAB RL Toolbox (RLTB) implementation	Matan	100%	15/2/25	25/2/25
Reward function design	Matan	100%	15/2/25	25/2/25
Design a PID baseline	Matan	100%	26/2/25	28/2/25
Testing	Matan	100%	3/3/25	14/3/25
SAC validation (learning performance)	Matan	100%	3/3/25	7/3/25
Nominal flight conditions (vs PID and RLTB)	Matan	100%	10/3/25	14/3/25
Off-nominal flight conditions (vs PID and RLTB)	Matan	100%	10/3/25	14/3/25
Various tasks (vs PID baseline and RLTB)	Matan	100%	10/3/25	14/3/25
writing Preliminary Analysis chapter	Matan	100%	10/3/25	21/3/25
Submit mid-term deliverable	Matan	100%	24/3/25	24/3/25
Mid-term Review	Matan + supervisors	100%	28/3/25	28/3/25
Research Phase 2				
Extend RL controller to pitch attitude control	Matan	100%	31/3/25	17/4/25
Sensor and actuator dynamics	Matan	100%	18/4/25	30/4/25
Adept reward function	Matan	100%	1/5/25	4/5/25
Develop LSTM actor-critic	Matan	100%	5/5/25	16/5/25
Extend PID baseline to pitch attitude control	Matan	100%	17/5/25	22/5/25
Validation & Verification	Matan	100%	23/5/25	26/5/25
Testing (LSTM vs FF vs PID)	Matan	100%	27/5/25	14/6/25
Sensitivity analysis	Matan	100%	27/5/25	6/6/25
CitAST vs linear dynamics	Matan	100%	7/6/25	14/6/25
Report writing	Matan	100%	15/6/25	6/7/25
Submit thesis draft	Matan	100%	7/7/25	7/7/25
Green Light review	Matan + supervisors	100%	14/7/25	14/7/25
Research Dissemination				
Request examination	Matan	100%	14/7/25	14/7/25
Implement feedback	Matan	100%	21/7/25	1/8/25
Research portfolio submission	Matan	100%	6/8/25	6/8/25
Defence presentation prep	Matan	0%	11/8/25	25/8/25
Thesis defence	Matan + supervisors	0%	26/8/25	26/8/25
Insert new rows ABOVE this one				

Vacation according to the academic calendar

Milestone

Weekend

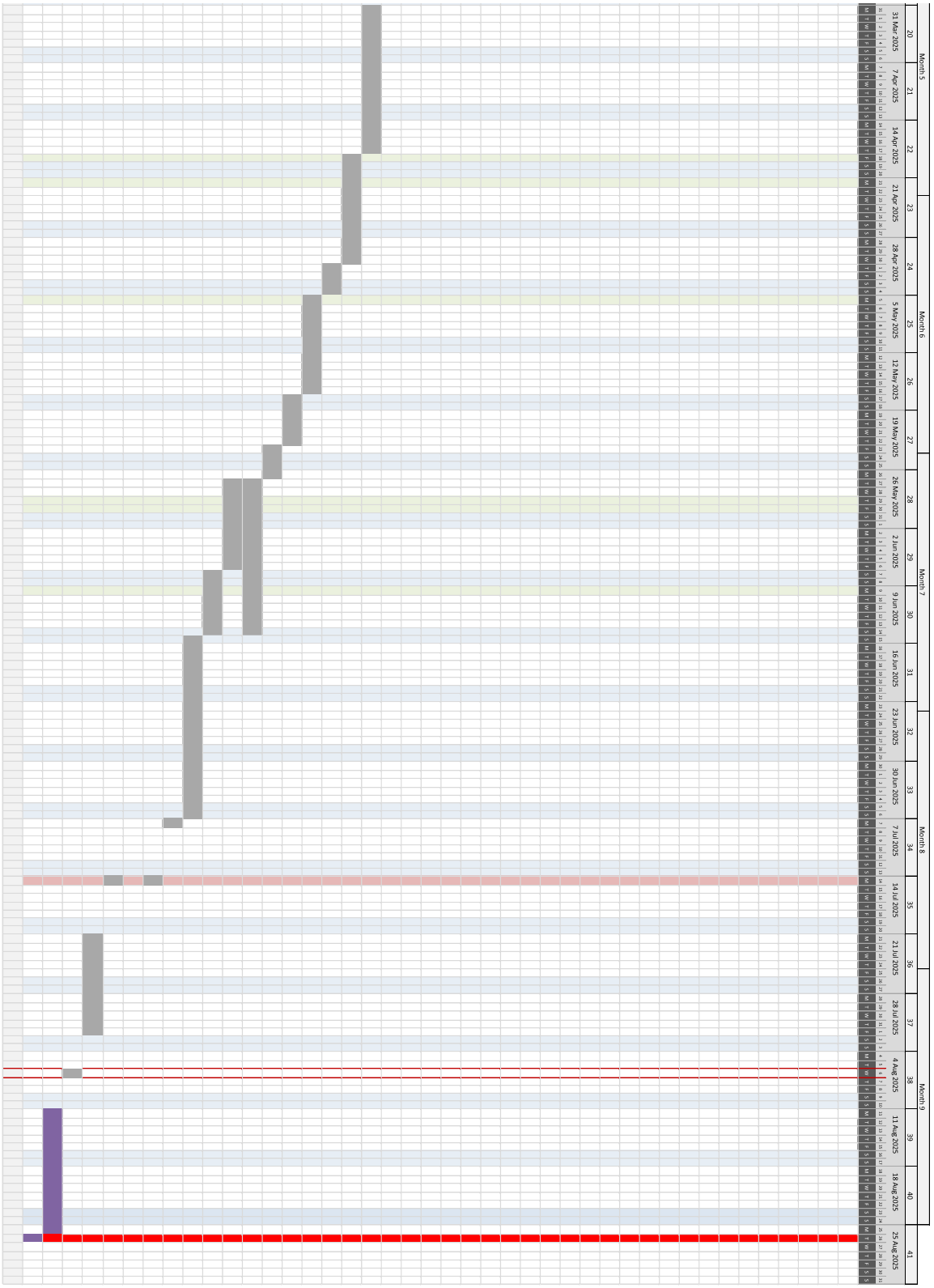


Figure B.1: Gantt chart of the thesis activities.