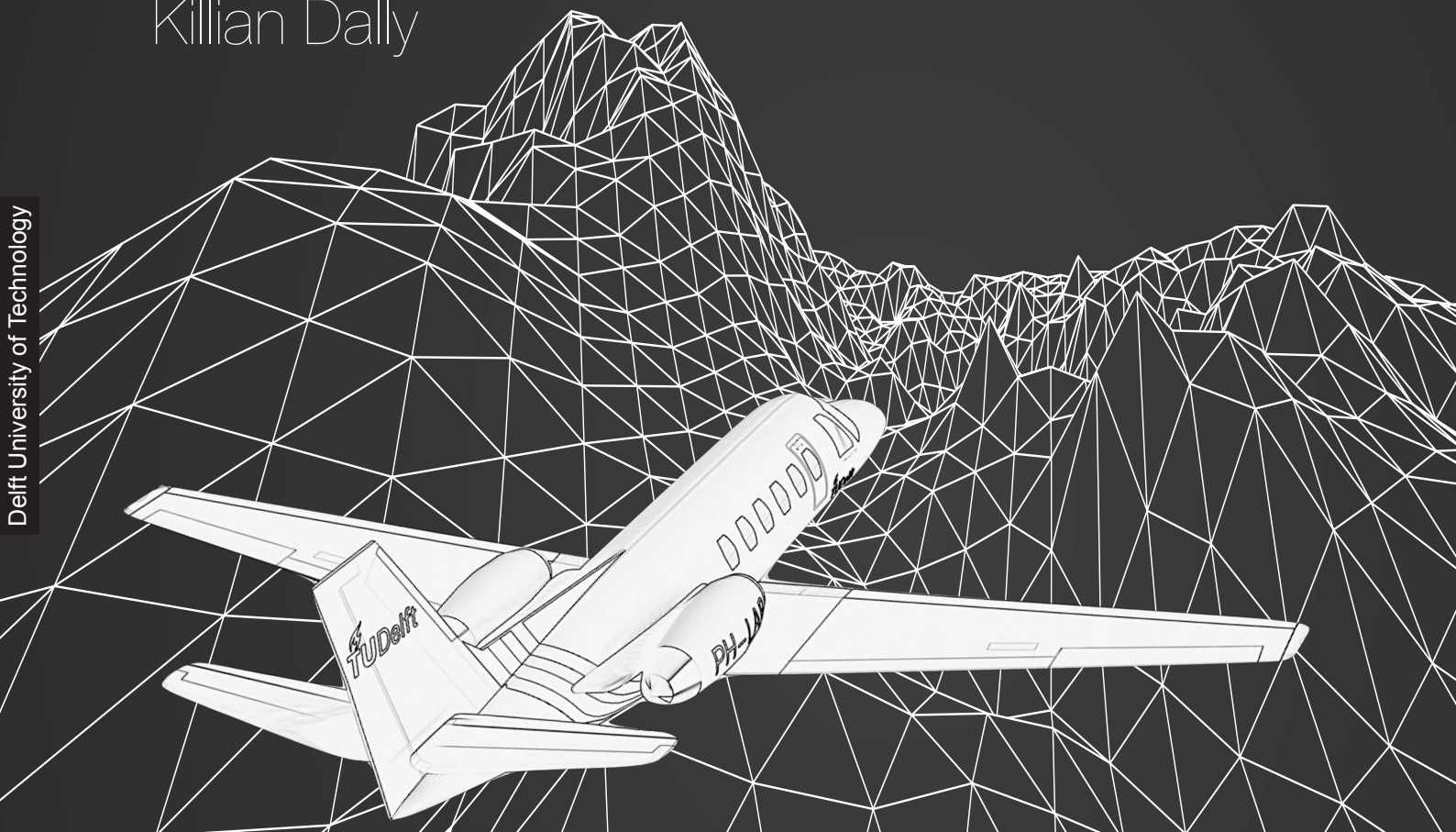
Deep Reinforcement Learning for Flight Control

Fault-Tolerant Control for the PH-LAB

Killian Dally



Delft University of Technology

TUDelft

Deep Reinforcement Learning for Flight Control

Fault-Tolerant Control for the PH-LAB

by

Killian Dally

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on February 24, 2021 at 14:00.

Student number:	4553373	
Project duration:	May 12, 2020 – February 24, 2021	
Thesis committee:	Dr. ir. M.M. van Paassen,	TU Delft, chair
	Dr. ir. E. van Kampen,	TU Delft, supervisor
	Dr. S.J. Hulshoff	TU Delft, external examiner
	B. Sun, MSc.,	TU Delft

TU Delft

Preface

Within the next decade, we will witness an exponential increase in the number of flying machines above our heads, from the growing civil aviation market to the advent of personal flying vehicles. This research makes use of the latest advancements in artificial intelligence and deep learning to enhance the safety of flight control systems and contribute towards making aircraft fully autonomous. In an effort to foster future developments in this field, the code developed as part of this research with its installation instructions is available publicly.[1]

With this thesis, I am concluding a five-year chapter of my life marked with the contribution of several individuals to whom I would like to express my gratitude. First and foremost, I would like to thank Dr. ir. Erik-Jan van Kampen for his continued support and guidance throughout this research. Your informed suggestions and constructive feedback have fundamentally shaped the outcome of this research. I am also grateful to the numerous and remarkable professors that, throughout my Bachelor's and Master's degrees, transmitted the so-called *Delft approach* to engineering that will be invaluable in my future career. This journey would not have been possible without the support of my roommates and friends in Delft, alongside whom I learned so much. From our frequent late-night work to countless adventures, to reshaping the world through our endless discussions, I will always cherish these memories. Thank you to all my family in France and to Maika in Yokohama for your relentless encouragements, much needed in these grim times of global pandemic. I would particularly like to thank my parents for devoting their utmost attention and effort to help me achieve my goals. This is all thanks to you.

Killian Dally
Delft, February 2021

[1] `https://github.com/kdally/fault-tolerant-flight-control-drl`

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

$\mathcal{A}$ Action space.

$\mathcal{S}$ State space.

N Number of steps in one episode.

t Time step.

$\mathbf{s}_t$ State vector at time step t.

$\mathbf{a}_t$ Action vector at time step t.

$\tilde{r}_t$ Instanteanous reward at time step t.

R_t Discounted return at time step t.

γ Discount factor.

ϵ Frequency at which actions are chosen randomly.

$\mathcal{P}\{x|y\}$ Probability of x occuring given y.

$\pi(\mathbf{a}_t|\mathbf{s}_t)$ Stochastic policy, probability of choosing action $\mathbf{a}_t$ given state $\mathbf{s}_t$.

$\mathbb{E}_\pi\{R_t|\mathbf{s}_t\}$ Expectation of the discounted return R_t given state $\mathbf{s}_t$ following policy π.

$V^\pi(\mathbf{s}_t)$ State value function at state $\mathbf{s}_t$ following policy π.

$V(\mathbf{s}_t)$ State value function estimate at state $\mathbf{s}_t$ following policy π.

$V^*(\mathbf{s}_t)$ State value function at state $\mathbf{s}_t$ following optimal policy π^*.

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ Action-state value function at state $\mathbf{s}_t$, choosing $\mathbf{a}_t$ and following policy π thereafter.

$Q(\mathbf{s}_t, \mathbf{a}_t)$ Action-state value function estimate at state $\mathbf{s}_t$, choosing $\mathbf{a}_t$ and following policy π thereafter.

$Q^*(\mathbf{s}_t, \mathbf{a}_t)$ Action-state value function at state $\mathbf{s}_t$, choosing $\mathbf{a}_t$ and following optimal policy π^* thereafter.

$A^\pi(\mathbf{s}_t, \mathbf{a}_t)$ Advantage function at state $\mathbf{s}_t$, choosing $\mathbf{a}_t$ and following policy π thereafter.

$A(\mathbf{s}_t, \mathbf{a}_t)$ Advantage function estimate at state $\mathbf{s}_t$, choosing $\mathbf{a}_t$ and following policy π thereafter.

$A^*(\mathbf{s}_t, \mathbf{a}_t)$ Advantage function at state $\mathbf{s}_t$, choosing $\mathbf{a}_t$ and following optimal policy π^* thereafter.

J Objective function.

L Loss function.

λ Learning rate.

$\mathcal{D}$ Memory buffer.

$\mathcal{B}$ Minibatch, subset of the memory buffer.

$\pi_{\boldsymbol{\theta}}$ Parameterized policy approximation.

$\boldsymbol{\theta}$ Policy parameter vector.

$\bar{\boldsymbol{\theta}}$ Target policy parameter vector.

$V_{\mathbf{k}}(\mathbf{s}_t)$ Parameterized state value function approximation.

$Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t)$ Parameterized action-state value function approximation.

$A_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t)$ Parameterized advantage function approximation.

$\mathbf{k}$ Value function parameter vector.

$\bar{\mathbf{k}}$ Target value function parameter vector.

ρ Probability ratio between the proposed and the current policy estimates.

KL Kullback–Leibler divergence constraint.

δ Kullback–Leibler divergence bound.

$\mathcal{H}$ Entropy.

$\bar{\mathcal{H}}$ Target entropy.

η Temperature hyperparameter trading off entropy and the expected return.

$\boldsymbol{\xi}$ Noise vector sampled from the Standard Normal Distribution.

τ Smoothing factor.

l Number of hidden units.

p Roll rate.

q Pitch rate.

r Yaw rate.

θ Pitch angle.

θ^R Reference pitch angle.

$\Delta\theta^R$ Reference pitch angle increment.

Φ Roll angle.

Φ^R Reference roll angle.

Ψ Yaw angle.

V Total air speed.

V_0 Initial total air speed.

γ Flight path angle.

α Angle of attack.

β Sideslip angle.

β^R Reference sideslip angle.

h Altitude.

h^R Reference altitude.

h_0 Initial altitude.

Δh Altitude tracking error.

δ_a Aileron deflection.

$\Delta\delta_a$ Aileron deflection increment.

δ_e Elevator deflection.

$\Delta\delta_e$ Elevator deflection increment.

δ_r Rudder deflection.

$\Delta\delta_r$ Rudder deflection increment.

x Aircraft state vector.

u Control input vector.

Δu Control input vector increment.

e Tracking error vector.

$n_{\mathbf{e}}$ Error vector dimension.

c Cost vector associated with the error vector.

C_L Lift coefficient.

C_D Drag coefficient.

C_m Pitching moment coefficient.

$C_{L_{\delta_e}}$ Lift coefficient derivative wrt. elevator deflection.

$C_{D_{\delta_e}}$ Drag coefficient derivative wrt. elevator deflection.

$C_{m_{\delta_e}}$ Pitching moment coefficient derivative wrt. elevator deflection.

$\boldsymbol{\sigma}$ Sample Standard Deviation vector.

$\boldsymbol{\mu}$ Mean vector.

List of Acronyms

ABS Adaptive Backstepping.

ACD Adaptive Critic Designs.

AD Action-Dependent.

ADDHP Action-Dependent Dual Heuristic Programming.

ADGDHP Action-Dependent Global Dual Heuristic Programming.

ADHDP Action-Dependent Heuristic Dynamic Programming.

ADP Approximate Dynamic Programming.

ANDI Adaptive Nonlinear Dynamic Inversion.

CPI Conservative Policy Iteration.

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.

EMA Exponentially (Weighted) Moving Average.

FDD Fault Diagnosis and Detection.

GDHP Global Dual Heuristic Programming.

HDP Heuristic Dynamic Programming.

i.i.d. Independent and Identically-Distributed.

iADP Incremental Approximate Dynamic Programming.

IDHP Incremental Dual Heuristic Programming.

IFC Initial Flight Conditions.

IGDHP Incremental Global Dual Heuristic Programming.

IHDP Incremental Heuristic Dynamic Programming.

INDI Incremental Nonlinear Dynamic Inversion.

KL Kullback–Leibler.

LTI Linear Time Invariant.

MAE Mean Absolute Error.

MC Monte-Carlo Method.

MDP Markov Decision Process.

MIMO Multi-Input Multi-Output.

MSE Mean Squared Error.

nMAE normalized Mean Absolute Error.

NN (Artificial) Neural Network.

PI Proportional-Integral.

PID Proportional-Integral-Derivative.

PPO Proximal Policy Optimization.

ReLu Rectified Linear Unit.

RL Reinforcement Learning.

RLS Recursive Least-Squares.

SAC Soft Actor-Critic.

SGD Stochastic Gradient Descent.

SISO Single-Input Single-Output.

SSD Sample Standard Deviation.

TD Temporal Difference.

TD3 Twin-Delayed Deep Deterministic Policy Gradient.

TRPO Trust Region Policy Optimization.

UAV Unmanned Air Vehicle.

VTOL Vertical Take-Off and Landing.

1 Introduction

1.1. Background

The aviation industry has benefited from an impressive safety record over the 2008-2019 period, with an accident rate decreasing by 36% [30]. Nonetheless, fatal accidents regularly occur, and among fatalities of commercial flight accidents between 2009 and 2018, over 61% were caused by in-flight loss of control [29]. Novel automation techniques can play a large role in reducing this number by making flight control systems more fault-tolerant.

A significant development in the aviation industry regards the introduction of Vertical Take-Off and Landing (VTOL) personal flying vehicles. Promised to revolutionize the urban mobility market, several start-ups have already emerged, such as PAL-V, Lilium and SkyDrive, while established manufacturers including Airbus and Rolls-Royce are also developing and testing prototypes [51]. These vehicles should integrate a high level of automation and avoid catastrophic failures in dense urban environments to aim at achieving autonomous flight for the general public.

Both the need for increased safety in civil aviation and the development of autonomous VTOL vehicles require flight control automation techniques to be fault-tolerant.

Up until today, automatic flight control systems typically consisted of linear uncoupled gain-scheduled controllers relying on look-up tables to control coupled-dynamics non-linear systems [69]. The controllers were tuned using a linearized plant dynamics model at specific known operating points within the flight envelope. This approach required many linearization points to perform best, making it cumbersome while also possibly leaving certain regions of the flight envelope with higher coupling effects out of reach of the controller. Furthermore, because they were based on known plant dynamics, they could not deal with sudden changes in dynamics such as failures [56]. It is desired to replace this framework with a fault-tolerant controller that can reach difficult regions of the flight envelope.

Adaptive flight control can, without a gain tuning phase, adapt to unforeseen plant dynamics changes [15]. Common methods such as Adaptive Nonlinear Dynamic Inversion (ANDI) and Adaptive Backstepping (ABS) methods require, however, knowledge of the failed plant dynamics to ensure fault-tolerance [25, 41]. While recent incremental-type implementations have been proven successful in flight-testing on business jet aircraft, some model dependency remains [22, 33].

Reinforcement Learning (RL) is a bio-inspired machine learning framework that enables a controller to learn from interaction with an environment. An RL agent learns by trial-and-error, not relying on environment dynamics knowledge. Fault tolerance can be achieved using online learning through adaptive control when the agent learns a task while executing it. Another way to achieve fault-tolerance in RL is through robust control, making use of the high generalization power of RL agents, provided adequate offline learning. This makes RL a promising control framework for this research. In its original form, RL is a tabular method with discrete action and state spaces, well suited for gaming environments. In control applications, however, large discretization errors are not permitted and difficulties are encountered when trying to approximate continuous environments. The so-called curse of dimensionality refers to the exponentially-increasing computational complexity of making action and state spaces larger [6].

Continuous spaces can be dealt with using function approximators, typically (Artificial) Neural Networks (NNs). The most common implementations are based on actor-critic structures, where the actor chooses an action to perform and the critic gives feedback on that action [72]. A common approach to actor-critic RL structures is Approximate Dynamic Programming (ADP). It has been applied to coupled-dynamics flight control for a business jet aircraft [18] and a helicopter [16] in simulations, yet both based on a known model of plant dynamics and thereby limiting their applicability to a wide range of unforeseen failures. Recent research using online incremental model-learning ADP techniques (iADP) has managed to provide adaptive control while eliminating all model dependence. The first implementation known as Incremental Heuristic Dynamic Programming (IHDP) was proposed by [83] for a simple angle-of-attack and pitch rate controller. The introduction of Incremental Dual Heuristic Programming (IDHP) demonstrated fault-tolerance on simple failure cases [84]. Using IDHP, [27] and [36] showed that the inner-loop dynamics of a business jet aircraft could be controlled in the presence of pre-tuned outer-loop PID controllers. As the controller's longitudinal and lateral motions were decoupled, severe failure cases where the coupling effects become too dominant were not tested. Coupled-dynamics control was enabled on the same aircraft in [35] with IDHP, yet limited to body rate control and therefore not removing the need for model-based outer-loop PID controllers. When the outer-loop PID controllers are replaced with IDHP agents for longitudinal control in [37], a failure rate of 24% suggests that this method alone does not yet have the reliability and sample efficiency to control online the outer-loop coupled dynamics of 6-DoF systems.

A novel type of actor-critic structure known as the field of Deep Reinforcement Learning (DRL) has gained popularity in recent years. It went from solving relatively simple Atari games in [46] to winning against humans at what is considered the most complex board game of all times, Go [66]. DRL was extended to complex control applications with the Deep Deterministic Policy Gradient (DDPG) algorithm thanks to its continuous control abilities [40]. Learning for DRL methods is usually conducted offline, thus requires a simulation model to learn from interaction but remains model-free as it treats the plant as a black box. DDPG was implemented on difficult robotics tasks such as bipedal walking with obstacles [68]. DDPG flight control applications have been limited to small-scale UAVs, for quadcopters [28, 34, 67] and unmanned flying-wing [74]. State-of-the-art algorithms such as Proximal Policy Optimization (PPO), Twin-Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) have aimed at reducing DDPG's learning instability and overestimation bias while keeping its high-sample efficiency advantage [19, 23, 60]. PPO and SAC's quick policy convergence were used for quadcopter applications [5, 42] and for unmanned flying-wing aircraft [8]. Despite DRL's ability to learn highly complex tasks, it has not been tested on coupled-dynamics flight control tasks for fixed-wing aircraft. Furthermore, due to its relatively long training time, online learning is expected to be difficult. For this reason, fault-tolerance is mainly achieved through robust control.

This research intends to advance state-of-the-art fault-tolerant flight control techniques by developing a model-free coupled-dynamics flight controller for a jet aircraft that can withstand multiple types of failures. This research proposes identifying the most promising RL method for this application between iADP and DRL, and subsequently testing it on a high-fidelity simulation model of the Cessna Citation seen in Figure 1.1.

Figure 1.1: Cessna Citation PH-LAB research business jet aircraft operated by Delft University of Technology. Its fly-by-wire flight control system allows for the use of experimental flight controllers.[1]

[1] Image from C. v. Grinsven (with permission).

1.2. Research Aim

The research objective of this thesis is presented below. For the scope of this research, the desired flight controller will be built as a proof-of-concept for a business jet aircraft.

Research Objective

To advance self-learning techniques for model-free coupled-dynamics flight control systems applied to fixed-wing aircraft with the purpose of enabling fault tolerance to unexpected failures, by means of developing a reinforcement learning flight controller for a business jet aircraft.

From this objective, several research questions have been identified.

Research Questions

Q1 Which RL framework can maximize the tracking performance in the control of a high-dimensional coupled-dynamics system in the presence of faults?

- *Q1.1* What are requirements on the RL algorithm in terms of its characteristics and training conditions?
- *Q1.2* Which RL algorithm satisfying the requirements has the highest potential to capture the full dynamics of a high-dimensional system?
- *Q1.3* How successful is the proposed algorithm applied to a simple coupled-dynamics system in terms of tracking error and stability?
- *Q1.4* To what degree is the proposed algorithm able to deal with unforeseen actuator and component faults on a simple coupled-dynamics system?

Q2 How can the proposed RL agent be integrated with the Cessna Citation 500?

- *Q2.1* At what control level should the RL controller be implemented, and what are the states and inputs to be controlled?
- *Q2.2* What simplifications to the simulation model can be made while keeping its relevant characteristics?
- *Q2.3* What type of task and maneuver is the system subject to experience?

Q3 What is the overall performance of the RL controller on a Cessna Citation 500 model?

- *Q3.1* How does it perform on coupled maneuvers in terms of tracking error and stability?
- *Q3.2* How does the controller adapt itself to unforeseen faults to the aircraft?
- *Q3.3* What is the sample efficiency of the control system?

1.3. Structure

This report is structured as follows. Part I contains a scientific article that concisely presents the main results of this research. The foundations of the chosen RL method are presented in section 2.2 and the controller structure in section 2.3. The results on the non-failed and failed system are presented in section 2.4, along with additional robustness tests. The gained insights are discussed in the conclusion in section 2.5.

Part II contains the preliminary research work of this thesis. Chapter 3 reviews the foundations and state-of-the-art algorithms of continuous-control RL methods. The most promising method for this research is then tested on a simple coupled-dynamics system as part of the preliminary analysis in Chapter 4.

In Part III, additional results to the scientific article including robustness tests in Chapter 5 and the

response to control disturbances in Chapter 6 are presented. A sensitivity analysis on the reward function is carried out in Chapter 7 and an alternative controller structure is proposed in Chapter 8. Lastly, verification and validation is performed on the high-fidelity simulation model and the controller in Chapter 9.

The thesis is wrapped-up in Part IV with the main conclusions and recommendations for further research.

I

Scientific Article

Soft Actor-Critic Deep Reinforcement Learning for Fault-Tolerant Flight Control

K. Dally*
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Fault-tolerant flight control faces challenges, as developing a model-based controller for each unexpected failure is unrealistic, and online learning methods can handle limited system complexity due to their low sample efficiency. In this research, a model-free coupled-dynamics flight controller for a jet aircraft able to withstand multiple failure types is proposed. An offline-trained cascaded Soft Actor-Critic Deep Reinforcement Learning controller is successful on highly coupled maneuvers, including a coordinated 40°-bank climbing turn with a normalized Mean Absolute Error of 2.64%. The controller is robust to six failure cases, including the rudder jammed at −15°, the aileron effectiveness reduced by 70%, a structural failure, icing and a backward c.g. shift as the response is stable and the climbing turn is completed successfully. Robustness to biased sensor noise, atmospheric disturbances, and to varying initial flight conditions and reference signal shapes is also demonstrated.

Nomenclature

$\mathbf{s}, \mathbf{a}$	=	environment state and actor action vectors
n, m	=	number of environment states and actor actions, respectively
$\tilde{r}(\mathbf{s}, \mathbf{a})$	=	instantaneous reward function
$t, \Delta t, N$	=	discrete time-step subscript, sample time and number of samples
$f(\mathbf{s}, \mathbf{a})$	=	state transition function
$\pi, \pi^*, \pi_{\boldsymbol{\theta}}$	=	policy, optimal policy and parameterized policy approximation
$Q^{\pi}, Q_{\mathbf{k}}$	=	action-state value function (Q-function) and parameterized Q-function approximation
$\boldsymbol{\theta}, \mathbf{k}, \bar{\mathbf{k}}$	=	policy, Q-function and target Q-function parameter vectors
γ	=	discount factor
η	=	temperature parameter
$\mathcal{H}, \tilde{\mathcal{H}}$	=	entropy and target entropy
$L(x), J(x)$	=	loss and objective functions for variable x
$\mathcal{D}, \mathcal{B}$	=	memory buffer and minibatch (subset of the memory buffer)
$\boldsymbol{\xi}$	=	noise vector
$\mathcal{N}$	=	standard normal distribution
$\boldsymbol{\mu}, \boldsymbol{\sigma}$	=	mean and Sample Standard Deviation (SSD) vectors
λ	=	learning rate
τ	=	smoothing factor
$\mathbf{x}, \mathbf{u}$	=	aircraft state and control input vectors
$p, q, r, \phi, \theta, \psi$	=	roll, pitch and yaw rates, and roll, pitch and yaw angles
V, α, β	=	total airspeed, angle-of-attack and sideslip angle
$h, \Delta h$	=	altitude and altitude tracking error
$\delta_{\mathrm{e}}, \delta_{\mathrm{a}}, \delta_{\mathrm{r}}$	=	control surface deflections (elevator, aileron, rudder)
R	=	reference signal superscript
$\Delta\mathbf{u}, \Delta\theta^R$	=	control input and reference pitch angle increments
$\mathbf{e}, \mathbf{c}$	=	error and error cost vectors
l	=	number of units per hidden layer
C_L, C_D, C_m	=	lift, drag and pitching moment coefficients
T, A	=	signal period and amplitude
$\odot$	=	element-wise multiplication

*M.Sc. Student, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology.

I. Introduction

In-flight loss of control was the cause of 61% of commercial flight accident casualties between 2009 and 2018, indicating the need for more fault-tolerant control systems [1]. At the same time, the advent of personal air vehicles in dense urban areas calls for the development of autonomous flight controllers that can withstand multiple types of failures.

Until now, flight control automation techniques have relied on gain-scheduling to switch between parallel linear controllers tuned at specific known operating points [2]. Because of their reliance on known plant dynamics, they cannot deal with sudden changes in dynamics such as failures. Model-free adaptive and intelligent control techniques offer the possibility of replacing this inconvenient controller structure with a more general and fault-tolerant approach.

Reinforcement Learning (RL) is a bio-inspired machine learning framework for intelligent control that can offer fault-tolerance. An RL agent learns through trial-and-error by interacting with the plant, also known as the environment [3]. In its original form, RL was a tabular method with discrete action and state spaces, well suited for gaming environments. To combat the curse of dimensionality encountered when making the action and state spaces larger [4], function approximators, typically Neural Networks (NNs), can be used with the actor-critic agent structure to enable continuous control.

A common approach to actor-critic RL structures is Approximate Dynamic Programming (ADP). It has been applied to coupled-dynamics flight control for a business jet aircraft [5] and a helicopter [6] in simulations, yet all based on a known model of plant dynamics and thereby limiting their applicability to a wide range of unforeseen failures. Recent research using online incremental model-learning ADP techniques (iADP) has managed to provide adaptive control while eliminating all model dependence. Longitudinal control was proposed first by [7], and then the introduction of Incremental Dual Heuristic Programming (IDHP) demonstrated fault-tolerance on simple failure cases [8]. Using IDHP for coupled-dynamics body rate control, [9] showed that that a business jet aircraft could be controlled successfully. It was extended to altitude and attitude control with pre-tuned PID controllers in [10] and [11], yet as the longitudinal and lateral motions were decoupled severe failures cases where the coupling effects become too dominant were not tested. When the outer-loop PIDs were replaced with IDHP agents for longitudinal control in [12], a failure rate of 24% suggested that this method alone does not yet have the reliability and sample efficiency to control online the outer-loop coupled dynamics of 6-DoF systems.

A novel approach to actor-critic structures has recently been introduced, known as the field of Deep Reinforcement Learning (DRL). Enabling end-to-end offline learning, high-dimensional input spaces such as images were used to surpass human performance on multiple Atari games, as shown by [13] with Deep Q-Network (DQN) for discrete action spaces. DRL was extended to control applications by [14] with the off-policy Deep Deterministic Policy Gradient (DDPG) algorithm thanks to its continuous control abilities. DDPG flight control applications have been limited to small-scale flying-wings [15] and quadcopters [16–18]. On-policy RL algorithm Proximal Policy Optimization (PPO), proposed by [19], has aimed at reducing DDPG's learning instability and showed improved policy convergence for a quadcopter UAV in [20] and for an unmanned flying-wing aircraft in [21]. State-of-the-art Twin-Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) have focused on reducing DDPG's critic overestimation bias [22, 23]. Unlike TD3, Soft Actor-Critic (SAC) uses a stochastic policy which was shown to encourage exploration and increase sample efficiency [23]. While it is unclear from the authors of TD3 and SAC which performed best, an independent study found that SAC outperformed TD3 in terms of sample efficiency on 4 out of 5 complex control tasks [24]. On a quadcopter control task, it recovered from unfavorable initial conditions in [25], demonstrating high robustness. SAC is identified as the most promising DRL algorithm for this flight control task. Despite DRL's ability to learn highly complex tasks, it has not been tested on coupled-dynamics flight control tasks for fixed-wing aircraft. Furthermore, due to its relatively long training time, online learning is expected to be difficult. For this reason, fault-tolerance is mainly achieved through robust control. SAC's generalization ability to multiple types of failures is unknown at this point and is to be better understood.

The contribution of this research is to advance state-of-the-art fault tolerant flight control methods by developing a model-free coupled-dynamics flight controller for a jet aircraft that can withstand multiple types of unexpected failures. This research explores the use of DRL controllers for CS-25 class aircraft generally only employed for small-scale UAVs. For this research, a high-fidelity simulation model of the Cessna Citation 500 will be used, paving the way for future test flights on the PH-LAB research aircraft thanks to its experimental fly-by-wire flight control system.

The foundations of RL and the SAC algorithm are explained in Section II, followed by a motivation for the controller design in Section III. The results are discussed in Section IV and the conclusions are presented in Section V.

II. Fundamentals

This section introduces the learning framework used in this research, actor-critic RL and the RL algorithm at hand.

A. Reinforcement Learning Problem

The actor-critic RL framework is composed of an agent that applies action $\mathbf{a}_t \in \mathbb{R}^m$ on an environment with state $\mathbf{s}_t \in \mathbb{R}^n$ at discrete time step t. The next state is determined by a state-transition function unknown to the agent in Eq. (1). It is assumed to have the Markov property, which implies that the environment's history is fully explained by its present state. The agent chooses actions based on the actor's policy, a mapping from the state space $\mathbb{R}^n$ to the action space $\mathbb{R}^m$. If it is stochastic, the action is sampled as shown in Eq. (2). The environment gives a scalar reward $\tilde{r}(\mathbf{s}_t, \mathbf{a}_t) \in \mathbb{R}$ to the agent as a feedback of its action $\mathbf{a}_t$ at each time step. The goal is to learn a policy that maximizes the reward over all states.

A critic, defined by an action-state value function, or Q-function, is introduced in Eq. (3) to characterize how beneficial it is to be in a given state $\mathbf{s}_t$ in terms of future expected reward when taking action $\mathbf{a}_t$ and following the policy thereafter. A discount factor, γ, is used to trade-off immediate and future rewards. The episode comprises of N time steps.

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t) \quad (1) \qquad \mathbf{a}_t \sim \pi(\cdot \mid \mathbf{s}_t) \quad (2) \qquad Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \mathop{\mathbb{E}}_{\mathbf{a}_{t+i} \sim \pi} \left[\sum_{i=0}^{N} \gamma^i \tilde{r}(\mathbf{s}_{t+i}, \mathbf{a}_{t+i}) \mid \mathbf{s}_t, \mathbf{a}_t \right] \quad (3)$$

B. Soft Actor-Critic Algorithm

Soft Actor-Critic (SAC) is a novel off-policy DRL algorithm and an extension of the DDPG algorithm that aims at optimizing a stochastic policy [23]. Unlike DDPG's deterministic policy, a stochastic policy ensures better exploration and was found to be applicable to broader types of control tasks [17]. At evaluation time, the mean of the policy distribution is selected to make actions deterministic and ensure consistent performance.

1. Entropy

SAC adds an entropy term to the standard optimal policy expression in Eq. (4), which is a measure of the randomness in its probability distribution. Policy distributions more spread over the action space have higher entropy, which can be measured with the log-likelihood according to Eq. (5). The entropy is traded-off against future rewards with the temperature parameter η[†].

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\mathbf{a}_{t+i} \sim \pi} \left[\sum_{i=0}^{N} \gamma^i \Big(\tilde{r}(\mathbf{s}_{t+i}, \mathbf{a}_{t+i}) + \eta \mathcal{H}\big(\pi(\cdot \mid \mathbf{s}_{t+i})\big) \Big) \right] \quad \forall\, \mathbf{s}_t \in \mathbb{R}^m \tag{4}$$

$$\mathcal{H}(\pi(\cdot \mid \mathbf{s}_t)) = \mathop{\mathbb{E}}_{\mathbf{a}' \sim \pi} [-\log \pi(\mathbf{a}' \mid \mathbf{s}_t)] \tag{5}$$

In other words, this objective favors the most random policy that still achieves a high return. This creates an inherent exploration mechanism that also prevents premature convergence to local optima. Multiple control strategies that achieve a near-optimal reward are captured, allowing for more robustness to disturbances.

2. Critic

The Q-function critic is modeled as a feed-forward Deep Neural Network (DNN) with parameter vector $\mathbf{k}$. The standard Bellman equation is modified with the expression of the entropy found in Eq. (5) to obtain a recursive expression of the soft Q-function in Eq. (6).

$$Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t) = \mathop{\mathbb{E}}_{\mathbf{a}_t, \mathbf{a}_{t+1} \sim \pi_{\boldsymbol{\theta}}} \left[\tilde{r}(\mathbf{s}_t, \mathbf{a}_t) + \gamma \left(Q_{\mathbf{k}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \eta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1}) \right) \right]. \tag{6}$$

Given that SAC is an off-policy algorithm, transition samples $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ can be collected in a memory buffer $\mathcal{D}$ and reused at a later stage. This can help ensure training samples are more independent and identically distributed, an assumption of the update method of RL algorithms. By sampling a minibatch $\mathcal{B}$ from the memory buffer, a minibatch

[†]The symbol η is used as temperature parameter instead of α in [23], the original SAC paper, to avoid confusion with the angle-of-attack.

gradient update can be performed instead of the computationally inefficient and noisy stochastic gradient update of on-policy algorithms.

To increase learning stability, a target network with parameter $\bar{\mathbf{k}}$ for the Q-function is introduced to make the gradient update follow a more constant direction. From time to time, the target network is synchronized with the current value network with an exponentially weighted moving average as a soft update mechanism, such that the target network is a delayed version of the current value network regulated by smoothing factor τ. The target network is used in the mean squared Bellman error for the Q-function loss function shown in Eq. (7). In an effort to reduce DDPG's overestimation bias of the Q-function, SAC makes use of the double Q-function trick by learning two approximators and using a pessimistic bound over the two. Transition samples are contained in minibatch $\mathcal{B}$ but because SAC is off-policy, fresh actions $\mathbf{a}_{t+1}$ can be sampled from the current policy to compute the Q-function targets.

$$L_Q\left(\mathbf{k}_i, \mathcal{B}\right) = \mathop{\mathbb{E}}_{\substack{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \sim \mathcal{B} \\ \mathbf{a}_{t+1} \sim \pi_{\boldsymbol{\theta}}}} \left[\left(Q_{\mathbf{k}_i}(\mathbf{s}_t, \mathbf{a}_t) - \left(\tilde{r}(\mathbf{s}_t, \mathbf{a}_t) + \gamma \left(\min_{i=1,2} Q_{\bar{\mathbf{k}}_i}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \eta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1}) \right) \right) \right)^2 \right] \tag{7}$$

3. Policy

The policy, or actor, is modeled as an m-dimensional multivariate Gaussian distribution with a diagonal covariance matrix. Its actions are passed to a tanh squashing function to ensure they are defined on a finite bound. The mean vector $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and the covariance matrix, or, in this case, vector $\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}$, are estimated for each state by a DNN with parameter vector $\boldsymbol{\theta}$.

Unlike DDPG's deterministic policy, no target policy is needed as the policy's stochasticity has a smoothing effect. The stochasticity also means that the policy objective in Eq. (4) depends on the expectation over actions and is therefore non-differentiable. A reparameterization trick is proposed by the SAC authors in [23] using the known mean and standard deviation of the stochastic policy along with independent noise vector $\boldsymbol{\xi}$, and applying the squashing function, as shown in Eq. (8). A policy objective with an expectation over noise instead of actions and making use of the Q-function as an approximation of expected future rewards and entropy is introduced in Eq. (9).

$$\tilde{\mathbf{a}}_{\boldsymbol{\theta}}(\mathbf{s}_t, \boldsymbol{\xi}) = \tanh\left(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}_t) + \boldsymbol{\sigma}_{\boldsymbol{\theta}}(\mathbf{s}_t) \odot \boldsymbol{\xi}\right), \quad \boldsymbol{\xi} \sim \mathcal{N}(\vec{\mathbf{0}}, \vec{\mathbf{1}}) \tag{8}$$

$$J_\pi\left(\boldsymbol{\theta}, \mathcal{B}\right) = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim \mathcal{B} \\ \boldsymbol{\xi} \sim \mathcal{N}}} \left[\min_{i=1,2} Q_{\mathbf{k}_i}(\mathbf{s}_t, \tilde{\mathbf{a}}_{\boldsymbol{\theta}}(\mathbf{s}_t, \boldsymbol{\xi})) - \eta \log \pi_{\boldsymbol{\theta}}(\tilde{\mathbf{a}}_{\boldsymbol{\theta}}(\mathbf{s}_t, \boldsymbol{\xi}) \mid \mathbf{s}_t) \right] \tag{9}$$

4. Automatic Temperature Adjustment

SAC can be unstable with respect to temperature parameter η, so the latter was proposed to be controlled automatically in [26]. Optimal entropy is not constant throughout training as the need for exploration is expected to decrease with increasing training steps. A loss function $L\left(\eta\right)$ is introduced in Eq. (10) to dynamically find the lowest temperature that still ensures a certain minimum target entropy $\bar{\mathcal{H}}$ while maximizing the return. A good empirical value for the target entropy is found to be connected to the action space dimension with $\log \bar{\mathcal{H}} = -m$ [26].

$$L\left(\eta\right) = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim \mathcal{B} \\ \mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}}} \left[-\eta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) - \eta \bar{\mathcal{H}} \right] \tag{10}$$

5. Overview

An overview of the SAC framework is shown in Fig. 1. With the critic loss and policy objective functions, a pseudocode can be constructed as shown in Algorithm 1.

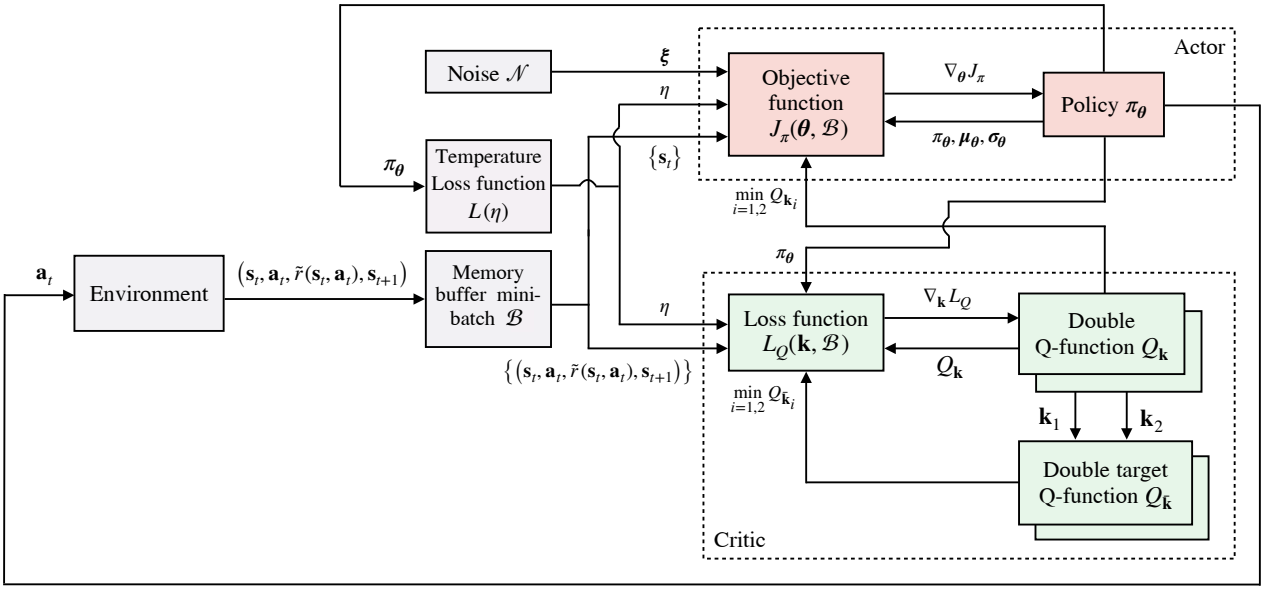


Fig. 1 SAC framework.

Algorithm 1: SAC. Adapted from [26].

Initialize $\boldsymbol{\theta}$, $\mathbf{k}_1$, $\mathbf{k}_2$ parameters for policy $\pi_{\boldsymbol{\theta}}$ and double Q-function $Q_{\mathbf{k}_{1,2}}$;
Set target parameters $\bar{\mathbf{k}}_1 \leftarrow \mathbf{k}_1$ and $\bar{\mathbf{k}}_2 \leftarrow \mathbf{k}_2$;
Initialize empty memory buffer $\mathcal{D}$, minibatch $\mathcal{B}$, learning rate λ and smoothing factor τ ;
Observe initial state $\mathbf{s}_0$;
for *each time step t* **do**
  Sample action $\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{s}_t)$;
  Observe next state and reward $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t), \tilde{r}(\mathbf{s}_t, \mathbf{a}_t)$;
  Store transition sample $(\mathbf{s}_t, \mathbf{a}_t, \tilde{r}(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})$ in $\mathcal{D}$;
  Sample a minibatch of transition samples $\mathcal{B} = \{(\mathbf{s}_t, \mathbf{a}_t, \tilde{r}(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ from $\mathcal{D}$;
  Update Q-function parameters: $\mathbf{k}_i \leftarrow \mathbf{k}_i - \lambda \nabla_{\mathbf{k}_i} \frac{1}{|\mathcal{B}|} \sum L_Q(\mathbf{k}_i, \mathcal{B})$ for $i = 1, 2$;
  Update policy parameter: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} \frac{1}{|\mathcal{B}|} \sum J_\pi(\boldsymbol{\theta}, \mathcal{B})$
  Update the temperature hyperparameter: $\eta \leftarrow \eta - \lambda \nabla_\eta L(\eta)$;
  Update target Q-function parameters: $\bar{\mathbf{k}}_i \leftarrow (1 - \tau)\mathbf{k}_i + \tau \bar{\mathbf{k}}_i$ for $i = 1, 2$;
end

III. Controller Design

With the SAC framework presented above, its integration with the flight controller is discussed in this section.

A. High-Fidelity Cessna Citation 500 Model

The system to be controlled in this application is a high-fidelity non-linear simulation model of the Cessna Citation 500 business jet aircraft. It was built with the Delft University Aircraft Simulation Model and Analysis Tool (DASMAT) based on flight data recorded on the PH-LAB research aircraft shown in Fig. 2. The model was validated in [27]. It is expected that the controller proposed in this research will be flight-tested on the PH-LAB at a later stage.

Fig. 2 Cessna Citation PH-LAB research aircraft.*

The full coupled-dynamics of the aircraft are to be controlled with a refresh rate of 100Hz. For this research, actuator dynamics are modeled with a low-pass filter and saturation limits, while ideal sensors are assumed. A yaw damper is already present on the aircraft. The aircraft is kept in a clean configuration for this simulation. The aircraft state and control input vectors are given in Eqs. (11) and (12), respectively. At the beginning of each simulation, the aircraft is untrimmed with null initial control inputs.

$$\mathbf{x} = [p, q, r, V, \alpha, \beta, \theta, \phi, \psi, h]^\intercal \tag{11}$$

$$\mathbf{u} = [\delta_\mathrm{e}, \delta_\mathrm{a}, \delta_\mathrm{r}]^\intercal \tag{12}$$

B. Interfacing

In reinforcement learning, the flight controller, the plant and the control input are known as the agent, the environment and the action, respectively.

A flight controller for automatic altitude and attitude tracking is to be built as part of this research so that it can be interfaced with existing navigation algorithms reviewed in [28]. To ensure compatibility with the experimental fly-by-wire system of the PH-LAB in future flight tests, the controller developed in this research will only command control surfaces while the airspeed is controlled with an independent PID auto-throttle. A flight control task for altitude, roll and sideslip angles tracking is proposed. Because of the difference in their dynamics, a more stable learning is expected by having the altitude and attitude controlled by two separate, cascaded controllers.

1. Attitude Control

An inner-loop SAC agent will track reference signals for the pitch, roll and sideslip angles, referred to with the R superscript. A reward function based on the clipped L1 norm of the error vector is proposed in Eq. (15). A cost vector $\mathbf{c}$, determined by trial and error, is associated with the tracked states in Eq. (14), where the sideslip angle is attributed a higher cost due to its generally low magnitude.

$$\mathbf{e}^{\mathrm{att}} = \left[\beta^R - \beta, \theta^R - \theta, \phi^R - \phi\right]^\intercal \tag{13}$$

$$\mathbf{c}^{\mathrm{att}} = \frac{6}{\pi}\,[1, 1, 4]^\intercal \tag{14}$$

$$\tilde{r}(\mathbf{e}^{\mathrm{att}}) = -\frac{1}{3}\left\|\mathrm{clip}\left[\mathbf{c}^{\mathrm{att}} \odot \mathbf{e}^{\mathrm{att}}, \vec{-1}, \vec{0}\right]\right\|_1 \tag{15}$$

Initial tests found that the aircraft control input was noisy when corresponding directly to the agent action. To smooth out the control input, the agent is set to command the control input increment $\mathbf{\Delta u}$ in Eq. (16). Because the tanh function squashes the action vector $\mathbf{a}^{\mathrm{att}}$ to $[-1, 1]^3$, it is mapped to the physical range of the control input increment in Eq. (17), chosen as a hundredth of actuator limits to prevent sharp variations.

$$\mathbf{u}_t = \mathbf{u}_{t-1} + \mathbf{\Delta u}_t \tag{16}$$

$$\mathbf{\Delta u} = \mathbf{\Delta u}_{\min} + (\mathbf{a}^{\mathrm{att}} + 1)\frac{\mathbf{\Delta u}_{\max} - \mathbf{\Delta u}_{\min}}{2} \tag{17}$$

A trade-off has to be made between making the environment state smaller to speed up learning and giving enough information to allow the agent to make informed decisions. As the environment is assumed to have the Markov property, only information of the current time step is needed to explain its state. The environment state is decided to contain the weighted error vector to ensure a satisfactory steady-state response, the three body rates to improve the transient

*Image from C. v. Grinsven (with permission).

response, and the current control input since the agent only controls its increment. The environment state is given in Eq. (18).

$$\mathbf{s}^{\text{att}} = \left[\left(\mathbf{c}^{\text{att}} \odot \mathbf{e}^{\text{att}}\right)^\intercal, \mathbf{u}^\intercal, p, q, r\right]^\intercal \tag{18}$$

2. Altitude Control

An outer-loop SAC agent is tasked with providing a reference pitch angle to track an altitude signal. The error and cost vectors are defined in Eqs. (19) and (20), respectively. Similarly to the inner-loop agent, the reward function is defined as the absolute clipped weighted error in Eq. (21). The cost was found empirically so that, despite the clipping, differences between large errors are perceived by the agent, while still incentivizing for a low steady-state error.

$$\mathbf{e}^{\text{alt}} = \left[h^R - h\right] = \left[\Delta h\right] \tag{19}$$

$$\mathbf{c}^{\text{alt}} = \begin{bmatrix} \frac{1}{240} \end{bmatrix} \tag{20}$$

$$\tilde{r}(\mathbf{e}^{\text{alt}}) = -\left\| \text{clip}\left[\mathbf{c}^{\text{alt}} \odot \mathbf{e}^{\text{alt}}, \vec{-\mathbf{1}}, \vec{\mathbf{0}}\right] \right\|_1 \tag{21}$$

For this agent too, the control policy is smoothed by controlling the reference pitch angle increment instead of its value in (22). The agent action $\mathbf{a}^{\text{alt}}$ defined on $[-1, 1]$ is mapped to the pitch angle increment in Eq. (23) such that the corresponding pitch rate does not exceed 10deg s^{-1}.

$$\theta_t^R = \theta_{t-1}^R + \Delta\theta_t^R \tag{22}$$

$$\Delta\theta^R = \Delta\theta_{\min}^R + (\mathbf{a}^{\text{alt}} + 1)\frac{\Delta\theta_{\max}^R - \Delta\theta_{\min}^R}{2} \tag{23}$$

The state environment in Eq. (24) is reduced because the agent only has to learn the kinematic relationship between altitude and pitch angle. No knowledge of lateral states is needed since the inner-loop controller is already fully coupled.

$$\mathbf{s}^{\text{alt}} = \left[\mathbf{c}^{\text{alt}} \odot \mathbf{e}^{\text{alt}}, \theta^R\right]^\intercal \tag{24}$$

A diagram of the cascaded controller structure is shown in Fig. 3. Feedback signals of plant state variables, current pitch reference angle and control surface deflections are from the previous time step.

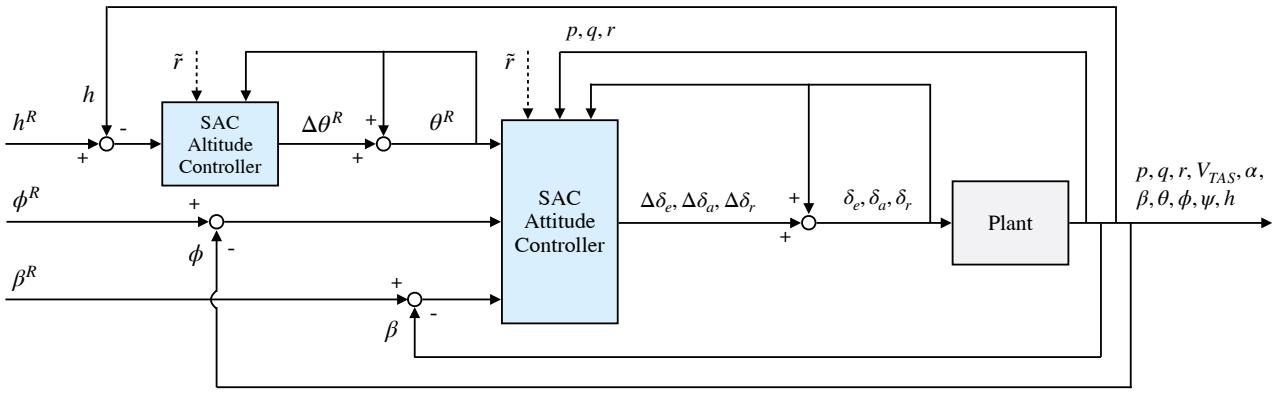


Fig. 3 Cascaded controller structure for altitude and attitude control. The SAC controllers observe the weighted errors, untracked states, and current control input and also receive a reward from the environment.

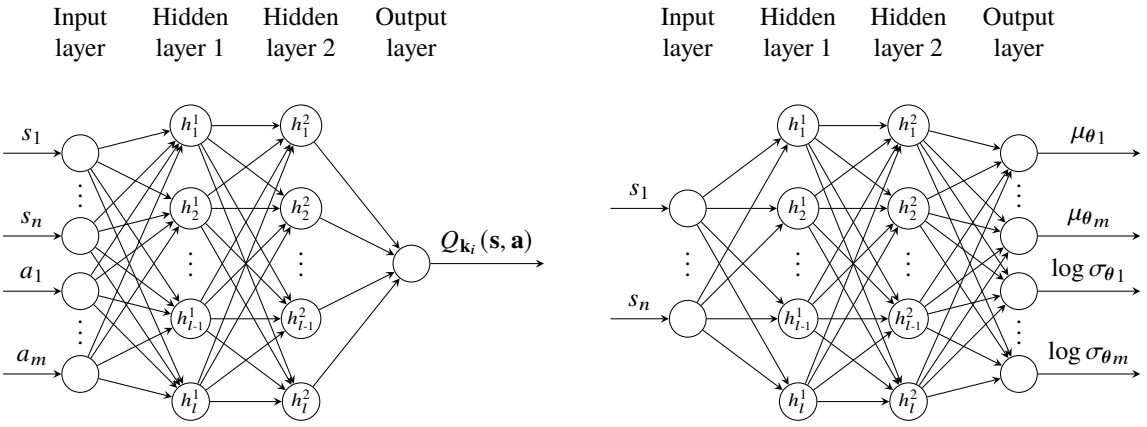
3. Hyperparameters

Choosing suitable hyperparameters for the SAC controllers and DNNs can significantly improve sample efficiency. A short selective-search hyperparameter optimization centered on default values was performed on the learning rate, number of hidden units, memory buffer and minibatch sizes, all known to be influential hyperparameters. In particular, it was found that the altitude controller, with smaller state and action spaces, performed better with fewer hidden units

Table 1 Cascaded SAC controller hyperparameters. Default values from [26].

Hyperparameter	**Altitude Controller**	**Attitude Controller** (if different)
Learning rate λ	$3 \cdot 10^{-4}$	Linearly decreasing from $4 \cdot 10^{-4}$ to 0
Hidden units lxl	32x32	64x64
Entropy target $\log \bar{\mathcal{H}}$	$-m = -1$ (default)	$-m = -3$ (default)
Discount factor γ	0.99 (default)	
Network activation	ReLu (default)	
Memory buffer size $\lvert\mathcal{D}\rvert$	$5 \cdot 10^4$	
Minibatch size $\lvert\mathcal{B}\rvert$	256 (default)	
Smoothing factor τ	0.995 (default)	

and a higher average learning rate. Several default hyperparameters are used from the original SAC implementation [26]. An overview of chosen hyperparameters is presented in Table. 1. Additionally, network initialization is executed following the Xavier method [29], and the gradient-descent employs the Adam optimizer, recognized for its superiority in terms of learning stability [30].



(a) Q-function i={1,2} **(b) Policy**

Fig. 4 Network topology of SAC controllers. The altitude controller has $n = 2$, $m = 1$ and $l = 32$ while the attitude controller has $n = 9$, $m = 3$ and $l = 64$. Subscript j of s_j represents the j-th element of vector s.

The network topology for the controllers is visualized in Fig. 4. The double Q-function corresponds to two independent networks with the structure shown in Fig. 4a. The policy's multivariate Gaussian distribution is defined according to the mean and log-standard deviation vectors from the network output in Fig. 4b. Using log-standard deviations allows the network to estimate the parameter on $\mathbb{R}$ and exponentiation is used to sample actions from the policy according to Eq. (8). At evaluation time, actions are made deterministic for consistent performance by setting the noise vector in Eq. (8) to zero.

Hidden units contain a linear combination of the input vector with bias and are fed to a normalization layer implemented according to [31]. This is to reduce the difficulty of training networks whose inputs have different scales and non-zero means. The normalized value is then given to a ReLu activation function, as seen in Fig. 5 for a generic hidden unit. The output layers, on the other hand, only contain a linear combination of the second hidden layer units with bias. All weights, biases and normalization factors are contained in network parameter vectors $\mathbf{k}$ and $\boldsymbol{\theta}$ for the Q-function and the policy, respectively.

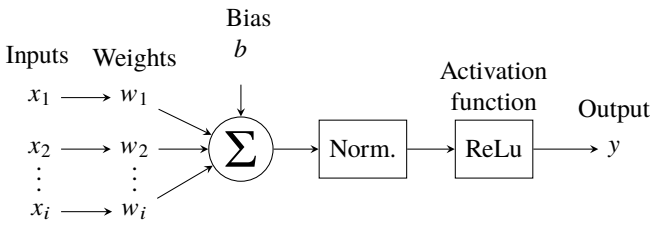


Fig. 5 Hidden unit h with input vector x and scalar output y.

C. Experiment Setup

In this research, the SAC controllers are trained offline on the normal plant dynamics and their adaptation to failures is subsequently evaluated online based on their robust response. For comparison purposes, the adaptive response to failures is also generated.

1. Offline Learning

DRL agent training is best performed offline as it typically requires 10^6 time steps or more. Normal plant dynamics with initial altitude and speed of 2000m and 90ms^{-1}, respectively, are used for the entire training process. For better learning stability, the inner-loop attitude controller is trained first alone with step reference signals for the pitch and roll angles, while the sideslip reference is always zero. After 500 20s-training episodes, or 10^6 time steps, the positive learning curve in Fig. 6 reaches a plateau, suggesting that no further training is required.

The altitude controller training is subsequently performed with the fully trained attitude controller in the inner loop. Successive climbing, constant altitude, and descending tasks allow the controller reach a converged policy after 10^6 time steps. As observed in Fig. 6, training is quite unstable as significant performance drops are experienced even in the last training stages. Benefiting from the offline learning environment, training for both controllers is repeated until a satisfactory policy is reached.

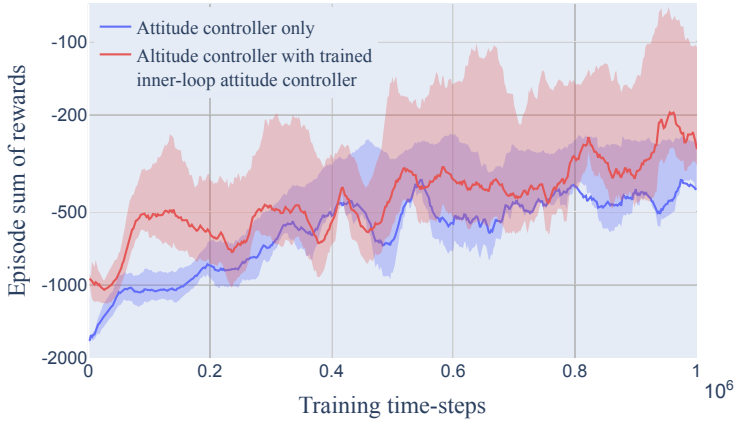


Fig. 6 Sum of rewards in function of training time-steps. The curves show the mean (smoothed with a window of length 20) and the shaded region the interquartile range over 5 random seeds of successful trials.

2. Online Robust Adaptation

The adaptation of the cascaded SAC controller to unseen flight conditions, atmospheric disturbances and sensor noise is evaluated with its robust response, as no controller parameters are changed. Similarly, six unknown failure cases are simulated online and the controller has to robustly adapt its response without changing its parameters.

As an additional experiment, the SAC attitude controller is also trained on the failed system. While this experiment falls outside the scope of the research objective since simulation models of failed dynamics are typically not obtainable, it provides insights into possible performance improvements. The previously-described offline training process of the attitude controller is repeated for each failure case to obtain six adaptive agents. The adaptive response will, however, not be used to assess the fault-tolerance of the SAC controller.

IV. Results and Discussion

The Cessna Citation jet aircraft response with the SAC controller is evaluated in this section, both on the non-failed and failed system. Moreover, the effect of biased sensor noise and atmospheric disturbances on the response is assessed. Lastly, additional robustness and reliability tests are conducted.

A. Non-Failed System

One of the goals of this experiment is to show that the controller can operate the non-failed aircraft on a representative coupled attitude tracking task. As depicted in Fig. 7, the attitude controller alone is able to keep the aircraft stable and minimizes the error overall. As the aircraft is initially untrimmed, rapid variations in the control input is seen close to t = 0s. A large roll angle is reached despite remaining 7.5°-off from the 70° reference, as the controller finds a compromise with the pitch angle error given the difficulty of keeping the aircraft horizontal during such a large bank maneuver. The roll angle suffers from a small yet consistently positive steady-state error, which could be due to the controller having failed to completely learn the asymmetricity of the aircraft.

The response of a similar aircraft on an analogous roll angle task in [5] with a model-based DHP controller shows less transient-response damping and a sideslip angle up to 50 times larger. The velocity, on the other hand, is better tracked there since it is directly controlled by the DHP agent, unlike here with an external auto-throttle.

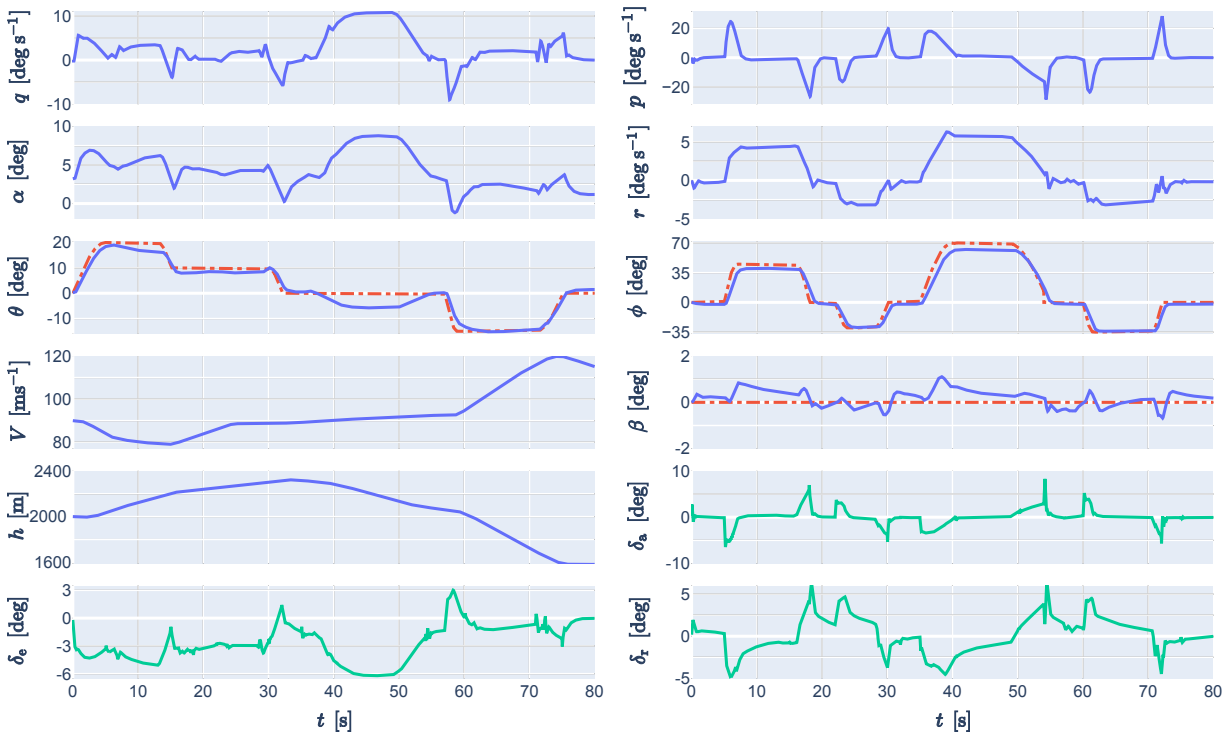


Fig. 7 Attitude tracking response with SAC attitude controller. Reference signals are shown with red dashed lines and control inputs with green solid lines.

The proposed framework should also be able to complete a representative altitude tracking task with the cascaded controller structure. As seen in Fig. 8, successive climbing turns are tracked well, with an altitude error of 15m or lower. This confirms that expert knowledge in flight control is successfully used to integrate the inner-loop and outer-loop controllers. Coupling effects in the attitude controller are observed as the elevator is pushed further up during the 40° bank turns at t = 35s and t = 65s, which is not sufficient to track the altitude controller's higher pitch angle reference. This could be due to the attitude controller avoiding regions of pitch rate higher than 5°/s as long as the pitch error is not too large, unlike in Fig. 7 where it reached 6°.

Similar performance to the Cessna Citation 500 response with uncoupled IDHP agent in [10] is achieved, this time

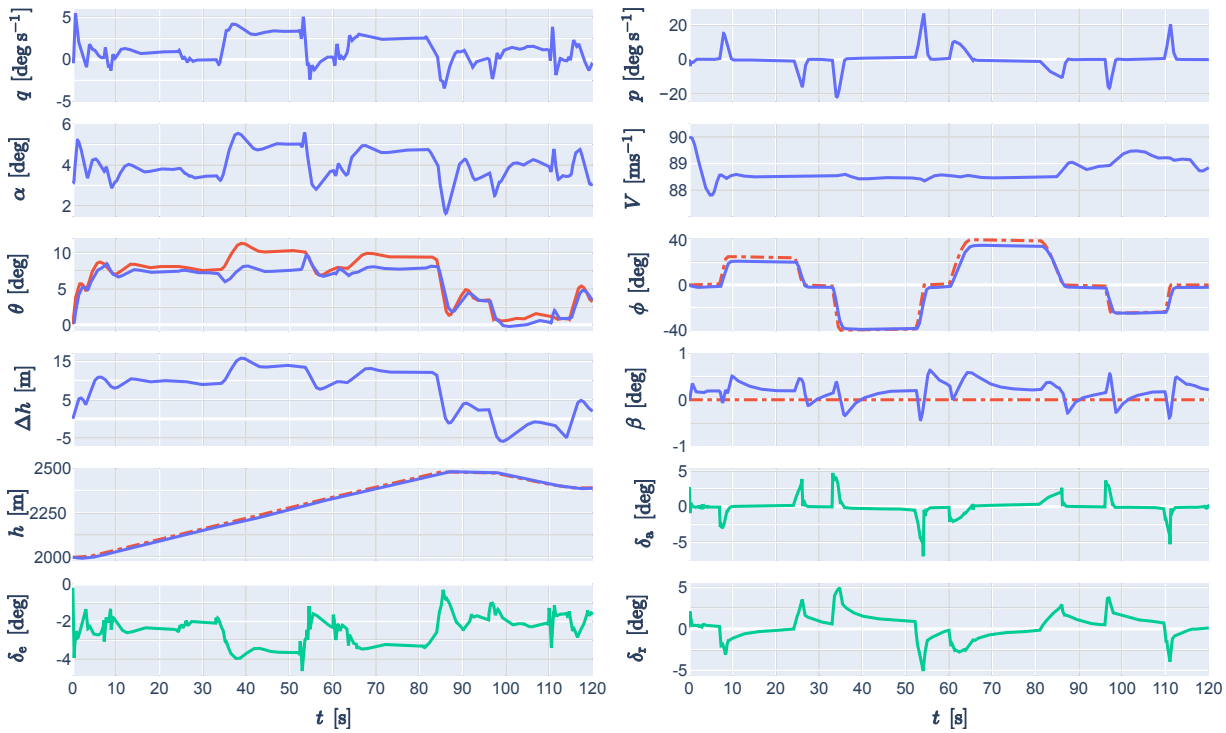


Fig. 8 Altitude tracking response with cascaded SAC controller. External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

with a maximum roll angle of 40° instead of 25°, benefiting from the coupled-dynamics SAC attitude controller.

B. Failed System

Another goal of this research is to evaluate the cascaded SAC controller on several types of failures. The robust response is shown first as it experiences an unexpected change in plant dynamics not seen during training. To identify possible performance gains, it is followed by the adaptive response, corresponding to a SAC controller trained on the failure case. For this research, however, the robust response is most important to determine if the SAC controller is successful since a plant model for every failure is typically not available for offline training.

1. Jammed Rudder

The response on a failure case with the rudder struck at $\delta_r = -15°$ from $t = 10$s onward is displayed in Fig. 9. The robust response until $t = 60$s is stable despite the severe failure. The sideslip error is inevitably big with this rudder deflection, but large and relatively constant 15° and 5° errors in the roll and pitch angles, respectively, are also observed. The attitude controller is affected by the unusual dynamics, resulting in large body rates oscillations at the failure time. They are nevertheless contained as it starts operating the elevator and ailerons around a new offset position, and as expected, it also tries to deflect the rudder in the opposite direction to the one it is stuck in, with no effect. The altitude controller, on the other hand, manages to track the climb task successfully by producing a higher-than-normal pitch reference, thereby almost removing the effect of the large pitch angle error. Overall, the robust controller leverages its knowledge of coupling effects jointly using the elevator and ailerons around a new offset to counteract the rudder failure.

The adaptive response was obtained by training the controller on the failed system and by stopping it both from tracking the sideslip and controlling the jammed rudder. The response then exhibits a tracking performance similar to the non-failed case in the pitch and roll angles even though the sideslip angle error is still large.

A similar failure case on a business-jet aircraft with a model-dependent DHP controller with the rudder stuck at $\delta_r = -15°$ and an asymmetric loss of thrust (possibly beneficial for this failure case) was investigated in [5]. The

adaptive response was stable but showed large 10°-amplitude undamped sideslip angle oscillations and a roll angle error of up to 50°. Oscillations in the longitudinal plane had five times higher amplitude and two times higher frequency than the robust controller in Fig. 9. Overall, the SAC robust response is more stable with smaller oscillations and lower error than the DHP adaptive one. This is partly attributable to the high generalization power of DNNs and the robustness of stochastic policies.

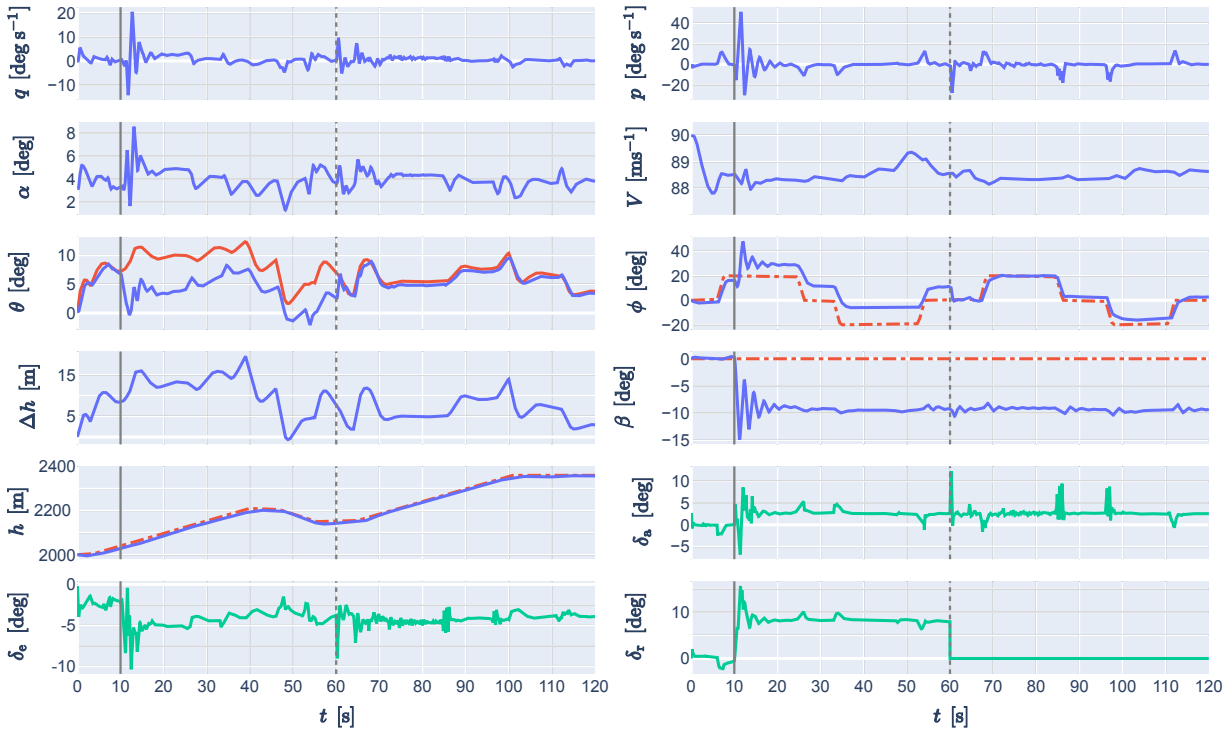


Fig. 9 Altitude tracking response with rudder stuck at $\delta_r = -15°$ from $t = 10$s (grey solid line). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines. Robust control until $t = 60$s, adaptive control thereafter (grey dotted line).

2. Reduced Aileron Effectiveness

An aileron failure case, implemented as a 70% reduced aileron effectiveness, is depicted in Fig. 10. The aileron failure has few effects on the response. Most notably, the robust controller more than doubles and prolongs non-zero aileron deflections to track the roll angle reference with almost no difference compared to the non-failed plant. The roll rate decreases by up to 6% with respect to the non-failed value.

The adaptive response from $t = 90$s exhibits almost identical performance, although its control policy is reduced in magnitude resulting in a slightly slower transient response in the roll angle tracking. It also suffers from some oscillations in the longitudinal plane, making its response less desirable than the robust controller. The lower performance likely comes from the increased difficulty to learn on the failed system.

In [10], an aileron failure with a 50% reduced effectiveness was introduced on a Cessna Citation 500 with an IDHP controller. Benefiting from its adaptive capability, the response was stable and well tracked in spite of the failure. It is worth noting that the robust SAC controller showed equivalent performance on a more severe failure, indicating again the high robustness of stochastic policies.

3. Reduced Elevator Range

Another actuator failure is tested, this time with the elevator range reduced from $[-20.05°, 14.90°]$ to $[-2.50°, 2.50°]$. The SAC agent is unaware of the failure and control inputs outside the new range are saturated. The robust response

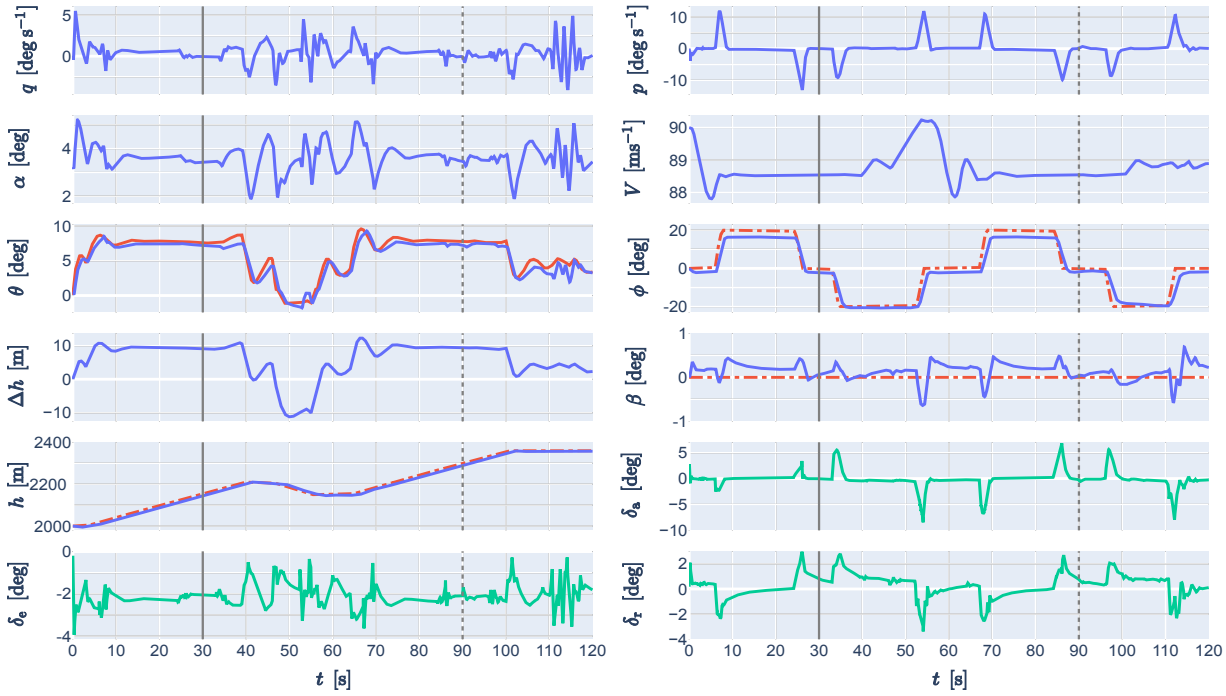


Fig. 10 Altitude tracking response with 70% reduced aileron effectiveness from $t = 30$s (solid grey line). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines. Robust control until $t = 90$s, adaptive control thereafter (grey dotted line).

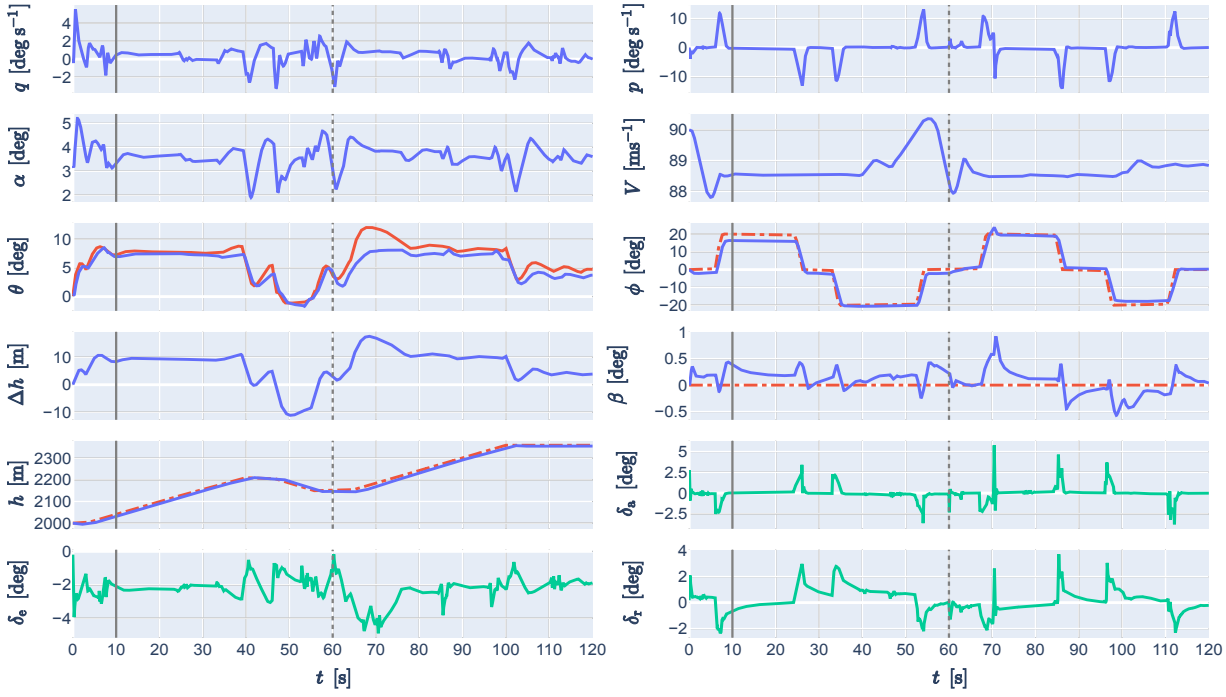


Fig. 11 Altitude tracking response with reduced elevator range from $t = 10$s (grey solid line). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines. Robust control until $t = 60$s, adaptive control thereafter (grey dotted line).

shown in Figure 11 is little-affected by the failure from t = 10s. While all states remain stable and well-tracked, the maximum pitch rate is reduced and remains at about half of the non-failed case seen in Fig. 8. This is explained by the close relationship between elevator deflection and pitch rate.

The adaptive response, from t = 60s, degrades more because of the failure. Despite having been trained on this failure, it can be explained by the more difficult task of initiating the climb with the failure already present. The controller instructs for more than 10s a control input outside the reduced elevator deflection range, which is not enough to avoid a large pitch error. The altitude error, in turn, doubles with respect to the non-failed case but the aircraft eventually completes the climb. It is worth noting that the agent did not explicitly learn the new saturation limits as its control input exceeds the bounds.

4. Partial Loss of Horizontal Tail

Structural failures are difficult to anticipate as they can affect various components of the aircraft. For this study, the structural failure of an essential part to flight control, the horizontal tail, is studied. As it mainly affects the elevator control effectiveness and pitch damping, it is implemented in the simulation model with a 70% reduction in $C_{(L/D/m)_{\delta e}}$ and C_{m_q}.

The failure at t = 10s generates a sudden loss in elevator effectiveness, which the robust agent immediately counteracts by quadrupling the elevator deflection. Despite that, large pitch angle and altitude errors are observed as the achieved pitch rate is too low for the climbing task. Lateral motion states are unaffected by this failure. This failure case shows the ability of the agent to adapt to this unexpected structural failure by offsetting its control input.

The adaptive response from t = 60s shows a similar control strategy with an unusually high elevator deflection while keeping most states well-tracked.

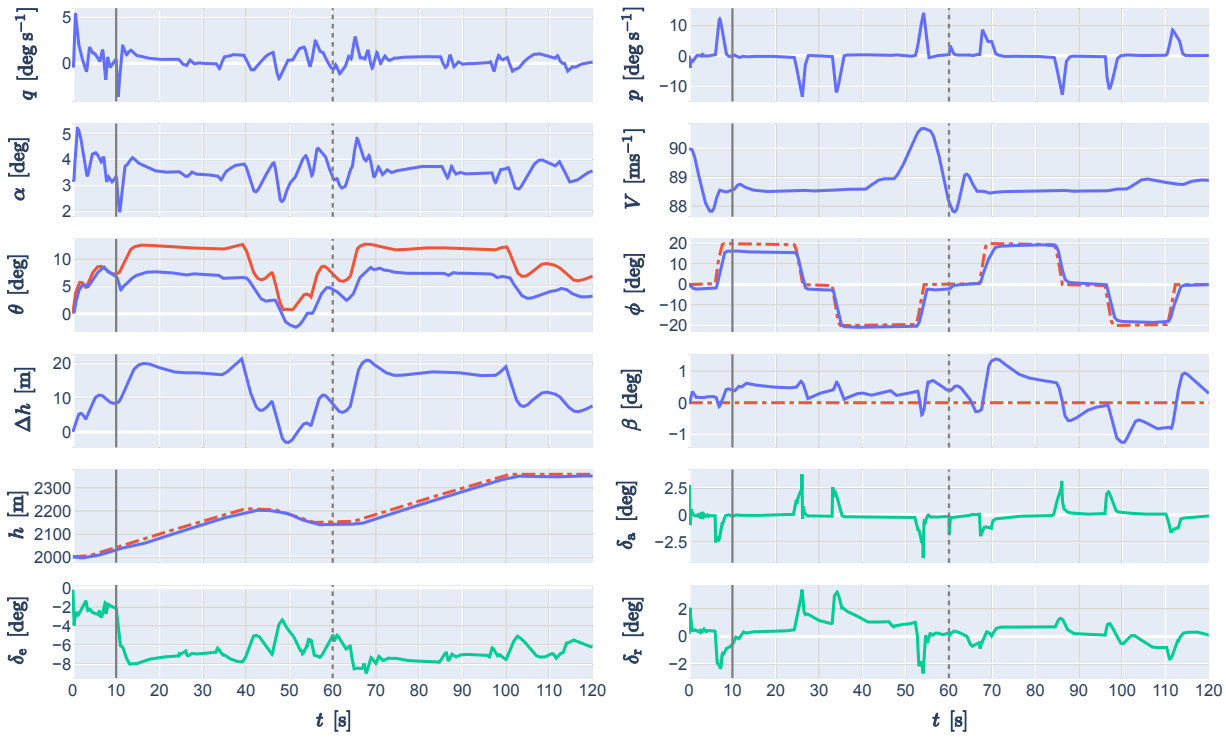


Fig. 12 Altitude tracking response with partial loss of horizontal tail from t = 10s (grey solid line). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines. Robust control until t = 60s, adaptive control thereafter (grey dotted line).

5. Icing

Icing is a common challenge faced by aircraft as ice accretions accumulate on wings, which main effects include a reduction in maximum lift coefficient and an increase in drag coefficient [32]. Conservative estimates suggest a reduction of $C_{L_{\max}}$ by 30% for mid-range Reynolds numbers and an increase of C_D by 0.06. As observed in Fig. 13, the decrease in lift and increase in drag due to icing cause large positive pitch and 20m-altitude errors on the robust response. The robust controller still manages to achieve the climbing task by doubling its elevator deflection, although it avoids deflecting it further up than $-4°$, most likely to avoid the stall region. The auto-throttle is unable to deal with both the increased drag and the climbing task, which leads to a reduction in velocity of 13% compared to the non-iced plant. Lateral states, on the other hand, are unaltered by icing.

The adaptive controller takes a more aggressive control strategy by deflecting the elevator further up, and despite trading kinetic energy, generates a larger altitude error. The roll angle transient response deteriorates and the sideslip error increases compared to the robust controller. It seems that the learning task with icing was difficult for the controller, which converged to a worse policy than the one of the robust controller.

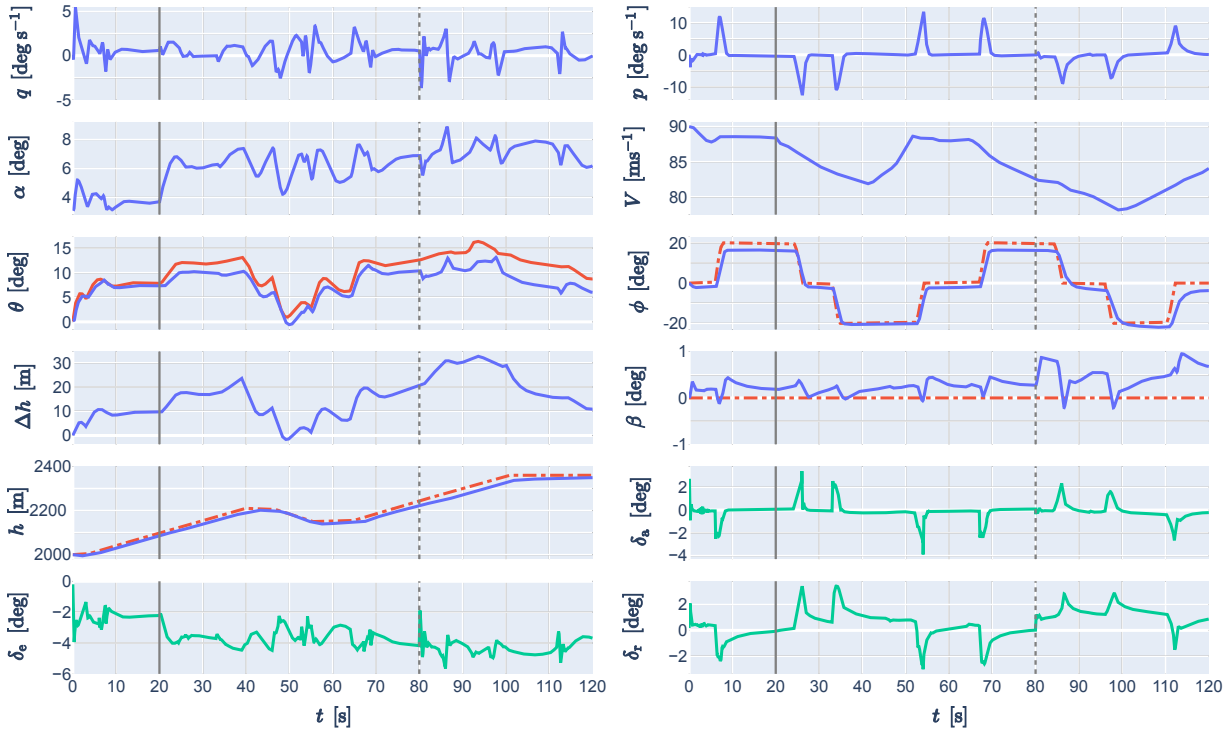


Fig. 13 Altitude tracking response with icing from t = 20s (solid grey line). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines. Robust control until t = 80s, adaptive control thereafter (grey dotted line).

6. Center-of-Gravity Shift

A sudden backward shift of heavy cargo in the aircraft can cause a diminution of the stability margin and create instability. The event of a 300kg payload moving from the front to the back of the passenger cabin is investigated, which translates to a backwards c.g. shift of 0.25m on the PH-LAB. As seen in Fig. 14, the c.g. shift at t = 20s causes a sudden change in the pitch rate and the controller immediately adapts the elevator deflection to the rare positive range. A constant negative pitch angle error appears as the c.g. moves backward, indicating that the controller's policy is stable but has not fully adapted to the new c.g. location. This is because the controller has only knowledge of the pitch error and not of the pitch angle itself, and has likely attributed higher elevator deflections to steep dive maneuvers on the aircraft with normal c.g. position. The robust attitude controller is unable to offset its elevator control input accurately. Consequently, the aircraft now has difficulty pitching down and experiences a large negative altitude error during descent. The lateral states, on the other hand, remain undisturbed by the c.g. shift.

The adaptive controller eliminates most of the pitch angle error, although it still suffers from sudden pitch-up events as the c.g. moved further back. This hints at instability issues should the c.g. be moved further back.

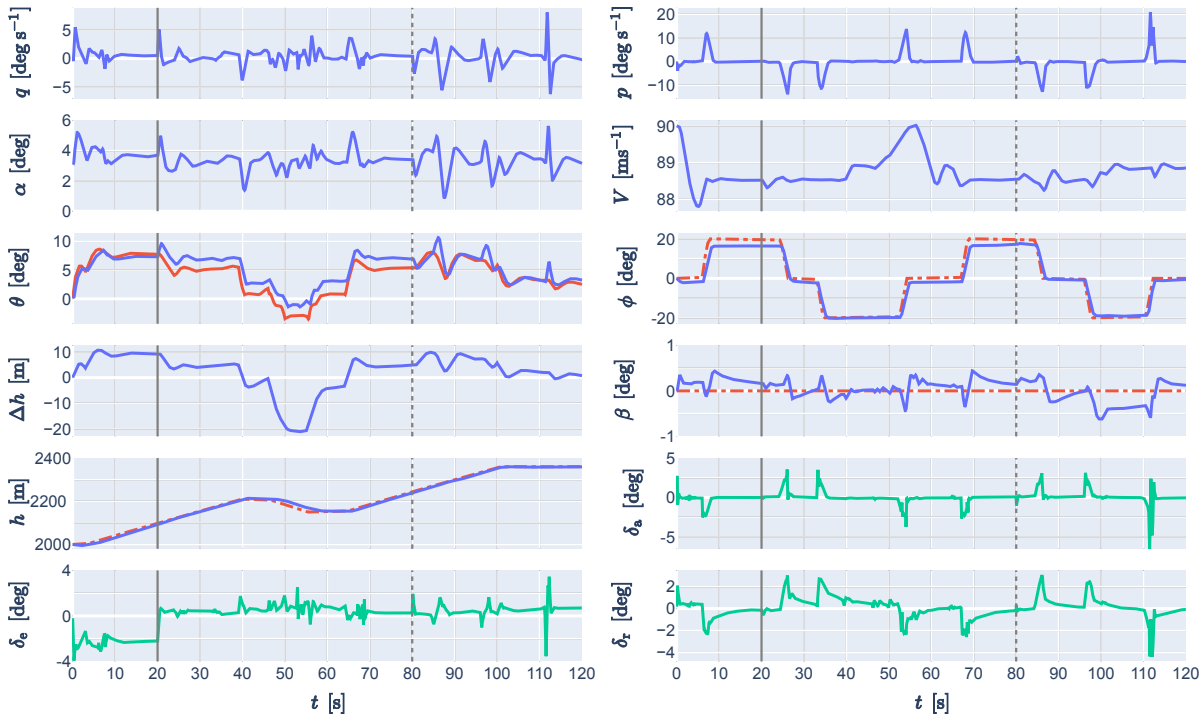


Fig. 14 Altitude tracking response with c.g. shift at t = 20s (solid grey line). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines. Robust control until t = 80s, adaptive control thereafter (grey dotted line).

C. Effect of Biased Sensor Noise and Atmospheric Disturbances

To evaluate the SAC controller in conditions closer to real-world phenomena, the addition of biased sensor noise and atmospheric disturbances is studied. A Gaussian white noise with standard deviation and bias obtained from the Cessna Citation PH-LAB's sensors in [33] are used to simulate sensor noise, with values shown in Table. 2. Noise from control surface deflection measurements is disregarded as the attitude controller observes the control input instead. Atmospheric disturbances in the form of discrete vertical gusts (15ft s^{-1}), specified in MIL-F-8785C [34], are also added to the system. They are implemented as 3s-step disturbances on the angle-of-attack at t = 20s and t = 75s.

Table 2 Cessna Citation PH-LAB aircraft sensor characteristics. Values from [33].

Observed state	p, q, r [rad s^{-1}]	θ, ϕ [rad]	β [rad]	h [m]
Noise SSD	$6.3 \cdot 10^{-4}$	$3.2 \cdot 10^{-5}$	$2.7 \cdot 10^{-4}$	$6.7 \cdot 10^{-2}$
Bias	$3.0 \cdot 10^{-5}$	$4.0 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$

As shown in Fig. 15, despite the controllers only having been trained with the ideal sensor assumption, the response remains stable and the errors are similar to the ideal-sensor scenario. The addition of biased sensor noise creates a slightly noisy control input and reference pitch angle while the aircraft states are unaffected.

The upwards vertical gusts cause a sudden increase in the angle-of-attack and disturb the pitch rate. As this makes the altitude error magnitude decrease, the outer-loop controller can instruct a lower pitch attitude, which subsequently leads to a downward elevator deflection. Conversely, when the gust vanishes, the sudden drop in angle-of-attack leads to

a sharp upward elevator deflection to limit the altitude error. During this time, the lateral states are unperturbed. This response demonstrates the ability to reject atmospheric disturbances in the presence of biased sensor noise.

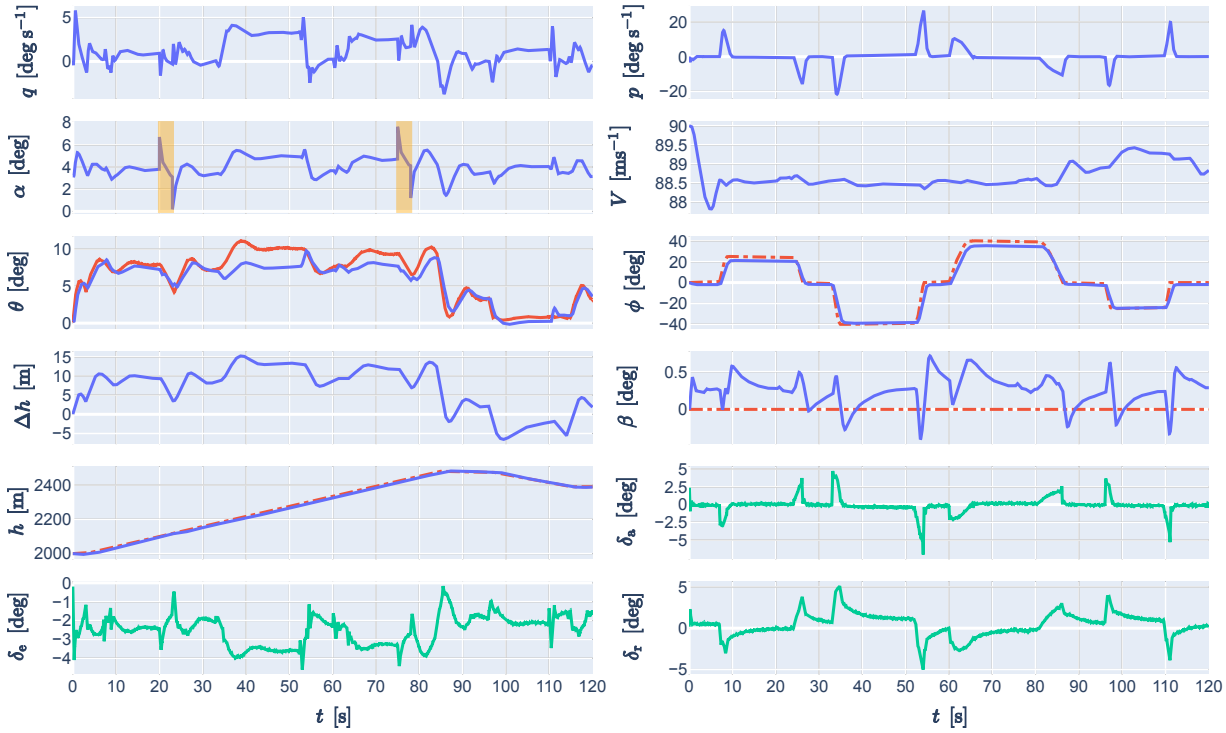


Fig. 15 Altitude tracking response with biased sensor noise and 15ft s^{-1} discrete vertical gusts (orange shaded region on α plot). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

D. Additional Tests

Further tests in terms of robustness and training reliability are conducted in this section, in an effort to explore some limitations of the SAC controller.

1. Robustness Analysis

The cascaded SAC controller was evaluated so far on the same nominal Initial Flight Conditions (IFC) and similar reference signal shapes. It is interesting to investigate how general the controller is by evaluating its robustness to IFC it has not been trained on. Initial altitude and speed combinations are chosen to match previous tests done in [10, 12] on the same aircraft. The robustness to various reference signal shapes and frequencies is also evaluated, namely on sinusoidal and triangular reference signals for the altitude and roll angle reference signals, while the sideslip reference angle remains zero. A performance metric is established as the average normalized Mean Absolute Error (nMAE) over externally tracked states (altitude, roll and sideslip angles). Normalization is done over the reference signals range, while for the zero-referenced sideslip angle an acceptable range of $[-5°, 5°]$ is used.

As reported in Table 3, the controller is robust on all four IFC, where the largest error corresponds to the condition with the lowest dynamic pressure. A higher flight speed leads to an increased performance while a higher altitude leads to a decreased one, explained by higher and lower dynamic pressure and aerodynamic damping, respectively. Different reference signal shapes are tracked with a normalized Mean Absolute Error (nMAE) no higher than 5%, with the error mostly caused by the slower transient response of altitude on the high-frequency signals. Overall, the controller is observed to maintain a relatively low error on various IFC and a multitude of reference signal shapes.

Table 3 Robustness analysis to varying IFC and reference signal shapes. The controller was trained on the nominal IFC.

Reference signals h^R, ϕ^R	Initial altitude [m]	Initial speed [ms^{-1}]	nMAE
As in Fig. 8	2000 (nominal)	90 (nominal)	2.64%
As in Fig. 8	2000	140	1.95%
As in Fig. 8*	5000	90	3.16%
As in Fig. 8*	5000	140	2.13%
Sinusoidal (low frequency)	2000	90	3.22%
Sinusoidal (high frequency)	2000	90	5.00%
Triangular (low frequency)	2000	90	3.00%
Triangular (high frequency)	2000	90	4.76%

low frequency: $T_{h^R} = 80$s, $A_{h^R} = 80$m, $T_{\phi^R} = 50$s, $A_{\phi^R} = 50°$

high frequency: $T_{h^R} = 40$s, $A_{h^R} = 40$m, $T_{\phi^R} = 25$s, $A_{\phi^R} = 25°$

*h^R is offset to new initial altitude

2. Training Reliability Analysis

Training the SAC controllers involves several random processes, including sampling from the stochastic policy to choose actions, parameter initialization for the DNNs and minibatch sampling. Since the performance can vary from training to training, it is interesting to evaluate how reliable the training process is. Although a large enough number of trials can be granted in an offline learning environment, a high success rate would indicate the convenience of such process.

Samples are generated by training pairs of cascaded SAC controllers in the same conditions as described in subsubsection III.C.1. Each sample is evaluated on the nominal IFC and altitude tracking task shown in Fig. 8 with a 5% nMAE success threshold. A global training success rate is obtained by generating enough samples ($n = 27$) until convergence to a stable success rate is obtained. It is found to be at 26%. This low value reveals the difficulty to train with consistency stochastic policy-based and random initialization-dependent DRL algorithms. The high sample efficiency, generalization power and robustness to failures presented so far can be seen to come at the expense of learning stability. It should be noted, however, that this does not compromise the online reliability and performance of the SAC controller, as long as it is trained in an offline environment.

V. Conclusion

A controller employing Soft Actor-Critic (SAC) Deep Reinforcement Learning (DRL) for the coupled-dynamics control of the Cessna Citation 500 jet aircraft is built as part of this research. Expert knowledge from classic flight control is used to create a cascaded SAC controller structure. The response is stable and keeps the tracking error low to successfully achieve coordinated 40°-bank climbing turns and 70°-bank flat turns. Severe failures such as the rudder jammed at −15°, the aileron effectiveness reduced by 70% or other unforeseen events such as icing and c.g. shift are handled by the robust SAC controller such that stability is maintained and the tracking task is achieved, demonstrating the SAC controller's strong fault tolerance. This high performance is further exhibited on varying initial flight conditions and tracking task types, and on biased sensor noise and atmospheric disturbances, none of which considerably degrade the controller response. This is achieved not having experienced any of these unexpected changes in dynamics, tracking task or flight conditions during training, indicating a high robustness not achievable with model-based controllers. It is believed that SAC's stochastic policy and deep neural networks allowing for better exploration and a higher generalization power, respectively, are the main contributors to this ability.

It is seen, however, to come at the expense of a stable and consistent offline training performance, which makes the development of SAC controllers more difficult but does not jeopardize online performance. This low training reliability is mainly attributed to the various random processes of the SAC algorithm, from network initialization to its stochastic policy. Interestingly, when training on the failed system, i.e. switching from robust to adaptive control, the performance of the SAC controller worsens or remains the same on five out of six failure cases. It is explained by the increased

difficulty of training on the failed plant dynamics, given the already low training reliability.

This research contributes to creating a coupled-dynamics model-free flight controller that allows for fault tolerance to various types of unforeseen failures. It is demonstrated that robust control through DRL can, unlike model-based controllers, adapt to various normal and failed flight conditions. While DRL is already widely used for small-scale UAV flight controllers, this research also paves the way for more applications to civil aircraft inner and outer-loop flight controllers. It is expected that DRL methods will be used more broadly in flight control applications to increase fault tolerance and help achieve safe fully-autonomous flight. It is suggested for further research to explore deterministic policy-based Twin-Delayed Deep Deterministic Policy Gradient or on-policy Proximal Policy Optimization DRL algorithms that, at the expense of enhanced exploration, can increase training reliability and may enable safe online learning, thereby removing the need for a plant model during offline learning.

References

[1] International Air Transport Association (IATA), "Loss of Control In-Flight Accident Analysis Report," , 2015.

[2] Stevens, B. L., Lewis, F. L., and Johnson, E. N., *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*, John Wiley & Sons, 2015.

[3] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, 2011.

[4] Bellman, R., "Dynamic Programming," *Science*, Vol. 153, No. 3731, 1966, pp. 34–37.

[5] Ferrari, S., and Stengel, R. F., "Online Adaptive Critic Flight Control," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 777–786.

[6] Enns, R., and Si, J., "Helicopter Trimming and Tracking Control using Direct Neural Dynamic Programming," *IEEE Transactions on Neural networks*, Vol. 14, No. 4, 2003, pp. 929–939.

[7] Zhou, Y., van Kampen, E.-J., and Chu, Q., "Nonlinear Adaptive Flight Control using Incremental Approximate Dynamic Programming and Output Feedback," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2016, pp. 493–496.

[8] Zhou, Y., van Kampen, E.-J., and Chu, Q. P., "Incremental Model Based Online Dual Heuristic Programming for Nonlinear Adaptive Control," *Control Engineering Practice*, Vol. 73, 2018, pp. 13–25.

[9] Konatala, R., Van Kampen, E.-J., and Looye, G., "Reinforcement Learning Based Online Adaptive Flight Control for the Cessna Citation II (PH-LAB) Aircraft," *AIAA Scitech 2021 Forum*, 2021, p. 0883.

[10] Heyer, S., Kroezen, D., and van Kampen, E.-J., "Online Adaptive Incremental Reinforcement Learning Flight Control for a CS-25 Class Aircraft," *AIAA Scitech 2020 Forum*, 2020, p. 1844.

[11] Kroezen, D., "Online Reinforcement Learning for Flight Control," MSc thesis, Delft University of Technology, 2019.

[12] Lee, J., and van Kampen, E.-J., "Online Reinforcement Learning for Fixed-Wing Aircraft Longitudinal Control," *AIAA Scitech 2021 Forum*, 2021, p. 0392.

[13] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing Atari with Deep Reinforcement Learning," *Neural Information Processing Systems Deep Learning Workshop*, 2013.

[14] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous Control with Deep Reinforcement Learning," *International Conference on Learning Representations*, 2016, p. 40.

[15] Tsourdos, A., Permana, I. A. D., Budiarti, D. H., Shin, H.-S., and Lee, C.-H., "Developing Flight Control Policy Using Deep Deterministic Policy Gradient," *2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, IEEE, 2019, pp. 1–7.

[16] Sohege, Y., Quinones-Grueiro, M., and Provan, G., "Unknown Fault Tolerant Control using Deep Reinforcement Learning: A Blended Control Approach," *Principles of Diagnostics (DX-19) Conference*, 2019, p. 23.

[17] Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M., "Control of a Quadrotor with Reinforcement Learning," *IEEE Robotics and Automation Letters*, Vol. 2, No. 4, 2017, pp. 2096–2103.

[18] Koch, W., Mancuso, R., West, R., and Bestavros, A., "Reinforcement Learning for UAV Attitude Control," *ACM Transactions on Cyber-Physical Systems*, Vol. 3, No. 2, 2019, pp. 1–21.

[19] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[20] Lopes, G. C., Ferreira, M., da Silva Simões, A., and Colombini, E. L., "Intelligent Control of a Quadrotor with Proximal Policy Optimization Reinforcement Learning," *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, IEEE, 2018, pp. 503–508.

[21] Bøhn, E., Coates, E. M., Moe, S., and Johansen, T. A., "Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs using Proximal Policy Optimization," *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2019, pp. 523–533.

[22] Fujimoto, S., van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," *International Conference on Machine Learning*, Vol. 80, 2018, pp. 1587–1596.

[23] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *International Conference on Machine Learning*, 2018, pp. 1856–1865.

[24] Achiam, J., "Spinning Up in Deep Reinforcement Learning, Benchmarks," , 2018. [Online; accessed 15-October-2020].

[25] Barros, G. M., and Colombini, E. L., "Using Soft Actor-Critic for Low-Level UAV Control," *arXiv preprint arXiv:2010.02293*, 2020.

[26] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al., "Soft Actor-Critic Algorithms and Applications," *arXiv preprint arXiv:1812.05905*, 2018.

[27] van den Hoek, M., de Visser, C., and Pool, D., "Identification of a Cessna Citation II Model Based on Flight Test Data," *Advances in Aerospace Guidance, Navigation and Control*, Springer, 2018, pp. 259–277.

[28] Delahaye, D., Puechmorel, S., Tsiotras, P., and Féron, E., "Mathematical Models for Aircraft Trajectory Design: A Survey," *Air Traffic Management and Systems*, Springer, 2014, pp. 205–247.

[29] Glorot, X., and Bengio, Y., "Understanding the Difficulty of Training Deep Feedforward Neural Networks," *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[30] Ruder, S., "An Overview of Gradient Descent Optimization Algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[31] Ba, J. L., Kiros, J. R., and Hinton, G. E., "Layer Normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[32] Lynch, F. T., and Khodadoust, A., "Effects of Ice Accretions on Aircraft Aerodynamics," *Progress in Aerospace Sciences*, Vol. 37, No. 8, 2001, pp. 669–767.

[33] Grondman, F., Looye, G., 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," *2018 AIAA Guidance, Navigation, and Control Conference*, 2018, p. 0385.

[34] Moorhouse, D. J., and Woodcock, R. J., "Background Information and User Guide for MIL-F-8785C, Military Specification-Flying Qualities of Piloted Airplanes," Tech. rep., Air Force Wright Aeronautical Labs Wright-Patterson AFB OH, 1982.

II

Preliminary Research[1]

[1] The Preliminary Research was assessed as part of the AE4020 Literature Study course.

3

Literature Review

This chapter presents the foundation of the methods used in this research, along with the review of state-of-the-art applications relevant to the research objective and challenges associated with them. In section 3.1, the fundamentals of Reinforcement Learning are laid out. Methods to deal with continuous spaces are explained in section 3.2. The two main approaches to continuous control are explained in section 3.3 and section 3.4. Lastly, section 3.6 concludes on the reviewed literature and selected algorithm by answering research questions *Q1.1* and *Q1.2*.

3.1. Reinforcement Learning Foundations

Reinforcement Learning (RL) is a machine learning framework inspired by the learning process of living organisms able to perform complex tasks. First, the basic concepts that define RL are explained in subsection 3.1.1.

3.1.1. Basic Concepts

RL is composed of two main elements, an agent and an environment. Similarly to animals that learn to walk by trial and error, the interaction between the RL agent and its environment allows learning a complex task over time.

The agent can execute actions on the environment and sense their effects to achieve a certain goal. In particular, the communication from the environment to the agent includes the environment's state and a reward. By trying to maximize the reward over time, the agent learns the correct actions that lead to achieving its goal. An overview of this interaction is given in Figure 3.1.

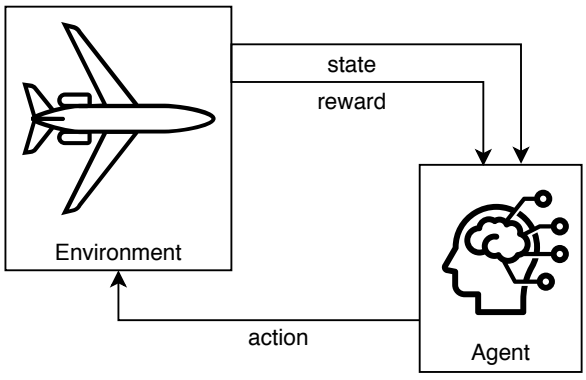


Figure 3.1: Agent-environment interaction in reinforcement learning.

For a flight control task, the environment includes the aircraft and the air surrounding it, while the agent is the flight controller. In more general engineering terms, the *environment*, the *agent*, the *ac-*

tion and the *state* correspond to the *plant*, the *controller*, the *control input* and the *feedback signal*, respectively.

Markov Decision Processes

The action, state and reward are communicated at discrete time-steps due to the iterative mechanism of RL algorithms. At time-step t, the action $\mathbf{a}_t$ and state $\mathbf{s}_t$ are vectors belonging to the action space $\mathcal{A}$ and state space $\mathcal{S}$, respectively. The instantaneous reward $\tilde{r}_t$ is a real number. Given a certain history, the probability that the environment reaches state $\mathbf{s}_{t+1}$ is expressed as shown in Equation 3.1.

$$\mathcal{P}\{\mathbf{s}_{t+1}, \tilde{r}_{t+1} | \mathbf{s}_t, \mathbf{a}_t, ..., \mathbf{s}_0, \mathbf{a}_0\} \tag{3.1}$$

A process has the Markov Property when the current state and action contain all the required information to predict its future state, without needing information about past states and actions [72]. This means the probability distribution function can be simplified to the expression in Equation 3.2.

$$\mathcal{P}\{\mathbf{s}_{t+1}, \tilde{r}_{t+1} | \mathbf{s}_t, \mathbf{a}_t, ..., \mathbf{s}_0, \mathbf{a}_0\} = \mathcal{P}\{\mathbf{s}_{t+1}, \tilde{r}_{t+1} | \mathbf{s}_t, \mathbf{a}_t\} \tag{3.2}$$

In other words, the future of a Markov Decision Process (MDP) is independent of its past and fully explained by its present. This property allows the agent to update an internal policy at every time-step to choose the ideal action that will lead to its goal. This requires the agent to have full access to the state at every time-step, neglecting any sensor noise. RL can be applied if enough relevant state variables are fed to the agent [72].

Reward Function

The agent learns by trying to maximize a reward. For state $\mathbf{s}_t$ and action $\mathbf{a}_t$, the environment will give the agent instantaneous reward $\tilde{r}_{t+1}$ at the next time-step. It is designed to help the agent reach a certain goal and may be sparse (e.g., non-zero only when the goal is reached) or more gradual (e.g., distance to the goal). The maximum reward should be received when the goal is reached, at the terminal state.

For complex tasks, it may be beneficial to sacrifice the instantaneous reward for higher future rewards. To make the agent take this long term planning into account, the discounted return R_t is introduced, which is the discounted sum of rewards until the terminal state at step N.

$$R_t = \tilde{r}_{t+1} + \gamma \tilde{r}_{t+2} + ... + \gamma^N \tilde{r}_{t+N+1} = \sum_{n=0}^{N} \gamma^n \tilde{r}_{t+N+1} \tag{3.3}$$

The discount factor $\gamma \in [0, 1)$ can be increased (decreased) to favor (disregard) long-term planning. For control applications, if the goal is not to reach a specific desired state but only striving to minimize the error, the task is then said to be continuing. The term N in Equation 3.3 becomes ∞ but R_t remains bounded as γ is less than one.

Policy

The policy π is a law to decide what action to take at every state, comparable to the control law in control theory. It can be represented by the probability of choosing action $\mathbf{a}_t$ at state $\mathbf{s}_t$, as seen in Equation 3.4.

To choose an action that maximizes the discounted return, the state value function V^π is introduced in Equation 3.5. It contains the expected discounted return if policy π is followed starting at state $\mathbf{s}_t$, as shown below. In other words, it tells the agent how beneficial it is to be in this position following policy π. Alternatively, the action-state value function of Equation 3.6 may be used, which describes the expected discounted return starting at state $\mathbf{s}_t$, taking any action $\mathbf{a}_t$ and following policy π thereafter.

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \mathcal{P}\{\mathbf{a}_t | \mathbf{s}_t\} \tag{3.4}$$

$$V^\pi(\mathbf{s}_t) = \mathbb{E}_\pi\{R_t | \mathbf{s}_t\} \tag{3.5}$$

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\pi}\{R_t | \mathbf{s}_t, \mathbf{a}_t\} \tag{3.6}$$

A policy π is optimal if its expected return for all states is higher or equal than any other policy π', which in this case is written π^*. The value functions, or expected discounted returns, are maximized if the optimal policy π^* is followed, yielding the optimal value functions in Equation 3.7.

$$V^*(\mathbf{s}_t) = \max_{\pi} V^{\pi}(\mathbf{s}_t) \qquad Q^*(\mathbf{s}_t, \mathbf{a}_t) = \max_{\pi} Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \tag{3.7}$$

Bellman Optimality

It is useful to describe the optimal value functions in a recursive form for the update algorithms discussed in the following sections. Using the expectation operator's linear properties, the Bellman equation is derived in Equation 3.8 yielding a recursive form of the state value function, as shown in [72].

$$\begin{aligned} V^{\pi}(\mathbf{s}_t) &= \mathbb{E}_{\pi}\{R_t | \mathbf{s}_t\} \\ &= \mathbb{E}_{\pi}\{\tilde{r}_{t+1} + \sum_{n=1}^{\infty} \gamma^n \tilde{r}_{t+N+1} | \mathbf{s}_t\} = \mathbb{E}_{\pi}\{\tilde{r}_{t+1} + \gamma \sum_{n=0}^{\infty} \gamma^n \tilde{r}_{t+n+2} | \mathbf{s}_t\} = \mathbb{E}_{\pi}\{\tilde{r}_{t+1} + \gamma \tilde{r}_{t+1} | \mathbf{s}_t\} \\ &= \tilde{r}_{t+1} + \gamma \mathbb{E}_{\pi}\{\tilde{r}_{t+1} | \mathbf{s}_{t+1}\} = \tilde{r}_{t+1} + \gamma V^{\pi}(\mathbf{s}_{t+1}) \end{aligned} \tag{3.8}$$

The optimal state value function can now be rewritten with Equation 3.8, which yields the Bellman optimality equation for the state value function in Equation 3.9. The Bellman optimality equation for the action-state value function can be deduced in Equation 3.10.

$$V^*(\mathbf{s}_t) = \max_{\mathbf{a}_t \in \mathcal{A}} Q^*(\mathbf{s}_t, \mathbf{a}_t) = \max_{\mathbf{a}_t} \mathbb{E}_{\pi^*}\{R_t | \mathbf{s}_t, \mathbf{a}_t\} = \max_{\mathbf{a}_t} [\tilde{r}_{t+1} + \gamma V^*(\mathbf{s}_{t+1})] \tag{3.9}$$

$$Q^*(\mathbf{s}_t, \mathbf{a}_t) = \tilde{r}_{t+1} + \gamma V^*(\mathbf{s}_{t+1}) = \tilde{r}_{t+1} + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}_{t+1}, \mathbf{a}') \tag{3.10}$$

A learning process can now be built based on these recursive expressions of the optimal value functions.

3.1.2. Learning Process

The learning process is what allows the agent to become successful in performing the task. This process can be carried out in several different ways, of which certain characteristics are relevant to point out. For this research's flight control application, some requirements on these characteristics can be drawn and are explained below.

Exploitation and Exploration Trade-Off

During the learning process, the agent can operate in two different modes, as explained below. The first one is exploitation, which describes an agent always choosing an action according to the optimal policy. The optimal policy, or policy that maximizes the value function, can be expressed as given in Equation 3.11.

$$\pi^*(\mathbf{s}_t) = \arg\max_{\mathbf{a}_t} Q^*(\mathbf{s}_t, \mathbf{a}_t) \tag{3.11}$$

The action corresponding to the optimal policy is the so-called greedy action. If the agent were always to perform the greedy action, it would never visit regions other than those that correspond to the maxima of the current value function. In the initial learning phase, the value function is not yet accurate, meaning that the agent may be stuck on a local maximum but will never find the global one if it exists. An optimal solution may never be found that way.

To counteract this, the agent can enter a second mode called exploration. The agent executes a random action in the exploration mode, most-likely sub-optimal, intending to reach yet unseen state-action combinations. This will allow the agent to overview its environment better and increase the chance to learn an optimal policy. In practice, this is often done by adding a noise component to the optimal policy if it is deterministic.

It is important to keep exploring during the entire learning process to avoid converging to a local maximum. A common strategy is to perform exploration with frequency ϵ and use exploitation the rest of the time $(1 - \epsilon)$, with ϵ decreasing over time while never reaching zero.

Model Dependence

The learning process is considerably different if knowledge of the environment, in particular, a state-transition model, is available to the agent. A common approach to designing algorithms for model-based agents is Dynamic Programming (DP) [80]. Some other algorithms first focus on learning a model of the system, a process known as system identification.

When no model knowledge of the environment is given to the agent, the learning process is said to be model-free. This means that the agent learns purely based on the interaction with the environment and treats it as a black box. Model-free techniques can also sometimes learn an approximation of the model and the task at the same time. This corresponds to online system identification and is still categorized as model-free since the RL algorithm creates its own approximation.

The requirement on model-dependence for this research's flight control application is pivotal to choosing a suitable RL algorithm. As part of the research objective, this flight controller should be fault-tolerant. It is known that actuator and component failures, including structural damage, can occur with varying degrees of severity and take different forms. Due to their inherent unpredictability, building models of the failed system covering all combinations of failures is unrealistic. To keep the agent flexible to various types of unpredicted failures, the RL algorithm used in this research should be model-free.

Online and Offline Learning

Learning can be done *online* while performing the task, i.e., inflight for flight control applications. Safety-critical states and crashes are therefore not permitted, requiring the learning process to be highly sample-efficient. Since agent parameters are updated while performing the task, this type of learning is classified as adaptive control.

Achieving the entire learning process online is sometimes difficult, depending on the complexity of the system and the algorithm used. Initial learning can then be performed *offline*, where safety is not critical and enough time can be given to the agent. In terms of flight control applications, this means learning is first performed on the ground in a simulated environment yet remains model-free as the agent has no knowledge of the environment. In a second phase, learning is continued online, where the agent has must to adapt to faults on the system.

A third option is to learn fully offline. This means that as much time as necessary can be given to the agent to learn the task. If failures occur when performing the task online, the agent may still adapt its response if it is robust enough. Since it cannot change its parameters online based on system faults, this type of learning is classified as robust control.

The RL agent for this flight control application should be fault-tolerant. As this may be achieved either through robust control or adaptive control, there is no requirement on whether the agent for this application should learn offline, online, or a combination of both.

On-Policy and Off-Policy

A distinction between on-policy and off-policy learning can be made. On-policy refers to an algorithm that learns based on the executed action, not necessarily the greedy action. These two are different in the exploration mode when a random action is executed. In more general terms, with on-policy algorithms, the policy that is being optimized corresponds to the behavioral policy.

On the other hand, off-policy learning can execute actions from a behavioral policy different from the one it optimizes. In general, it uses the greedy action and not the executed action to update its policy. When exploring, this may lead to dangerous behavior as the agent reaches a boundary of the safe state region. If higher rewards are present near a boundary, the agent will learn to stay close to it

while neglecting the risk of trespassing it due to exploration noise. When evaluating the agent, safety is ensured by removing exploration noise. Off-policy learning is thus better-suited for offline learning. Another benefit is that the optimization process can be done not only on the current sample but also on old data, which allows the use of a memory buffer. This is beneficial for sample efficiency, defined by the performance obtained after a given number of training steps [64].

Update Rule

In RL, several methods exist to learn the value function for policy π, i.e. have estimate $Q(\mathbf{s}_t, \mathbf{a}_t)$ approach $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$. For this research, one of the requirements on the RL agent found above was that it has to be model-free. This excludes DP as an update rule which is model-based. In flight control, the task is generally continuing as it does not have a specific final goal. This characteristic excludes Monte-Carlo methods, which are only available for episodic tasks.

Another method widely used in RL control applications is Temporal Difference (TD) learning. It is a model-free method that bootstraps, meaning that is it uses values from previous iterations to compute current values, well-suited for continuing tasks. The principle is to compute the error between the target and the current estimate of the value function and increment the current estimate by this error factored by a learning rate λ, as seen in Equation 3.12 and Equation 3.13 for on-policy and off-policy learning, respectively.

$$V(\mathbf{s}_t) \leftarrow V(\mathbf{s}_t) + \lambda \left[\tilde{r}_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t)\right] \tag{3.12}$$

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \lambda \left[\tilde{r}_{t+1} + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}_{t+1}, \mathbf{a}') - Q(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{3.13}$$

In this case, the target is given by Equation 3.8 and the TD-error corresponds to the expression between square brackets. The action-state value function, also known as Q-function, is used for the off-policy update as it is well suited for a dichotomy between the greedy action and the executed action.

With TD as a baseline for the update rule, RL algorithms can be built to learn the optimal solution. In discrete state and action spaces, this is typically done with Q-Learning using Equation 3.13 or SARSA using Equation 3.12, methods that do not learn an explicit policy but only a value function and find the action argument that maximizes it. For advanced control applications such as flight control, however, continuous action and state spaces are required. For that purpose, the next section introduces methods to solve continuous-space RL tasks that integrate the TD update rule as a baseline.

In subsection 3.1.2, it was found that the RL agent should be model-free and either offline or online learning-based, as long as it can ensure fault-tolerance. It should also have both continuous action and state spaces. With this set of requirements, research question *Q1.1* is answered.

3.2. Continuous-Space Reinforcement Learning

RL was initially conceived for discretized state and action spaces, allowing for the easy evaluation of the value function and policy in every point with look-up tables. As task complexity increases, smaller discretization errors are permitted, leading to growing state and action spaces. This growth makes the computational complexity increase at an exponential rate, an effect also known as the curse of dimensionality [6]. Instead of look-up tables, function approximators can be used to deal with continuous spaces [9]. Their working principle is explained in subsection 3.2.1 and their integration within the RL framework is laid out in subsection 3.2.2.

3.2.1. Function Approximation Methods

Function approximators can be used as a tool to alleviate the curse of dimensionality. They generalize unknown values to neighboring known ones in an effort to reduce computations and memory usage. A requirement to use such generalizations is that the spaces should be smooth and exclude any discontinuity [54]. This is generally the case for flight control, as long they do not include any physical boundaries.

Parametrization

Approximation methods may be classified either as parametric or non-parametric. Non-parametric methods have the advantage of being more flexible as they do not constraint the approximation to a specific class of functions. Usually memory-based, such methods have the disadvantage of exceeding reasonable memory capacities as the space dimensions increase. They are therefore deemed unsuitable for this continuous flight control application. On the other hand, parametric function approximators have a set of tunable parameters from a pre-defined function class, making them more memory-efficient for large state spaces.

Linearity

Parametric function approximators can be further divided into two categories, between linear and non-linear ones. It should be noted that the linearity refers only to the features of those functions, but the resulting functions are not linear due to non-linearities in the way their features are connected. Linear functions have the advantage of being well-understood and extensively studied, some of which have seen their rate of convergence proven under certain conditions [9]. Some of the most commonly used linear methods are tile coding, radial-basis functions, or fuzzy-sets [4]. A major limitation is that they all require some knowledge of the environment. For instance, radial-basis functions and fuzzy-sets need to have the number of membership functions and their range defined by the user. For that reason, adaptability to unforeseen states in case of failure is more difficult, while also not being practical on large state and action spaces encountered in flight control [40]. For this research, linear function approximators are therefore discarded.

Artificial Neural Networks

This leaves non-linear parametric function approximators as a remaining candidate, which in the field of RL mainly entails artificial (Artificial) Neural Network (NN). While non-linear function approximation methods do not guarantee convergence, they have been seen in practice to provide satisfactory results. Although NNs having gained much attention in recent years, it is not a new concept [26].

A neural network typically consists of three layers, the first one being the input layer, followed by a hidden layer, and finally an output layer. An example NN is presented in Figure 3.2. The input layer is a one-to-one mapping of the input vector, therefore has as many neurons as variables in the input vector (three in this example). The hidden layer has a variable size, representing a hyperparameter of the network (five in the example). Lastly, the number of neurons in the output layer is determined by the output vector size.

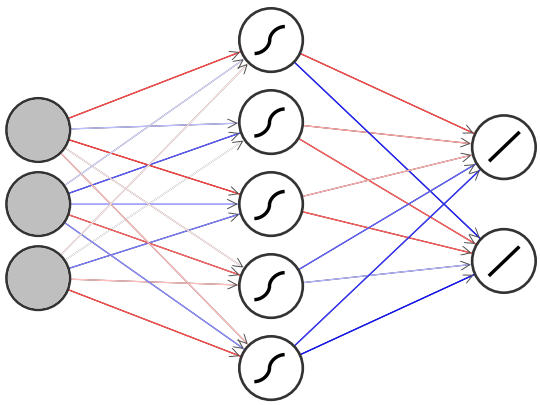


Figure 3.2: NN example with one tanh-activated bias-free hidden layer and linear output layer. Different line colors represent different weights. Each circle represents a neuron.

Each connection between neurons is attributed a weight, and neurons in the hidden layer generally add a bias to the sum of their inputs. The linear combination of the weights, inputs and bias is then fed to an activation function, typically a hyperbolic tangent or a Rectified Linear Unit (ReLu) function for the hidden layer and a linear or sigmoid function for the output layer. While the hyperbolic tangent is

bounded, it suffers from the exploding and vanishing gradient problem [50]. On the other hand, ReLu fixes this issue, yet it generally requires more neurons and its unboundedness may cause stability issues. NNs are universal approximators in the sense that with the adequate activation function and network architecture, they can approximate any real-valued function [61].

Weights are typically initialized with random values on the range $[-1, 1]$ while biases usually start with the zero value. A forward pass through the network is executed, activating each neuron according to the current weights and biases. The error between the network output and target values is then estimated. Because NNs are fully differentiable, partial derivatives based on the error function can be calculated to update the weights, starting from the output layer to the input layer, through a backward pass. This process, also called backpropagation, constitutes the basis for network training [26].

Recent advancements in computing power have allowed NNs to be applied to more complex problems in a myriad of domains. More than one hidden layer may be used to approximate more complex functions, resulting in a network known as a Deep Neural Network (DNN) used in the field of Deep Learning. This feature notably enables a superior generalization power [32]. The drawback of adding more layers is the increased training time and the vanishing and exploding gradient problem [50].

3.2.2. Using Function Approximation for Reinforcement Learning

Having identified a suitable function approximation method, it becomes necessary to know what exactly is to be approximated. The agent's goal is to learn an optimal policy π^* for every state. For that purpose, the state or action-state value functions may help to do so. It is common to label the structure that learns the policy and the value function as actor and critic, respectively.

Three methods therefore emerge, one that only learns a value function (also known as critic-only), one that only learns a policy (or actor-only) and one that learns both (actor-critic).

Value-Based (Critic-Only)

Value-based methods only learn a value function and no policy is explicitly learned. Using the Q-function $Q(\mathbf{s}_t, \mathbf{a}_t)$, the greedy action can be derived by selecting the action argument that maximizes $Q(\mathbf{s}_t, \mathbf{a}_t)$. Even though $Q(\mathbf{s}_t, \mathbf{a}_t)$ is stochastic, the implied policy is mostly deterministic as the derived action does not follow a probability distribution. These methods have the advantage of being intrinsically simple as only one structure is required. In discrete spaces, its implementations known as SARSA or Q-Learning have been largely used in the field of RL [72]. Function approximation methods, NNs in particular, can generalize continuous state spaces and successfully learn to approximate a value function [80].

A difficulty arises when used in conjunction with continuous action spaces. Finding the greedy action becomes non-trivial as the algorithm has to search through an infinite action space, making the task computationally unpractical. Since flight control needs continuous control, value-based methods are not considered in this research.

Policy-Based (Actor-Only)

Policy-based methods are also composed of a single structure, which only learns a policy without using a value function. Gradient-free methods with evolutionary strategies will not be explored as they are less suitable for high-dimensional tasks due to the high variance and computational cost of the fitting function [80]. Instead, since NNs are fully differentiable, a gradient ascent approach can be used to find the optimal action through a continuous action space.

To solve this optimization problem, the policy is parameterized with vector $\boldsymbol{\theta}$ such that it can be expressed as $\pi(\mathbf{a}_t|\mathbf{s}_t, \boldsymbol{\theta}) = \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$. When using NNs as function approximator, $\boldsymbol{\theta}$ corresponds to the weights and biases of the network. To measure the performance of the policy function, the expected discounted return $V^{\pi_{\boldsymbol{\theta}}}$ is used as objective function $J(\boldsymbol{\theta})$. Parameter $\boldsymbol{\theta}$ is updated in the direction of steepest ascent of the objective function by step size λ, shown in Equation 3.14 using the stochastic gradient of Equation 3.15, according to the works of [52].

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.14}$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t) = E\{R_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\} \tag{3.15}$$

As seen in Equation 3.15, the discounted return R_t requires knowledge of all future rewards such that the actor has to wait for the task to end to update the policy. This Monte-Carlo policy-gradient method forms the basis of the REINFORCE algorithm [81]. It is, however, impractical for continuing tasks and online learning.

Another drawback of this method is that the discounted return R_t suffers from high variance, making it difficult to converge towards a deterministic policy, although research has been done to limit this effect [21, 52]. It should be noted that learning a stochastic policy is beneficial in uncertain environments and allows to ensure a minimum of exploration [52].

Overall, policy parameterization allows generalizing between states while allowing for continuous actions. The method presented below capitalizes on these benefits.

Actor-Critic

The actor-critic structure uses a critic to compensate for the actor's weaknesses. It is implemented with a TD update method to allow for continuing tasks. The critic receives the state and the reward from the environment to compute an estimate of the value function. It then calculates the TD-error to update the actor's policy parameters. The actor, based on the state, chooses an action that maximizes its parameters. The overall structure can be seen in Figure 3.3.

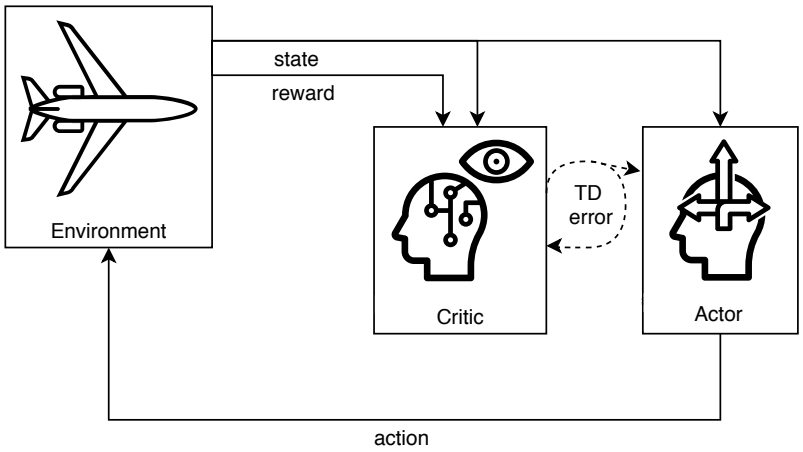


Figure 3.3: Actor-critic structure and environment interaction in reinforcement learning. Adapted from [72].

Several approaches exist to reduce the variance of R_t. One of them, Q-function Actor-Critic, makes use of the critic to approximate the Q-function. The Q-function is used instead of R_t in Equation 3.15 to reduce the variance. The approximation is enabled with parameterization $\mathbf{k}$, such that $Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t) \approx Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t, \mathbf{a}_t)$. The Q-function is updated with the TD-error and the policy parameter is updated in the direction suggested by the critic, hence its name. The resulting algorithm can be seen in algorithm 3.1.

Algorithm 3.1: Q-function Actor-Critic. Adapted from [73].

Initialize $\mathbf{s}_0$, parameters $\boldsymbol{\theta}$, $\mathbf{k}$ and learning rate λ ;
Sample initial action $\mathbf{a}_0 \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_0|\mathbf{s}_0)$;
for *each step* t **do**
  Execute action $\mathbf{a}_t$;
  Sample R_t and $\mathbf{s}_{t+1} \sim \mathcal{P}\{\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\}$;
  Sample next action $\mathbf{a}_{t+1} \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$;
  Update policy parameter: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)\, Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t)$;
  Compute TD-error for Q-function approximation: $\delta = R_t + \gamma Q_{\mathbf{k}}(\mathbf{s}_{t+1}\ \mathbf{a}_{t+1}) - Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t)$;
  Update Q-function parameter: $\mathbf{k} \leftarrow \mathbf{k} + \lambda \delta \nabla_{\mathbf{k}} Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t)$;
end

The actor-critic structure provides a solution to deal with continuous state and action spaces while allowing for continuing tasks, which is suitable for this research. Still, various actor-critic algorithms exist, which differ, among others, in their learning setup, update rule, and parameterization. The next two sections present the main approaches to actor-critic algorithms in the field of RL .

3.3. Approximate Dynamic Programming

Approximate Dynamic Programming (ADP) is an algorithmic strategy that aims at solving high-dimensional optimization problems in continuous spaces, often implemented online. In other words, it is a Dynamic Programming approach that uses function approximation to deal with the curse of dimensionality. Because it is more closely related to engineering, the environment and the agent may be referred to as the plant and the controller, respectively.

In this research, ADP is used for actor-critic design employing NNs for function approximation. While the actor's structure is fixed as a mapping from state to action, the critic structure can vary along with requirements on a state-transition model. Various on-policy Adaptive Critic Designs (ACD) are implemented in the context of ADP, generally taking one of these three structures: Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP) or Global Dual Heuristic Programming (GDHP). Action-Dependent (AD) versions exist, yielding abbreviations ADHDP, ADDHP and ADGDHP [63]. Each ACD type is studied in subsection 3.3.1 and an overview is presented in Table 3.1. Their state-of-the-art applications to flight control are discussed in subsection 3.3.2.

3.3.1. Adaptive Critic Designs Overview

(Action-Dependent) Heuristic Dynamic Programming

HDP is a form of ACD where the critic receives the state $\mathbf{s}_t$ as input and outputs the estimated state value function $V(\mathbf{s}_t)$.

Critic training can be done directly by accessing the environment state $\mathbf{s}_t$. Actor training, on the other hand, requires the derivative of $V(\mathbf{s}_t)$ wrt. the action $\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{a}_t} = \frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}\frac{\partial \mathbf{s}_t}{\partial \mathbf{a}_t}$. While the first partial derivative $\left(\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}\right)$ can be retrieved by backpropagation through the critic network, the second one $\left(\frac{\partial \mathbf{s}_t}{\partial \mathbf{a}_t}\right)$ requires a state-transition model.

The AD version, however, receives both the state $\mathbf{s}_t$ and action $\mathbf{a}_t$ as inputs, allowing it to output $Q(\mathbf{s}_t, \mathbf{a}_t)$ instead. This means that the derivative of $Q(\mathbf{s}_t, \mathbf{a}_t)$ wrt. the action $\frac{\partial Q(\mathbf{s}_t, \mathbf{a}_t)}{\partial \mathbf{a}_t}$ can be obtained by backpropagation through the critic network, eliminating the need for a state-transition model [63].

ADHDP was found to handle better external disturbances but had a lower convergence probability and an overall worse performance than HDP [77]. Considering that action-dependence adds to memory usage and increases computational cost, ADHDP is not preferred.

(Action-Dependent) Dual Heuristic Programming

DHP is similar to HDP except from the fact that the critic outputs the gradient of the value function $\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$, instead of the value function itself. This requires both a state-transition model and the actor to provide partial derivative targets during critic training. On the other hand, actor training is simplified since $\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$ is directly obtained from the critic's output but still requires a state-transition model for $\frac{\partial \mathbf{s}_t}{\partial \mathbf{a}_t}$.

Estimating the derivative of $V(\mathbf{s}_t)$ instead of its value avoids finding its gradient through backpropagation, the latter often resulting in a coarse function [55]. As a result, DHP yields a higher approximation accuracy and more optimal control than HDP and performs better upon sudden changes in the environment [78].

In ADDHP, the critic outputs both $\frac{\partial Q(\mathbf{s}_t, \mathbf{a}_t)}{\partial \mathbf{s}_t}$ and $\frac{\partial Q(\mathbf{s}_t, \mathbf{a}_t)}{\partial \mathbf{a}_t}$. Actor training is therefore even further simplified by not requiring a state-transition model and directly using $\frac{\partial Q(\mathbf{s}_t, \mathbf{a}_t)}{\partial \mathbf{a}_t}$. Critic training, however, still requires a state-transition model. A summary of model requirements for each structure is shown in Table 3.1.

(Action-Dependent) Global Dual Heuristic Programming

GDHP is more complex in the sense that is combines HDP and DHP, outputing both $V(\mathbf{s}_t)$ and $\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$. It is implemented by having an HDP-like critic that outputs $V(\mathbf{s}_t)$, supplemented by a dual network to estimate $\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$. The dual network receives $V(\mathbf{s}_t)$ as input from the HDP-like critic network and the state of all its hidden neurons. Because GDHP is partially based on DHP, it also needs a state-transition model for training.

As GDHP minimizes the error wrt. both $V(\mathbf{s}_t)$ and its gradient, the tracking performance is slightly higher than DHP. Still, the more complex training structure and resulting increased computational complexity does not outweigh the increased performance for most applications [55].

Similarly to ADDHP, ADGDHP requires a state-transition model for the critic's training. It was seen that faster training could be achieved compared to GDHP in the case of recurrent action networks [55].

Incremental Approximate Dynamic Programming

Some past research has explored techniques to reduce model dependence using AD structures. Based on ADHDP's delayed TD-error of the critic objective function first introduced in [62], a finite difference method is used to estimate the gradient of the value function in the critic network error for ADDHP in [49]. While the implementation is successful, a more accurate estimate of the value function gradient is needed for successful online learning applied to complex systems [49, 84].

Another approach to alleviating the model dependence of ADP structures is the use of incremental model identification, a technique known as Incremental Approximate Dynamic Programming (iADP). The environment, or non-linear plant, is approximated by a local, time-varying, linearized incremental model, avoiding expensive non-linear approximators and resulting in a low computational complexity. It suffers, however, from instability when subject to external disturbances in the initial learning phase, which can compromise its online applicability due to safety concerns.

When adapted to HDP, the method is known as IHDP. It uses a quadratic cost-to-go function and requires an initial offline training phase [71]. It was successful on simple angle-of-attack and pitch rate controllers in [83] and more complex pitch and roll rate controllers in [14].

It was also implemented to DHP, thus becoming IDHP. A major improvement is that learning can now be fully online. Provided a high enough sampling rate, plant dynamics of a failed simple system can be identified online, thereby enabling fault-tolerance [71]. Performance in terms of learning efficiency and accuracy was improved compared to DHP on a simple angle-of-attack and pitch rate control task [84]. To reduce instability in the initial learning phase, Recursive Least-Squares (RLS) over ordinary least-squares are used to help keep model parameters stable. Still, on this simple task, failures reached 7% of online trials. It is recognized that IDHP applied to more complex systems may fail to adapt to sudden system faults online [84].

Incremental model identification was also applied to GDHP (i.e. IGDHP) on a simple angle-of-attack and pitch rate tracking task. Similarly to the comparison between IDHP and DHP, the performance of IGDHP was improved compared to GDHP [71].

It should be noted that iADP is only applied to non-AD structures. The actor's partially reduced model dependence in AD versions is outweighed by the ability of iADP to eliminate model dependence altogether.

Incremental online model identification for ADP is a promising method for model-free RL applications, compared to other ADP structures shown in Table 3.1. To the author's best knowledge, no direct comparison of IHDP, IDHP, and IGDHP have been made. It is, however, safe to assume that, similarly to their non-AD counterparts, IDHP remains the best compromise in terms of performance and complexity. Still, it should be noted that the controller gives an unreliable optimal policy in the initial learning phase, which can result in failures. To increase safety, a switching control strategy with a PID controller for the initial learning phase was proposed [38]. This approach, however, is not model-free as it requires an approximate model for PID tuning.

The next subsection explores how ADP was applied to relevant flight control applications.

Table 3.1: ACD structures overview and characteristics comparison.

Structure	Critic Structure		Model-Based		Source
	Input	Output	Actor	Critic	
Heuristic					
HDP	$\mathbf{s}_t$	$V(\mathbf{s}_t)$	✓		[78]
ADHDP	$\mathbf{s}_t, \mathbf{a}_t$	$Q(\mathbf{s}_t, \mathbf{a}_t)$			[62]
IHDP	$\mathbf{s}_t$	$V(\mathbf{s}_t)$			[83]
Dual Heuristic					
DHP	$\mathbf{s}_t$	$\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$	✓	✓	[78]
ADDHP	$\mathbf{s}_t, \mathbf{a}_t$	$\frac{\partial Q(\mathbf{s}_t,\mathbf{a}_t)}{\partial \mathbf{s}_t}, \frac{\partial Q(\mathbf{s}_t,\mathbf{a}_t)}{\partial \mathbf{a}_t}$		✓	[63]
IDHP	$\mathbf{s}_t$	$\frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$			[84]
Global Dual Heuristic					
GDHP	$\mathbf{s}_t$	$V(\mathbf{s}_t), \frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$	✓	✓	[55]
ADGDHP	$\mathbf{s}_t, \mathbf{a}_t$	$Q(\mathbf{s}_t, \mathbf{a}_t), \frac{\partial Q(\mathbf{s}_t,\mathbf{a}_t)}{\partial \mathbf{s}_t}, \frac{\partial Q(\mathbf{s}_t,\mathbf{a}_t)}{\partial \mathbf{a}_t}$		✓	[63]
IGDHP	$\mathbf{s}_t$	$V(\mathbf{s}_t), \frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{s}_t}$			[71]

Note: the actor structure is identical for all structures with input $\mathbf{s}_t$ and output $\mathbf{a}_t$.

3.3.2. State-of-the-art ADP Applications to Flight Control

ADP methods were reviewed in the previous subsection. It remains to be seen if they are suitable for this research. The lateral and longitudinal coupled-dynamics of the Cessna Citation 500 aircraft correspond to a large MIMO system with 6 Degrees of Freedom (DoF). To help answer this question, ADP applied in previous flight control applications to 6-DoF fixed-wing aircraft or helicopter models are investigated. Applications to quadcopters, 3-DoF, or small-scale aircraft models are excluded from this study as it is difficult to generalize their findings to systems with higher complexity or slower dynamics.

Coupled-Dynamics Business Jet Flight Controller: DHP

The first ACD application to a simulated coupled-dynamics 6-DoF flight control task was done using DHP in [17] for a business jet aircraft. Learning is split between an offline phase using prior well-established knowledge of control theory and an online phase where the weights are updated to reflect actual non-linear plant dynamics.

The initial offline learning phase uses known Linear Time Invariant (LTI) control laws that satisfy safety and performance requirements, initialized at known operating points both in the actor and critic networks.

In the second online learning phase, backpropagation through actor and critic networks is used. Differently from classic DHP, the actor network structure aims at representing a Proportional-Integral (PI) controller. For that purpose, it is split into three distinct networks representing the forward gain matrix (fixed during online training), feedback gain matrix, and command-integral gain matrix.

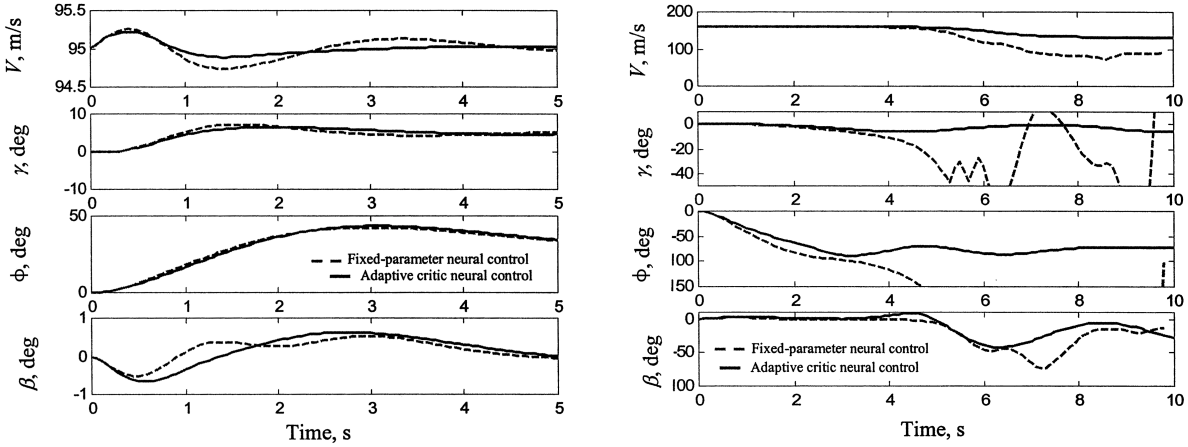
The performance of the DHP-based controller is compared to that of a non-adaptive fixed Proportional-Integral (PI) controller, using the same prior knowledge at the same operating points. It is seen that on a simple task with little coupling, the adaptive controller rapidly but only slightly outperforms the linear one. This research, however, shows that 6-DoF coupled flight control tasks can be performed through adaptive control, partially online.

Follow-up research using the same simulation setup has been conducted in [18] to investigate further different flight scenarios. Four scenarios are presented, one with a low-bank angle climbing

turn, one with a large-bank angle climbing turn, one with system failures, and one with parameter variations.

The first scenario corresponds to a climb angle $\gamma = 5°$ and roll angle $\Phi = 30°$, a maneuver that involves little coupling. Again, the adaptive controller's performance is measured against a fixed Proportional-Integral (PI) controller. While the amplitude of the velocity V, flight path angle, and oscillations of the sideslip angle β are reduced as shown in Figure 3.4a, overall performance remains similar.

In the second scenario, a horizontal high-bank angle turn with $\Phi = -70°$ is simulated, involving much more coupling as the lift vector cannot sustain the altitude anymore. In this case, the fixed Proportional-Integral (PI) controller is unable to deal with non-linearities and coupling effects, resulting in an unrecoverable stall seen from the flight path angle in Figure 3.4b. On the other hand, the adaptive controller quickly learns the coupling effects to remain stable and reaches $\Phi = -70°$ after 3s. While the velocity V, flight path angle γ remain almost constant, the sideslip angle β is large and exhibits little-damped oscillations even after 10s.



(a) Reference signals: $\Phi = 30°$ and $\gamma = 5°$.

(b) Reference signals: $\Phi = -70°$ and $\gamma = 0°$.

Figure 3.4: Comparison between DHP (solid line) and fixed PID controller (dashed line). From [18].

The third scenario investigates the controller's adaptiveness to system failures, including asymmetric loss of thrust and immobile rudder at $\delta_r = -15°$. It is seen that reference signals are followed more closely, the oscillation amplitude is reduced and control resources are used more efficiently with the adaptive controller.

The fourth scenario explores the effect of reduced control surface effectiveness parameters. It is seen that the adaptive controller adapts itself to these unforeseen changes by showing a very similar performance to a Proportional-Integral (PI) controller with constant effectiveness parameters.

This experiment has shown that for three out of four flight scenarios, the fixed PI controller had worse performance but remained stable. It became clear that a coupled-dynamics adaptive controller was required in the second scenario, where coupling effects became dominant. This research also demonstrates that a large system can learn online from sudden environment changes. It should be noted, however, that this implementation is not model-free as these results were obtained based on a reference (linear) model used in the initialization phase.

Coupled-Dynamics Helicopter Flight Controller: ADHDP

Another application of ADP to a 6-DoF coupled-dynamics flight controller for a non-linear sophisticated model of the Apache helicopter is shown in [16]. A modified version of ADHDP is employed, thus not requiring a model of the environment in theory, as per Table 3.1.

The structure of the actor network is modified to sample learning efficiency while integrating coupling effects. Inspired by traditional flight controllers, the three encapsulated loops of translational velocities (u, v, w), attitudes (θ, Φ, Ψ) and body rates (p, q, r), going from the outer to the inner one, are replaced by individual NNs in series as seen in Figure 3.5. This way, physical relationships between states

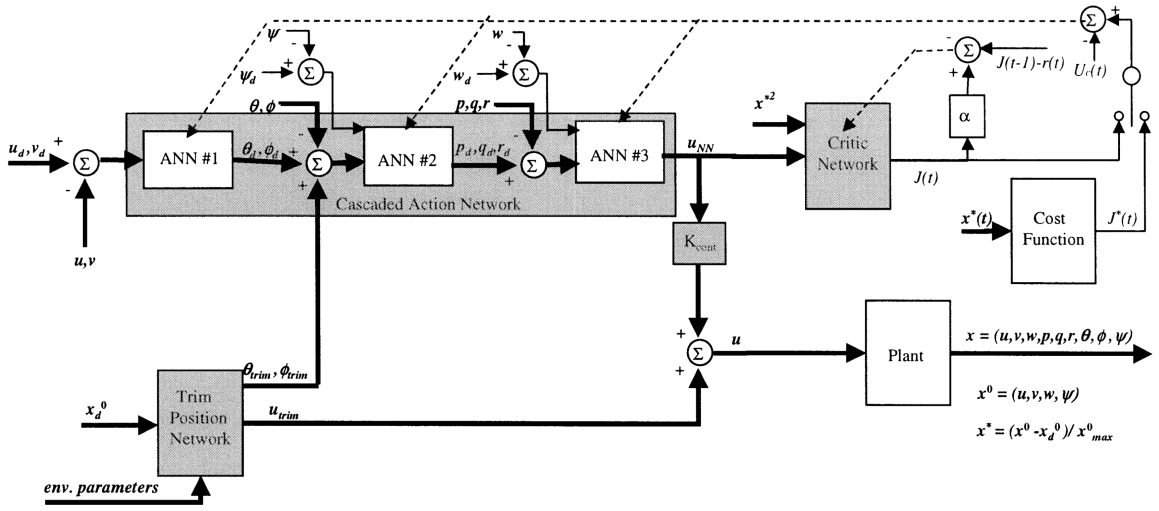


Figure 3.5: Cascaded ADHDP structure for helicopter flight control. Difference in nomenclature: x is used as state $\mathbf{s}$, u as action $\mathbf{a}$ and $J(t)$ as $Q(\mathbf{s}_t, \mathbf{a}_t)$. Subscript d means 'desired'. From [16].

are being taken advantage of. Compared to having multiple SISO disconnected controllers, coupling effects between longitudinal and lateral motions can be learned in this cascaded structure. On the other hand, one large network receiving all states at once may capture coupled dynamics but would most likely make learning more difficult.

Another particularity of this experiment is the use of a pre-trained trim position network, required for such non-linear systems, according to the authors. This structure provides nominal trim control positions at desired operating conditions using prior knowledge and accelerates learning. The control action is a combination of these trim positions u_{trim} and the deviations u_{NN} calculated by the actor, as seen Figure 3.5. The critic and actor networks, on the other hand, are model-free and initialized randomly.

The learning process is separated into two phases. An initial offline phase allows for critic and actor training, which needs many trials with tolerance for failures. In a second online phase, learning continues with restrictions on weight updates to avoid unlearning.

The resulting controller is successful in tracking and stabilization tasks for varying turbulence conditions in forward decelerating flight, provided deceleration remains low. Interestingly, the controller learns more quickly and reliably with turbulence, explained by the higher excitation in the networks, favorable for weight update. With turbulence, it maintains a 97% success rate with an acceleration of $0.19g$, while it drops to 73% without wind when experiencing an acceleration of $0.25g$. This indicates that the model is quite sensitive to turbulence conditions and maneuver type.

This research shows an innovative structure for the NN to improve the learning efficiency yet capture all coupling effects. Its design, however, requires expert knowledge of the system as it cannot be readily applied to any complex non-linear system. Lastly, although the ADHDP structure is model-free, this application makes use of a pre-defined trim model for initial offline training.

Decoupled-Dynamics Business Jet Flight Controller: IDHP

Recent research has focused on the online model-free coupled-dynamics control of a high-fidelity non-linear model of the Cessna Citation 500 [35], yet as the controller was limited to body rate control only, it will not be reviewed here. Other research has investigated the online model-free decoupled control of the same aircraft for attitude and altitude tracking [27, 36, 37]. In [27], IDHP is employed for on a 6-DoF control task, building on the work of [84]. It should be noted, however, that longitudinal and lateral motions are controlled by two separate controllers, considerably simplifying learning. Initial learning instability is reduced by adding a separate target critic network that slows down the policy update at the expense of a lower convergence rate. This method, however, is not fully online, as pre-tuned PID controllers provide inner-loop body rate reference signals allowing for reference altitude and roll angle

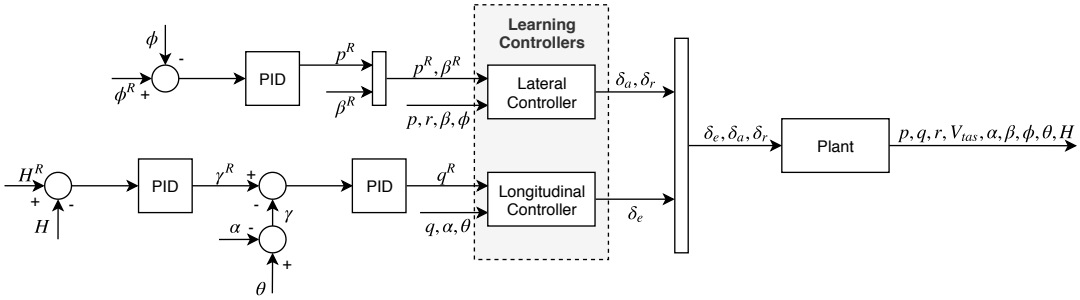


Figure 3.6: Uncoupled flight controller structure for 6-DoF altitude and roll angle control task. From [27].

tracking. The online system identification of velocity-related parameters is not simple and velocity is therefore not tracked. The resulting flight controller structure is displayed in Figure 3.6.

A short online training phase is performed during which the system is excited to learn an incremental model of the aircraft and initialize the networks. The online evaluation phase using these pre-trained values is successful in several flight scenarios. It can also adapt to a simple actuator failure implemented as a reduction in aileron effectiveness, demonstrating the fault tolerance ability of IDHP. No failures are encountered online, thanks to the separate target critic network, without which a failure rate of 5% is found.

A similar flight control application is proposed in [36], with a few key changes. While the same aircraft model and RL method are used, a shared critic between longitudinal and lateral control modes is introduced. This allows taking some coupling effects into account, as the critic can share these effects through the update to the actors. Nevertheless, such ability is not explicitly demonstrated in the simulations as the maneuvers maintained the roll angle within $\pm 25°$.

In [36], the findings of [27] that failures caused by the inherent instability of IDHP can be reduced with a separate critic target network were confirmed. Still, in this application, a failure rate of up to 27% is reached. The choice of the unbounded ReLu activation function for the critic and actor networks may have played a role in this abnormally high failure rate. Furthermore, if the simulation time was extended to approach continued learning, the author acknowledges the system might have become unstable due to the growth of these network weights.

Another research, still using IDHP on the same aircraft model, has investigated the effect of removing the outer-loop pre-tuned PID controllers [37]. It is implemented on a longitudinal-motion controller only, one version with a standard actor structure and one with a cascaded actor structure. It is seen that the standard structure in still conditions yields a better tracking performance (RMSE-measured) but has a higher failure rate. The more successful cascaded structure still has a 24% failure rate with loose requirements, indicating that learning instability is high. Confirming the observations of [16], it is found that learning is more reliable in the presence of noise in certain conditions.

Applications in [27, 36] have contributed to extend IDHP to complex 6-DoF flight control. However, its dependence on an offline-tuned PID controller diminishes the claimed benefit of online control with IDHP. When this outer controller is removed in [37], learning instability suggests that IDHP is not yet suited for outer-loop coupled-dynamics 6-DoF flight control tasks.

3.4. Deep Reinforcement Learning

A novel type of reinforcement learning called Deep Reinforcement Learning (DRL) has been developed thanks to the advancements of deep learning techniques. The term *deep* refers to the Deep (Artificial) Neural Networks (DNNs) that are at its base. It boasts the ability to deal with high-dimensional inputs such as images, identify complex patterns, and extract their features [2]. Still, DRL's ability to deal with large continuous state spaces and strong generalization power is useful in control applications. The addition of hidden layers, however, leads to a considerably longer training time. The most relevant DRL methods for this research are discussed below.

3.4.1. Deep Q-Network

DRL was created founded with the introduction of Deep Q-Network (DQN) in [45]. It builds on the foundations of Q-learning, an off-policy discrete tabular method using the update rule presented in Equation 3.13. Instead of a look-up table, a Deep Neural Network is used for function approximation, referred to as Q-network. As a value-based method, it is not able to deal with continuous action-spaces. Still, two major changes introduced compared to Q-learning are pillars for other continuous-control DRL methods. These two features are experience replay and target networks.

Experience Replay

In RL, model-free methods assume that samples are independent (i.e. uncorrelated) and identically distributed (i.e. follow the same probability distribution), also called i.i.d. [61]. As samples are usually obtained chronologically and resemble the preceding or following ones, this assumption is often violated. The Q-network weights would then be updated in a very specific direction resulting in biased training, either not reaching the global maximum or diverging [45]. Experience replay is introduced to alleviate this problem.

Transitions $(\mathbf{s}_t, \mathbf{a}_t, \tilde{r}_{t+1}, \mathbf{s}_{t+1})$ are temporarily stored in a memory-buffer in chronological order. The Q-network training is performed on a random minibatch of fixed size from the memory-buffer, as seen in algorithm 3.2. When the memory limits are reached, the memory-buffer is emptied. Because learning is dissociated from acting, DQN is classified as an off-policy method, justifying the choice of Q-learning as foundation [45]. Overall, experience replay reduces the variance of the updates and improves learning stability. It has been tested on 49 Atari games where it was able to achieve human-like performance, receiving the high-dimensional visual interface as sole input [46].

Experience replay has been improved further with the addition of prioritized experience replay [58]. By sampling more frequently from transitions with a higher TD-error, learning is intensified where the most improvement is needed. The resulting algorithm outperformed DQN with uniform experience replay on 41 of the 49 Atari games.

Target Network

DQN also integrates a separate target network [45]. A target refers to the bellman optimality for the Q-function given in Equation 3.10 and used in Q-learning for the TD-error in Equation 3.13. As the Q-function's update uses a target itself dependent on the Q-function, the target might change too rapidly, creating learning instability. Fast-moving targets are avoided by slowing down their update with a separate target network synchronized only once in a while with the Q-network. It is implemented in algorithm 3.2 by copying the current network weights to the target network weight every c steps. While learning is slowed down with a delay added between the Q-network's and the targets' updates, learning stability increases.

The target is used to compute the mean-squared Bellman error, which is used as a loss function as shown in algorithm 3.2.

3.4.2. Deep Deterministic Policy Gradient

To extend DRL to continuous control, actor-critic methods discussed in subsection 3.2.2 are presented in this section in the context of DRL. Deep Deterministic Policy Gradient (DDPG) combines DPG's actor-critic structure and DQN's learning method [40].

Deterministic Policy Gradient

DPG is an actor-critic approach that proposes a more efficient approach by reducing computations compared to traditional stochastic gradient methods [65]. Stochastic policies integrating over both the action and state spaces are usually required to explore the entire action-state. DPG reduces computations by integrating only over the state space, resulting in a deterministic policy. Exploration is ensured by making the algorithm off-policy and executing actions according to a stochastic Gaussian behavior policy during learning. This method significantly outperforms both on and off-policy stochastic gradient methods in terms of sample efficiency [65]. Nevertheless, this method employs tile-coding as a linear approximation method, which comes with the disadvantages studied in subsection 3.2.1.

Algorithm 3.2: DQN. Adapted from [45].

Initialize randomly-chosen weights $\mathbf{k}$ for critic network $Q_{\mathbf{k}}$;
Initialize weights $\bar{\mathbf{k}} \leftarrow \mathbf{k}$ for target critic network $Q_{\bar{\mathbf{k}}}$;
Initialize initial state $\mathbf{s}_0$ and memory buffer $\mathcal{D}$;
for *each step* t **do**
  Execute the ϵ-greedy action: $\mathbf{a}_t = \begin{cases} \arg\max_{\mathbf{a}'} Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}') & \text{with probability } (1-\epsilon) \\ \text{a random action} & \text{with probability } \epsilon \end{cases}$;
  Sample R_t and $\mathbf{s}_{t+1} \sim \mathcal{P}\{\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\}$;
  Store transition $(\mathbf{s}_t, \mathbf{a}_t, R_t, \mathbf{s}_{t+1})$ in $\mathcal{D}$;
  Sample a random minibatch of n transitions $(\mathbf{s}_i, \mathbf{s}_i, \tilde{r}_i, \mathbf{s}_{i+1})$ from $\mathcal{D}$;
  Compute targets: $y_i = r_i + \gamma \max_{\mathbf{a}'} Q_{\bar{\mathbf{k}}}(\mathbf{s}_{i+1}, \mathbf{a}')$;
  Update critic with one-step gradient descent by minimizing the loss:
  $L = \frac{1}{n} \sum_{i=0}^{n} \left(y_i - Q_{\mathbf{k}}(\mathbf{s}_i, \mathbf{a}_i)\right)^2$;
  Every c steps, set $\bar{\mathbf{k}} \leftarrow \mathbf{k}$;
end

Integrating DPG's structure and DQN's learning methods

DDPG takes the actor-critic strucure of DPG while implementing the learning method of DQN. Modifications to DPG to use Deep Neural Networks as non-linear function approximators are introduced with DDPG [40]. Both the actor and the critic are represented by DNNs.

Similarly to DQN, experience replay is used so that learning is done in minibatches. The purpose is to facilitate a more i.i.d. learning environment so that an unbiased value function and policy can be built.

While DQN's target network was designed for a critic-only structure, it has to be adapted to accommodate the actor-critic structure of DDPG. Both the critic and the actor are attributed a target network, found to be a key feature to avoid critic divergence. Instead of the 'hard' copy used in DQN which introduces sudden changes in the targets, 'soft' copying is proposed. The target network parameters change more slowly applying an Exponentially (Weighted) Moving Average (EMA) with previous values with smoothing factor τ, increasing learning stability overall [53].

Another challenge that DDPG faces is the disparity of units, and thus orders of magnitude, in the state vector. This is the case in flight control, with angles, positions, and velocities all being part of the state vector. It is difficult for the network to learn parameters that approximate over this heterogeneous space. Instead of manually scaling each dimension to have a uniform range, one can employ batch [31] or layer [3] normalization that makes each dimension have zero mean and unit variance. It is also useful to reduce the so-called internal covariate shift in deep networks, the effect changing layer parameters have on the following layers that receive them as inputs, requiring a slower learning rate.

Lastly, similarly to DPG, exploration is ensured by having off-policy learning. The stochastic behavioral policy corresponds to the learned deterministic policy supplemented with noise. The resulting algorithm is shown in algorithm 3.3.

Online Implementability

DDPG is successful on complex tasks such as legged locomotion, benefiting from the high abstraction ability of deep neural networks applied to continuous control [40]. Convergence was only possible after a long interaction with the environment (up to $2.5 \cdot 10^6$ steps), yet it required substantially fewer steps than DQN for the same Atari games ($50 \cdot 10^6$) [46]. This poses a challenge to implement this algorithm even partially online, a way of making the flight controller more fault-tolerant.

Nevertheless, online learning was tested on a network traffic optimization problem [82]. Similarly to [58], prioritized experience replay was used to improve stability in the initial learning phase. Exploration

Algorithm 3.3: DDPG. Adapted from [40].

Initialize randomly-chosen weights $\boldsymbol{\theta}$ and $\mathbf{k}$ for policy $\pi_{\boldsymbol{\theta}}$ and critic $Q_{\mathbf{k}}$ networks, respectively ;
Initialize weights $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$ and $\bar{\mathbf{k}} \leftarrow \mathbf{k}$ for policy $\pi_{\bar{\boldsymbol{\theta}}}$ and critic $Q_{\bar{\mathbf{k}}}$ target networks, respectively;
Initialize initial state $\mathbf{s}_0$, random process $\mathcal{N}$ and smoothing factor τ ;
Initialize memory buffer $\mathcal{D}$;
Sample initial action $\mathbf{a}_0 \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_0|\mathbf{s}_0)$;
for *each step* t **do**
  Execute action $\mathbf{a}_t = \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) + \mathcal{N}$;
  Sample R_t and $\mathbf{s}_{t+1} \sim \mathcal{P}\{\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\}$;
  Store transition $(\mathbf{s}_t, \mathbf{a}_t, R_t, \mathbf{s}_{t+1})$ in $\mathcal{D}$;
  Sample a random minibatch of n transitions $(\mathbf{s}_i, \mathbf{s}_i, \tilde{r}_i, \mathbf{s}_{i+1})$ from $\mathcal{D}$;
  Compute targets: $y_i = r_i + \gamma Q_{\bar{\mathbf{k}}}(\mathbf{s}_i, \pi_{\boldsymbol{\theta}}(\mathbf{s}_i))$;
  Update critic with one-step gradient descent by minimizing the loss:
  $L = \frac{1}{n}\sum_{i=0}^{n}\left(y_i - Q_{\mathbf{k}}(\mathbf{s}_i, \mathbf{a}_i)\right)^2$
  Update the actor policy using the sampled policy gradient:
  $\nabla_{\boldsymbol{\theta}} J \approx \frac{1}{n}\sum_{i=0}^{n} \nabla_a Q_{\mathbf{k}}(\mathbf{s}, \mathbf{a})|_{\mathbf{s}=\mathbf{s}_i, \mathbf{a}=\mathbf{a}_i} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{s})|_{\mathbf{s}=\mathbf{s}_i}$
  Update the target networks weights: $\bar{\boldsymbol{\theta}} \leftarrow (1-\tau)\boldsymbol{\theta} + \tau\bar{\boldsymbol{\theta}}$;
  $\bar{\mathbf{k}} \leftarrow (1-\tau)\mathbf{k} + \tau\bar{\mathbf{k}}$;
end

is guided by suggesting known well-performing yet sub-optimal actions, such as the shortest path one. After less than 1000 steps, the agent had reached 75% of the maximum reward encountered for a simple traffic setting, while it required 2000 steps on a more difficult scenario. This research demonstrates that online learning with DDPG is possible with a guided exploration after several thousand training steps.

DDPG was applied to quadcopter flight control, yet only in offline implementations [28, 67]. The higher task complexity makes online applicability difficult. Used for a quadcopter inner-loop flight controller, a converged policy was reached after at least 5min of training with a modified actor-critic structure [28]. While relatively short, this training time is expected to grow for the flight control of a business-jet with slower dynamics. Another challenge of online flight control applications is DDPG's off-policy aspect and its safety implications. Because of the behavioral policy's noise component, the chosen action may lead to unsafe states when learning online. Exploration may, therefore, have to be reduced.

A promising learning setup for DDPG was proposed in [39] where a combination of offline and online weight training was used. It was applied to a data processing system where processing time has to be minimized. Offline training was based on $10,000$ samples and allowed both to ensure enough exploration and faster convergence during online training. Online learning still required up to 10min to converge to a near-optimal solution. This research's main takeaway is that learning in DRL can successfully be split between an offline and online learning phase. If implemented online at all, learning on DRL algorithms is therefore suggested to be first conducted offline.

3.4.3. Proximal Policy Optimization

A new on-policy DRL approach, named Proximal Policy Optimization (PPO), benefits from increased learning stability yet aims at improving the poor sample efficiency of on-policy DRL methods. It is based on Trust Region Policy Optimization (TRPO), another continuous control DRL technique, and claims to be more sample efficient than TRPO itself and DDPG [60].

Fundamentals of Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) is a policy gradient method that is recognized for its reliable performance as it improves learning stability thanks to its on-policy approach [59]. The policy is explicitly estimated and its gradient is updated according to an objective function. Unlike previous policy-gradient methods, TRPO makes use of the advantage function $A(\mathbf{s}_t, \mathbf{a}_t) = Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t)$, which represents

the benefit of choosing action $\mathbf{a}_t$ over the action suggested by the policy at state $\mathbf{s}_t$.

Monotonic improvements are guaranteed by considering a Conservative Policy Iteration (CPI) on the surrogate objective function $J(\boldsymbol{\theta})$, given in Equation 3.16. To avoid the risk of destructively high policy updates, a Kullback–Leibler (KL) divergence constraint bounded by scalar δ is applied between the proposed and the current policy estimates given in Equation 3.17. The probability ratio $\rho_t(\boldsymbol{\theta})$ between the two estimates is introduced.

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad J^{CPI}(\boldsymbol{\theta}) = \mathbb{E}_t\left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t|\mathbf{s}_t)} A(\mathbf{s}_t, \mathbf{a}_t)\right] = \mathbb{E}_t\left[\rho_t\left(\boldsymbol{\theta}\right) A\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{3.16}$$

$$\text{subject to} \quad \mathbb{E}_t\left[\mathrm{KL}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}\left(\cdot \mid \mathbf{s}_t\right), \pi_{\boldsymbol{\theta}}\left(\cdot \mid \mathbf{s}_t\right)\right]\right] \leq \delta \tag{3.17}$$

While learning stability is improved, TRPO is rather complicated and suffers from a high sample complexity [60].

PPO Implementation

PPO aims at leveraging the benefits of TRPO yet simplifying its implementation to make it more sample efficient. The surrogate objective function makes use of the advantage function to stabilize the update. It can be computed from the critic's $\mathbf{k}-$parameterized state value function estimate $V_{\mathbf{k}}(\mathbf{s}_t)$ in Equation 3.18 as proposed in [47], where j is bounded by the maximum number of time-steps of the episode.

$$A_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{i=0}^{j-1} \gamma^i \tilde{r}_{t+i} + \gamma^j V_{\mathbf{k}}\left(\mathbf{s}_{t+j}\right) - V_{\mathbf{k}}\left(\mathbf{s}_t\right) \tag{3.18}$$

The surrogate objective function is then simplified by removing the KL divergence constraint. Instead, it is clipped as shown in Equation 3.19, with clipping threshold ϵ (0.2 by default).

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad J^{CLIP}(\boldsymbol{\theta}) = \mathbb{E}_t\left[\min\left(\rho_t\left(\boldsymbol{\theta}\right) A\left(\mathbf{s}_t, \mathbf{a}_t\right), \text{clip}\left[\rho_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon\right] A\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)\right] \tag{3.19}$$

The probability ratio $\rho_t(\boldsymbol{\theta})$ is clipped to the interval $[1-\epsilon, 1+\epsilon]$ while the final surrogate objective is the pessimistic bound with the unclipped objective. As a result, it was found that the algorithm is cautious enough with actions displaying a positive advantage while giving enough chance to actions with a negative one [8]. The maximization is implemented with a stochastic gradient ascent. An entropy bonus may be added to the surrogate objective to increase exploration.

The task can be run on multiple (N) threads at the same time to reduce training time. This means that each actor-learner will likely propose dissimilar updates, which in practice ensures samples are i.i.d.. As a result, the need for a memory buffer is eliminated. Still, to avoid each actor-learner overwriting each other's update, several gradients are accumulated over T time-steps, to be updated all at once in a minibatch of size NT. The resulting algorithm is presented in algorithm 3.4. It should be noted that learning occurs offline and no research so far has focused on implementing it online.

Comparison with DDPG and TRPO

It is interesting to compare how PPO performs wrt. DDPG and TRPO on a complex task such as flight control. Each method is implemented in [34] for the attitude control of a quadcopter. The average reward during training is shown in Figure 3.7.

Algorithm 3.4: PPO. Adapted from [60].

Initialize randomly-chosen weights $\boldsymbol{\theta}$ and $\mathbf{k}$ for policy $\pi_{\boldsymbol{\theta}}$ and critic $V_{\mathbf{k}}$ networks, respectively ;
Initialize clipping threshold ϵ ;
for *each step* t **do**
  for *each thread* $j = 1, 2, ..., N$ **do**
    Run policy $\pi_{\boldsymbol{\theta}}$ in environment for T time-steps ;
    Compute advantage estimates $A(\mathbf{s}_i, \mathbf{a}_i)$ for $i = 1, 2, \ldots, T$;
  end

  Update the actor policy by maximizing the surrogate objective function:
  $\boldsymbol{\theta}_{t+1} = \arg\max_{\boldsymbol{\theta}} \frac{1}{NT} \sum_{j=0}^{N} \sum_{i=0}^{T} \min\left(\rho_i\left(\boldsymbol{\theta}_t\right) A\left(\mathbf{s}_i, \mathbf{a}_i\right), \text{clip}\left[\rho_i(\boldsymbol{\theta}_t), 1-\epsilon, 1+\epsilon\right] A\left(\mathbf{s}_i, \mathbf{a}_i\right)\right)$;

  Update critic with one-step gradient descent by minimizing the loss:
  $L = \frac{1}{NT} \sum_{j=0}^{N} \sum_{i=0}^{T} \left(V_{\mathbf{k}}(\mathbf{s}_i) - R_i\right)^2$;
end

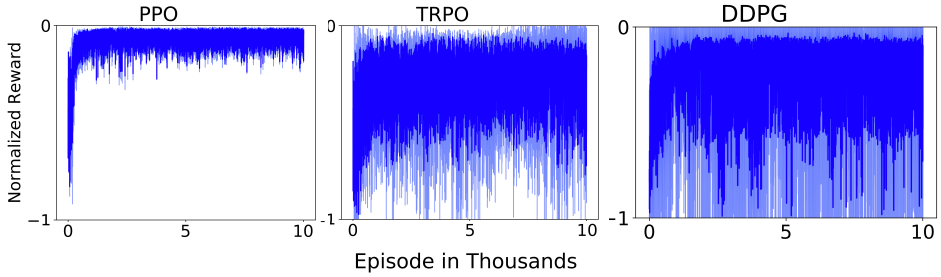


Figure 3.7: Comparison between PPO, TRPO and DDPG on the average normalized reward during training for a quadcopter flight control task. Light blue represents 95% confidence interval. Adapted from [34].

It is clear that PPO not only converges faster than the two other methods but also receives higher rewards on average. The training was done using 10^7 steps (on 10^4 episodes) and was 72% faster for PPO than for DDPG. PPO has been shown to outperform DDPG (and classical PID controllers) for a quadcopter flight control task, on all performance metrics including rise time, overshoot, error and stability [34].

3.4.4. Soft Actor-Critic

Recent works have aimed at improving the learning stability of DDPG while retaining its off-policy approach for increased sample efficiency. First, an overview of extensions to DDPG is proposed. The most promising option, Soft Actor-Critic (SAC) is explained in more detailed hereafter.

Extentions to DDPG

One of the drawbacks of DDPG is its tendency to overestimate the value function. An extension to DDPG named Twin-Delayed Deep Deterministic Policy Gradient (TD3) was published in 2018, aiming at diminishing this effect [19]. One of its features is that it learns two twin Q-functions and chooses a pessimistic bound over the two during the policy update. It was found that to reduce further the variance in the presence of target networks, the policy should be updated at a lower frequency than the Q-function. This is so that the Q-function error is first reduced before using it to update the policy. The authors suggest updating the policy at half the rate of the Q-function. Lastly, a small amount of noise, clipped around the chosen action, is added to the target action to deter the agent from choosing overestimated Q-function values. It outperforms DDPG on several control tasks such as a 2D walker and an inverted pendulum in terms of learning speed and average return [19].

Another extension to DDPG was proposed in 2018, called Soft Actor-Critic (SAC) [23]. Unlike DDPG and TD3, it uses a stochastic policy that is inherently more stable during learning yet keeping

the off-policy update of DDPG for better sample efficiency. It has been compared to DDPG and PPO on multiple control tasks, showing more stable learning with increased average return and sample efficiency [23].

It is not clear according to the respective authors which of these two methods performed best, as their results may be biased by a lack of hyper-parameter tuning, stochasticity and unalike network characteristics. An independent study tested both algorithms on five complex robotic tasks from the Mujoco environment [1]. Each algorithm was run on 10^6 training steps on 10 different seeds, with identical network and minibatch size for a fair comparison. SAC outperformed TD3 in terms of final return on 4 out of 5 tasks. Furthermore, while TD3 has been successful on a legged robotic task only in a simulated environment [12], SAC was able to teach walking within two hours of online learning in a real-world experiment [24]. Current research results suggest that SAC has an overall higher sample efficiency than TD3, and is therefore selected as the most promising extension to DDPG.

Implementation

SAC is an off-policy method that optimizes a stochastic policy. Exploration is an inherent mechanism to SAC thanks to the stochastic nature of its policy. An entropy term $\mathcal{H}$ is added to the objective function to control this mechanism. It measures the randomness of any probability distribution P as seen in Equation 3.20.

$$\mathcal{H}(P) = \mathop{\mathbb{E}}_{x\sim P}[-\log P(x)] \tag{3.20}$$

In the context of SAC, it measures the randomness in the policy at a given state. Incorporating it into the Q-function, exploration of high-entropy regions is encouraged and early convergence to sub-optimal solutions is avoided. It is traded-off against the expected return with the temperature hyperparameter η in the updated Bellman equation in Equation 3.21 as proposed in [23].

$$\begin{aligned} Q_{\mathbf{k}}(\mathbf{s}_t, \mathbf{a}_t) &= \mathop{\mathbb{E}}_{\substack{\mathbf{s}'\sim\mathcal{P}\\ \mathbf{a}'\sim\pi_{\boldsymbol{\theta}}}} \left[\tilde{r}' + \gamma \left(Q_{\mathbf{k}}\left(\mathbf{s}', \mathbf{a}'\right) + \eta\mathcal{H}\left(\pi_{\boldsymbol{\theta}}\left(\cdot \mid \mathbf{s}'\right)\right)\right)\right] \\ &= \mathop{\mathbb{E}}_{\substack{\mathbf{s}'\sim\mathcal{P}\\ \mathbf{a}'\sim\pi_{\boldsymbol{\theta}}}} \left[\tilde{r}' + \gamma \left(Q_{\mathbf{k}}\left(\mathbf{s}', \mathbf{a}'\right) - \eta \log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}' \mid \mathbf{s}'\right)\right)\right] \end{aligned} \tag{3.21}$$

The expectation term can be dropped by sampling states from the memory buffer while actions are obtained from the current policy. Two Q-functions called twin critics are learned to avoid overestimation. From Equation 3.21, a single critic target is computed with the minimum value of the twin critics to obtain the mean-squared Bellman error, as shown in algorithm 3.5. Similarly to DDPG, the critic target is updated with an EMA.

The policy is updated at each step according to Equation 3.22, using the objective in Equation 3.23 with a KL divergence, similarly to TRPO, as proposed in [24]. $Z_{\mathbf{k}}$ is a normalization function.

$$\pi_{\boldsymbol{\theta}_{\text{new}}} = \arg\min_{\boldsymbol{\theta}} J_{\pi}(\boldsymbol{\theta}) \tag{3.22}$$

$$\begin{aligned} J_{\pi}(\boldsymbol{\theta}) &= \mathop{\mathbb{E}}_{\mathbf{s}'\sim\mathcal{P}} \left[KL\left[\pi_{\boldsymbol{\theta}}\left(\mathbf{a}' \mid \mathbf{s}'\right), \frac{\exp\left(Q_{\mathbf{k}}^{\pi_{\text{old}}}\left(\mathbf{s}', \mathbf{a}'\right)\right)}{Z_{\mathbf{k}}^{\pi_{\text{old}}}\left(\mathbf{s}'\right)}\right]\right] \\ &= \mathop{\mathbb{E}}_{\mathbf{s}'\sim\mathcal{P}} \left[\mathop{\mathbb{E}}_{\mathbf{a}'\sim\pi_{\boldsymbol{\theta}}} \left[\log \pi_{\boldsymbol{\theta}}\left(\mathbf{a}' \mid \mathbf{s}'\right) - Q_{\mathbf{k}}\left(\mathbf{s}', \mathbf{a}'\right) + \log Z_w\left(\mathbf{s}'\right)\right]\right] \end{aligned} \tag{3.23}$$

The objective is optimized with a reparameterization trick, to make sure that sampling from the policy $\pi_{\boldsymbol{\theta}}$ is a differentiable process. Actions are sampled from $\mathbf{a}' = f_{\boldsymbol{\theta}}(\boldsymbol{\xi}; \mathbf{s}')$, where $\boldsymbol{\xi}$ is an input noise vector sampled from random process $\mathcal{N}$. The objective can be rewritten as shown in Equation 3.24, where $Z_{\mathbf{k}}$ is omitted since it does not depend on the policy parameter $\boldsymbol{\theta}$ [23].

$$J_{\pi}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{s}'\sim\mathcal{P}} \left[\mathop{\mathbb{E}}_{\boldsymbol{\xi}\sim\mathcal{N}} \left[\log \pi_{\boldsymbol{\theta}}\left(f_{\boldsymbol{\theta}}(\boldsymbol{\xi}; \mathbf{s}') \mid \mathbf{s}'\right) - Q_{\mathbf{k}}\left(\mathbf{s}', f_{\boldsymbol{\theta}}(\boldsymbol{\xi}; \mathbf{s}')\right)\right]\right] \tag{3.24}$$

The policy does not have targets, nor it is manually smoothed like in DDPG since the policy's inherent stochasticity produces a similar effect. When testing the policy, stochasticity is removed by taking the mean action of the probability distribution.

It was found that the SAC algorithm is brittle with respect to the temperature hyperparameter η. Optimal entropy is not only dependent on the learning task but also on the learning state of the policy. An updated version of SAC proposes to adjust it automatically based on another optimization problem [24]. Objective J_η is introduced to ensure that a minimum entropy is satisfied while maximizing the return.

Algorithm 3.5: SAC. Adapted from [24].

Initialize randomly-chosen weights $\boldsymbol{\theta}$, $\mathbf{k}_1$ and $\mathbf{k}_2$ for policy $\pi_{\boldsymbol{\theta}}$ and twin critic $Q_{\mathbf{k}_j}$ networks, respectively ;
Initialize weights $\bar{\mathbf{k}}_j \leftarrow \mathbf{k}_j$ for twin critic $Q_{\bar{\mathbf{k}}_j}$ target networks, with $j = 1, 2$;
Initialize initial state $\mathbf{s}_0$ and smoothing factor τ ;
Initialize memory buffer $\mathcal{D}$;
Sample initial action $\mathbf{a}_0 \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_0|\mathbf{s}_0)$;
for *each step* t **do**
  Execute action $\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$;
  Sample R_t and $\mathbf{s}_{t+1} \sim \mathcal{P}\{\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\}$;
  Store transition $(\mathbf{s}_t, \mathbf{a}_t, R_t, \mathbf{s}_{t+1})$ in $\mathcal{D}$;
  if *it's time to update* **then**
    Sample a random minibatch of n transitions $(\mathbf{s}_i, \mathbf{s}_i, \tilde{r}_i, \mathbf{s}_{i+1})$ from $\mathcal{D}$;
    Compute targets for the twin critics: $y_i = \tilde{r}_i + \gamma \left(\min_{j=1,2} Q_{\bar{\mathbf{k}}_j}(\mathbf{s}_i, \mathbf{a}') - \eta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}'|\mathbf{s}_i) \right)$ with $\mathbf{a}' \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}'|\mathbf{s}_i)$;
    Update twin critics with one-step gradient descent by minimizing the loss wrt. $\mathbf{k}_j$: $L = \frac{1}{n} \sum_{i=0}^{n} \left(y_i - Q_{\mathbf{k}_j}(\mathbf{s}_i, \mathbf{a}_i) \right)^2$ for $j = 1, 2$;
    Update the actor policy with one-step gradient ascent using objective J_π from Equation 3.24 wrt. $\boldsymbol{\theta}$;
    Update the temperature hyperparameter η with one-step gradient ascent using objective J_η ;
    Update the target networks weights: $\bar{\mathbf{k}}_j \leftarrow (1 - \tau)\mathbf{k}_j + \tau \bar{\mathbf{k}}_j$ for $j = 1, 2$;
  end
end

SAC is recognized as one of the state-of-the-art DRL algorithms with a proven record of successful real-world applications on continuous robotic tasks. Although not safety-critical in offline learning, a high sample efficiency is desired to make controller development easier. It has consistently outperformed previous state-of-the-art DRL algorithms such as DDPG, PPO, and TD3 in terms of sample efficiency, partly thanks to its stochastic policy that encourages exploration and avoids early convergence to sub-optimal policies.

SAC was successfully applied to UAV path planning in [11] and later to the low-level control of a quadcopter in [5]. The latter demonstrated SAC's robustness to unfavorable initial conditions. It has, however, not yet been applied to fixed-wing aircraft flight control tasks to the author's best knowledge. It has also not been implemented for online learning yet, although it is expected to face the same difficulties as DDPG. Its use on failed systems for robust control also remains largely unexplored.

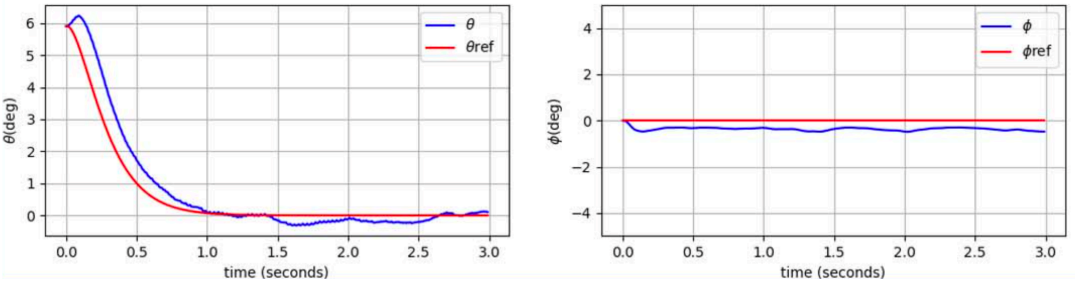
3.4.5. State-of-the-art DRL Applications to Flight Control

DRL was initially applied to games with a raw visual interface as input [40, 46], and was later expanded to robotics with camera-based sensors, and lately to flying devices on quadcopters [5, 28, 34, 42, 67]. Only recently have DRL algorithms been employed for 6-DoF fixed-wing aircraft flight control, which is most relevant for the Cessna Citation model. These few state-of-the-art applications are discussed below.

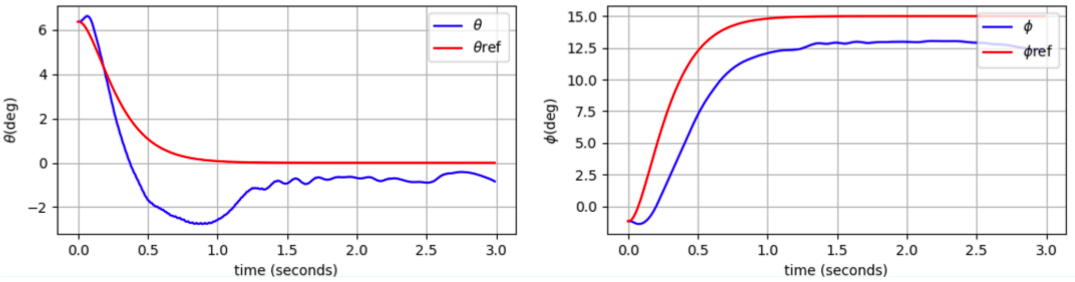
Coupled-Dynamics Fixed-Wing UAV Flight Controller: DDPG

DDPG was applied for the coupled-dynamics flight control of a 6-DoF small fixed-wing UAV model in [74]. Learning is done offline, on four different consecutive training sets that excite either the longitudinal, lateral, or directional mode. In each mode, one or two control surfaces had a reduced allowable range to reduce the space to explore and make convergence faster. In the testing phase, the behavioral policy's noise component was removed to ensure safe and optimal actions are executed.

It was seen that after 500 training episodes on the first training set, the policy had converged while still making mistakes resulting in a sparsely negative reward. Nevertheless, the following training sets inherited the previously trained networks and the agent could achieve higher rewards on average.



(a) Single attitude angle change tracking task.



(b) Multiple attitude angle changes tracking task.

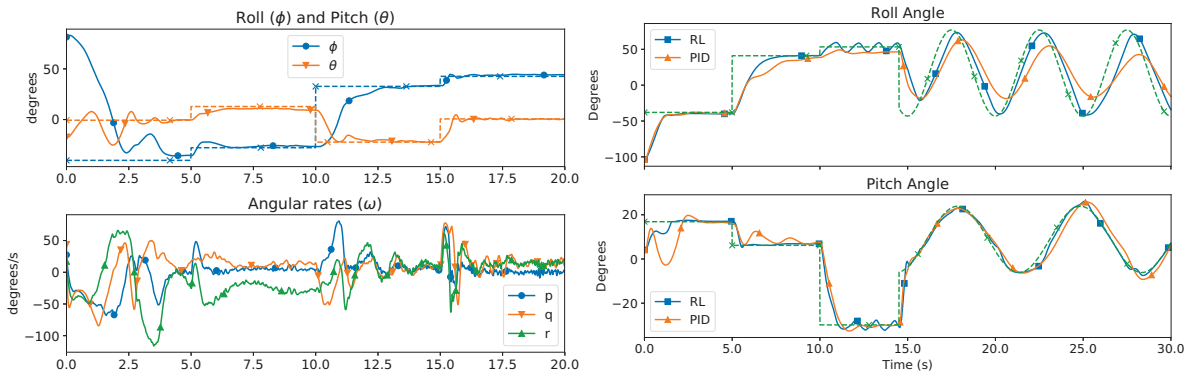
Figure 3.8: Attitude reference tracking task for a DDPG agent. From [74].

The agent response was evaluated on several tracking tasks. It is relatively successful when tracking one attitude angle change as displayed in Figure 3.8a, with only small steady-state errors and little oscillations. Its performance was poorer when tracking multiple attitude angle changes simultaneously, as shown in Figure 3.8b. A large overshoot in the pitch response and steady-state error on the roll angle were present. Despite its mixed results, this application showed that consecutive offline training for different modes was possible and allowed to speed up learning.

Coupled-Dynamics Unmanned Flying-Wing Flight Controller: PPO

Another DRL algorithm, PPO, was used for the coupled-dynamics flight control of an unmanned small flying-wing model in [8]. The motivation for choosing PPO came from its superior performance for quadcopter attitude control compared to DDPG and TRPO [34]. Training was performed offline in an episodic fashion, with the level of difficulty increasing over time as initial conditions are made more difficult. It required $2 \cdot 10^6$ steps to perform training, equivalent to 1hr on the hardware at hand.

The agent was tested in an environment with severe wind and turbulence, while it was only trained in a disturbance-free environment. Figure 3.9a showed that the agent could deal with this unseen



(a) Severe wind conditions ($20 ms^{-1}$) and turbulence.

(b) Comparison with PID controller response without turbulence.

Figure 3.9: Attitude reference (dashed-lines) tracking task for a PPO agent. From [8].

situation and track the attitude references relatively well. Compared to a PID controller on Figure 3.9b, the PPO agent showed similar performance overall between the roll and pitch angle, in terms of rise time and overshoot. Nonetheless, the PPO agent exhibited a smaller error in the transient response compared to the PID controller for both attitude angles.

This research showed that PPO is a suitable RL technique for coupled-dynamics flight control and even demonstrated its ability to generalize to unforeseen situations. It remains to be seen if this approach can be applied to larger aircraft and to what extent part of its learning can be brought online.

3.5. Challenges of RL for Flight Control

Past research on RL for flight control has demonstrated the ability to provide model-free fault-tolerant control. State-of-the-art research implementing iADP or DRL techniques, however, show some common limitations.

All RL methods suffer from a relatively low sample efficiency, making the learning of complex tasks difficult. Two approaches to this problem emerge. One is to reduce the controller size by using uncoupled, parallel or cascaded RL controllers, possibly in conjunction with outer-loop PID controllers. A higher sample efficiency for each sub-controller is obtained and allows for online learning on adequate tasks. This approach, mostly used in iADP methods, comes with the difficulty of capturing the full dynamics of complex coupled systems due to the limited sample collection time.

Another approach is to keep the full controller complexity entailing a lower sample efficiency, and learn over more training steps. For safety and practical reasons, a large part of the learning process has to be done offline and thus requires a simulation model. This approach is typically used in DRL, as their DNNs can capture the dynamics of more complex systems but require a large number of samples for training.

Both iADP or DRL generally use NNs as function approximators. Despite their superior approximation power, they suffer from high variability due to parameter initialization. This effect will be reduced by using Xavier initialization, which minimizes the risk of exploding or vanishing gradients [20]. iADP online performance was found to considerably vary with initial parameter variance, imposing its tuning to ensure successful runs [36]. On the other hand, offline learning has the advantage of allowing for longer training and thus cope better with unfavorable parameter initialization.

As a non-unsupervised machine learning framework, successful RL applications heavily rely on the designer's choice for reward function and observations. For this research, commonly used reward functions will be tested and iterated if necessary. Similarly, observation signals and their scaling will be adjusted to minimize steady-state errors while allowing for fast initial learning.

3.6. Conclusion

This literature review presented the foundations of the Reinforcement Learning (RL) framework and explored state-of-the-art applications to flight control. RL was identified as a self-learning technique for control systems in section 3.1. Research question *Q1.1* was answered by studying various RL algorithm characteristics and learning conditions. It was found that the RL algorithm should be model-free, able to deal with failures by being either robust or adaptive, and have both continuous action and state spaces. For continuous control, the actor-critic structure was identified as the most suitable in section 3.2.

Various implementations of the actor-critic structure were explored above, in section 3.3 and section 3.4. Three options satisfy the requirements detailed as part of research question *Q1.1*, which are summarized below.

The first option is Action-Dependent Heuristic Dynamic Programming (ADHDP), a type of Approximate Dynamic Programming (ADP). It has the advantage of being model-free and can learn online, potentially adapting to unexpected failures. Despite that, it has a more complex structure and worse performance than DHP.

The second option is a more recent development in ADP, Incremental Dual Heuristic Programming (IDHP). It is a model-learning technique that can learn online such that it can adapt to unexpected failures. Previous applications to 6-DoF business jet aircraft with altitude and attitude-tracking flight controllers only proposed uncoupled structures dependent on pre-tuned outer-loop PID controllers. Recent research demonstrated coupled-dynamics control, yet, was limited to inner-loop body rates control and would likely still depend on outer-loop PID controllers. It is foreseen to be difficult to implement IDHP for an altitude and attitude-tracking coupled-dynamics flight controller to a 6-DoF aircraft with a fully model-free structure.

The third option is the Soft Actor-Critic (SAC) algorithm, state-of-the-art Deep Reinforcement Learning (DRL) algorithm making use of recent advancements in deep neural networks. It benefits from a high generalization power and exploration that is beneficial to capture the dynamics of high-dimensional systems. Although SAC's sample efficiency is higher than most other DRL algorithms, all DRL algorithms still require relatively long training times. Fault tolerance may be achieved to a certain extent with robust control if the learned policy can be generalized to the failed system dynamics. Adaptive control through online learning may be implemented too, but it is expected to be difficult given DRL algorithms' need for many training samples and the relative instability of SAC's stochastic policy.

The solution is chosen as the one with the highest potential to capture the full inner and outer-loop dynamics of a coupled 6-DoF aircraft. In offline learning conditions, SAC is expected to perform better than ADHDP or IDHP due to the superior generalization power of Deep Neural Network (DNN)s. SAC is thus selected as the main agent for the coupled-dynamics Cessna Citation flight controller, thereby answering research question *Q1.2*. While fault tolerance is a key requirement for this application, the SAC agent can only provide such ability through its robustness. If needed, it can be enhanced by training offline on predictable failures. The extent to which the SAC agent is robust to failures is to be tested as part of next chapter's preliminary analysis.

One approach to make the proposed solution more fault-tolerant is to train online on the failed system. This ability will also be evaluated in the preliminary analysis. In case SAC is unable to learn online successfully and is not robust enough on unforeseen failures, an IDHP add-on agent is proposed for adaptive control. As soon as the fault is identified, the IDHP agent is expected to build an incremental model of the failed system fast-enough to regain control of the aircraft. This solution would be hybrid, such that it benefits from the advantages of different methods that complement each other.

The next chapter presents the preliminary analysis, where the SAC agent's robustness and ability to learn online is evaluated on a simple coupled-dynamics system.

4

Preliminary Analysis

In the previous chapter, a promising RL framework was identified for the coupled-dynamics Cessna Citation flight control task, Soft Actor-Critic (SAC). This helped answer research questions *Q1.1* and *Q1.2*. Now research questions *Q1.3* and *Q1.4* remain to be answered. In particular, it is to be investigated whether the offline-trained SAC agent is successful on a simple coupled-dynamics system control task. This is to gain confidence that it will also be successful on a more complex coupled-dynamics system such as flight control for the Cessna Citation. Moreover, its ability to deal with unforeseen failures on the system is to be evaluated.

As part of a preliminary analysis, a simple coupled-dynamics system is simulated. The environment is presented in section 4.1. The SAC agent implementation is detailed in section 4.2. Then, the simulation results are discussed in section 4.3. The conclusion of this analysis and answers to research questions *Q1.3* and *Q1.4* are given in section 4.4.

4.1. Simulation Environment

The environment consists of a mechanical system laid-out in subsection 4.1.1. The environment state and reward are detailed in subsection 4.1.2.

4.1.1. System Dynamics

The environment should correspond to a coupled-dynamics system while remaining simple to validate the chosen algorithm's learning abilities. A 2 Degrees-of-Freedom (DoF) cart-ball system is proposed, where the ball's and the cart's momentums influence that of the other through their spring-damper connection. The system and its reference points are presented in Figure 4.1.

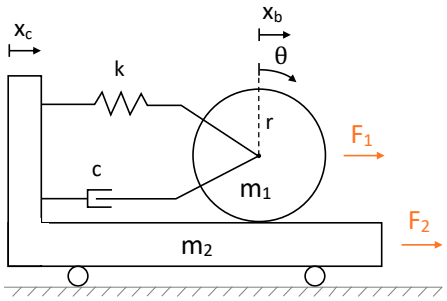


Figure 4.1: Cart-ball system. The spring and damper are at rest at shown reference positions.

Two continuous actions can be exerted on the ball and the cart, respectively. As the SAC agent squashes the policy output with a hyperbolic tangent function, it was found in [34] that clipping and normalizing the action by the maximum allowed force on the interval $[-1000\text{N}, 1000\text{N}]$ helped the learning

process. For simplicity, friction is neglected and the motion is constrained horizontally in a 2D plane. The cart is assumed to be infinitely long, while the left wall is a hard boundary that should not be hit.

The system is converted to a state-space system as shown in Equation 4.1. This state vector is referred to as the system state-vector. A simple forward Euler integration is used to solve the ordinary differential equation, and the hard boundary is implemented after the integration in a conditional statement. The outputs of interest are the cart horizontal position x_c and the ball angular rotation θ, calculated in Equation 4.2 from the system state vector.

$$\begin{bmatrix} \dot{x}_b \\ \dot{x}_c \\ \ddot{x}_b \\ \ddot{x}_c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -k/m_1 & k/m_1 & -c/m_1 & c/m_1 \\ k/m_2 & -k/m_2 & c/m_2 & -c/m_2 \end{bmatrix} \begin{bmatrix} x_b \\ x_c \\ \dot{x}_b \\ \dot{x}_c \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m_1 & 0 \\ 0 & 1/m_2 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{4.1}$$

$$\begin{bmatrix} x_c \\ \theta \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1/r & -1/r & 0 & 0 \end{bmatrix} \begin{bmatrix} x_b \\ x_c \\ \dot{x}_b \\ \dot{x}_c \end{bmatrix} \tag{4.2}$$

4.1.2. Environment State and Reward

Based on the system state vector, an environment state fed to the RL agent is determined. It is primordial to give enough relevant information to the agent to allow it to learn while not overflowing it with information to maximize sample efficiency.

The agent is to be trained on a control task with reference signals for the two system states x_c and θ. Because the motion is linear in the most part (constant spring constant and damping coefficient), the state errors e_{x_c} and e_θ are fed to the agent instead of the states and reference signals. Additionally, both state velocities $\dot{x}_c$ and $\dot{\theta}$ are provided to the agent to help improve the transient response. The state environment, therefore, consists of the error and the velocity vectors. The environment state is normalized by the absolute maximum of the reference signals for the errors and pre-determined maximum values for the velocities so that all signals have the same order of magnitude, facilitating the learning process.

The reward for this environment is only dependent on the environment state, namely the negative of the sum of the errors. As suggested in [34], training stability is increased by clipping the reward on the interval $[-1, 1]$ instead of using the quadratic error.

4.2. SAC Agent Implementation

A SAC agent is implemented to control this environment. The training task consists of step inputs on each state at different times on a system initially at rest. The episode length is $20s$ which corresponds to 2000 time-steps with a time resolution of $\Delta t = 0.01s$. For each episode, the seed is randomly selected. To evaluate the performance of the SAC agent, the response of a tuned PID controller is used as a benchmark. The performance metric used for comparison is the return, or sum of rewards over one episode. The choice of hyperparameters is explained in subsection 4.2.1 and the training process in subsection 4.2.2.

4.2.1. Hyperparameter Settings

The implementation follows algorithm 3.5, and adopts some default hyperparameters suggested in [23]. In particular, default values are used for the smoothing factor $\tau = 0.995$ and the depth of the critics and policy networks, with two hidden layers. The default Adam stochastic gradient-descent optimizer algorithm is employed, recognized for its superiority [57]. While the original paper suggests using a ReLu activation function, initial testing was done to compare it to a hyperbolic tangent activation. These tests reaffirmed the superiority of ReLu for a SAC agent in terms of final reward for a given number of training steps.

The learning rate is known to be a highly influential hyperparameter, and optimum values differ for different training tasks [72]. A high value might theoretically increase the convergence rate but also

increases the chance of overfitting. This occurs if the agent learns patterns in the observations that are too sample-specific and fails to see global trends. It is beneficial to have a high learning rate at the beginning of training to quickly build a rough policy and a lower one after many training samples to avoid observational overfitting. For this reason, it is common to decay the learning rate as a function of training steps [26]. In this implementation, it is brought linearly from an initial value at the beginning of training to zero at the last training step.

Initial trials with default values for other hyperparameters were successful but required a very large number of training steps, $6 \cdot 10^6$, to yield an equal or higher return than the PID controller. Hyperparameter optimization is employed to reduce the number of training steps for a given performance.

Hyperparameters to be optimized are chosen among the ones that have a determining factor in the learning process. SAC is known to be brittle wrt. the temperature parameter that regulates the exploration-exploitation trade-off. The temperature and the initial learning rate are thus selected. The network width (i.e. number of neurons per layer) is also tested, along with the buffer and minibatch size. Lastly, the discount factor is added as an optimization variable since it is a well known key hyperparameter in RL . To reduce the optimization space, most variables are restricted to a few discrete options, mostly centered around their default value or other suggested values in literature [1, 23, 24]. Only the learning rate is chosen on a continuous logarithmic scale.

455 trials are run, each trained for 10^5 steps. A tree-structured Parzen estimator approach is used for the hyperparameter search, which was proven more efficient than random search [7]. When unpromising runs are identified, they are aborted early to save time. The results are shown in Figure 4.2 under the form of a parallel coordinates plot.

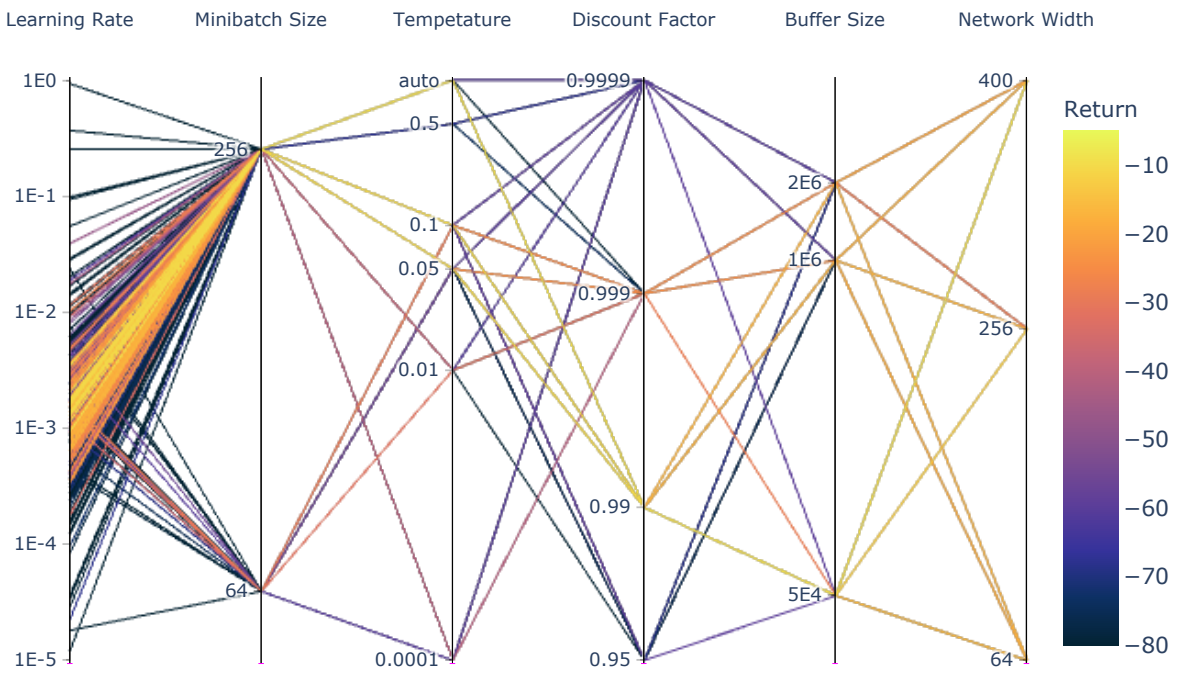


Figure 4.2: Hyperparameter optimization with 455 trials. Each line starting on the left represents one trial. Color scaling is averaged for lines representing multiple trials.

The results show the return for multiple combinations of hyperparameter settings. Some optimization space regions are more explored than others due to the selective search method. The highest return was $R = -14.55$ with a learning rate of $9.4 \cdot 10^{-4}$, minibatch size 256, temperature coefficient 0.1, buffer size $5 \cdot 10^4$, discount factor 0.99 and network width of 400 neurons. Due to the stochasticity in the learning process, this combination is not necessarily the best on average. Furthermore, the results show that more training is required as the best performance is not on par with the PID controller yet. This means that the optimal hyperparameter values may be different for a longer training.

Nonetheless, it is clear from Figure 4.2 that a learning rate close to 10^{-3} yields the highest return. Similarly, the best trials converge to a minibatch size of 256.

It is less clear, however, what setting the entropy regularization coefficient, or temperature, should be. Both the automatic setting from the updated version of SAC, 0.05 and 0.1 are well-performing settings for this parameter. Given that more training is required, the automatic setting is preferred for its robustness to various training lengths [24].

The discount factor confirms to be an influential parameter. A setting of $\gamma = 0.99$ consistently gives better returns.

The buffer size is not as determining as the discount factor since all of its options yield values within the top 20% trials. Despite that, $5 \cdot 10^4$ leads to slightly better results.

Lastly, the network width shows that a larger size generally leads to marginally better results. It should be considered, however, that training the 400 and 300 neurons-wide networks takes about three times and twice as long as the 64 neuron-wide version, respectively. From a training time performance perspective and given the proximity in the results, a small-width network is preferred.

This hyperparameter optimization analysis allowed to identify optimum settings, balancing tracking and training performance. Two settings differ from the best trial, namely the temperature parameter and the policy network size, to improve training time and robustness to further training. A summary of those settings, compared to default values from the original paper, is given in Table 4.1.

Table 4.1: SAC agent hyperparameter summary. Default values from [23].

Hyperparameter	**Default Value**	**Chosen Value**
Learning rate λ	$3 \cdot 10^{-4}$	Linearly decreasing from $9.4 \cdot 10^{-4}$ to 0
Minibatch size $\|\mathcal{B}\|$	256	256
Temperature coefficient η	0.2	automatic
Discount factor γ	0.99	0.99
Buffer size $\|\mathcal{D}\|$	10^6	$5 \cdot 10^4$
Network width	256	64
Network depth	2	2
Network activation	ReLu	ReLu
Smoothing factor τ	0.995	0.995
Optimizer	Adam	Adam

4.2.2. Training

Training is performed using the tuned hyperparameters for $5 \cdot 10^5$ training steps, taking 48min on an 8GB-RAM 2.6GHz-processor configuration. The evolution of the return with training steps can be seen in Figure 4.3. Despite the training process being unstable especially at the beginning of training, instabilities are reduced as learning progresses. Large dropouts are still observed but overfitting is avoided as the return keeps progressing to a maximum of $R = -6$ at the end of training. The relatively short amount of training steps highlight a fast convergence rate for DRL algorithms standards.

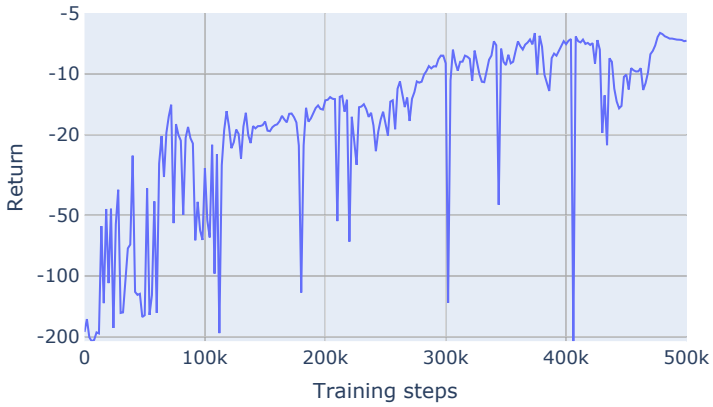


Figure 4.3: Training of SAC agent on cart-ball system with optimized hyperparameters.

4.3. Simulation Results

Several simulations are run to evaluate the robustness and the adaptiveness of the SAC agent. The evaluation task differs from the one it was trained on in terms of step height and time to reflect better unpredictable situations a system might encounter during operations. Simulations with the offline and online-trained agents are discussed in subsection 4.3.1 and subsection 4.3.2, respectively.

4.3.1. Offline-Trained Agent

The agent is first evaluated on the system with normal dynamics, which it was trained on offline. Three other simulations are proposed to evaluate its ability to cope with failures in the system dynamics.

Faults are categorized in [15] as either sensor, actuator, or component faults. Given that sensor faults relate more to the field of system identification, they are not explored in this research. Actuator faults and component faults, the latter including structural failures represented by changed aerodynamic coefficients and stiffness parameters, are investigated.

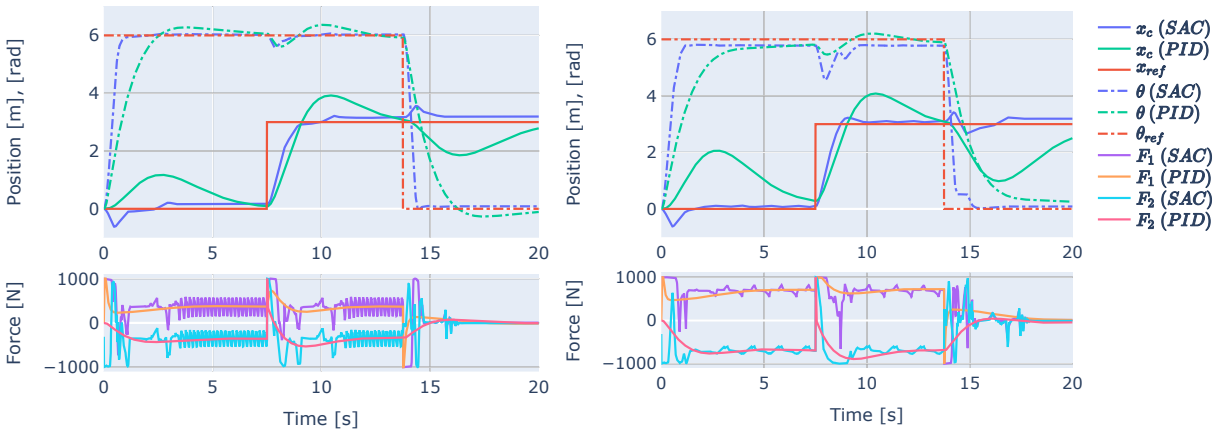
For comparison purposes, the SAC agent response is shown against a PID controller. The latter was tuned on the training task and not on the evaluation task for a fair comparison. Test results are presented in Figure 4.4.

Simulation 1: Normal Dynamics

The first simulation corresponds to the cart-ball system with normal dynamics. The SAC agent has to generalize its learning to an unseen task. The agent response and the applied force are shown in Figure 4.4a. For the ball position state (θ), corresponding to the element with smaller mass, the rise time is short, no overshoot is present, steady-state oscillations are completely damped and no steady-state error remains. The cart position state (x_c) has a longer rise time due to the higher cart mass but the same force saturation. While steady-state oscillations are also fully damped out, a small steady-state error is observed. This is most likely due to the agent having found a compromise between both errors and being stuck on this local optimum.

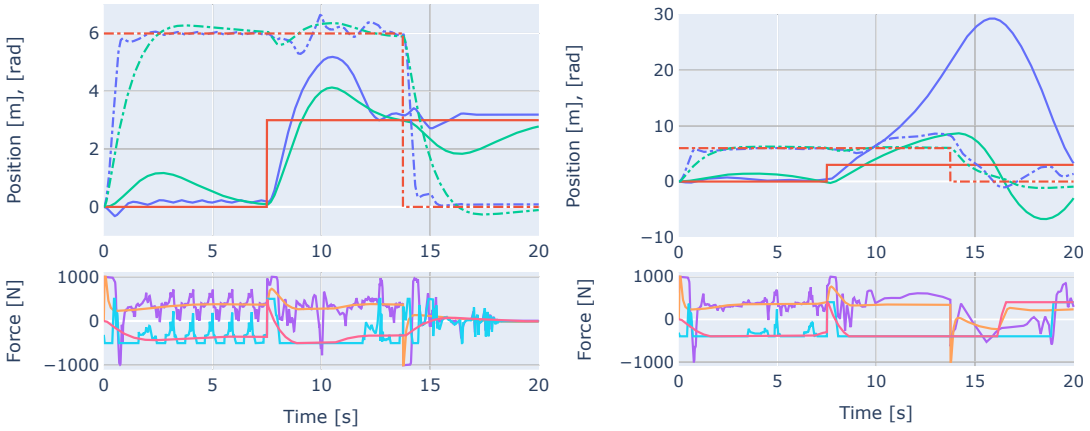
The coupling effects are clearly visible, particularly on the force plots where a step in one of the states results in a maximum force input on both elements. Despite small-amplitude force oscillations in the steady-state phase when θ is non-zero, the phase alignment and frequency is such that no negative effects are seen in the agent response. This is caused by the high-gain control strategy of the SAC agent, which allows it to have a fast transient response.

The PID controller, on the other hand, has a longer rise time, poorly-damped transient response but no steady-state error thanks to the integral term. The force inputs are more steady than that of the SAC agent and rarely reach saturation limits, which corresponds to a lower gain control strategy. Around $t = 2$s and $t = 16$s, the PID cart position response is seen to suffer from large coupling effects, as the PID controller does not have full system knowledge and is only reacting to errors in the state it controls. This highlights the SAC's agent ability to actively control the system and apply a preemptive



(a) Simulation 1: Normal system dynamics.

(b) Simulation 2: Double spring stiffness and damping coefficient.



(c) Simulation 3: Saturated F_2 to 50% of maximum value.

(d) Simulation 4: Saturated F_2 to 40% of maximum value.

Figure 4.4: SAC agent and PID controller response comparison on the evaluation task. Identical legend for all four plots.

action on a state to diminish disturbances from the other state, observed at $t = 1$s, $t = 14$s for x_c and at $t = 8$s for θ.

The SAC agent has an overall better response, reinforcing the choice of an RL controller for this coupled-dynamics system. Given the low tracking error of this response, which was obtained after only $5 \cdot 10^5$ training steps, SAC's sample efficiency is deemed excellent for this task.

Simulation 2: Stiffer and More Damped Dynamics

The second simulation tests the response to a component structural fault. The spring stiffness and the damping coefficient are doubled, while the evaluation task remains unchanged. The SAC agent response seen in Figure 4.4b is similar to the one with normal dynamics. Noticeable differences are small steady-state errors in both states when their reference is non-zero. The coupling effects at $t = 8$s and $t = 16$s are more present in the state response but quickly corrected.

The high-frequency oscillation in the force input is replaced by a low-frequency one, which creates very low-amplitude oscillations in the x_c state. Both forces have higher magnitudes in the steady-state phase than with normal dynamics to counteract the increased force in the cart-ball connection.

The PID controller response is more impacted by the changes in dynamics, as the transient response is less damped and coupling disturbances are more severe. Overall, the PID controller response degrades more than that of the SAC agent, while both show enough robustness to maintain good performance.

Simulation 3: Saturated Force (50%)

The third simulation evaluates the agent on an actuator failure. This type of failure is relevant for flight control when, for instance, a control surface is partially blocked. It is implemented on the cart-ball system by saturating the absolute force on the cart F_2 to 50% of its maximum value, at 500N. The SAC agent response displayed in Figure 4.4c shows more degradation than in simulation 2. As expected, the ball position state θ changes very little while the cart position x_c suffers from a high overshoot, as not enough force allows to damp it. The repercussion is higher coupling effects seen on the θ state at $t = 10$s. Still, both input forces are adapted as F_2 hits the saturation limit some of the time and the steady-state response remains satisfactory.

Contrastingly, the PID controller sees little impact from the force saturation. As its transient response was slower and less damped than the SAC agent with normal dynamics, it does not utilize such high force and almost never reaches saturation limits. As a result, the damping of the transient response for both states remains poor, yet the overshoot for x_c is two times smaller than for the SAC agent. The PID controller shows good performance as its control input is almost unchanged from Figure 4.4a. The SAC agent, on the other hand, can successfully generalize its control strategy to deal with this actuator failure.

Simulation 4: Saturated Force (40%)

The last simulation investigates an aggravated version of the actuator failure discussed in simulation 3. This time, the force on the cart F_2 is saturated to 40% of its maximum value. The SAC agent response is seen in Figure 4.4d for the ball position state θ, on which enough force is exerted to control it. Conversely, the cart position state x_c rises in an uncontrolled manner to unacceptably high values when it receives a step input, as the force F_2 on the cart is too low to stop it. Previous oscillations in F_2 seen in Figure 4.4c are replaced by a nearly constant input as the saturation limit became stricter. Although it seems to recover from $t = 16$s, this control strategy is deemed unstable.

The PID controller response shows more robustness than the SAC agent in the x_c state. Still, F_2 hits the saturation limit most of the time, and x_c reaches highly negative values that are not acceptable for this simple control task. Restricting the force on the cart from 50% to 40% shows that a breaking point in both the SAC agent and the PID controller has been reached.

Overall, this indicates that the SAC agent is robust to changes in the dynamics, including actuator failures, but only to a certain extent. On aggravated failures, the SAC agent can become unstable. One approach is to train offline the agent on predictable failures such as actuator faults so that it becomes more robust. One may also try to train the agent online on the failed system. Such ability is tested in the next subsection.

4.3.2. Online-Trained Agent

Online learning can potentially be used to achieve adaptive control, making the RL agent more fault-tolerant when robust control is not sufficient to keep the system stable. As the purpose of this research is to build an RL agent that can deal with faults in the system, the task of identifying the fault falls outside of the scope of this research. For this reason, it is assumed that a fault identification module is already implemented and gives a signal to the RL agent to switch to the adaptive control mode.

The online evaluation task simulates the cart-ball system with normal dynamics until $t = 5$s, after which the aggravated actuator failure occurs. From that time, F_2 is saturated to 40% of its maximum value, and the fault identification mechanism informs the agent to start online learning. Training is performed on the earlier offline-trained agent so that previous knowledge can be built on. The agent is given 15s of constant reference signals to have the opportunity to learn before receiving a negative step input in each signal.

Given the need for rapid improvements in the policy, the batch size is reduced from 256 to 64 samples so that gradient updates are more frequent. Since very few training steps are used, over-fitting is less problematic, and to improve the learning efficiency the learning rate is kept constant at 0.001. Lastly, a trick on the stochastic policy is used to deal with its inherent instability, given the off-policy characteristic of the SAC algorithm. The behavioral policy is made deterministic by taking the mean of

its probability distribution, which considerably stabilizes it at the expense of exploration. The response is shown in Figure 4.5.

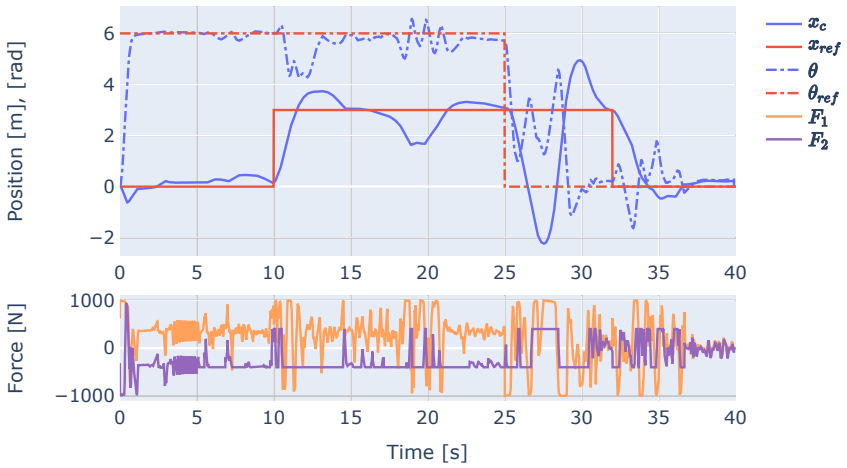


Figure 4.5: SAC agent response with saturated F_2 to 40% of maximum value from $t = 5$s with online learning.

This time, the SAC agent shows a stable response. The agent shows small instabilities in both states' responses due to the online learning from $t = 5$s, even as the reference signals are held constant. As F_2 reaches its saturation limit for most of the time, the other force F_1 receives a higher gain than in Figure 4.4a to stabilize the cart using coupling effects. The force exerted on the cart F_2 is not enough to resist the ball's momentum after $t = 25$s resulting in large instabilities in both states. Eventually, the agent converges to a close-to-zero steady-state error, showing that the agent successfully learned to generalize its previous knowledge and control the failed system.

This positive result, however, has to be put into perspective. Due to the inherent instability of off-policy learning and various random processes during training, success is not guaranteed at every trial. In fact, out of 50 trials of online learning, only 6% remained within safety bounds, defined as ± 3m and ± 3rad of the extremum of the reference signals. This underlines the difficulty the SAC agent has to learn on very few training steps, online. This is mainly attributable to the fact that DRL algorithms generally need a lot more training steps, as discussed in subsection 3.4.2. As the replay buffer reaches only 8% of its maximum capacity during this online training phase, its stabilizing effects are reduced. Furthermore, SAC's off-policy characteristic means that the behavioral policy during training differs from the one being optimized. This is another obstacle to good online performance, as seen in Figure 4.5 at $t = 5$s when the control input suddenly changes.

Overall, the SAC agent's online performance is poor given the extremely low success rate. Given that reliability of the algorithm is primordial to ensure safety at all times, online learning for SAC is not seen as a viable option to make it more fault-tolerant.

4.4. Conclusion

This chapter presented a preliminary analysis of the RL algorithm chosen in the literature study, Soft Actor-Critic (SAC). The environment on which it was tested was described and motivated in section 4.1. The SAC agent implementation was discussed in section 4.2, with a hyperparameter optimization conducted to improve sample efficiency.

The results of the simulations were analyzed in section 4.3. On normal system dynamics, which it was trained on, the SAC agent response showed excellent performance in the transient response and little error on the steady-state one. Sample efficiency was high as this response was achieved with only $5 \cdot 10^5$ training steps compared to other training setups discussed in section 3.4. Research question *Q1.3* is thereby answered, confirming the performance superiority in terms of tracking error and stability of the SAC algorithm.

The agent was robust both on component and actuator faults. Its generalization power allowed it to maintain a similar response with modified internal dynamics and a force saturation. On a more severe actuator fault, the agent's response became unstable. Online learning was tested as to whether it can help deal with an aggravated fault. With a success rate of 6% for maintaining stability, this option was discarded. This experiment provides insights to answer research question *Q1.4*. Ultimately, fault-tolerance was found to depend on the fault intensity, making it difficult to answer the research question with a definite answer. Another simpler option to dealing with aggravated failures is to make the SAC more robust by training offline the agent on some predictable failure.

With answers to research questions *Q1.3* and *Q1.4*, research question **Q1** is now answered. The remainder of this thesis work in Part I and Part III will focus on research questions **Q2** and **Q3**, where the SAC RL framework will be interfaced with a high-fidelity simulation model of the Cessna Citation 500 aircraft to test its fault tolerance.

III

Additional Results

5

Robustness Analysis

In the scientific article presented in Part I, robustness to various Initial Flight Conditions (IFC) and reference signal shapes was assessed as part of subsection 2.4.4. In this chapter, the response on each of IFC and reference signal shape is displayed in section 5.1 and section 5.2, respectively.

5.1. Initial Flight Conditions

The information presented in Table 2.3 is recalled in Table 5.1 to be shown against the aircraft responses depicted below. The performance metric used is the normalized Mean Absolute Error (nMAE) averaged over all externally tracked states (altitude, roll and sideslip angles) and normalized over the reference signals range. An acceptable range of $[-5°, 5°]$ is used for the zero-referenced sideslip angle. Four IFC are investigated, with IFC 1 corresponding to the nominal training conditions.

Table 5.1: Robustness analysis to varying IFC. The controller was trained on the nominal IFC.

Condition	Initial altitude [m]	Initial airspeed [ms^{-1}]	nMAE
IFC 1 (nominal)	2000	90	2.64%
IFC 2	2000	140	1.95%
IFC 3	5000	90	3.16%
IFC 4	5000	140	2.13%

5.1.1. IFC 1

As elaborated in subsection 2.4.1, the response on the nominal training conditions is stable and yields a low error on externally tracked states. A sudden jump in the control input at $t = 0$s is observed given that the trimming conditions are unknown.

The internally-tracked pitch attitude suffers from a significant error during the high-bank turns at $t = 40$s and $t = 70$s as the controller has no knowledge of the actual roll angle. This effect is mitigated as the outer-loop altitude controller increases its reference pitch angle to maintain the altitude error within 15m.

5.1.2. IFC 2

The robust response of the SAC controller on IFC 2, corresponding to an initial altitude of $h_0 = 2$km and initial airspeed of $V_0 = 140$ms^{-1}, is displayed in Figure 5.2. The controller was not trained on these conditions.

The 2.4-times higher dynamic pressure is mitigated by reducing the requested pitch angle to obtain a lower angle-of-attack, such that the altitude error remains low. The attitude controller, however, has difficulty reducing the pitch attitude, leading to a negative steady-state pitch error. It is especially difficult to stay level or descend as observed from $t = 85$s given the constant airspeed. The aileron and rudder

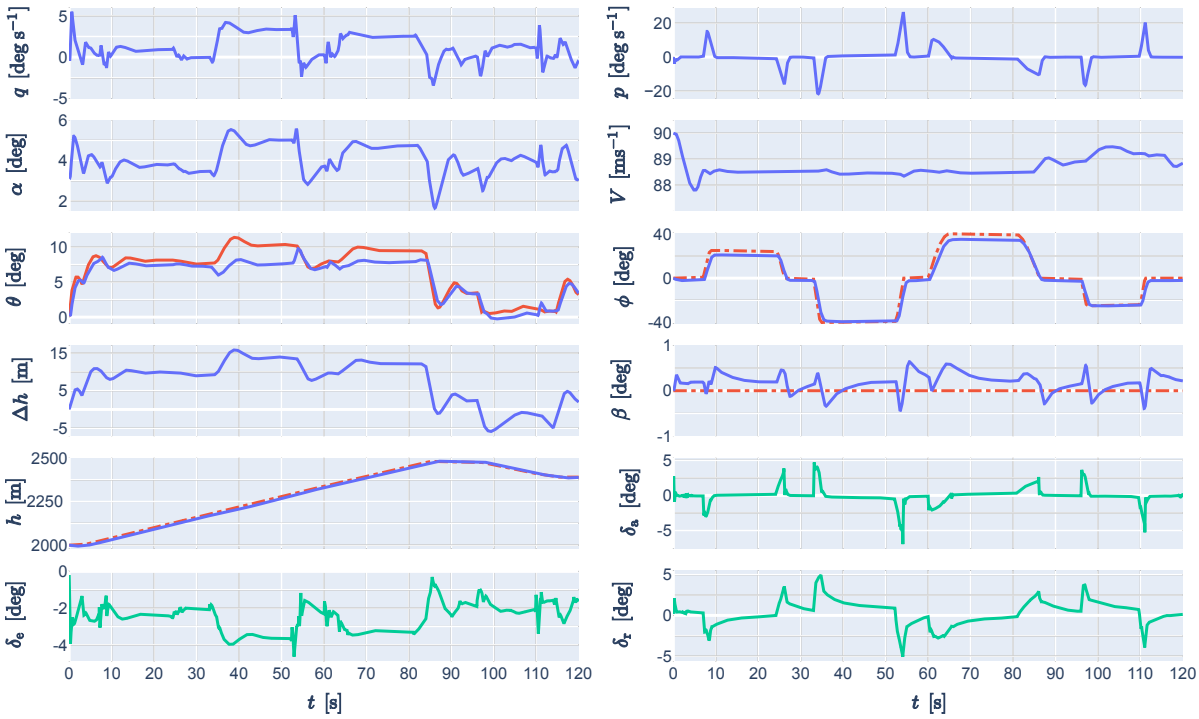


Figure 5.1: Altitude tracking response with $h_0 = 2$km and $V_0 = 90$ms^{-1} (IFC 1). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

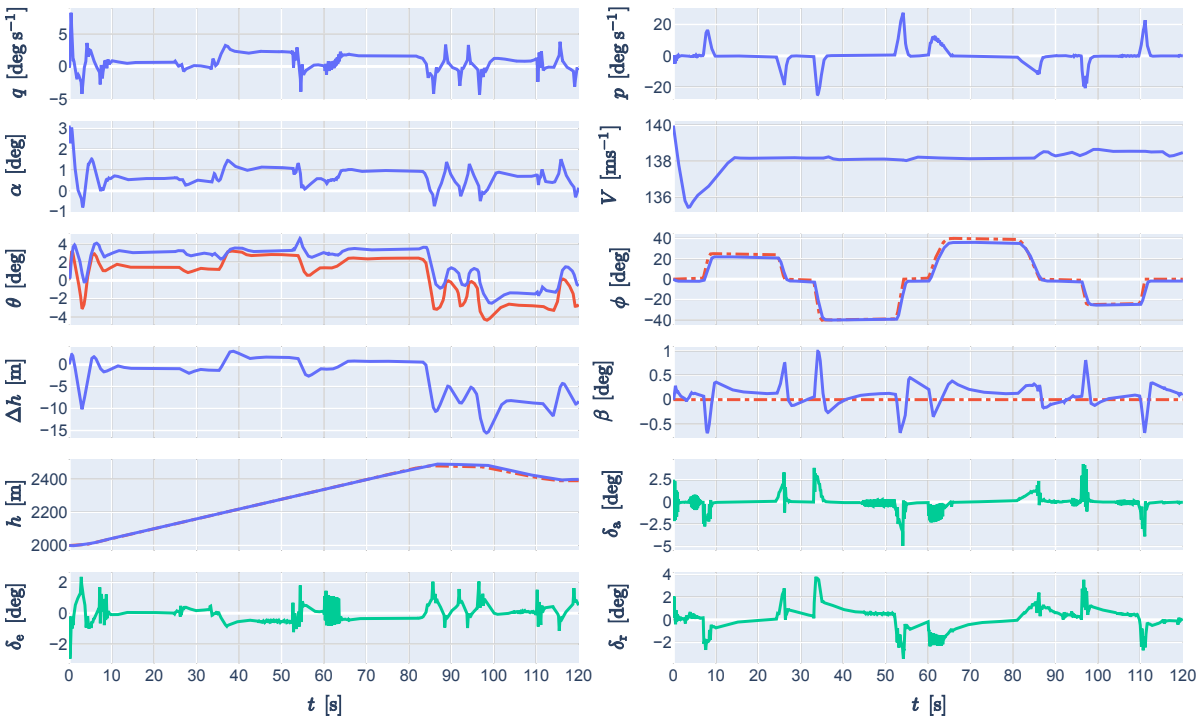


Figure 5.2: Altitude tracking response with $h_0 = 2$km and $V_0 = 140$ms^{-1} (IFC 2). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

commands are also slightly reduced given their higher effectiveness with the higher dynamic pressure. The response is overall stable and shows a low error on externally-tracked states despite the higher dynamic pressure than the nominal training conditions.

5.1.3. IFC 3

The controller's robust response on IFC 3, corresponding to an initial altitude of $h_0 = 5\text{km}$ and initial airspeed of $V_0 = 140\text{ms}^{-1}$, is shown in Figure 5.3. This IFC represents a 27% decrease in dynamic pressure compared to the nominal training conditions, which may bring the aircraft closer to the stall region. It also reduces control surface effectiveness and aerodynamic damping, possibly deteriorating the transient response.

Altitude tracking is more challenging on this task, with the error magnitude reaching values as large as 25m. The reduction in dynamic pressure is compensated with a higher-than-nominal pitch reference, although the attitude controller has a large pitch error as it seems to avoid deflecting the elevator upwards more than $-6°$, most likely to stay outside the stall region. The diminished rudder and aileron effectivenesses are counteracted by slightly higher-magnitude deflections such that lateral state tracking is not degraded. The overall response is stable and the transient response seems unaffected by the non-nominal IFC.

5.1.4. IFC 4

The last IFC corresponds to an initial altitude of $h_0 = 5\text{km}$ and initial airspeed of $V_0 = 140\text{ms}^{-1}$. The robust response on this non-nominal condition is visible in Figure 5.4. The increase in both initial altitude and airspeed leads to a 80%-higher dynamic pressure than nominal training conditions.

Similarly to IFC 2 the pitch reference is reduced and the altitude error is small, but the descent task is slightly more difficult to achieve.

On all four IFC, the response was stable and error low for all states, showing that the controller is robust to a large area of the altitude-airspeed envelope.

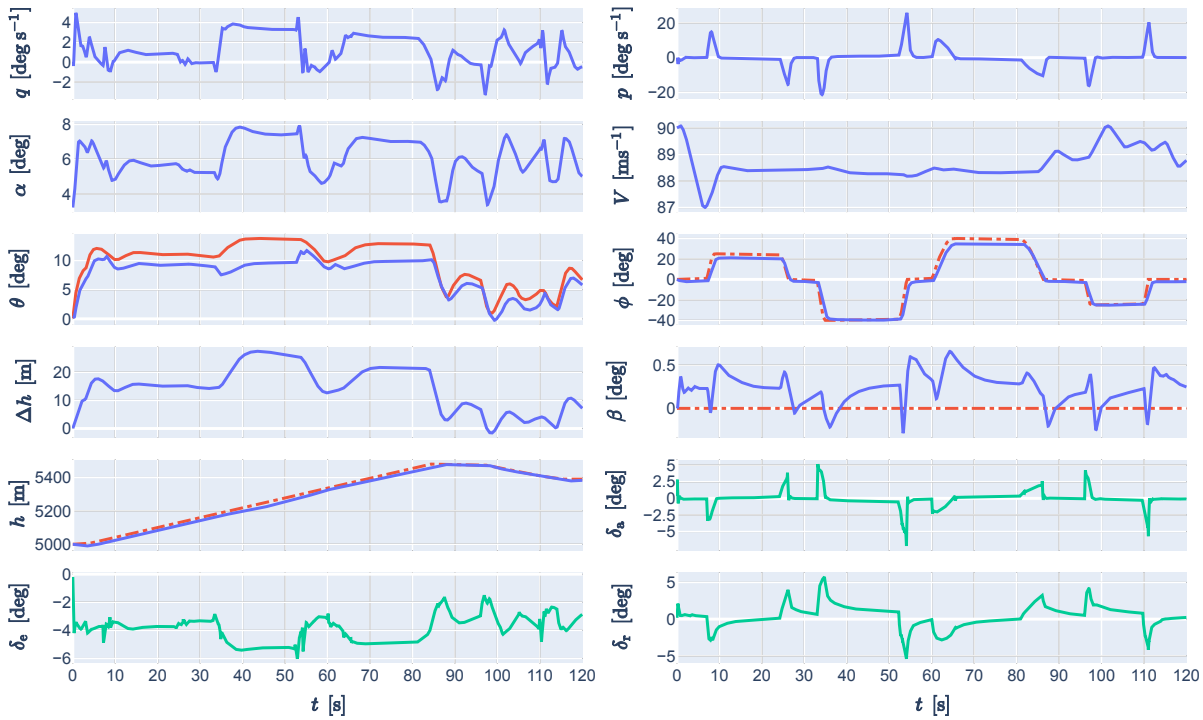


Figure 5.3: Altitude tracking response with $h_0 = 5\text{km}$ and $V_0 = 90\text{ms}^{-1}$ (IFC 3). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

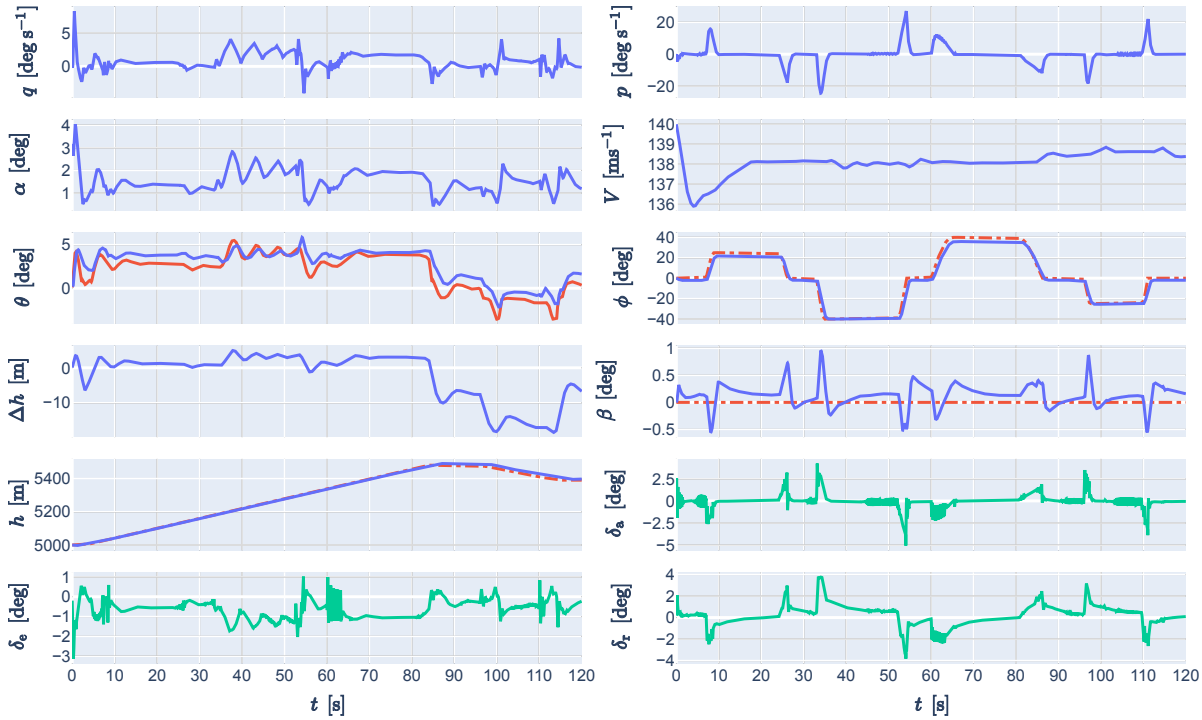


Figure 5.4: Altitude tracking response with $h_0 = 5$km and $V_0 = 140\text{ms}^{-1}$ (IFC 4). External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

5.2. Reference Signal Shapes

In Part I subsection 2.4.4, the robustness to various reference signal shapes was presented in terms of nMAE performance. In this section, the response to each signal shape is presented. Sinusoidal signals in subsection 5.2.1 and triangular signals in subsection 5.2.2 are used as altitude h^R and roll angle ϕ^R reference signals. The sideslip angle is asked to track a zero reference to ensure coordinated turns.

The information given in Table 2.3 is restated in Table 5.2 to be visualized against the aircraft responses shown below.

Table 5.2: Robustness analysis to varying reference signal shapes. The controller was trained on the nominal IFC.

Reference signals h^R, ϕ^R	**Period [s]**		**Amplitude [m], [deg]**		**nMAE**
	T_{h^R}	T_{ϕ^R}	A_{h^R}	A_{ϕ^R}	
Sinusoidal	80	50	80	50	3.22%
Sinusoidal	40	25	40	25	5.00%
Triangular	80	50	80	50	3.00%
Triangular	40	25	40	25	4.76%

5.2.1. Sinusoidal

The response to high and low-frequency sinusoidal signals is shown in Figure 5.6 and Figure 5.5, respectively. In both cases, the error is low and the aircraft states are stable.

Several instances of noisy control inputs are observed on both responses, which seem to occur as error vector components change sign. These strong non-linearities embedded in the policy network may be due to the controller not being used to smooth periodic reference signals. The latter make the error oscillate slowly between the positive and negative range, unlike the sudden changes introduced by step and ramp references seen in section 5.1. Reference signals during training should therefore be diversified, to include sinusoidal-like smooth signals.

The states, however, are little affected by this noisy control input as the actuator dynamics, modeled with a low-pass filter, remove most of the noise. Overall robustness to sinusoidal signals is thus demonstrated.

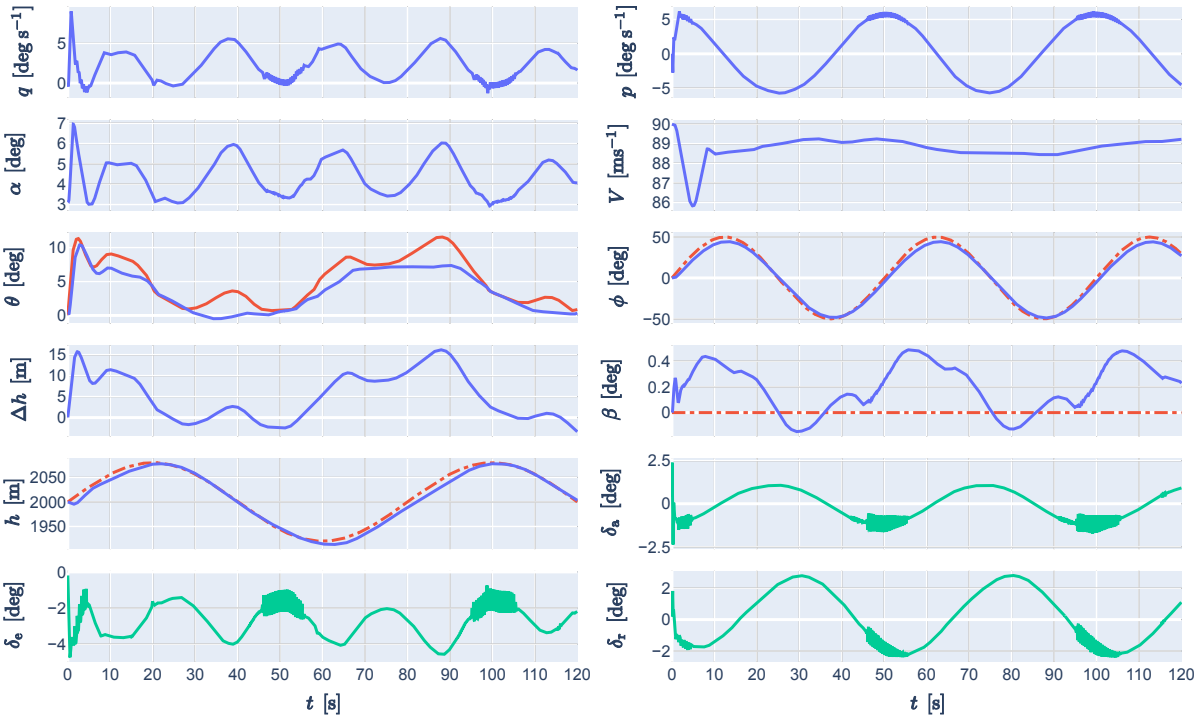


Figure 5.5: Altitude tracking response to low-frequency sinusoidal external reference signals. External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

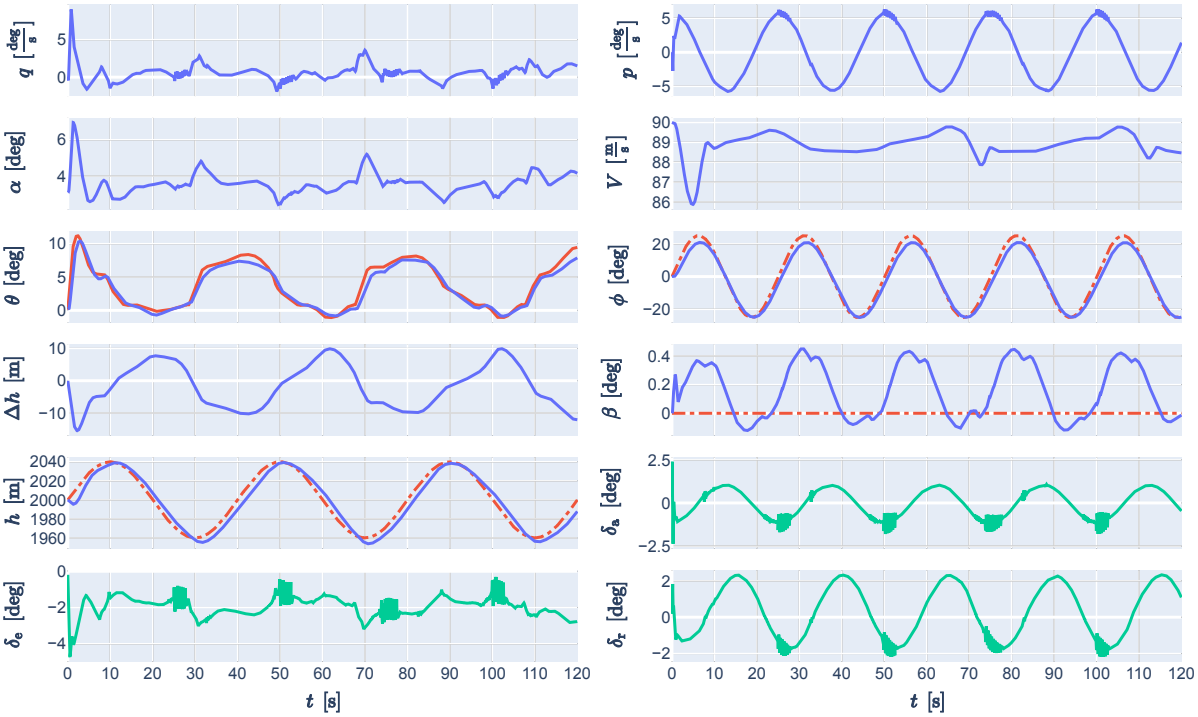


Figure 5.6: Altitude tracking response to high-frequency sinusoidal external reference signals. External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

5.2.2. Triangular

The response to triangular reference signals is displayed in this section for high and low-frequencies in Figure 5.8 and Figure 5.7, respectively. The error is again low and the response is stable.

The control inputs do not suffer from noisy values seen in subsection 5.2.1 as the triangular signals are closer to the reference signals the controller was trained on. One still observes small-amplitude noise for both cases as the error changes sign on the aileron command, although not degrading the response. Robustness is therefore also demonstrated on triangular reference signals of varying frequency and amplitude.

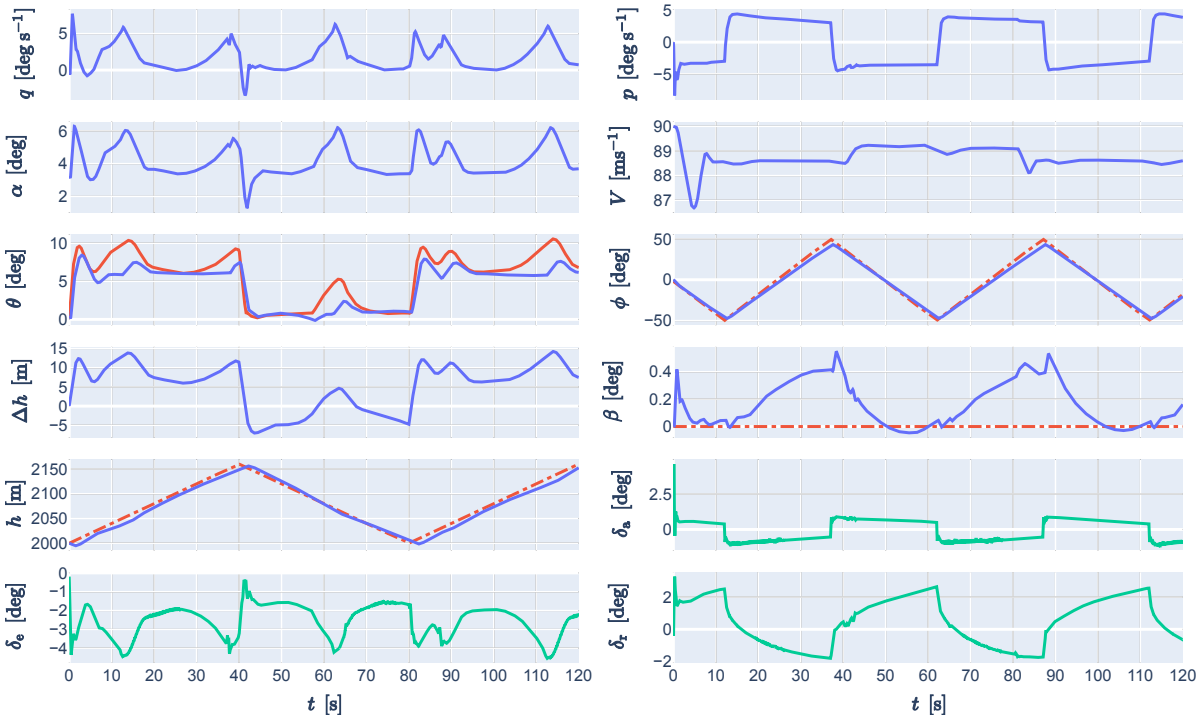


Figure 5.7: Altitude tracking response to low-frequency triangular external reference signals. External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

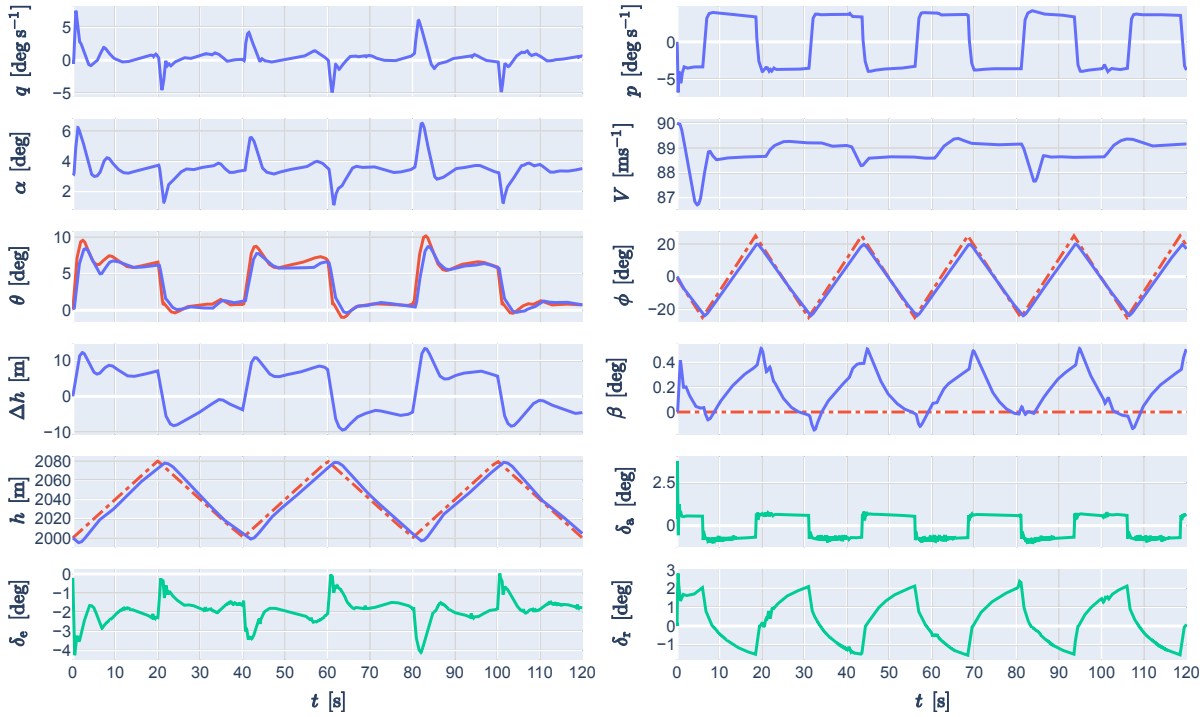


Figure 5.8: Altitude tracking response to high-frequency triangular external reference signals. External and self-generated reference signals are shown with red dashed and solid lines, respectively, and control inputs with green solid lines.

6

Control Disturbances

The paper presented in Part I investigated the controller's robustness to atmospheric disturbances in the form of vertical gusts. In this chapter, another look at disturbances is proposed in the form of control disturbances. While further away from real-world phenomena, they allow for a more direct evaluation of the controller's disturbance rejection abilities. The robust responses with and without sensor noise are studied in section 6.1 and section 6.2, respectively.

6.1. Disturbance Rejection with Ideal Sensors

The control disturbance is implemented as a 3211 doublet on the elevator and aileron input but remains hidden to the controller. The controllers were trained in steady flight conditions. For better visualization of the response, a zero-reference attitude tracking task is selected.

As seen in Figure 6.1, the control disturbance is immediately counteracted by a control input offset with a step of opposite sign and similar magnitude than that of the disturbance. A large disturbance at $t = 16$s creates some oscillations in the aileron command, although it is quickly damped. The disturbances create a larger error in the pitch than in the roll angle tracking, possibly due to the difficulty of taking both the elevator's non-zero trim position and disturbance into account. Disturbance rejection is nevertheless demonstrated as the controller generates a counteracting control input to minimize the error.

6.2. Disturbance Rejection with Biased Sensor Noise

The effect of biased sensor noise on disturbance rejection is evaluated in this section. Gaussian white noise with SSD and bias from the PH-LAB sensors reported in Table 2.2 is used as sensor noise, in conjunction with the same control disturbance introduced in section 6.1.

Aircraft states are unaffected by the noise as they are almost identical to Figure 6.1. Noise seems to degrade more the rudder and aileron commands than the elevator one, most likely because the roll and yaw-related noisy observations are more coupled than with the longitudinal states.

Given the aileron command oscillations at $t = 16$s, it is projected that a larger control disturbance would lead to an unstable response. However, with the present sensor characteristics, the addition of sensor noise did not lower the disturbance rejection ability of the controller.

This analysis concludes the robustness tests on the controller. The next chapter will investigate potential sample efficiency gains.

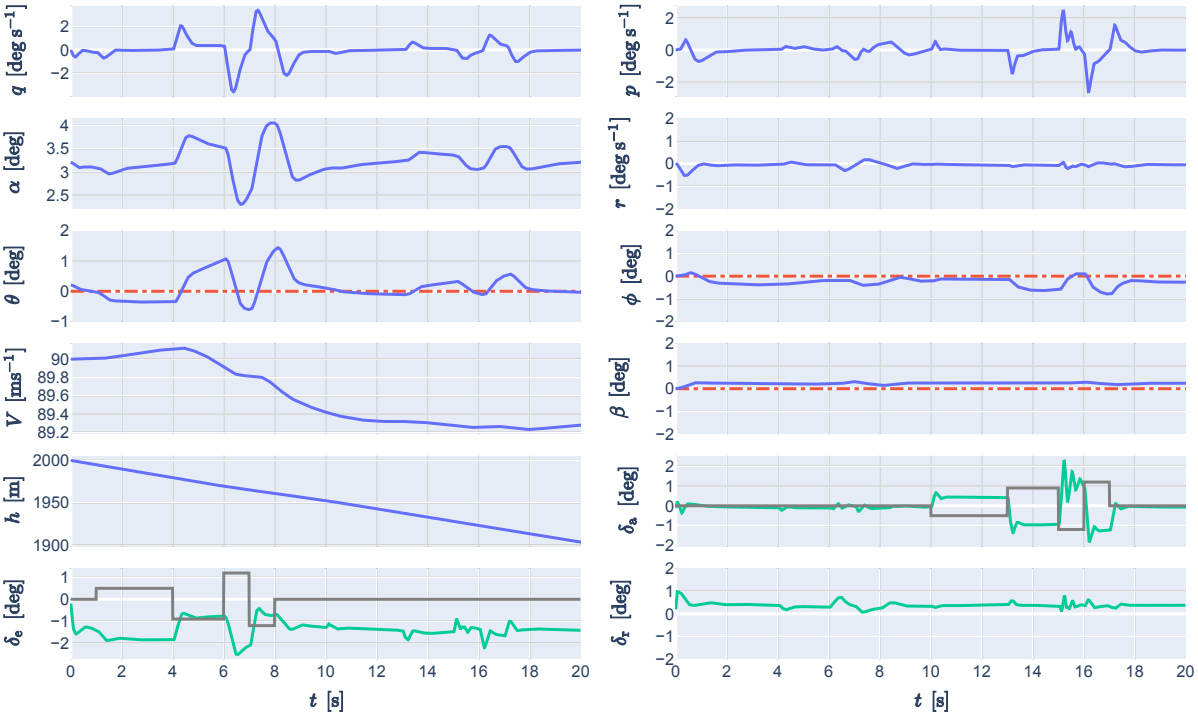


Figure 6.1: Attitude tracking response with ideal sensors and control disturbances (grey lines). Reference signals and control inputs are shown with red and green lines, respectively.

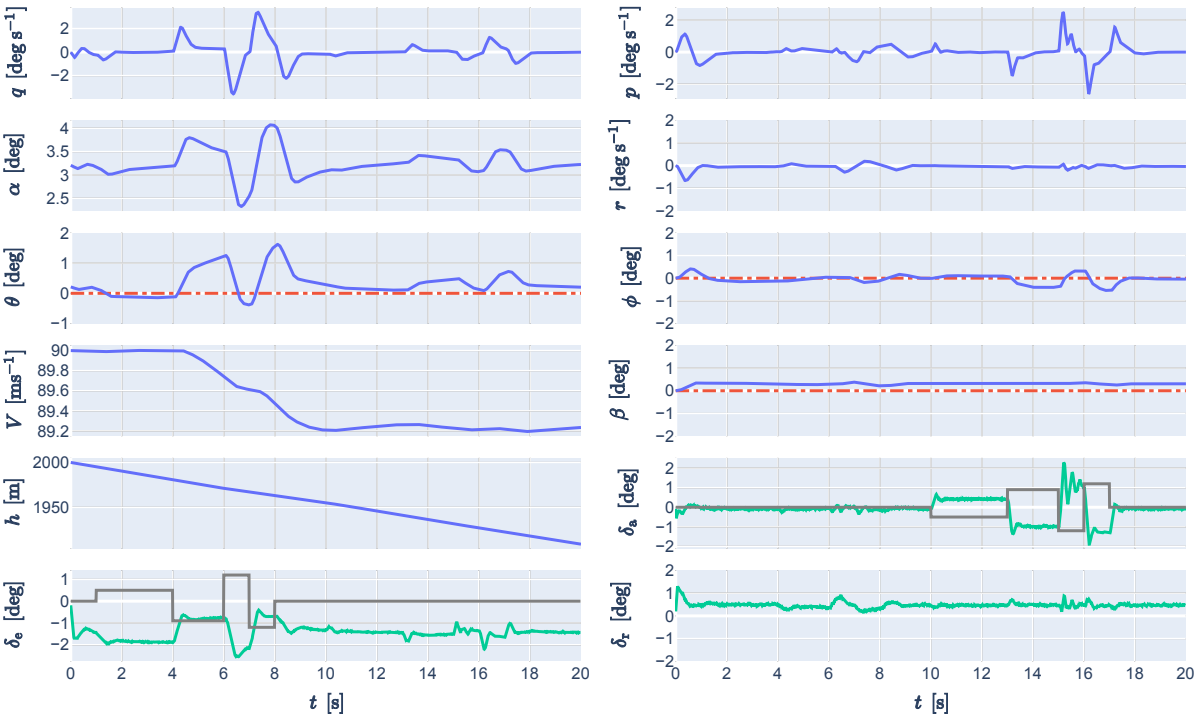


Figure 6.2: Attitude tracking with biased sensor noise and control disturbances (grey lines). Reference signals and control inputs are shown with red and green lines, respectively.

7

Reward Function Sensitivity Analysis

Additional tests on the controller can help increase its sample efficiency during offline training. For this purpose, several controller characteristics have been optimized through hyperparameter optimization and some through trial and error. Among them, the reward function has a large influence on the learning ability of the agent. In this chapter, a sensitivity analysis on different types of reward functions is proposed, with first an overview of function candidates in section 7.1 and the associated results in section 7.2.

7.1. Reward Function Types

RL applied to control tasks uses reward functions of various types. It is preferable to use continuous reward functions to facilitate the learning process, but a sparse reward may be the only option for certain environments with limited situational awareness. In this application, aircraft sensors and accelerometers allow for continuous tracking error estimation, which is the most relevant performance metric for low-level control. Before being passed to the reward function, the error vector **e** is scaled with the cost vector **c** as explained in subsection 2.3.2. Still, converting the weighted error vector to a scalar leaves room for choice.

Common approaches include using the negative of the Mean Squared Error (MSE) or the Mean Absolute Error (MAE) across all tracked states at a given time-step. Since these functions are unbounded, each element of the error vector can be clipped on $[-1, 0]$, as suggested in [45], to avoid unreasonably large gradient updates. This comes with the disadvantage of the agent not being able to differentiate between very negative and negative rewards outside the clipping range. Both clipped and unclipped MSE and MAE functions are tested as part of the sensitivity analysis. They are mathematically represented as the L-1 and squared L-2 norm of the weighted error vector in Table 7.1.

An inherently-clipped reward function can be an alternative to the problem described above. A rational function making use of the absolute value function and outputting values on the range $(0, 1]$ for any error vector is therefore tested as well. An updated cost vector $\mathbf{c}'$ is found by trial-and-error to fit this rational function.

7.2. Results

The results are presented after inner-loop and subsequent outer-loop training with varying reward functions. The controllers are trained on 10^6 steps in the same conditions and with same hyperparameters as described in subsection 2.3.2, and for each selecting the best of run out of 10 trials. The results are presented in Table 7.1. Each function is evaluated based on the episode nMAE averaged across tracked states.

Table 7.1: Reward function sensitivity analysis. $n_{\mathbf{e}}$ represents the number of tracked states.

Reward type	Function	Episode nMAE
Clipped MAE	$-\frac{1}{n_{\mathbf{e}}} \left\| \text{clip}\left[\mathbf{c} \odot \mathbf{e}, -\vec{\mathbf{1}}, \vec{\mathbf{0}}\right] \right\|_1$	2.64%
Unclipped MAE	$-\frac{1}{n_{\mathbf{e}}} \left\| \mathbf{c} \odot \mathbf{e} \right\|_1$	unstable
Clipped MSE	$-\frac{1}{n_{\mathbf{e}}} \left\| \text{clip}\left[\mathbf{c} \odot \mathbf{e}, -\vec{\mathbf{1}}, \vec{\mathbf{0}}\right] \right\|_2^2$	4.26%
Unclipped MSE	$-\frac{1}{n_{\mathbf{e}}} \left\| \mathbf{c} \odot \mathbf{e} \right\|_2^2$	unstable
Rational	$\frac{1}{n_{\mathbf{e}}} \left\| \frac{1}{\lvert \mathbf{c}' \odot \mathbf{e} \rvert + 1} \right\|_1$	5.27%

It is clear that clipping the reward function is beneficial as both unclipped MAE and MSE reward functions do not allow for a converged policy. This is most likely explained by the decreased stability of larger gradient updates from unclipped rewards, as suggested in [45]. This is particularly important for the SAC algorithm, which suffers from poor learning stability as discussed in subsection 2.4.4.

The clipped MAE reward function provides the highest sample efficiency, yielding a 38% and 50% lower error compared to the clipped MSE and rational reward functions, respectively, for the same number of training samples. A reward function giving proportionally the same importance to large and small errors within the bounds seems to be advantageous. The clipped MAE is the preferred reward function for this application.

It is worth noting that whether the reward is defined on the positive range $(0, 1]$, or on the negative range $[-1, 0]$, no significant effect is observed as the performance of the clipped MSE and the rational reward functions are similar.

8

Single Controller Structure

The controller tested so far had a cascaded structure. This chapter discusses an alternative controller structure. Instead of splitting altitude and attitude control, a single controller is proposed to track altitude, roll and sideslip angle references. Its structure is detailed in section 8.1 and the response is presented in section 8.2.

8.1. Structure Overview

The structure of the single controller is presented in Figure 8.1. The major differences with the cascaded structure, displayed in Figure 2.3, are the removal of the outer-loop controller and the addition of the altitude error feedback to the remaining single controller. Because altitude has slower dynamics than attitude angles, the pitch angle is fed to the SAC agent as an observation to help improve the transient response.

The single SAC controller is trained on the same task as the SAC altitude controller and with the same hyperparameters as the SAC attitude controller listed in Table 2.1.

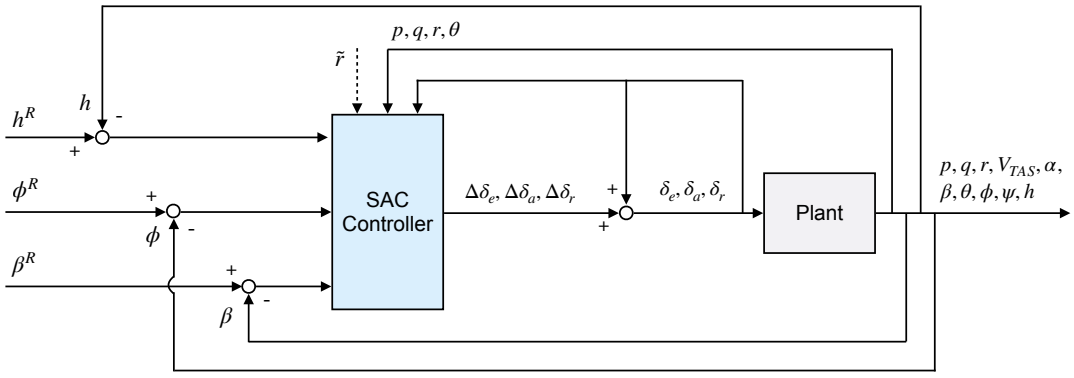


Figure 8.1: Single controller structure for altitude and attitude control. The SAC controller observes the weighted errors, untracked states, and current control input and also receives a reward from the environment.

8.2. Altitude Tracking Response

The altitude tracking response in Figure 8.2 has a relatively low nMAE of 3.86%, averaged over tracked states. Compared to the cascaded controller response, the single controller shows slightly larger altitude and sideslip errors and slower roll angle tracking. Longitudinal states suffer from low-frequency oscillations even during periods of constant references. Since these oscillations do not create a larger altitude error, the controller has no incentive to suppress them. It is therefore not a flaw of the controller itself but of the controller design instead.

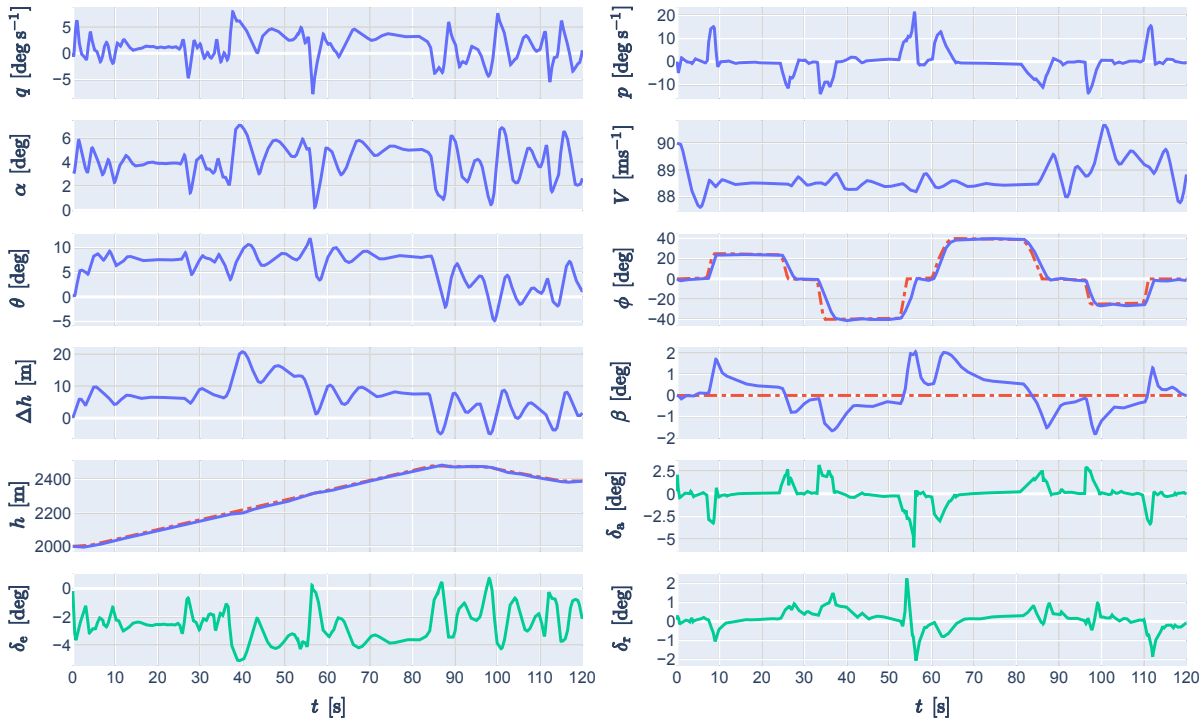


Figure 8.2: Altitude tracking response with single SAC controller. Reference signals and control inputs are shown with red and green lines, respectively.

Despite the small performance loss, having a single controller can be advantageous due to simplified training and implementation. Testing it on non-nominal initial conditions and on severe failure cases, however, the controller was not robust enough to maintain stability. For an initial altitude of 5000m and and speed of 90ms^{-1}, the response diverges from $t = 45$s. This low robustness is believed to be caused by the oscillations seen in the longitudinal states that are amplified in difficult flight conditions. The mismatch between the dynamics of altitude and of attitude angles, all tracked by the same controller, are also believed to also contribute to the controller's lower robustness. This single controller structure is, therefore, not preferred.

9

Verification & Validation

The simulation setup used throughout this research must undergo verification and validation steps to ensure the validity of results and understand their limitations. The verification and validation of both the high-fidelity simulation model and SAC controller are presented in section 9.1 and section 9.2, respectively.

9.1. Verification

Verification entails ensuring the simulation tools were implemented as intended.

9.1.1. High-Fidelity Simulation Model

The Cessna Citation 500 simulation model was built with the Delft University of Technology Aircraft Simulation Model and Analysis Tool (DASMAT) in Simulink [76]. To be compatible with Python, the model is exported to C code and compiled as a shared object executable. The correct compilation is checked by comparing the aircraft response from the shared object executable with the one from Simulink. The comparison is displayed in Figure 9.1 where the system is excited with two step commands. The two responses are almost indistinguishable, underlined by a mean-normalized RMSE of 0.67% averaged over all states.

The correct implementation of the simulation model can be checked by performing a sanity check using basic aircraft dynamics knowledge. In the standard body reference frame, a negative elevator deflection is expected to induce a positive pitch rate and result in a pitch-up maneuver. This effect can be seen at $t = 1$s. A positive aileron deflection creates, as expected, a negative roll rate and leads to a negative bank angle around $t = 4$s. A near-zero roll rate should translate to a constant roll angle, which is observed from $t = 8$s.

9.1.2. SAC Controller

The SAC RL algorithm was reproduced from [23]. It is verified by making sure a positive learning curve is present during agent training. The trend towards increasing returns throughout training seen in Figure 2.6 and Figure 4.3 indicates that the agent learned from interaction with the environment and confirms the correct algorithm implementation.

9.2. Validation

Validation is about investigating the representational power of the combined simulation model-controller environment.

9.2.1. High-Fidelity Simulation Model

The DASMAT simulation model of the Cessna Citation 500 was compared to flight test data collected on the Cessna Citation 550 PH-LAB research aircraft, an aircraft of similar size and engine power.

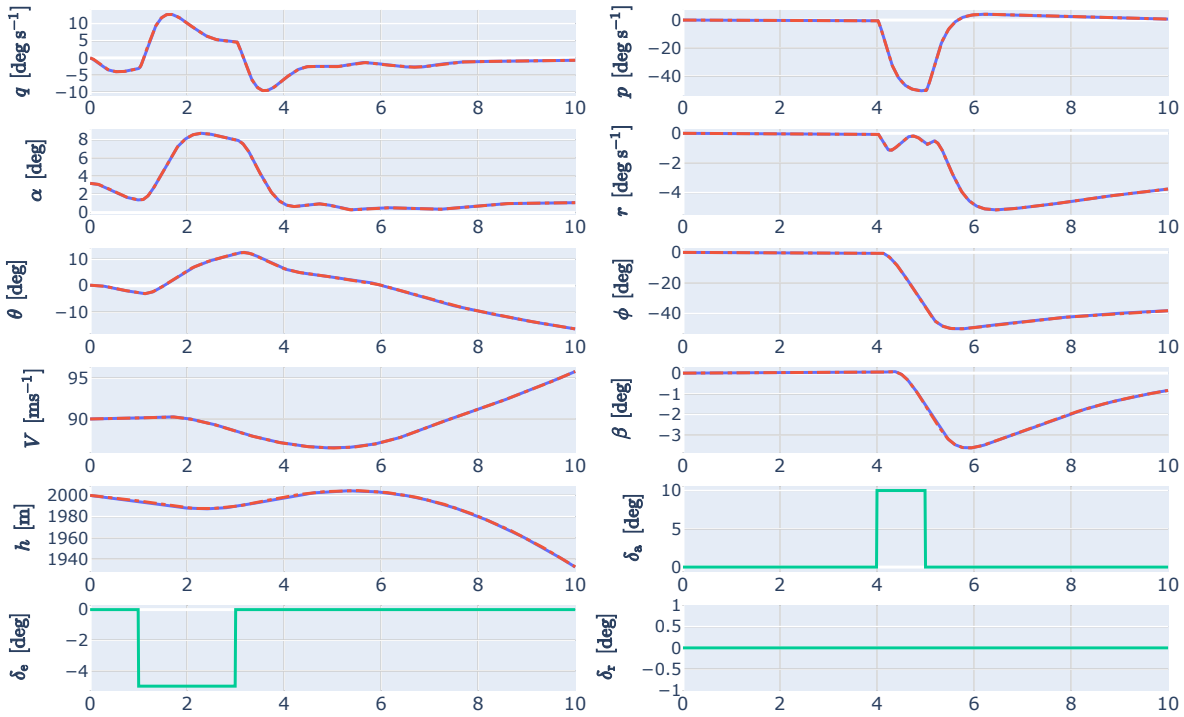


Figure 9.1: Step response comparison between the DASMAT Simulink model (dashed red lines) and the shared object executable (solid blue line). Control inputs (shared) with green solid lines.

In the pre-stall flight envelope, the relative RMSE was measured to be 8.38% and 12.65% for the longitudinal force and moment coefficients, and 7.34% and 8.58% for the lateral force and moment coefficients, respectively [75]. The relatively low RMSE means that the DASMAT simulation model is representative of the PH-LAB aircraft in the pre-stall flight envelope.

9.2.2. SAC Controller

The controller can be validated by assessing whether its performance meets the expected standards in various scenarios. In Table 2.3, an nMAE of 2.64% on the nominal training flight conditions and up to 3.16% on three other initial flight conditions were reported. It was also tested in conditions closer to real-world phenomena by adding biased sensor noise, based on the PH-LAB sensor characteristics estimated in [22]. Along with atmospheric disturbances specified the MIL-F-8785C specifications in [48], the response of the SAC controller remained stable and the tracking error small. The controller is therefore robust to various real-world phenomena flight conditions.

9.2.3. Limitations

Several limitations remain. Actuators are modeled with saturated first-order low-pass filters but no transport delay is taken into account. The environment is assumed fully observable and sensor delay is neglected. Continuous atmospheric disturbance models such as the Dryden model were not tested. The effect of simulation and controller frequency, both set at 100Hz, are not studied either. Although quantization effects were neglected, they are not expected to be an issue in future flight tests given the available computational resources.

Despite the demonstrated robustness on various flight conditions and tasks, further exploration of the standard flight envelope is required to satisfy safety standards.

IV

Closure

10

Conclusion

In-flight loss-of-control is the cause for the majority of civil aviation casualties. With the rapid development of personal flying vehicles, the need for enhanced flight controllers tolerant to various types of failures becomes apparent. Developing a model-based controller for each failure type is unrealistic, and online adaptive solutions lack the sample efficiency to control in the inner and outer-loop of coupled 6-DoF systems. This research contributes to model-free coupled-dynamics flight control with the aim of achieving fault tolerance to various unexpected failures.

The first research question, recalled below, is answered in Part II.

Research Question

Q1 Which RL framework can maximize the tracking performance in the control of a high-dimensional coupled-dynamics system in the presence of faults?

Q1.1 What are requirements on the RL algorithm in terms of its characteristics and training conditions?

Q1.2 Which RL algorithm satisfying the requirements has the highest potential to capture the full dynamics of a high-dimensional system?

Q1.3 How successful is the proposed algorithm applied to a simple coupled-dynamics system in terms of tracking error and stability?

Q1.4 To what degree is the proposed algorithm able to deal with unforeseen actuator and component faults on a simple coupled-dynamics system?

In Chapter 3, it is found that the reinforcement learning framework for this research should be model-free, fault-tolerant through either robust or adaptive control and enable continuous control. These characteristics answer research question *Q1.1*. After reviewing RL algorithms satisfying these requirements, the one with the highest expected generalization power on complex coupled-dynamics systems is identified to be Soft Actor-Critic (SAC). Although relatively sample efficient for a DRL algorithm, SAC's deep NNs require a large number of samples, compelling training to be conducted offline. This provides an answer to research question *Q1.2*.

The selected SAC algorithm is evaluated on a simple coupled-dynamics cart-ball system in Chapter 4. After a relatively short offline training, its response outperforms a tuned PID controller in terms of tracking error, making use of full-state knowledge and a high-gain control strategy. The transient response is fast and well-damped, while the steady-state one is stable, only showing a small error on one of the states. This analysis allows answering research question *Q1.3*.

The agent is robust both on component and actuator faults thanks to its high generalization power. On a more severe actuator fault, however, the agent's response became unstable. Online learning with SAC allows to regain stability with a success rate of only 6%, thereby disqualifying this option for fault-tolerant control. This experiment provides insights to answer research question *Q1.4*. Ultimately,

fault-tolerance is found to depend on the fault intensity, making it difficult to answer the research question with a definite answer. With these four sub-questions, research question **Q1** is now answered.

The next research question concerns the integration of the SAC controller with the simulation model of the aircraft, discussed in section 2.3.

Research Question

Q2 How can the proposed RL agent be integrated with the Cessna Citation 500?

Q2.1 At what control level should the RL controller be implemented, and what are the states and inputs to be controlled?

Q2.2 What simplifications to the simulation model can be made while keeping its relevant characteristics?

Q2.3 What type of task and maneuver is the system subject to experience?

With the goal of making flight control more fault-tolerant and reach full autonomy, the controller should allow for the highest-level control possible. Since trajectory planning for CS-25 class aircraft can be carried out according to methods reviewed in [13], a controller tracking altitude and bank angle signals that can be easily interfaced with these existing navigation algorithms is to be built. A single controller structure is explored in Chapter 8 but proved to have little robustness to failures. A cascaded structure is developed instead in section 2.3, with the purpose of separating altitude and attitude control, which have largely different dynamics. The attitude controller did not command the control surface deflections themselves but their time increment instead, to stabilize the control input. Besides the weighted errors, the cascaded controller also receives feedback signals from the current control surface deflections and the body rates to improve the transient response. This choice for controller design answers research question *Q2.1*.

The simulations are restricted to the in-cruise clean configuration with limited atmospheric disturbances considering that this research is a proof-of-concept for a SAC controller, thereby answering research question *Q2.2*. Lastly, the system is asked to perform representative strongly-coupled maneuvers that may be required in emergency situations, including climbing and descending turns and high-bank turns. With these characteristics for the controller, research question **Q2** is answered.

The last research question is about the performance of the built SAC controller, which is evaluated in section 2.4 and Part III.

Research Question

Q3 What is the overall performance of the RL controller on a Cessna Citation 500 model?

Q3.1 How does it perform on coupled maneuvers in terms of tracking error and stability?

Q3.2 How does the controller adapt itself to unforeseen faults to the aircraft?

Q3.3 What is the sample efficiency of the control system?

The first evaluation setting with the non-failed system is on attitude and altitude tracking tasks involving large coupling effects. The response is stable and keeps the tracking error low to achieve coordinated climbing $40°$-bank turns successfully. On a $70°$-bank flat turn, steady-state errors in the pitch and roll angles appear as the controller finds a compromise between the two difficultly-tracked signals. Testing the controller on initial flight speed and altitudes it has not seen during training, the aircraft remains stable and performance is similar. Moreover, the controller can track various reference signal shapes of varying frequency and amplitude, although sinusoidal reference signals are seen to generate a noisy control input. This does not degrade the step response but suggests that the training task should be diversified to include smoother reference signals. Lastly, disturbance rejection to both atmospheric and control disturbances is quick and effective, even in the presence of biased sensor noise. These tests demonstrate the overall high robustness of the SAC controller in normal conditions. These sets of observations provide an answer to research question *Q3.1*.

The robust response to the failed-system is evaluated on six different faults. A severe actuator

failure involving the rudder jammed at -15° is handled well, as a climbing turn is successfully executed despite large roll and sideslip angle errors. On other actuator failures, one implicating a reduced aileron effectiveness and one a reduced elevator range, the controller adapts its control input scaling and offset to track the climbing turn task with almost no performance degradation. A structural failure involving the partial loss of horizontal tail, a sudden backward c.g. shift and icing are all unexpected changes in aircraft dynamics that the robust controller can also adapt to. This is possible as the attitude controller up to quadruples its elevator deflection and the altitude controller increases its reference to cope with the higher pitch attitude error. Despite the controller not having experienced any fault during training, this high robustness to failures is explained by SAC's built-in exploration feature, i.e. its stochastic policy, and large generalization power. Research question *Q3.2* is now answered.

The failed response is also evaluated with an adaptive controller, which is trained on the failed system. While this experiment falls outside the scope of research question *Q3.2* since simulation models of failed dynamics are typically not obtainable, it provides insights into the possible improvement in the response. While the adaptive SAC controller is able to track better the roll angle during the rudder failure, the adaptive response provides equal or worse performance than the robust response on all other five failure cases. This is mostly explained by the increased difficulty of training on the failed system, given the already low training reliability of SAC on the normal system.

The SAC controller's sample efficiency is relatively low given its two hidden layers and large amount of parameters, which means that it has to be trained offline. Still, it performs well in terms of sample efficiency compared to other DRL algorithms reviewed in section 3.4 when taking the tracking error and the convergence rate into account. After 10^6 training steps for each controller, or $2 \cdot 10^6$ in total, a normalized Mean Absolute Error (nMAE) of 2.64% is achieved on the altitude tracking task. This result, however, has to be put into perspective given that the training performance is not consistent and a nMAE of 5% or lower is achieved on only 26% of 27 training runs. DRL's multiple random processes from network parameter initialization to minibatch sampling, and SAC's stochastic policy in particular, are all believed to contribute to this low training reliability. When taking this effect into account, the sample efficiency advantage with respect to other DRL algorithms is contrasted. While this is not safety-critical in an offline learning environment, it makes the controller development process more difficult.

The sample efficiency is thought to vary based on various RL hyperparameters, the reward function being one of the most influential ones. A sensitivity analysis on the reward function in Chapter 7 finds that the sample efficiency of a clipped MAE function is at least 1.6 times higher than that of clipped Mean Squared Error, rational and unclipped functions. With these last indicators on the training performance, research question *Q3.3* is answered, which also concludes research question **Q3**.

The research objective of this thesis is reviewed.

Research Objective

To advance self-learning techniques for model-free coupled-dynamics flight control systems applied to fixed-wing aircraft with the purpose of enabling fault tolerance to unexpected failures, by means of developing a reinforcement learning flight controller for a business jet aircraft.

This research presents the development and the testing of a model-free coupled-dynamics flight controller that can withstand multiple types of unexpected failures for a high-fidelity simulation model of the Cessna Citation 500. The controller is able to perform highly-coupled maneuvers including coordinated climbing turns and high-bank flat turns, despite biased sensor noise, atmospheric disturbances or varying initial flight conditions. Its robustness is demonstrated on severe actuator failures, structural failures and other unexpected events such as icing and c.g. shift as the controller maintains stability and completes the required maneuver. This research proposes the use of a novel RL framework, SAC, to overcome the difficulty model-based methods have to adapt to unexpected failures and the limited system complexity online learning methods can handle.

The results presented in this research pave the way for future applications of SAC and other DRL algorithms to improve fault tolerance in flight control systems. This will not only increase safety standards in civil aviation but also help achieve safe fully-autonomous flight. Ultimately, this research can help propel the current developments in the urban air mobility market that will require enhanced control systems that can withstand failures in dense traffic.

11

Recommendations

The results presented as part of this research allow to gain the following insights for further research.

- The largest area of improvement for the SAC controller presented in this research is the training reliability, directly caused by the low stability of SAC's training process. Less sample efficient but more stable state-of-the-art deterministic-policy DRL algorithms such as TD3 should be investigated. On-policy DRL algorithm PPO is also a good candidate for increased training stability. These alternative algorithms are expected, however, to suffer from a lower robustness due to reduced exploration but open the door to safe online learning. In turn, this may allow to make training fully online and thereby eliminate any model dependence.
- The hyperparameters of the SAC controller should be optimized with the goal of making training more stable. For instance, this could include increasing the smoothing factor and the minibatch size, and changing the entropy target throughout training. For the latter option, a novel version named meta-SAC , developed in [79], proposes to tune the entropy target hyperparameter automatically with a meta-gradient method.
- The limitations in the flight conditions of the SAC controller should be explored. For this purpose, the simulation of the entire standard flight envelope of the Cessna Citation 500, in clean and landing configurations, and the addition of continuous atmospheric disturbances is suggested. Possible instabilities caused by reference signal shapes should also be investigated, as noisy control inputs were discovered on specific reference signal shapes.
- DRL algorithms suffer from lower sample efficiency than online-learning iADP methods. The development of a hybrid offline learning-DRL and online learning-IDHP structure can be investigated such that, together, they can enable online adaptive outer and inner-loop control for complex coupled 6-DoF systems. Both having an actor-critic structure, it is projected that the sharing of information from the critic can make this solution attractive.
- In this research, several assumptions were made to simplify the simulation. To reduce the discrepancy with real-world phenomena and prepare for flight tests on the PH-LAB, sensor and transport delays should be modeled. The run time performance should also be investigated to ensure that enough computational power is available in-flight, especially given the use of deep NNs.
- In the prospect of enabling autonomous control for fixed-wing aircraft, the SAC flight controller should be interfaced with a navigation algorithm. Even though in this research the airspeed is tracked by the aircraft's internal auto-throttle to ensure compatibility with the PH-LAB on future flight tests, a SAC controller should eventually be able to command the throttle setting to track the airspeed.

Bibliography

[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning, Benchmarks, 2018. [Online; accessed 15-October-2020].

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Robert Babuška and Jens Kober. Knowledge-Based Control Systems - Lecture Notes, 2019.

[5] Gabriel Moraes Barros and Esther Luna Colombini. Using soft actor-critic for low-level UAV control. *arXiv preprint arXiv:2010.02293*, 2020.

[6] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[7] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[8] Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533. IEEE, 2019.

[9] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[10] YM Chen and ML Lee. Neural networks-based scheme for system failure detection and diagnosis. *Mathematics and Computers in Simulation*, 58(2):101–109, 2002.

[11] Yan Cheng and Yong Song. Autonomous decision-making generation of UAV based on soft actor-critic algorithm. In *2020 39th Chinese Control Conference (CCC)*, pages 7350–7355. IEEE, 2020.

[12] Stephen Dankwa and Wenfeng Zheng. Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, pages 1–5, 2019.

[13] Daniel Delahaye, Stéphane Puechmorel, Panagiotis Tsiotras, and Eric Féron. Mathematical models for aircraft trajectory design: A survey. In *Air Traffic Management and Systems*, pages 205–247. Springer, 2014.

[14] Pedro Miguel Dias, Ye Zhou, and Erik-Jan van Kampen. Intelligent nonlinear adaptive flight control using incremental approximate dynamic programming. In *AIAA Scitech 2019 Forum*, page 2339, 2019.

[15] Christopher Edwards, Thomas Lombaerts, Hafid Smaili, et al. Fault tolerant flight control. *Lecture Notes in Control and Information Sciences*, 399:1–560, 2010.

[16] Russell Enns and Jennie Si. Helicopter trimming and tracking control using direct neural dynamic programming. *IEEE Transactions on Neural networks*, 14(4):929–939, 2003.

[17] Silvia Ferrari and Robert F Stengel. An adaptive critic global controller. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 4, pages 2665–2670. IEEE, 2002.

[18] Silvia Ferrari and Robert F Stengel. Online adaptive critic flight control. *Journal of Guidance, Control, and Dynamics*, 27(5):777–786, 2004.

[19] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, volume 80, pages 1587–1596, 2018.

[20] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[21] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

[22] Fabian Grondman, Gertjan Looye, Richard O Kuchar, Q Ping Chu, and Erik-Jan van Kampen. Design and flight testing of incremental nonlinear dynamic inversion-based control laws for a passenger aircraft. In *2018 AIAA Guidance, Navigation, and Control Conference*, page 0385, 2018.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1856–1865, 2018.

[24] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[25] Ola Härkegård. *Backstepping and control allocation with applications to flight control*. PhD thesis, Linköpings universitet, 2003.

[26] Mohamad H Hassoun et al. *Fundamentals of artificial neural networks*. MIT press, 1995.

[27] Stefan Heyer, Dave Kroezen, and Erik-Jan van Kampen. Online adaptive incremental reinforcement learning flight control for a cs-25 class aircraft. In *AIAA Scitech 2020 Forum*, page 1844, 2020.

[28] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[29] International Air Transport Association (IATA). Loss of Control In-Flight Accident Analysis Report, 2015.

[30] International Civil Aviation Organization (ICAO). Accident Statistics, 2019. [Online; accessed 15-June-2020].

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[32] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *Mathematics of Deep Learning, Cambridge University Press*, 2018.

[33] Twan Keijzer, Gertjan Looye, Q Ping Chu, and Erik-Jan van Kampen. Design and flight testing of incremental backstepping based control laws with angular accelerometer feedback. In *AIAA Scitech 2019 Forum*, page 0129, 2019.

[34] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for UAV attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21, 2019.

[35] Ramesh Konatala, Erik-Jan Van Kampen, and Gertjan Looye. Reinforcement learning based online adaptive flight control for the cessna citation ii (ph-lab) aircraft. In *AIAA Scitech 2021 Forum*, page 0883, 2021.

[36] Dave Kroezen. Online reinforcement learning for flight control. MSc thesis, Delft University of Technology, 2019.

[37] Jun Lee and Erik-Jan van Kampen. Online reinforcement learning for fixed-wing aircraft longitudinal control. In *AIAA Scitech 2021 Forum*, page 0392, 2021.

[38] Hangxu Li, Liguo Sun, Wenqian Tan, Baoxu Jia, and Xiaoyu Liu. Switching flight control for incremental model-based dual heuristic dynamic programming. *Journal of Guidance, Control, and Dynamics*, pages 1–7, 2020.

[39] Teng Li, Zhiyuan Xu, Jian Tang, and Yanzhi Wang. Model-free control for distributed stream data processing using deep reinforcement learning. *Proceedings of the VLDB Endowment VLDB Endowment Hompage archive*, 11(6):705–718, 2018.

[40] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, page 40, 2016.

[41] TJJ Lombaerts, GHN Looye, QP Chu, and Jan Albert Mulder. Design and simulation of fault tolerant flight control based on a physical approach. *Aerospace Science and Technology*, 23(1): 151–171, 2012.

[42] Guilherme Cano Lopes, Murillo Ferreira, Alexandre da Silva Simões, and Esther Luna Colombini. Intelligent control of a quadrotor with proximal policy optimization reinforcement learning. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, pages 503–508. IEEE, 2018.

[43] Peng Lu, Laurens Van Eykeren, Erik-Jan van Kampen, Coen de Visser, and Qiping Chu. Double-model adaptive fault detection and diagnosis applied to real flight data. *Control Engineering Practice*, 36:39–57, 2015.

[44] Frank T Lynch and Abdollah Khodadoust. Effects of ice accretions on aircraft aerodynamics. *Progress in Aerospace Sciences*, 37(8):669–767, 2001.

[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Neural Information Processing Systems Deep Learning Workshop*, 2013.

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[47] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[48] David J Moorhouse and Robert J Woodcock. Background information and user guide for mil-f-8785c, military specification-flying qualities of piloted airplanes. Technical report, Air Force Wright Aeronautical Labs Wright-Patterson AFB OH, 1982.

[49] Zhen Ni, Haibo He, Xiangnan Zhong, and Danil V Prokhorov. Model-free dual heuristic dynamic programming. *IEEE transactions on neural networks and learning systems*, 26(8):1834–1839, 2015.

[50] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

[51] Orbis Market Reports. Global Flying Cars Market Size, Status And Forecast 2019-2025, 2019.

[52] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[53] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[54] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[55] Danil V Prokhorov and Donald C Wunsch. Adaptive critic designs. *IEEE transactions on Neural Networks*, 8(5):997–1007, 1997.

[56] César Castro Rendón, Cristina Verde, and Alejandro Mora Hernández. Actuator fault tolerant pid controllers. *IFAC Proceedings Volumes*, 45(3):721–726, 2012.

[57] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[58] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR 2016 : International Conference on Learning Representations 2016*, 2016.

[59] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[61] Mohit Sewak. *Deep Reinforcement Learning*. Springer, 2019.

[62] Jennie Si and Yu-Tsung Wang. Online learning control by association and reinforcement. *IEEE Transactions on Neural networks*, 12(2):264–276, 2001.

[63] Jennie Si, Andrew G Barto, Warren B Powell, and Don Wunsch. *Handbook of learning and approximate dynamic programming*, volume 2. John Wiley & Sons, 2004.

[64] Olivier Sigaud. Reinforcement Learning, Off-policy versus on-policy RL, Lecture Slides, 2018.

[65] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms, 2014.

[66] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[67] Yves Sohege, Marcos Quinones-Grueiro, and Gregory Provan. Unknown fault tolerant control using deep reinforcement learning: A blended control approach. In *Principles of Diagnostics (DX-19) Conference*, page 23, 2019.

[68] Doo Re Song, Chuanyu Yang, Christopher McGreavy, and Zhibin Li. Recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 311–318. IEEE, 2018.

[69] Brian L Stevens, Frank L Lewis, and Eric N Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. John Wiley & Sons, 2015.

[70] Robert Sullivan. Cover picture, 2016.

[71] Bo Sun and Erik-Jan van Kampen. Incremental model-based global dual heuristic programming with explicit analytical calculations applied to flight control. *Engineering Applications of Artificial Intelligence*, 89:103425, 2020.

[72] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2011.

[73] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[74] Antonios Tsourdos, Ir Adhi Dharma Permana, Dewi H Budiarti, Hyo-Sang Shin, and Chang-Hun Lee. Developing flight control policy using deep deterministic policy gradient. In *2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, pages 1–7. IEEE, 2019.

[75] MA van den Hoek, CC de Visser, and DM Pool. Identification of a cessna citation ii model based on flight test data. In *Advances in Aerospace Guidance, Navigation and Control*, pages 259–277. Springer, 2018.

[76] CAAM van Der Linden. Dasmat-delft university aircraft simulation model and analysis tool: A matlab/simulink environment for flight dynamics and control analysis. *Series 03: Control and Simulation 03*, 1998.

[77] Erik-Jan van Kampen, QP Chu, and JA Mulder. Continuous adaptive critic flight control aided with approximated plant dynamics. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 6429, 2006.

[78] Ganesh K Venayagamoorthy, Ronald G Harley, and Donald C Wunsch. Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator. *IEEE Transactions on Neural Networks*, 13(3):764–773, 2002.

[79] Yufei Wang and Tianwei Ni. Meta-sac: Auto-tune the entropy temperature of soft actor-critic via metagradient. *arXiv preprint arXiv:2007.01932*, 2020.

[80] Marco Wiering and Martijn van Otterlo. Reinforcement learning, state-of-the-art. *Adaptation, learning, and optimization*, 12:3, 2012.

[81] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[82] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1871–1879. IEEE, 2018.

[83] Ye Zhou, Erik-Jan van Kampen, and QiPing Chu. Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback. *Journal of Guidance, Control, and Dynamics*, 40(2):493–496, 2016.

[84] Ye Zhou, Erik-Jan van Kampen, and Qi Ping Chu. Incremental model based online dual heuristic programming for nonlinear adaptive control. *Control Engineering Practice*, 73:13–25, 2018.