Reinforcement Learning for Flight Control

Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance

Casper Teirlinck

TU Delft

Reinforcement Learning for Flight Control

Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance

by

Casper Teirlinck

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday September 14, 2022 at 14:00.

Student number:	4680723	
Project duration:	October 25, 2021 – September 14, 2022	
Thesis committee:	Dr. ir. C.C. de Visser,	TU Delft, chair
	Dr. ir. E. van Kampen,	TU Delft, supervisor
	Dr. ir. E. Mooij,	TU Delft

Preface

The world of artificial intelligence, machine learning and especially reinforcement learning can be daunting, but especially exciting with the tremendous progress made in recent years. In an effort to satiate my own curiosity, this thesis aims to continue the chain of research and new insights in this field within the context of safety and autonomy in aerospace. The code developed during this research has been made public[1] to serve the reproducibility of this thesis, and with the hope that it can be of further help for future students and researchers.

This thesis marks the end of my studies in Delft, a greatly fulfilling and exciting chapter of my life where I have learned more than my high-school self could have ever imagined only five years ago. The passion and excitement for engineering, conveyed by the numerous exceptional professors and fellow students in Delft, has given me the drive to push through this challenging study. First and foremost, I would like to thank my supervisor, Dr. ir. E. van Kampen for his continued guidance and supervision throughout the entire research. Your input and expertise have provided a backbone for the successful completion of my thesis. None of this would have been possible without the unconditional support of my family, especially my parents, who I would like to thank for their continued encouragement, support, and love. Thanks to all of you.

Casper Teirlinck
Delft, August 2022

[1]Code available at `https://github.com/CasperTeirlinck/RLFC-SACIDHP`

Contents

List of Figures

List of Tables

List of Symbols

T	Episode period, [s]
N	Number of time steps
t	Time step
a_t	Action vector at time step t
s_t	Observation vector at time step t
x_t	State vector at time step t
r_t	Reward at time step t
Δa_t	Action vector increment at time step t
Δx_t	State vector increment at time step t
$\mathcal{A}$	Action space
$\mathcal{S}$	Observation space
$\mathcal{R}$	Reward space
γ	Discount factor
G_t	Discounted return at time step t
A	State matrix
B	Input matrix
F	Incremental state matrix estimate
G	Incremental input matrix estimate
Θ	Incremental parameter matrix
X	Incremental measurement matrix
Λ	Covariance matrix
ϵ	Innovation
$\mathcal{P}\{x \mid y\}$	Probability of x occurring given y
$\mathbb{E}[x]$	Expectation of random variable x
$\pi(a_t \mid s_t)$	Stochastic policy, probability of selecting an action a_t given the observation s_t
$\pi(s_t)$	Deterministic policy, gives an action a_t given the observation s_t
π_*	Optimal policy
π_θ	Parameterized policy with parameter vector θ
$v_\pi(s_t)$	State-value function following policy π
$v_*(s_t)$	State-value function following the optimal policy π_*
$V(s_t)$	State-value function estimate of v_π
$V_w(s_t)$	Parameterized state-value function estimate with parameter vector w
$V'_{w'}(s_t)$	Parameterized target state-value function estimate with parameter vector w'
$q_\pi(s_t, a_t)$	Action-value function following policy π
$q_*(s_t, a_t)$	Action-value function following the optimal policy π_*
$Q(s_t, a_t)$	Action-value function estimate of q_π
$Q_w(s_t, a_t)$	Parameterized action-value function estimate with parameter vector w
$Q'_{w'}(s_t, a_t)$	Parameterized target action-value function estimate with parameter vector w'
$\lambda(s_t)$	State-value derivative w.r.t the state vector

$\lambda_w(s_t)$	Parameterized state-value derivative w.r.t the state vector with parameter vector w
$\lambda'_{w'}(s_t)$	Parameterized target state-value derivative w.r.t the state vector with parameter vector w'
L	Loss function
δ	Temporal-Difference error
∇_x	Gradient w.r.t x
$\mathcal{D}$	Replay buffer
$\mathcal{B}$	Mini-batch sampled from the replay buffer
$\mathcal{H}$	Entropy
$\bar{\mathcal{H}}$	Target entropy
η	Entropy/Temperature coefficient
η_a	Learning rate actor
η_c	Learning rate critic
τ	Target mixing factor
κ	Reward scaling factor
γ_{RLS}	Recursive Least Squares forgetting factor
h	Hidden layer unit
o	Output layer unit
α	Angle of attack, [deg or rad]
q	Pitch rate, [deg/s or rad/s]
δ_e	Elevator deflection, [deg or rad]
V	Airspeed, [m/s]
$\bar{c}$	Mean aerodynamic chord, [m]
μ_c	Relative density for symmetric motions
K_Y	Non-dimensional radius of gyration about the Y -axis
C_{Z_α}	Z-component coefficient derivative w.r.t angle of attack
$C_{Z_{\dot{\alpha}}}$	Z-component coefficient derivative w.r.t $\frac{\dot{\alpha}\bar{c}}{V}$
C_{Z_q}	Z-component coefficient derivative w.r.t pitch rate
$C_{Z_{\delta_e}}$	Z-component coefficient derivative w.r.t elevator deflection
C_{m_α}	Pitching moment coefficient derivative w.r.t. angle of attack
$C_{m_{\dot{\alpha}}}$	Pitching moment coefficient derivative w.r.t. $\frac{\dot{\alpha}\bar{c}}{V}$
C_{m_q}	Pitching moment coefficient derivative w.r.t. pitch rate
$C_{m_{\delta_e}}$	Pitching moment coefficient derivative w.r.t. elevator deflection
$\Delta x_{c.g.}$	Shift in centre of gravity, [m]
μ	Mean vector
σ	Standard deviation vector
σ^2	Variance vector

List of Acronyms

(e)VTOL (Electrical) Vertical Take-Off and Landing.

ABS Adaptive Backstepping.

ACD Adaptive Critic Design.

AD Action-Dependent.

ADDHP Action-Dependent Dual Heuristic Programming.

ADGDHP Action-Dependent Global Dual Heuristic Programming.

ADHDP Action-Dependent Heuristic Dynamic Programming.

ADNI Adaptive Nonlinear Dynamic Inversion.

ADP Approximate Dynamic Programming.

ANN Artificial Neural Network.

BS Backstepping.

CE Cross-Entropy.

CPI Conservative Policy Iteration.

DASMAT Delft University of Technology Aircraft Simulation Model and Analysis Tool.

DDPG Deep Deterministic Policy Gradient.

DHP Dual Heuristic Programming.

DL Deep Learning.

DNN Deep Neural Network.

DP Dynamic Programming.

DPG Deterministic Policy Gradient.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

GD Gradient Descent.

GDHP Global Dual Heuristic Programming.

GPI Generalised Policy Iteration.

HDP Heuristic Dynamic Programming.

i.d.d. Independent and identically distributed (i.i.d).

iADP Incremental Approximate Dynamic Programming.

IBS Incremental Backstepping.

IDHP Incremental Dual Heuristic Programming.

IGDHP Incremental Global Dual Heuristic Programming.

IHDP Incremental Heuristic Dynamic Programming.

INDI Incremental Nonlinear Dynamic Inversion.

KL Kullback–Leibler.

LSTD Least Squares TD Learning.

LTI Linear Time Invariant.

MC Monte-Carlo.

MDP Markov Decision Process.

ML Machine Learning.

NDI Nonlinear Dynamic Inversion.

NN Neural Network.

PER Prioritized Experience Replay.

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.

SPG Stochastic Policy Gradient.

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

New automation techniques play a vital role in both the safety and economics of current and future aerospace industry needs. Developments in urban air mobility initiatives focusing on (e)VTOL aircraft have a need for novel automation techniques. Many safety and autonomy challenges have to be overcome to make this viable [10]. In commercial air transportation, there is also significant interest in better fault tolerance and automation techniques. Accident rates have dropped significantly over the last two decades, however of all fatal accidents in commercial flights from 2009 to 2018, 60.4% are still caused by in-flight loss of control [1].

Flight control systems that are currently in use mainly use classical control theory. These techniques use linear controllers and require gain scheduling in order to cover the flight envelope of non-linear systems. This gain-scheduling process can be tedious, especially for complex coupled-dynamics systems, and also relies on an accurate plant dynamics model [3]. These classical controllers also lack adaptive behaviour and are not sufficient for increasingly autonomous systems that need to be able to deal with unexpected failures. Hence, there is a need for methods that better handle non-linear systems and can enable fault-tolerant control.

Developing methods that better deal with system non-linearity has been an active research topic in the past decades. Several non-linear control methods have emerged, like Nonlinear Dynamic Inversion (NDI) [35] [16] and Backstepping (BS) [75]. These methods avoid the gain scheduling step of classical controllers. NDI and BS can also be implemented to have adaptive functionality in order to increase fault tolerance and react better to failures[46] [18]. This results in the Adaptive Nonlinear Dynamic Inversion (ADNI) and Adaptive Backstepping (ABS) methods being researched, for example for spacecraft attitude control [89]. One limitation that remains however is the dependence on an accurate plant model. This is what incremental versions try to address, Incremental Nonlinear Dynamic Inversion (INDI) [21] and Incremental Backstepping (IBS) [30] have recently been developed and decrease the model dependence by estimating an incremental plant model. These incremental methods have already been proven successful with flight tests on CS-25 class aircraft [21].

Reinforcement Learning (RL) from the field of Machine Learning (ML) [62] is now being researched for adaptive control applications. Traditional tabular RL methods use discrete state and action spaces, which are infeasible for controlling most complex airborne systems. Thanks to the developments in continuous function approximators such as Artificial Neural Networks (ANNs), continuous state and action spaces are possible and several methods have been developed.

Adaptive and robust control can be achieved with RL by using online and offline learning techniques. While training online can be highly adaptive, it can also be a safety concern due to a continually changing policy. Instead, RL controllers can also be trained offline, where the generalization power of the function approximators provides robust control.

The field of Approximate Dynamic Programming (ADP) [71] [86] uses mostly shallow ANNs as function approximators and contains the class of Adaptive Critic Designs (ACDs) [43]. These actor-critic designs have been successfully applied in simulation to flight control of a business jet aircraft

[19] and helicopter tracking control [17]. The ADP methods do however still require an accurate model of the plant dynamics in an offline training phase, which limits the ability to deal with random failures. Newer ACDs implement an incremental approach to system identification similar to INDI and IBS, which makes them highly adaptive and not reliant on accurate system models. Incremental Heuristic Dynamic Programming (IHDP) [96] and Incremental Dual Heuristic Programming (IDHP) [98] both are incremental methods that can be applied fully online and provide adaptive fault-tolerant control by identifying an incremental plant model in real-time. Recent works [28] [34] [36] on these Incremental Approximate Dynamic Programming (iADP) methods have further explored their applicability to flight control of the PH-LAB research aircraft, but still require more validation and high fidelity simulations before real-world flight tests are feasible. The main advantage for (i)ADP methods is the high sample efficiency, with the potential to react to severe failure cases.

Thanks to the increasing popularity of Deep Learning (DL) in recent years, the advancements made in training Deep Neural Networks (DNNs) have found their way into RL, characterizing the field of Deep Reinforcement Learning (DRL). A major development by DeepMind in 2015 used DRL to achieve human-level performance on a number of classic Atari games using a deep Q-network [48]. With a similar achievement in 2017, DeepMind's AlphaGo [73] also demonstrated the ability of DRL algorithms to outperform a human by defeating a world-class GO player. For the continuous control of an aircraft, DRL methods that specifically can handle continuous spate and action spaces can be used. Trust Region Policy Optimization (TRPO) [68] and Deep Deterministic Policy Gradient (DDPG) [42] are two notable extensions of the classic Q-learning approach that provide continuous control. TRPO has been improved on by the state-of-the-art algorithm Proximal Policy Optimization (PPO) which has been successfully demonstrated in control of a fixed-wing UAV [5] and path generation for an aircraft guidance task[88]. DDPG has also been improved with state-of-the-art algorithms such as Twin-Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) which improve the learning instability of DDPG. UAV path planning and tracking control have been demonstrated using TD3 [70] [26] and SAC [9] [38] [4]. SAC has recently also been researched in the context of a coupled flight controller for the PH-LAB research aircraft [11]. The main advantage of DRL methods is the generalization power of the DNNs and the scalability to high-dimensional spaces.

1.2. Research Objective and Questions

The research objective as seen below describes the main goal of the research. Several sub-goals have also been identified (RO1, RO2, RO3, RO4) which contain the high-level tasks required to complete the research objective.

The term model-independent is used here to describe a requirement for the reinforcement learning flight controller to be developed. Since literature can vary on the definition of model-free, it is clarified here that for this research, the term *model-independent* is used to describe algorithms that contain no knowledge of the environment, as well as algorithms that learn a model of the environment as part of the learning process.

Research Objective

Contribute to the development of *model-independent*, *adaptive* and *robust* flight control with the purpose of progressing towards enabling flight tests **by** investigating the challenges in reducing the simulation reality-gap of current reinforcement learning implementations and developing a reinforcement learning-based flight controller for the PH-LAB (Cessna Citation II) research aircraft.

RO1 Identify an RL framework best suited for flight control on the Cessna Citation II through a state-of-the-art review.

RO2 Identify the main challenges in reducing the simulation reality-gap of current implementations.

RO3 Implement the RL framework into a flight controller for the Cessna Citation II.

RO4 Evaluate the performance of the RL controller in the presence of realistic sensor dynamics and external disturbances.

Following the research objective, a set of three main research questions is formulated (RQ1, RQ2, RQ3). Each research question contains several sub-questions, yielding an answer to the main question. Answering these questions will help in achieving the research objective.

Research Questions

RQ1 What RL framework is best suited for implementation on the Cessna Citation II?

RQ1.1 What is the current state-of-the-art for continuous adaptive and robust flight control?

RQ1.2 What are the main challenges in reducing the simulation reality-gap of current implementations?

RQ1.3 How can the proposed RL framework be implemented for a simple dynamic system?

RQ1.4 How does the proposed RL framework perform for a simple dynamic system?

RQ2 How can the proposed RL controller be integrated into the Cessna Citation II?

RQ2.1 At what control level should the RL controller be implemented?

RQ2.2 What are the architectural characteristics of the RL controller to ensure applicability to the control level?

RQ2.3 What are the characteristic system dynamics, sensor dynamics and actuator dynamics that need to be accounted for to ensure applicability to the real system?

RQ3 What is the overall performance and stability of the proposed RL controller on the Cessna Citation II?

RQ3.1 How are the performance and stability metrics defined and what are their requirements to ensure applicability to the real system?

RQ3.2 How does it perform in standard manoeuvres?

RQ3.3 How does it perform on fault tolerance?

RQ3.4 How does it compare to baseline?

RQ3.5 What is the effect of sensor dynamics and external disturbances on performance and stability?

1.3. Report Outline

This thesis report consists of four main parts. The scientific article in Part I presents the most important processes and outcome of the final results. Additional results from this process are presented in Part III and include results on additional failure modes in chapter 9, an alternative controller structure in chapter 5 and robustness analyses in chapter 6, chapter 7 and chapter 8. The processes used for verification and validation are discussed inchapter 10.

In Part II, the preliminary research is presented. This includes a literature review and preliminary analysis. In chapter 3, the literature review provides the fundamentals of reinforcement learning theory in order to support the theory in the rest of the report. Then, chapter 4 contains the preliminary analysis where the most promising RL agents are implemented and tested on a simple dynamical system and provide the starting point for implementation for the final thesis work. Finally, Part IV wraps up the thesis with the conclusion and recommendations for future research.

I

Scientific Article

Hybrid Soft Actor-Critic and Incremental Dual Heuristic Programming Reinforcement Learning for Fault-Tolerant Flight Control

C. Teirlinck*
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Recent advancements in fault-tolerant flight control have involved model-free offline and online Reinforcement Learning (RL) algorithms in order to provide robust and adaptive control to autonomous systems. Inspired by recent work on Incremental Dual Heuristic Programming (IDHP) and Soft Actor-Critic (SAC), this research proposes a hybrid SAC-IDHP framework aiming to combine adaptive online learning from IDHP with the high complexity generalization power of SAC in controlling a fully coupled system. The hybrid framework is implemented into the inner loop of a cascaded altitude controller for a high-fidelity, six-degree-of-freedom model of the Cessna Citation II PH-LAB research aircraft. Compared to SAC-only, the SAC-IDHP hybrid demonstrates an improvement in tracking performance of 0.74%, 5.46% and 0.82% in nMAE for nominal case, longitudinal and lateral failure cases respectively. Random online policy initialization is eliminated due to identity initialization of the hybrid policy, resulting in an argument for increased safety. Additionally, robustness to biased sensor noise, initial flight condition and random critic initialization is demonstrated.

Nomenclature

$\mathbf{x}, \mathbf{s}, \mathbf{a}$	=	environment state, reinforcement-learning state and environment action vectors
$\mathbf{x}^e, \mathbf{x}^c$	=	environment state error and cost vectors
n, k, m	=	number of reinforcement-learning states, environment states and environment actions
r, γ	=	instantaneous reward and discount factor
τ	=	target smoothing factor
δ	=	temporal difference error
$t, \Delta t, N, T$	=	current time-step, sample time [s], number of samples and simulation time [s]
$f(\mathbf{s}, \mathbf{a})$	=	state transition function
λ_T, λ_S	=	temporal and spacial scaling coefficients
$\pi, \pi_\theta, \mu_\theta, \sigma_\theta$	=	policy, parametric policy and stochastic policy parameterized mean and standard deviation
Q_π, Q_w, V_π	=	action-state value function, parameterized action-state value function and state value function
λ_w	=	parameterized state-derivative of the state value function
θ, w, w'	=	policy, critic and target critic parameter vectors
$\mathcal{H}, \bar{\mathcal{H}}, \eta$	=	entropy, entropy target and temperature coefficient
η_a, η_c	=	actor and critic learning rates
$\mathcal{D}, \mathcal{B}$	=	replay buffer and mini-batch
$L_\pi, L_Q, L_\lambda, L_\eta$	=	loss functions for policy, SAC-critic, IDHP-critic and temperature coefficient
F, G, Θ, Λ, X	=	state, input, parameter, covariance and measurement matrices of the incremental model
$\kappa, \boldsymbol{\epsilon}$	=	incremental model forgetting factor and innovation or error vector
$\delta_e, \delta_a, \delta_r$	=	elevator, aileron and rudder deflection [deg]
p, q, r	=	roll rate, pitch rate and yaw rate [deg/s]
$\alpha, \beta, \theta, \phi, \psi$	=	angle of attack, sideslip angle, pitch angle and heading angle [deg]
V, h	=	airspeed [m/s], altitude [m]

*M.Sc. Student, Faculty of Aerospace Engineering, Department of Control & Operations, Section Control & Simulation, Delft University of Technology.
Code available at `https://github.com/CasperTeirlinck/RLFC-SACIDHP`

I. Introduction

New automation techniques play a vital role in both the safety and economics of current and future aerospace industry needs. Developments in urban air mobility initiatives focusing on (e)VTOL aircraft have a need for novel automation techniques. Many safety and autonomy challenges have to be overcome to make this viable [1]. In commercial air transportation, there is also significant interest in better fault tolerance and automation techniques. Accident rates have dropped significantly over the last two decades, however of all fatal accidents in commercial flights from 2009 to 2018, 60.4% are still caused by in-flight loss of control [2]. Flight control systems that are currently in use mainly use classical control theory. These techniques use linear controllers and require gain scheduling in order to cover the flight envelope of non-linear systems. This gain-scheduling process can be tedious, especially for complex coupled-dynamics systems, and also relies on an accurate plant dynamics model [3]. These classical controllers also lack adaptive behaviour and are not sufficient for increasingly autonomous systems that need to be able to deal with unexpected failures. Hence, there is a need for methods that better handle non-linear systems and can enable fault-tolerant control. Reinforcement Learning (RL) from the field of Machine Learning (ML) [4] is now being researched for adaptive control applications. Traditional tabular RL methods use discrete state and action spaces, which are infeasible for controlling most complex airborne systems. Thanks to the developments in continuous function approximators such as Artificial Neural Networks (ANNs), continuous state and action spaces are possible and several methods have been developed. Adaptive and robust control can be achieved with RL by using online and offline learning techniques. While training online can be highly adaptive, it can also be a safety concern due to a continually changing policy. Instead, RL controllers can also be trained offline, where the generalization power of the function approximators provides robust control.

The field of Approximate Dynamic Programming (ADP) [5] [6] uses mostly shallow ANNs as function approximators and contains the class of Adaptive Critic Designs (ACDs) [7]. These actor-critic designs have been successfully applied in simulation to flight control of a business jet aircraft [8] and helicopter tracking control [9]. The ADP methods do however still require an accurate model of the plant dynamics in an offline training phase, which limits the ability to deal with random failures. Newer ACDs implement an incremental approach to system identification, which makes them highly adaptive and not reliant on accurate system models. Incremental Heuristic Dynamic Programming (IHDP) [10] and Incremental Dual Heuristic Programming (IDHP) [11] both are incremental methods that can be applied fully online and provide adaptive fault-tolerant control by identifying an incremental plant model in real-time. Recent works [12] [13] [14] on these Incremental Approximate Dynamic Programming (iADP) methods have further explored their applicability to flight control of the PH-LAB research aircraft, but still require more validation and high fidelity simulations before real-world flight tests are feasible. The main advantage for (i)ADP methods is the high sample efficiency, with the potential to react to severe failure cases.

Thanks to the increasing popularity of Deep Learning (DL) in recent years, the advancements made in training Deep Neural Networks (DNNs) have found their way into RL characterizing the field of Deep Reinforcement Learning (DRL). A major development by DeepMind in 2015 used DRL to achieve human-level performance on a number of classic Atari games using a deep Q-network [15]. With a similar achievement in 2017, DeepMind's AlphaGo [16] also demonstrated the ability of DRL algorithms to outperform a human by defeating a world-class GO player. For the continuous control of an aircraft, DRL methods that specifically can handle continuous spate and action spaces can be used. Trust Region Policy Optimization (TRPO) [17] and Deep Deterministic Policy Gradient (DDPG) [18] are two notable extensions of the classic Q-learning approach that provide continuous control. TRPO has been improved on by the state-of-the-art algorithm, Proximal Policy Optimization (PPO) which has been successfully demonstrated in control of a fixed-wing UAV [19] and path generation for an aircraft guidance task[20]. DDPG has also been improved with state-of-the-art algorithms such as Twin-Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) which improve the learning instability of DDPG. UAV path planning and tracking control have been demonstrated using TD3 [21] [22] and SAC [23] [24] [25]. SAC has recently also been researched in the context of a coupled flight controller for the PH-LAB research aircraft [26], making the SAC controller the baseline for this research. The main advantage of DRL methods is the generalization power of the DNNs and the scalability to high-dimensional spaces.

The contribution of this paper is to advance the development of model-independent, adaptive and robust flight controllers. More specifically, by developing an RL-based flight controller for the Cessna Citation II. This is achieved by presenting a hybrid framework that aims to combine the advantages of the state-of-the-art SAC and IDHP frameworks in providing fault-tolerance to unexpected failures and providing robust flight control.

The theoretical foundations for the SAC and IDHP algorithms are presented in section II followed by the flight controller design in section III. The results of the hybrid method compared to SAC-only are discussed in section IV followed by the conclusion in section V.

II. Fundamentals

This section first formulates the flight control task as a reinforcement learning problem. Additionally, a detailed overview of the two baseline algorithms used in finding a suitable control policy is provided.

A. Reinforcement Learning Problem Formulation

RL frameworks involve working inside the agent-environment interface, which is mathematically described by a Markov Decision Process (MDP). This consists of a RL agent that at time t selects an action $\mathbf{a}_t \in \mathbb{R}^m$ which acts on the environment with state $\mathbf{s}_t \in \mathbb{R}^n$. The next state is determined by the state-transition function as seen in Equation 1 which is governed by the environment dynamics. The MDP has the Markov Property, meaning that the current state and action carry all the information to predict the next state. A scalar reward $r_{t+1} \in \mathbb{R}$ is then determined based on the environment state and used as feedback for the agent. The goal of the agent is to maximize the reward over the time of the episode of N time-steps, or the discounted return G_t defined in Equation 2. The discount factor γ is used to trade-off future to immediate rewards.

Actor-critic RL frameworks consists of an actor that learns the policy, and a critic that learns a value function. The policy can be stochastic as in Equation 3, or deterministic as in Equation 5. Both policy types are used in the hybrid framework presented in this research. Additionally, the critic can estimate an action-value function, or Q-function Q_π as seen in Equation 4. The Q-function maps a given state and action to a scalar value representing the value of being in that state and taking the given action, while following the policy π thereafter. This type of value function is used in the SAC framework. A state value function as seen in Equation 6 works similarly, but only maps a given state to the value of being in that state. The state derivative of the state value function is used in the IDHP framework.

Note that in practice, the state $\mathbf{s}_t$ is often a subset of the full environment state $\mathbf{x}_t$ and/or is augmented with additional samples as shown in section III, however the terms for RL-state and environment state are often used interchangeably.

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t) \qquad (1) \qquad\qquad G_t = \sum_{k=0}^{N} \gamma^k r_{t+k+1} \qquad (2)$$

$$\mathbf{a}_t \sim \pi(\cdot \mid \mathbf{s}_t) \qquad (3) \qquad\qquad Q_\pi(\mathbf{s}_t, \mathbf{a}_t) = \underset{\pi}{\mathbb{E}}\left[G_t \mid \mathbf{s}_t, \mathbf{a}_t\right] \qquad (4)$$

$$\mathbf{a}_t = \pi(\mathbf{s}_t) \qquad (5) \qquad\qquad V_\pi(\mathbf{s}_t) = \underset{\pi}{\mathbb{E}}\left[G_t \mid \mathbf{s}_t\right] \qquad (6)$$

B. Soft Actor-Critic Framework

Soft Actor-Critic (SAC) is a state-of-the-art offline-learning off-policy DRL algorithm [27]. The main characteristics of SAC are the use of soft policy iteration which includes an entropy term in the policy objective function, and the use of a stochastic policy during training. Hence, a high level of exploration is achieved and the SAC agent is trained offline. When evaluated, the policy is sampled using only the mean of the policy distribution, making it deterministic at evaluation. Since SAC is off-policy, experience replay is used by storing samples in the replay buffer $\mathcal{D}$. Every learning step, a mini-batch $\mathcal{B}$ of experience samples can be sampled from the replay buffer.

1. Actor

The actor learns the SAC policy π_θ, parameterized by the parameter vector θ representing the parameters of a DNN. The stochastic policy distribution is implemented by having two outputs of the policy, being the standard deviation σ_θ and mean μ_θ. This is then used to sample an action from a normal distribution with σ_θ and μ_θ. Note that this sampling requires a "reparameterization trick" to ensure differentiability of the sampled action, which is necessary for the gradient calculations. This is usually implemented using an input Gaussian noise vector ϵ_t and sampling an action using $\mathbf{a}_t = f_\theta(\epsilon_t, \mathbf{s}_t) = \mu_\theta(\mathbf{s}_t) + \epsilon_t \cdot \sigma_\theta(\mathbf{s}_t)$. In [28] this reparameterization is implemented manually, but the implementation of the normal distribution used here applies this under the hood.

The loss function for the policy can be seen in Equation 7. It depends on the Q-function critics and also includes the entropy term with the coefficient η. The log probabilities $\log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$ are also derived from the normal distribution and used in the update rules of both the actor and the critic.

$$L_\pi = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim \mathcal{B} \\ \mathbf{a}_t \sim \pi}} \left[\min_{i=1,2} Q'_{w'_i}(\mathbf{s}_t, \mathbf{a}_t) - \eta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \right] \tag{7}$$

In complex control tasks like 6-degree-of-freedom flight control, the action of the SAC policy tends to be highly noisy and oscillatory. A method to smooth out the control input can be implemented by letting the policy control the action increment $\Delta\mathbf{a}$ instead of the action directly [26]. The development of a hybrid framework in this research however requires both the SAC and the IDHP frameworks to operate inside the same control loop. Initial tests showed the IDHP framework has difficulty controlling an action derivative as opposed to the direct action of a complex system. Hence, a different method is chosen to smooth out the policy that is compatible with direct action control using an additional policy regularization term.

Using temporal and spatial regularization terms, described by Conditioning for Action Policy Smoothness (CAPS) [29] forces the policy to keep new actions close to the previous action, and keeps actions close to actions corresponding to similar states. The temporal regularization loss is defined in Equation 8 and computes the distance between the previous and current actions. The spacial regularization loss is defined in Equation 9 and computes the distance between the action and the action based on a normally sampled state $\bar{\mathbf{s}} \sim N(\mathbf{s}, \sigma = 0.05)$. The distances are implemented as the L2-norm. The total CAPS loss term is defined in Equation 10 and includes two additional scaling parameters λ_T and λ_S for the temporal and spacial terms respectively. Note that only the mean, or the deterministic action of the policy is used in computing the distances and not the entire policy distribution

$$L_T = D(\pi(\mathbf{s}_t), \pi(\mathbf{s}_{t+1})) = ||\pi(\mathbf{s}_t) - \pi(\mathbf{s}_{t+1})||_2 \tag{8}$$

$$L_S = D(\pi(\mathbf{s}), \pi(\bar{\mathbf{s}})) = ||\pi(\mathbf{s}) - \pi(\bar{\mathbf{s}})||_2 \tag{9}$$

$$L_\pi^{CAPS} = L_\pi + \lambda_T L_T + \lambda_S L_S \tag{10}$$

2. Critic

The critic learns the Q-function and in this case consists of two separate critics Q_{w_i} with their respective target critics $Q'_{w'_i}$ and parameters vectors w_i and w'_i for $i \in [1, 2]$. Each Q-function is updated separately using its own loss function, and the minimum of the two Q-values is used in the update rules to prevent overestimation. The target critics are used to slow down the gradient updates with the purpose of increasing learning stability. This is achieved by updating the target weights according to a soft update mechanism $w'_{t+1} = \tau w_t + (1 - \tau) w'_t$ using the smoothing factor τ. The loss function for the critic can be seen in Equation 11. As already seen in the actor update rule, the minimum of the twin target critics is used in the update rule of actor and critic. This is done to prevent over-estimation of the value.

$$L_{Q_i} = \mathop{\mathbb{E}}_{\substack{(\mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{B} \\ \mathbf{a}_{t+1} \sim \pi}} \left[\left(Q_{w_i}(\mathbf{s}_t, \mathbf{a}_t) - \left(r_{t+1} + \gamma \left(\min_{i=1,2} Q'_{w'_i}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \eta \log \pi_\theta(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1}) \right) \right) \right)^2 \right] \tag{11}$$

3. Entropy

The SAC framework tries to maximize entropy in addition to maximizing the expected return. This ensures a high level of exploration. The entropy of the policy distribution is defined in Equation 12.

$$\mathcal{H}(\pi_\theta(\cdot \mid \mathbf{s}_{t+1})) = \mathop{\mathbb{E}}_{\mathbf{a} \sim \pi} \left[-\log \pi_\theta(\mathbf{a} \mid \mathbf{s}_t) \right] \tag{12}$$

In the update rules of the actor and critic, the entropy term is weighted using the entropy coefficient or temperature coefficient η. The SAC algorithm has been shown to be highly sensitive to the temperature coefficient, thus it was proposed by [27] to learn η automatically. The loss function for this automatic learning process can be seen in Equation 13. The term $\bar{\mathcal{H}}$ is a constant entropy target and is set to the negative of the action space size [28]. In the case of an aircraft environment with three control surfaces, $\bar{\mathcal{H}} = -3$.

$$L_\eta = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim \mathcal{B} \\ \mathbf{a}_t \sim \pi}} \left[-\eta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) - \eta \bar{\mathcal{H}} \right] \tag{13}$$

4. Overview

In Figure 1, an overview of the SAC framework can be seen showing the interactions between the actor, critic, entropy and environment. Data flow is depicted by solid arrows, while update processes are shown using dashed arrows. The off-policy design is made clear by having two kinds of forward passes through the policy, one where the environment observation generates a new action to take every time-step, and one where the replay buffer is sampled to perform the updates. Also, different notations of the replay buffer signals are used where $\{a_t\}$ is an action batch sampled from the replay buffer and $\{a_t\} \sim \pi$ is a batch of newly generated actions using the observations from the replay buffer. The policy and Q-functions are modelled using DNNs. The update rules described above update the network parameters according to Stochastic Gradient Descent (SGD) using the gradients $\nabla_\theta L_\pi$, $\nabla_{w_i} L_{Q_i}$ and $\nabla_\eta L_\eta$ for the actor, critics and temperature coefficient respectively.

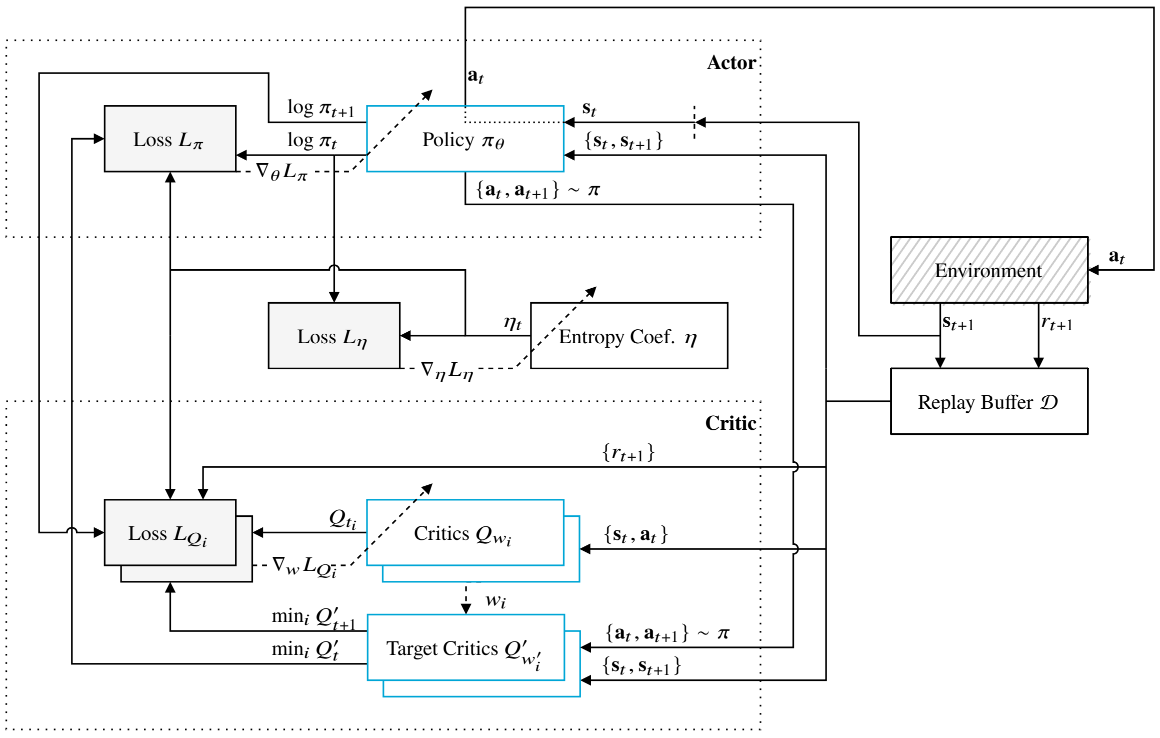


Fig. 1 SAC framework architecture, adapted from [26]

C. Incremental Dual Heuristic Programming Framework

Incremental Dual Heuristic Programming (IDHP) is a state-of-the-art online on-policy ACD algorithm [7]. IDHP is characterized by a linearized, time-varying incremental model of the environment. This model is estimated online and part of the learning process. The agent assumes no prior knowledge of the environment dynamics, hence this method is still considered model-free in the context of this research. Furthermore, the policy is deterministic and the critic consists of the derivative of the state value function as opposed to a Q-function critic in SAC. Compared to SAC, the sample efficiency is considerably higher, assuming a sufficiently high sampling rate [30], and this method can be trained online providing an adaptive fault-tolerant control policy.

Note that the distinction between the RL-state $\mathbf{s}$ and the environment state $\mathbf{x}$ is important to make here. Since $\mathbf{s}$ is often only a subset or augmented version of $\mathbf{x}$ by design, the state vector used by the partial state derivatives and incremental model has to be explicitly defined as the environment state vector $\mathbf{x}$ in order for the incremental model to retain a meaningful estimation of the system dynamics.

1. Actor

The actor learns the deterministic IDHP policy π_θ parameterized by the parameter vector θ. The loss function for the policy can be seen in Equation 14 and consists of the next Bellman value estimate with γ the discount factor. Since the output of the critic is the state partial derivative of V, the gradient of L_π does not need backpropagation through the critic network and can use the output of the critic directly in the gradient. The gradient of the loss function can then be derived as seen in Equation 15 where the critic value comes from the target critic $\lambda'_{w'}$. The term $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{a}_t}$ can be replaced by the incremental model input matrix G_{t-1} as per definition of the input matrix according to the model discussed in section II.C.3. The term $\frac{\partial \mathbf{a}_t}{\partial \theta}$ is calculated using backpropagation on the actor.

$$L_\pi = -V(\mathbf{s}_t) = -\left[r_{t+1} + \gamma V(\mathbf{s}_{t+1})\right] \tag{14}$$

$$\begin{aligned}\nabla_\theta L_\pi = \frac{\partial L_\pi}{\partial \theta} &= -\left[\frac{\partial r_{t+1}}{\partial \mathbf{x}_{t+1}} + \gamma \lambda'_{w'}(\mathbf{s}_{t+1})\right]\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{a}_t}\frac{\partial \mathbf{a}_t}{\partial \theta} \\ &= -\left[\frac{\partial r_{t+1}}{\partial \mathbf{x}_{t+1}} + \gamma \lambda'_{w'}(\mathbf{s}_{t+1})\right] G_{t-1}\frac{\partial \mathbf{a}_t}{\partial \theta}\end{aligned} \tag{15}$$

2. Critic

The IDHP critic estimates the partial derivative of the state value function with respect to the state $\lambda_w(\mathbf{s}_t) = \frac{\partial V(\mathbf{s}_t)}{\partial \mathbf{x}_t}$ with parameter vector w.

The loss is defined as the mean squared error of the state derivative of the TD error $\frac{\partial \delta_t}{\partial \mathbf{x}_t}$ as seen in Equation 16. In Equation 17, the formulation of the TD error for the critic can be seen, which corresponds to the next value function estimate called the TD target minus the current value estimate $V(\mathbf{s}_t)$. Taking the state partial derivative of the TD error results in Equation 18 where the TD target is calculated using the target critic value $\lambda'_{w'}$. The state derivative of the reward is provided by the environment, while the term $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$ can be computed by using the incremental model as seen in Equation 19. The term $\frac{\partial \mathbf{a}_t}{\partial \mathbf{x}_t}$ or $\frac{\partial \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\partial \mathbf{x}_t}$ can be obtained by backpropagation through the policy network. The loss gradient can then be derived using these previous definitions, as seen in Equation 20. The target critics are updated according to the same smooth update method used in SAC.

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

$$\delta_t = r_{t+1} + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t) \tag{17}$$

$$\frac{\partial \delta_t}{\partial \mathbf{x}_t} = \left[\frac{\partial r_{t+1}}{\partial \mathbf{x}_{t+1}} + \gamma \lambda'_{w'}(\mathbf{s}_{t+1})\right]\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} - \lambda_w(\mathbf{s}_t) \tag{18}$$

$$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} = F_{t-1} + G_{t-1}\frac{\partial \mathbf{a}_t}{\partial \mathbf{x}_t} \tag{19}$$

$$\nabla_w L_\lambda = \frac{\partial L_\lambda}{\partial w} = \frac{\partial L_\lambda}{\partial \lambda_w(\mathbf{s}_t)}\frac{\partial \lambda_w(\mathbf{s}_t)}{\partial w} = -\frac{\partial \delta_t}{\partial \mathbf{x}_t}\frac{\partial \lambda_w(\mathbf{s}_t)}{\partial w} \tag{20}$$

3. Incremental Model

The incremental model provides a future estimate of the environment state to be used in the update rules for the actor and critic. This model is derived from a first-order Taylor series expansion [31] and can be seen in Equation 25. Here, the state matrix F_{t-1} and input matrix G_{t-1} are time-varying and are updated every time-step using an RLS estimator.

The RLS update rule of the incremental model can be seen in Equation 22 with Θ the parameter matrix as defined in Equation 21 and $\kappa \in [0, 1]$ the forgetting factor. The measurement matrix X contains the increments of the previous state and action as seen in Equation 24. The error or innovation ϵ is defined in Equation 26 and represents the prediction error between the actual state and the predicted state. Finally, the covariance matrix Λ estimates a measure of the covariance of the parameter estimates and is updated according to Equation 23. Both the parameter matrix and covariance matrix are expressed recursively and thus need an initial value. In this case, the parameter matrix is initialized as zero's and the covariance matrix as an identity matrix of magnitude Λ_0 as no prior knowledge of the parameter covariances is assumed. The magnitude Λ_0 is usually set to a large value, as the uncertainty of the parameters is high at the initial stage. The state and input matrices of the incremental model are used in the update rules of both the actor and the critic.

$$\Theta_{t-1} = \begin{bmatrix} F_{t-1}^T \\ G_{t-1}^T \end{bmatrix} \quad (21) \qquad \Theta_t = \Theta_{t-1} + \frac{\Lambda_{t-1} X_t}{\kappa + X_t^T \Lambda_{t-1} X_t} \boldsymbol{\epsilon}_t \quad (22) \qquad \Lambda_t = \frac{1}{\kappa} \left[\Lambda_{t-1} - \frac{\Lambda_{t-1} X_t X_t^T \Lambda_{t-1}}{\kappa + X_t^T \Lambda_{t-1} X_t} \right] \quad (23)$$

$$X_t = \begin{bmatrix} \Delta \mathbf{x}_t \\ \Delta \mathbf{a}_t \end{bmatrix} \quad (24) \qquad \Delta \mathbf{x}_{t+1} = F_{t-1} \Delta \mathbf{x}_t + G_{t-1} \Delta \mathbf{a}_t \quad (25) \qquad \boldsymbol{\epsilon}_t = \Delta \mathbf{x}_{t+1}^T - \Delta \hat{\mathbf{x}}_{t+1}^T = \Delta \mathbf{x}_{t+1}^T - X_t^T \Theta_{t-1} \quad (26)$$

4. Overview

An overview of the IDHP framework can be seen in Figure 2 which shows the interactions between the actor, critic, incremental model and the environment. Compared to SAC, the actor and critic networks are much shallower and narrower networks, using only a single layer of neurons. The update rules described above update the network parameters according to SGD using the gradients $\nabla_\theta L_\pi$ and $\nabla_w L_\lambda$ for the actor and critic respectively. Additionally, the incremental model is updated using the RLS update rule.

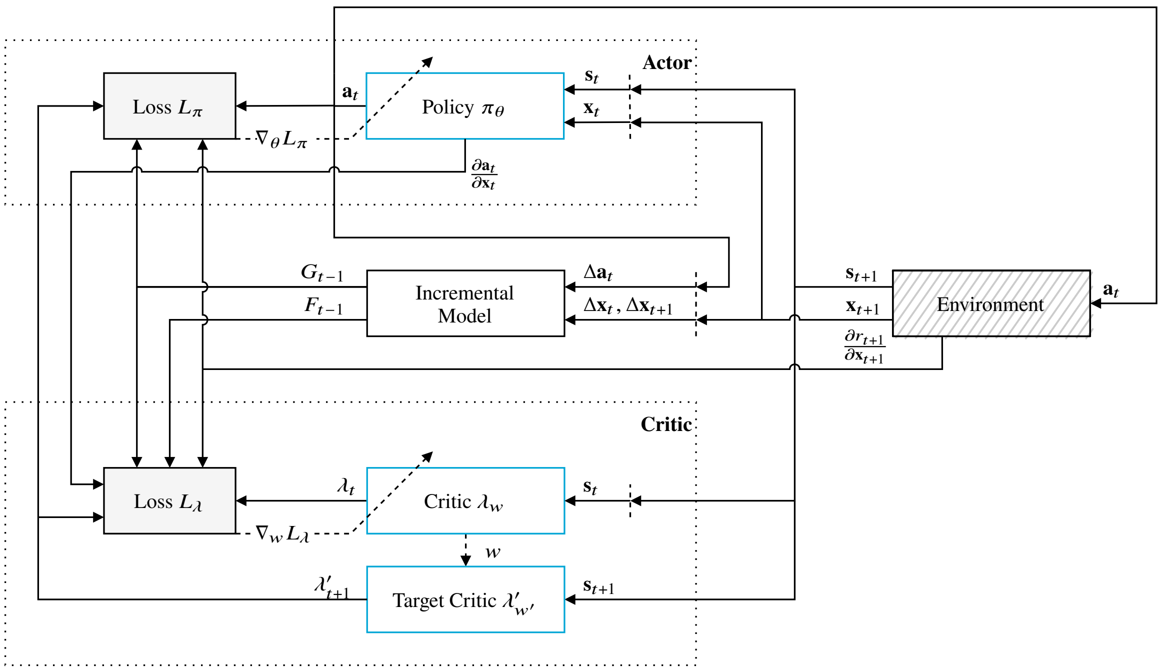


Fig. 2 IDHP framework architecture, adapted from [12][13][14]

III. Flight Controller Design

The flight controller design involves integrating the RL algorithms into a suitable altitude control loop designed to interface with a simulation model of the PH-LAB research aircraft.

A. High-Fidelity Environment Model

The environment is modelled by a high-fidelity non-linear fully-coupled simulation model of the Cessna Citation 500 jet aircraft, built using the DASMAT tool by the Delft University of Technology [32] based on real world flight data. This model can be considered equivalent to the Cessna 550 Citation II PH-LAB aircraft, which is the target platform for the developed controller, despite the difference in fuselage size, engine power and wing size [33]. All simulations are performed with the controller and environment model running at $100Hz$.

The environment state $\mathbf{x} \in \mathbb{R}^k$ and input vector $\mathbf{a} \in \mathbb{R}^m$ can be seen in Equation 27 and Equation 28 respectively. The environment state $\mathbf{x}$ used throughout this paper is the observed state as seen by the RL agent, while the full aircraft state available from the simulation model is denoted by **x'**. A clean configuration is used for all simulations. Additionally, a yaw damper and auto-throttle are provided by the simulation model. The auto-throttle tries to maintain a constant airspeed set by the initial flight condition. The initial control inputs are always untrimmed and zero at the start of a simulation.

$$\mathbf{x'} = [p, q, r, V, \alpha, \beta, \theta, \phi, \psi, h]^T \Rightarrow \mathbf{x} = [p, q, r, \alpha, \theta, \phi, \beta, h]^T \quad (27) \qquad \mathbf{a} = [\delta_e, \delta_a, \delta_r]^T \quad (28)$$

B. Network Architecture

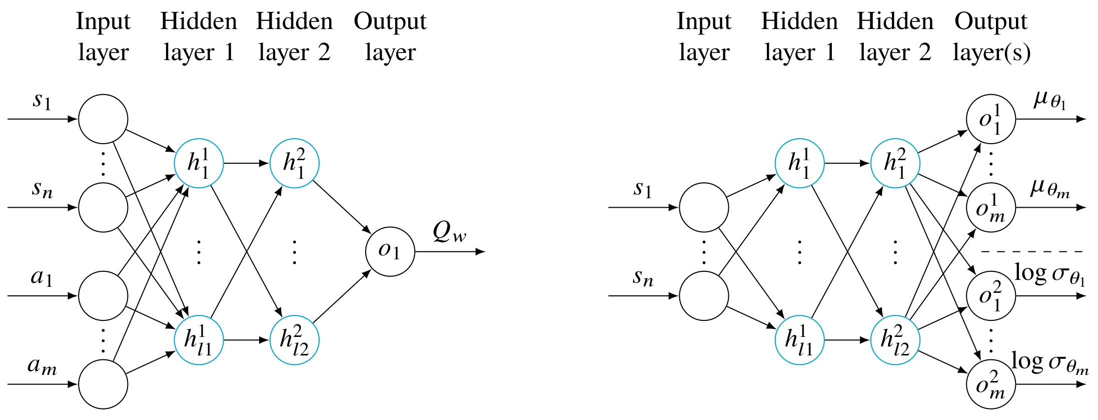
The following two sections describe in more detail the neural network architecture of the SAC and IDHP actors and critics. The SAC network architectures are used in the offline SAC agent of the attitude and altitude controllers, while the SAC-IDHP networks are only used in the inner attitude controller of the final controller structure during online learning.

1. SAC Network Architecture

The network topology of actor and critic can be seen in Figure 3. Hidden layer neurons are identified as h and output layers by o with superscript for layer number and subscript for neuron number.

The network of a single Q-function critic in Figure 3a takes both the RL-state and the action as inputs per definition of the Q-function with a single scalar output. The policy network in Figure 3b is constructed using two separate output layers μ_θ and $\log \sigma_\theta$ for the policy mean and standard deviation respectively. The network estimates the log of the standard deviation in order to stay in $\mathbb{R}$ and the exponential is taken in order to build the policy distribution.

Both the actor and critic contain two hidden layers of sizes $l1$ and $l2$. Each hidden neuron h consists of a weighted linear combination of the input vectors with an additional bias term. The signal is subsequently passed to a LayerNormalization layer [34] and finally passed to a ReLU activation function. The output neurons have a linear activation function. All the weights, biases and normalization parameters form the parameter vectors θ and w_i for the actor and critics respectively.



(a) SAC critic Q-function architecture **(b) SAC policy Architecture**

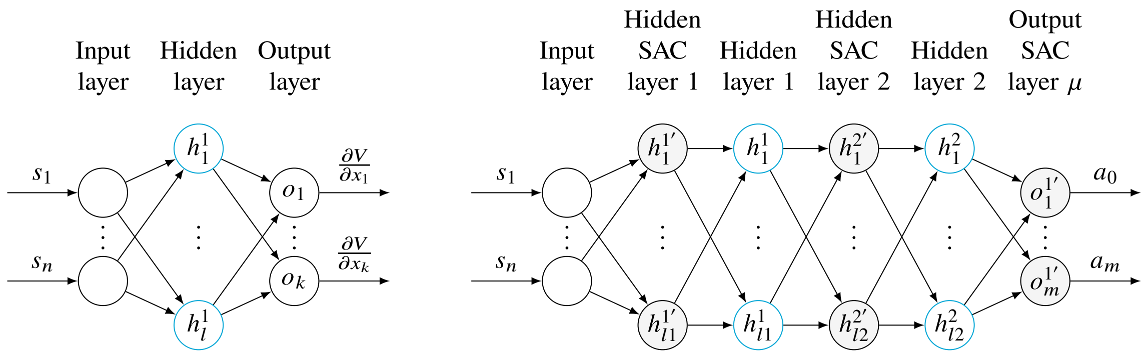
Fig. 3 SAC network architectures

2. Hybrid SAC-IDHP Network Architecture

The novelty of the hybrid SAC-IDHP controller developed in this research lies in large part in the network structure of the hybrid policy, as seen in Figure 4b. Contrary to a traditional policy network used in IDHP agents, the hybrid policy includes the pre-trained layers $h^{1'}$, $h^{2'}$ and $o^{1'}$ corresponding to h^1, h^2 and o^1 of the SAC policy. For the output layer, only the policy mean output of the SAC policy is used. The parameters of these pre-trained SAC layers are frozen during the IDHP learning process. Furthermore, the IDHP hidden units h^1 and h^2 do contain learnable parameters consisting of only weights and no bias. Similarly to the SAC policy, the activation function used on the additional hidden layers are ReLU functions, but no LayerNormalization is used in the IDHP neurons. Linear activation functions are used on the output neurons. The hybrid policy parameters vector thus only contains the weights of the hidden units h^1 and h^2.

By constructing the hybrid policy in this way, the goal is to maintain as much of the pre-learned information of the robust SAC policy during online learning. In order to achieve this, the learnable IDHP policy layers are initialized using the identity matrix as opposed to random initialization.

The critic on the other hand relates to a more traditional network structure used in Dual Heuristic Programming as seen in Figure 4a consisting of a single hidden layer. The IDHP critic accepts the RL-state as input and estimates the partial state derivative of the state value function $\frac{\partial V}{\partial \mathbf{x}}$ with k elements. The hidden units are constructed identically to the policy hidden units but with hyperbolic tangent activation functions, while the output neurons also have a linear activation function. The initialization of the critic weights is random as opposed to the actor and sampled from a truncated normal distribution with standard deviation σ_c.



(a) IDHP critic architecture **(b) Hybrid IDHP-SAC combined policy architecture**

Fig. 4 Hybrid IDHP-SAC network architectures

C. Attitude Controller

The attitude or inner control loop tracks reference signals for pitch, roll and sideslip angles as seen in Equation 29, and outputs the environment action vector. The tracking error vector can then be defined as in Equation 30 where P is a binary selection matrix defined in Equation 32 mapping the aircraft state to the tracked states. In order to keep the error signals in similar order of magnitude, the scaling vector from Equation 31 is determined by trial and error, where the sideslip signal receives a larger scale due to its lower overall magnitude resulting from the zero-sideslip hold task.

$$\mathbf{x}^{r^{att}} = [\theta^r, \phi^r, \beta^r]^T \tag{29}$$

$$\mathbf{x}_t^{e^{att}} = \mathbf{x}_t^{r^{att}} - P^{att}\mathbf{x}_{t+1} = [\theta^r - \theta, \phi^r - \phi, \beta^r - \beta]^T \tag{30}$$

$$\mathbf{x}^{c^{att}} = \frac{180}{\pi}\left[\frac{1}{30}, \frac{1}{30}, \frac{1}{7.5}\right]^T \tag{31}$$

$$P^{att} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{32}$$

1. *SAC Attitude Controller*

In the context of SAC, a reward function and RL-state vector are defined by Equation 33 and Equation 34 respectively. The reward function represents the negative of the L1 norm of the scaled error vector clipped on the $[-1, 1]$ interval. The RL-state vector consists of the scaled error vector to ensure good steady-state response and additionally the pitch, roll and yaw rates for improved transient response [26].

$$r_{t+1}^{SAC^{att}} = -\frac{1}{3}\left\|\text{clip}\left[\mathbf{x}_t^{e^{att}} \odot \mathbf{x}^{c^{att}}, \vec{\mathbf{-1}}, \vec{\mathbf{1}}\right]\right\|_1 \tag{33}$$

$$\mathbf{s}_{t+1}^{SAC^{att}} = \left[p, q, r, \left(\mathbf{x}_t^{e^{att}} \odot \mathbf{x}^{c^{att}}\right)^T\right]^T \tag{34}$$

2. *IDHP Attitude Controller*

The IDHP framework utilizes a reward function defined as the squared scaled error vector, defined in Equation 35. The RL-state for the IDHP critic is defined in Equation 36 and contains the three body rates, pitch, roll, sideslip and angle of attack angles and the scaled error vector. Note that the IDHP framework requires the state derivative of the reward function as defined in Equation 37 which can be derived directly using the definitions of the reward function and error vector.

Note that the RL-state vector for the IDHP actor is the same as the state vector for the SAC actor due to the hybrid policy architecture, and that the environment state vector for the incremental model excludes the altitude from the vector defined in Equation 27.

$$r_{t+1}^{IDHP} = -\left[\mathbf{x}_t^{e^{att}}\right]^T \left[diag\ \mathbf{x}^{c^{att}}\right] \left[\mathbf{x}_t^{e^{att}}\right] \quad (35)$$

$$\mathbf{s}_{t+1}^{IDHP} = \left[p, q, r, \alpha, \theta, \phi, \beta, \left(\mathbf{x}_t^{e^{att}} \odot \mathbf{x}^{c^{att}}\right)^T\right]^T \quad (36)$$

$$\frac{\partial r_{t+1}^{IDHP}}{\partial \mathbf{x}_{t+1}} = 2\left[\mathbf{x}_t^{e^{att}}\right]^T \left[diag\ \mathbf{x}^{c^{att}}\right] P^{att} \quad (37)$$

D. Altitude Controller

The altitude or outer control loop only tracks the altitude reference signal and outputs the reference signal for the pitch angle. The reference vector is defined in Equation 39 with the error vector in Equation 39 using the selection matrix from Equation 41. Also, the altitude error signal requires a scaling factor, as seen in Equation 40.

$$\mathbf{x}^{r^{alt}} = [h^r] \quad (38)$$

$$\mathbf{x}_t^{e^{alt}} = \mathbf{x}_t^{r^{alt}} - P^{alt}\mathbf{x}_{t+1} = [h^r - h,] \quad (39)$$

$$\mathbf{x}^{c^{alt}} = \left[\frac{1}{240}\right] \quad (40)$$

$$P^{alt} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (41)$$

1. SAC Altitude Controller

The outer control loop only implements the SAC algorithm with a reward function similar to the SAC attitude controller as seen in Equation 42. The RL-state vector defined in Equation 43 is simpler than the attitude controller, with only containing the scaled error vector. This is because only the kinematic relationship between the pitch angle and altitude has to be learned.

$$r_{t+1}^{SAC^{alt}} = -\left\|\text{clip}\left[\mathbf{x}_t^{e^{alt}} \odot \mathbf{x}^{c^{alt}}, \vec{\mathbf{-1}}, \vec{\mathbf{1}}\right]\right\|_1 \quad (42)$$

$$\mathbf{s}_{t+1}^{SAC^{alt}} = \left[\mathbf{x}_t^{e^{alt}} \odot \mathbf{x}^{c^{alt}}\right] \quad (43)$$

E. Hybrid SAC-IDHP Cascaded Controller

In Figure 5 a control loop diagram can be seen of the complete cascaded SAC-IDHP hybrid altitude-attitude controller. Only the inner attitude controller implements the hybrid architecture, as it is assumed the majority of dynamic relations that will change during failure modes are learned in the inner loop. Hence, an adaptive element is most useful in the inner loop and the outer loop is only trained offline with the SAC algorithm in order to limit the overall complexity. The dotted lines in the hybrid attitude controller represent the signal flow during online operation, where the IDHP attitude controller controls the aircraft and the SAC attitude controller only provides its pre-learned policy weights.

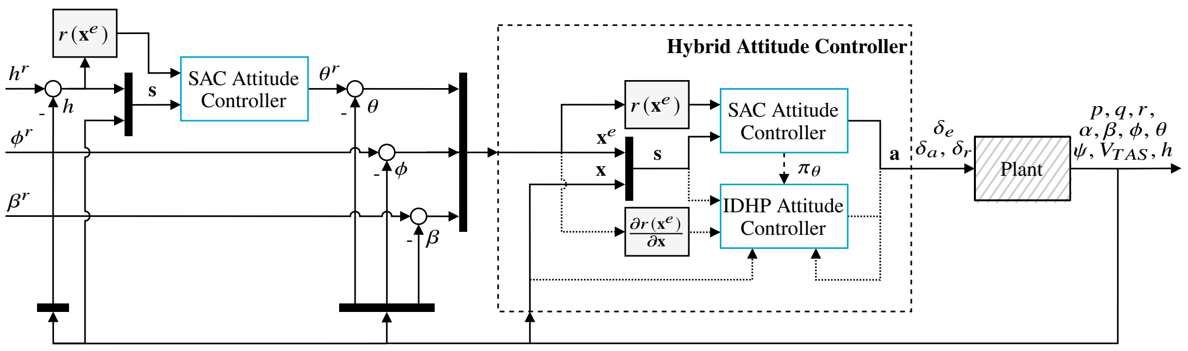


Fig. 5 SAC-IDHP Cascaded Altitude and Attitude Controller Structure

F. Training Strategy

The hybrid RL controller involves multiple stages of training. The SAC attitude and altitude controllers are trained offline, while the SAC-IDHP attitude controller requires an initial online training phase. The training strategy and hyperparameters used for all phases are discussed in this section.

1. Offline SAC Training

The SAC attitude agent is trained first using a maximum of 1000000 time steps. This corresponds to 500 $20s$ training episodes with a double step reference signal for the pitch and roll angles. The magnitude and sign of the step signals is uniformly sampled from $[20°, 10°]$ and $[40°, 20°]$ for the pitch and roll angles, respectively. After every training episode, the agent is evaluated using the same task, but without the randomized reference signal magnitudes and using the maximum values instead. The sideslip reference signal is always held at zero. A batch of at least 5 agents with differing random seeds is trained, where the best performing agent is selected to train the altitude controller.

The altitude controller follows the same training strategy with alternating climb, descend and altitude hold reference signals using $50m$ altitude differences. The roll and sideslip reference signals are equal to the attitude training task.

The hyperparameters used for both SAC controllers can be seen in Table 1 consisting of default values from the original SAC papers and empirically determined values. The parameters $\gamma, \tau, l1, l2, \eta_a, \eta_c, |\mathcal{B}|$ and η_0 are taken from the previously successful SAC implementation [26], while the maximum replay buffer size $|\mathcal{D}|$ has been increased compared to that paper as better learning stability was observed with a larger buffer size. The CAPS regularization coefficients λ_T and λ_S have been determined by trial and error, whereby a trade-off is made between increased smoothness of the policy's action, and decreased tracking performance with increasing coefficient values. The altitude controller requires smaller CAPS scaling coefficients, likely due to the decreased learning complexity of the outer altitude control. Note that the learning rates are linearly decreased to 0 over the total number of time steps.

In Figure 6 the reward curves for attitude and altitude controllers can be seen, showing the average and interquartile ranges over 5 random seeds of converged runs. The attitude and altitude controllers plateau at around -250 and -30 respectively, with little improvements after the initial 200000 time steps.

Compared to the equivalent training curves shown in previous research on the same system [26] the interquartile range of the current results is considerably smaller. This improved learning stability is assumed to be caused by the utilization of direct control and CAPS regularization as opposed to an incremental control approach in the previous research.

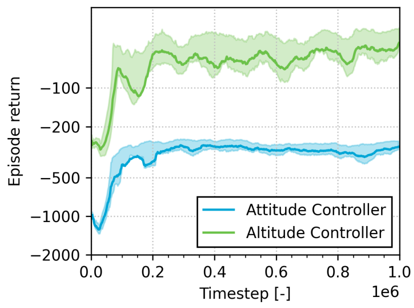


Fig. 6 SAC offline training curves of attitude and altitude controllers. Mean smoothed by a window of size 20 and the interquartile range over 5 random seeds are shown by the solid lines and shaded regions respectively.

Table 1 SAC Hyperparameters, adapted from [26] [27] [28]

Param.	**Value Attitude Agent**	**Value Altitude Agent**	**Description**
γ	0.99	0.99	Discount factor
τ	0.005	0.005	Target critic mixing factor
$l1, l2$	$[64, 64]$	$[32, 32]$	Actor/Critic hidden layer sizes
η_a, η_c	$4.4e-4$	$3.0e-4$	Actor/Critic initial learning rate
$\lvert\mathcal{B}\rvert$	256	256	Replay buffer mini-batch size
$\lvert\mathcal{D}\rvert$	1000000	1000000	Replay buffer maximum size
λ_T, λ_S	$400, 400$	$10, 10$	CAPS scaling coefficients
η_0	1.0	1.0	Initial temperature coefficient

2. Online IDHP Training

The IDHP framework requires initial excitation of all the environment states in order to successfully identify the incremental model. In previous IDHP-only frameworks [12] [13], an exponentially decaying sinusoidal excitation signal is added to the agent's action in order to excite the system during the initial training phase. An advantage of the hybrid framework is the presence of the pre-existing converged SAC policy, as obtained in subsubsection III.F.1 at the start of the online learning phase. Because of the identity initialization of the hybrid policy, the initial excitation can be provided by the SAC policy, driven by the reference signals without the use of additional excitation signals. The hybrid attitude controller thus requires an initial online training task using in this case sinusoidal reference signals on the pitch, roll and sideslip angles. The reference signal for the sideslip is decaying in order to better preserve aircraft stability. The IDHP specific hyperparameters used are seen in Table 2, while the SAC portion only involves inference and no updates to the SAC layers are performed during online learning. The IDHP hyperparameters are taken from previous IDHP-only configurations [12] with the exception of the layer size l and learning rates η_a and η_c. The hidden layer size is empirically determined at a small value for increased learning stability, but without observing significant loss in learning ability. The learning rates are determined using trial and error by increasing learning rates until noticeable oscillations or instability appears in the training tasks.

The response on the training task can be seen in Figure 7 where also the SAC-only response is shown. The progression of the actor/critic weights and the incremental model parameters can be seen in Figure 8. Note that weights of identity initialized networks remain near the identity matrix [35], hence the actor weights seen in the parameter plots are plotted separately around 1 and 0. The parameters' progression demonstrates the hybrid method can converge on the training task, with the actor-critic weights and the parameter matrices of the incremental model all converging. The F and G matrices take approximately 8s to converge, while the critic weights are stable after 30s. The actor weights are converging more slowly, but have reached the converging range within the 60s training task.

The nMAE metric as later defined in section IV can already be used here to compare the initial tracking performance with SAC-only. With an nMAE of 7.41% for the hybrid and 10.31% for SAC-only, a noticeable improvement of 2.9% in tracking performance is already demonstrated. Looking at the visual tracking response, after approximately 20s, the hybrid agent has successfully corrected for the steady-state error of the SAC policy with a smaller tracking error near the sinusoidal peaks of the pitch and roll reference signals. The sideslip tracking performance shows little difference, as the SAC policy is already providing a low error. Note that before 20s, the initial converge phase results in temporary divergence from the tracking signal, hence the importance of a controlled training task before performing flight manoeuvre tasks. Also note that the airspeed tracking is handled by the auto-throttle from the DASMAT model and is not managed by the RL-agent.

The resulting weights and parameters are stored and used as initial condition for the hybrid attitude agent in the following flight manoeuvre demonstrations.

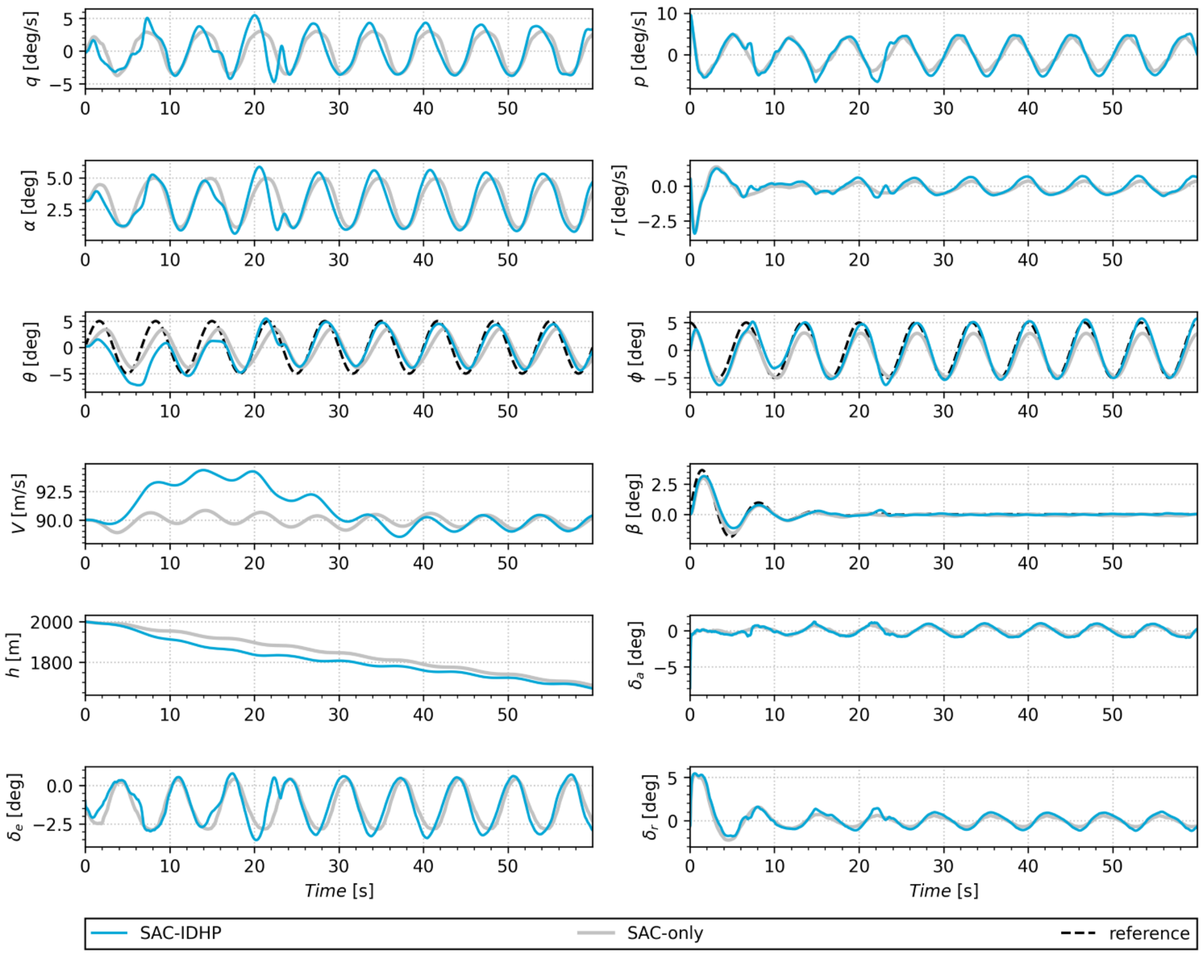


Fig. 7 SAC-IDHP online training response of attitude controller. nMAE = 7.41% for SAC-IDHP and 10.31% for SAC-only.

Table 2 IDHP Hyperparameters, adapted from [12] [13]

Parameter	**Value**	**Description**
γ	0.8	Discount factor
τ	0.01	Target critic mixing factor
l	8	Critic hidden layer size
η_a, η_c	0.2, 1.0	Actor/Critic learning rate
κ	1.0	Incremental model forgetting factor
Λ_0	$1 \cdot 10^8$	Initial covariance matrix magnitude

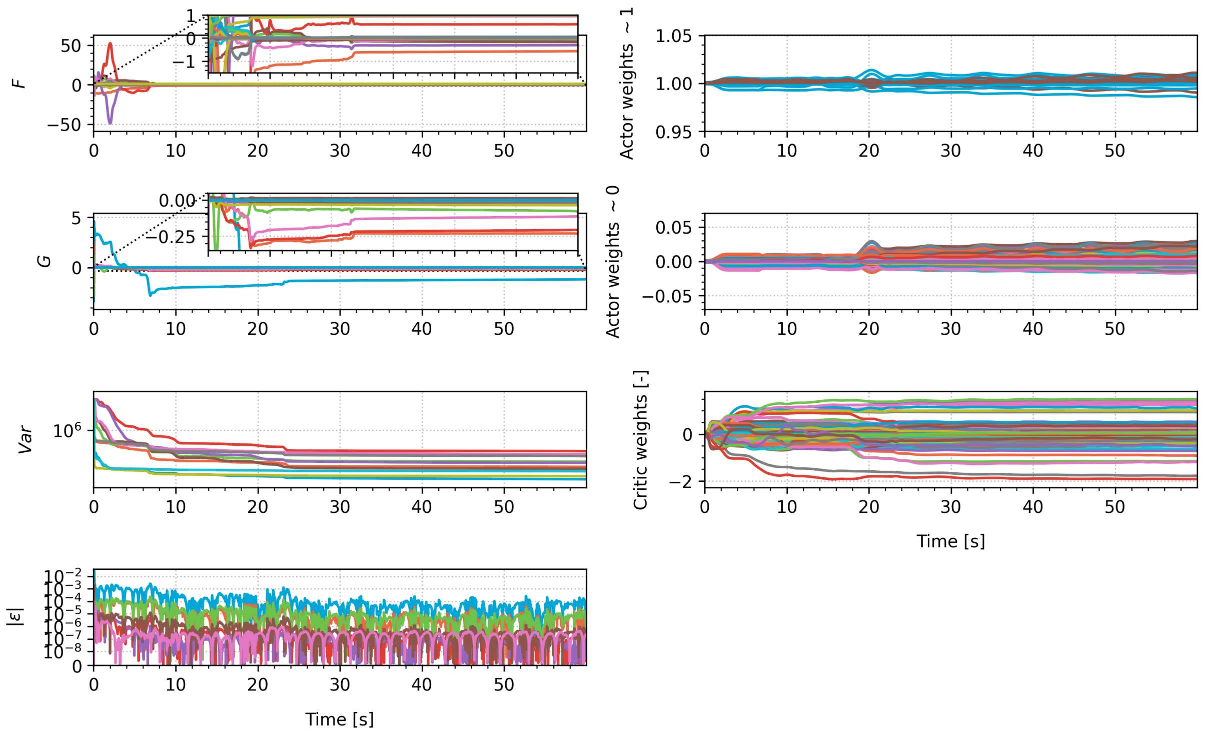


Fig. 8 SAC-IDHP online training of attitude controller, actor/critic weights and incremental model parameters.

IV. Results and Discussion

The response of the Hybrid controller and the SAC-only controller are compared on the simulation model of the Cessna Citation jet aircraft. First, the aircraft in nominal state is evaluated after which several aircraft failures are simulated and the performance between SAC-IDHP and SAC-only is compared. All evaluation runs are performed using an initial condition of $h = 2000m$ and $V = 90\frac{m}{s}$. Additional results concerning varying initial flight conditions, critic initialization and sensor noise are discussed in subsubsection IV.C.1.

A performance metric used in addressing tracking performance is the normalized Mean Absolute Error (nMAE) averaged over externally tracked states which are altitude, roll and sideslip angles for the altitude tasks, and pitch, roll and sideslip angles for the attitude tasks. The normalization is done over the maximum reference signal range with the exception of the sideslip angle, where the normalization range is set at $[-5°, 5°]$ as its reference signal is always 0.

A. Nominal System

The proposed flight controller should be able to control the aircraft in nominal condition without failures. This section presents the control response on the altitude task by comparing the performance of the SAC-only controller against the hybrid SAC-IDHP controller.

In Figure 9 the response on the altitude task can be seen. The external reference signal for the altitude is set at a steady climb and descend over 250m with a 15s hold in between. The bank angle reference is set at alternating 20° and 40° turns motivated by CS-25 specifications for nominal coordinated turns [36]. The sideslip reference is set at zero per definition of a coordinated turn. The pitch angle reference signal shown in the response plots is generated by the SAC outer loop controller based on the altitude error.

Comparing tracking performance, the SAC-only agent achieves a nMAE of 2.77% and the hybrid 2.03% showing a small improvement of 0.74%. Most notably, the sideslip angle has reduced peaks, but the longitudinal states show increased oscillatory behaviour in the hybrid response while keeping closer to the reference signal. The SAC agent shows similar tracking performance to previously developed SAC controllers on the same system [26]. This shows that both SAC-only and the hybrid agent have satisfactory tracking performance on the nominal altitude tracking task.

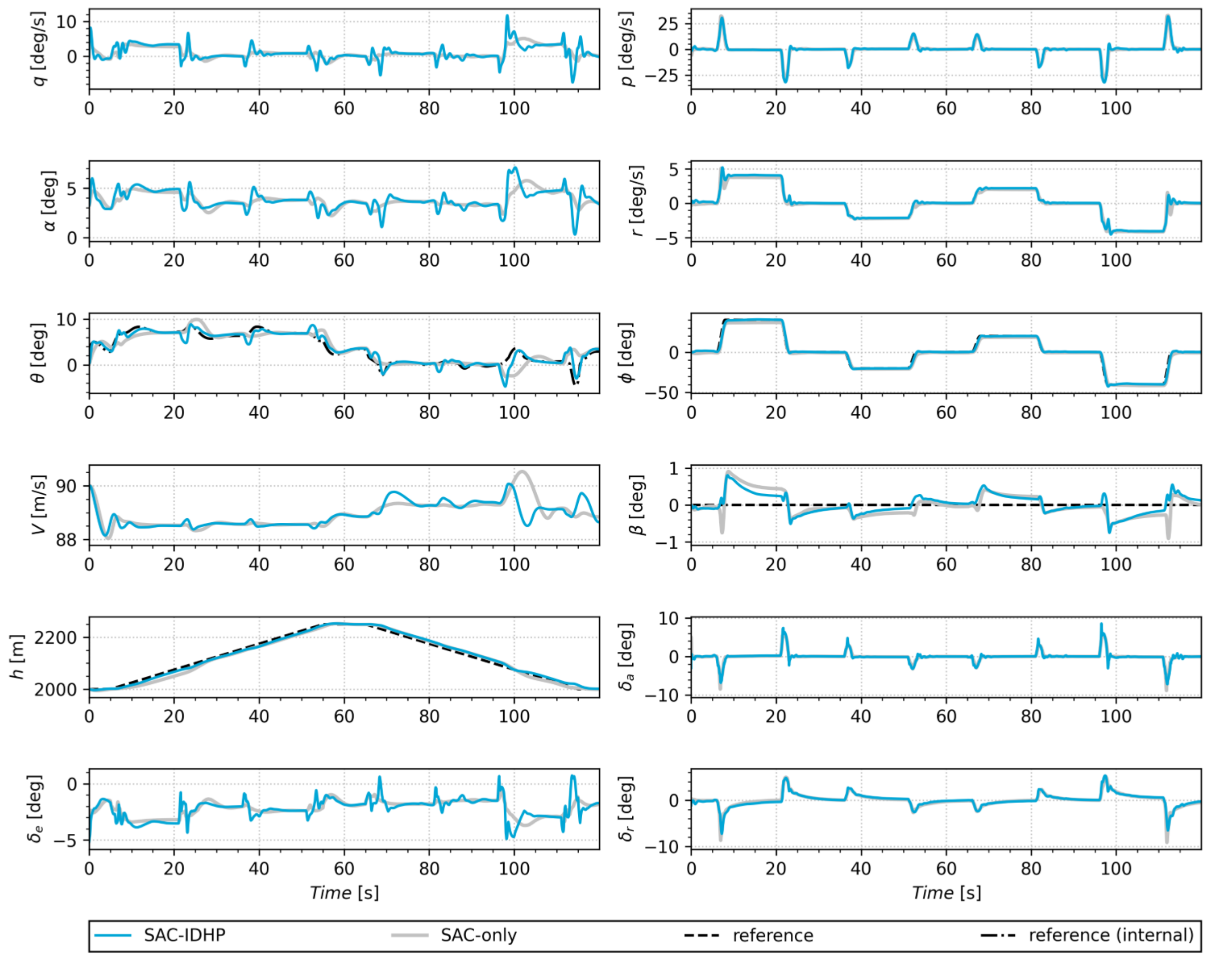


Fig. 9 Altitude tracking response on nominal system. SAC-IDHP and SAC-only compared. nMAE = 2.03% for SAC-IDHP and 2.77% for SAC-only.

B. Failed System

The two most important failure cases tested in this section are the reduced effectiveness of the elevator and ailerons to demonstrate longitudinal and lateral failures respectively. The same reference signals are used as for the nominal case except for the maximum bank angle which is set at 20°.

1. Reduced Elevator Effectiveness

In Figure 10 the response of 70% reduced elevator effectiveness at t=30s can be seen. The SAC agent achieves a nMAE of 7.99% while the hybrid agent maintains an nMAE of 2.53%, an improvement of 5.46%. This shows the hybrid policy is successful in correcting for performance degradation present in the robust response of the SAC policy. Looking at the response, the SAC agent remains stable, but with a considerable tracking error on the altitude due to the heavily reduced elevator effectiveness. The response of the hybrid agent remains close to the altitude reference while also improving on a small steady state error appearing on the roll angle for SAC. Overall, the hybrid agent shows greatly improved tracking performance, but with slightly increased oscillations mainly in the longitudinal states.

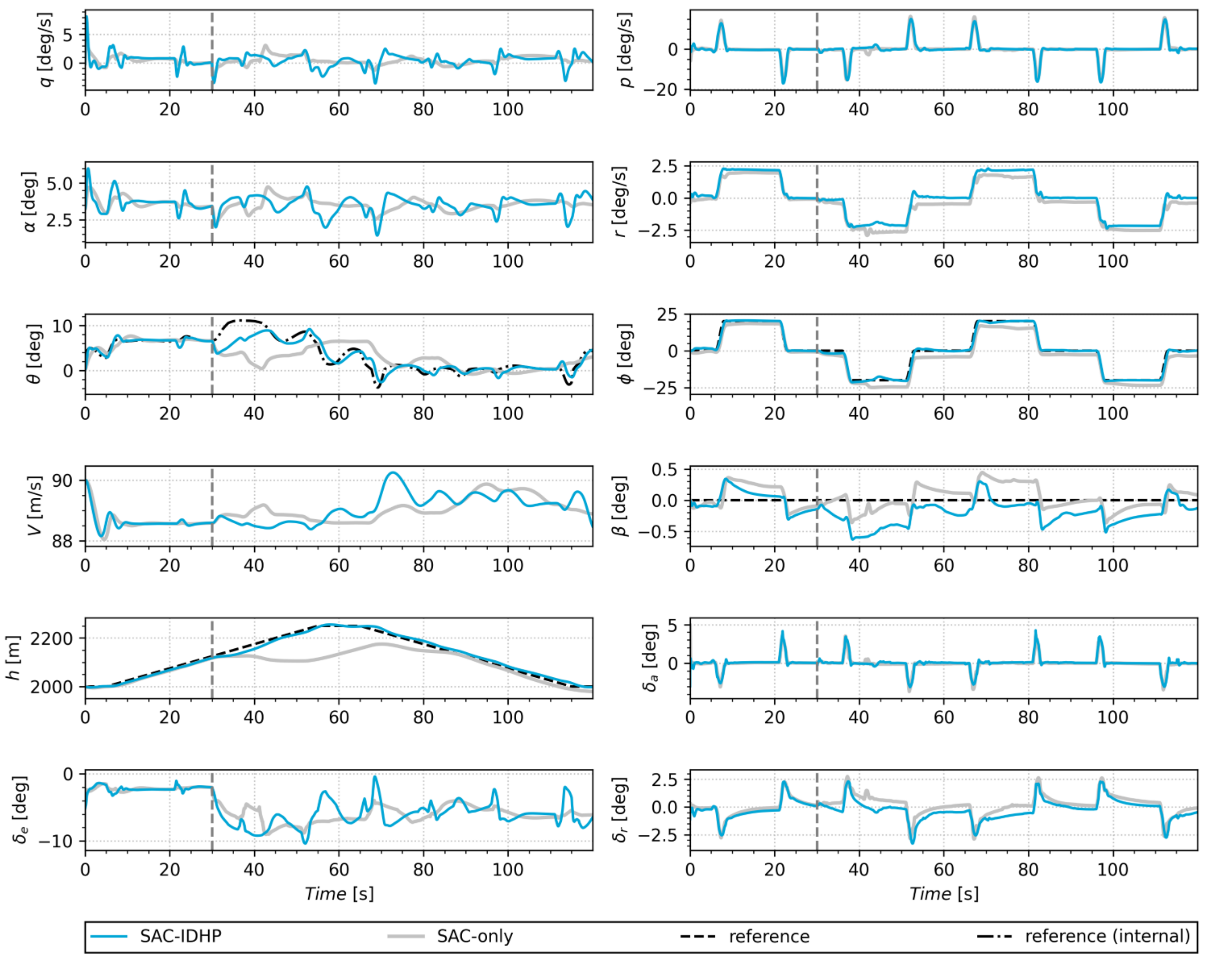


Fig. 10 Altitude tracking response on system with 70% reduced elevator effectiveness from t=30s. SAC-IDHP and SAC-only compared. nMAE = 2.53% for SAC-IDHP and 7.99% for SAC-only.

2. Reduced Aileron Effectiveness

In Figure 11 the response to 90% reduced aileron effectiveness from t=30s can be seen. Comparing tracking error between SAC and SAC-IDHP, the respective nMAE of 3.28% and 2.46% only show a 0.82% improvement for the hybrid agent. This is in line with the nominal case as can also be seen from visual inspection. The robust response of the SAC policy successfully corrects for the reduced elevator effectiveness by increasing the maximum aileron deflection from approximately 4° to 26° and keeping the tracking error small. Nonetheless, the hybrid agent still provides small improvements in rise time of the bank angle, and keeping slightly closer to the zero sideslip reference. Note that for the hybrid agent, increased oscillations are noticeably prevalent in the longitudinal states.

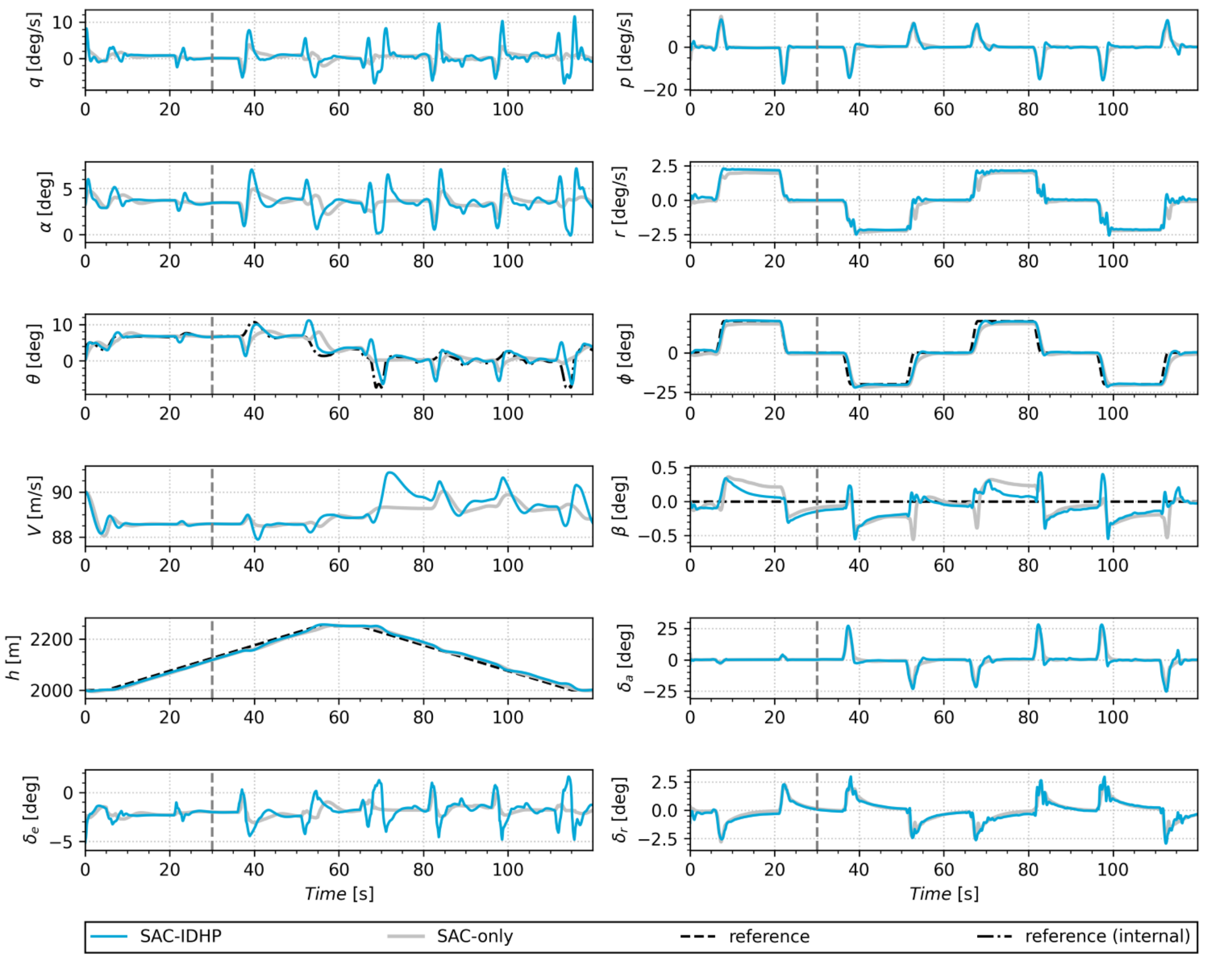


Fig. 11 Altitude tracking response on system with 90% reduced aileron effectiveness from t=30s. SAC-IDHP and SAC-only compared. nMAE = 2.46% for SAC-IDHP and 3.28% for SAC-only.

C. Additional Results

Additional experiments are performed in order to judge the robustness and reliability of the hybrid SAC-IDHP controller compared to a SAC-only controller. Robustness to varying initial flight condition, biased sensor noise and random critic initialization are explored.

1. Robustness to Initial Flight Condition

The SAC offline and the hybrid online training tasks are all performed on an initial condition of $h_0 = 2000m$ and $V_0 = 90\frac{m}{s}$. This section explores variability in tracking performance with changing initial conditions different from the training conditions, all performed on the altitude tracking task with 20° maximum bank angle.

In Table 3 the nMAE for 4 different flight conditions can be seen with FC2 being the nominal condition. The flight conditions are in order of increasing dynamic pressure. It can be seen that tracking error and error variability across the flight conditions is lower for the hybrid method. For FC1 with the lowest dynamic pressure, the SAC agent has the largest error with decreasing error with increasing dynamic pressure, except that FC4 has a slightly higher error than FC2. This pattern is not present for the hybrid method, which has a considerably lower variance and stays within 0.27% nMAE. Note that the scope of this analysis excludes variability over multiple SAC reference policies and IDHP training seeds, the latter being discussed independently in section IV.C.3.

Table 3 Robustness to initial flight conditions of cascaded altitude controllers.

Flight Condition	Initial Altitude [m]	Initial Airspeed [m/s]	nMAE SAC-only	nMAE SAC-IDHP
FC1	5000	90	4.71%	1.96%
FC2 (nominal)	2000	90	2.76%	2.02%
FC3	5000	140	2.05%	1.75%
FC4	2000	140	2.27%	2.01%

2. Biased Sensor Noise

Sensor noise is an essential element in assessing a closer to real-world environment. Biased sensor noise is applied to the observed state using measurement values derived from the PH-LAB aircraft [32] as seen in Table 4. It was noticed during initial testing that the incremental model identification of the IDHP framework produces inconsistent results when high frequency oscillation are present in the states. Hence, a low-pass filter with $\omega_0 = 40deg$ is applied to the observation for the IDHP update rule, with an equivalent filter applied to the SAC-only update rules for a fair comparison. The exact nominal altitude task from subsection IV.A is used, with a resulting nMAE of 2.69% for SAC-only and 2.00% for the hybrid agent. This corresponds to a respective 0.08% and 0.03% reduction in nMAE compared to the case without noise, attributed to the bias having a positive effect on the error, but also indicating both controllers maintain performance in the presence of sensor noise. Note however that the hybrid agent suffers from increased oscillations compared to SAC-only and the case without sensor noise, but shows to have no negative effect on tracking performance. Again, the scope of this analysis excludes variability over IDHP training seeds and SAC reference policies.

Table 4 Cessna Citation PH-LAB sensor noise characteristics [32]

State	Bias μ	Variance σ^2
p, q, r $[rad/s]$	$3.0 \cdot 10^{-5}$	$4.0 \cdot 10^{-7}$
θ, ϕ $[rad]$	$4.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-9}$
β $[rad]$	$1.8 \cdot 10^{-3}$	$7.5 \cdot 10^{-8}$
h $[m]$	$8.0 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$

3. Sensitivity to Random Critic Initialization

Compared to IDHP-only frameworks, the hybrid framework has a lesser degree online random initialization because of the identity initialization of the IDHP actor layers. The online critic however is still initialized by sampling from a truncated normal distribution with zero mean and a standard deviation σ_c as seen in Equation 44. The effect of this random initialization and the effect of varying standard deviations is evaluated in this section.

All training runs are performed using same hyperparameters from Table 2. A total of 150 runs are performed, 50 for each of three different standard deviations.

An additional metric is used in order to arrive at a total success rate per batch. Relating to the CAPS temporal loss function used in the SAC offline training from Equation 8, a temporal loss metric is calculated per training run according to Equation 45. The success threshold for the temporal loss metric is set at $L_T \leq 0.01$.

$$w \sim \mathcal{N}_{trunc}(\mu = 0, \sigma = \sigma_c) \qquad (44) \qquad L_T = D(\pi(\mathbf{s}_t), \pi(\mathbf{s}_{t+1})) = \sum_{t=1}^{N} ||\mathbf{a}_{t-1} - \mathbf{a}_t)||_2 \qquad (45)$$

Looking at the results in Table 5, the success rates for three critic standard deviations can be seen with $\sigma_c = 0.05$ the nominal case. The convergence rate tracks the number of runs that become unstable in the parameters, while the total success rate combines the convergence and temporal metrics. A lower standard deviation results in a more stable training experience, indicated by the 100% success rate of a lower standard deviation compared to the nominal case. A higher standard deviation then results in a lower success rate.

To show the effect of random critic initialization on the response, the interquartile range of the states over 50 runs of the nominal case is presented in Figure 12. Only the successful runs are presented meaning 72% or 36 runs. This shows the level of variation the random seed causes during the training task. This shows a satisfactory and safe training response with the states remaining close to the reference signal, even during the initial convergence period before t=20s, but keeping in mind the success rate.

Due to the hybrid policy design and identity initialization of online learning policy layers, it is proposed that a failed online training run can safely be reset due to the presence of the pre-trained SAC policy layers. This is in contrast with IDHP-only methods which start with zero knowledge at the start of the training phase.

Table 5 Success rates on SAC-IDHP attitude training task with varying random critic initialization. Analysed over 50 runs per configuration.

Network Initialization	Convergence Rate	Temporal Loss Rate	Total Success Rate
$\sigma_c = 0.01$	100.0%	100.0%	100.0%
$\sigma_c = 0.05$ (nominal)	92.00%	78.26%	72.00%
$\sigma_c = 0.10$	68.00%	70.59%	48.00%

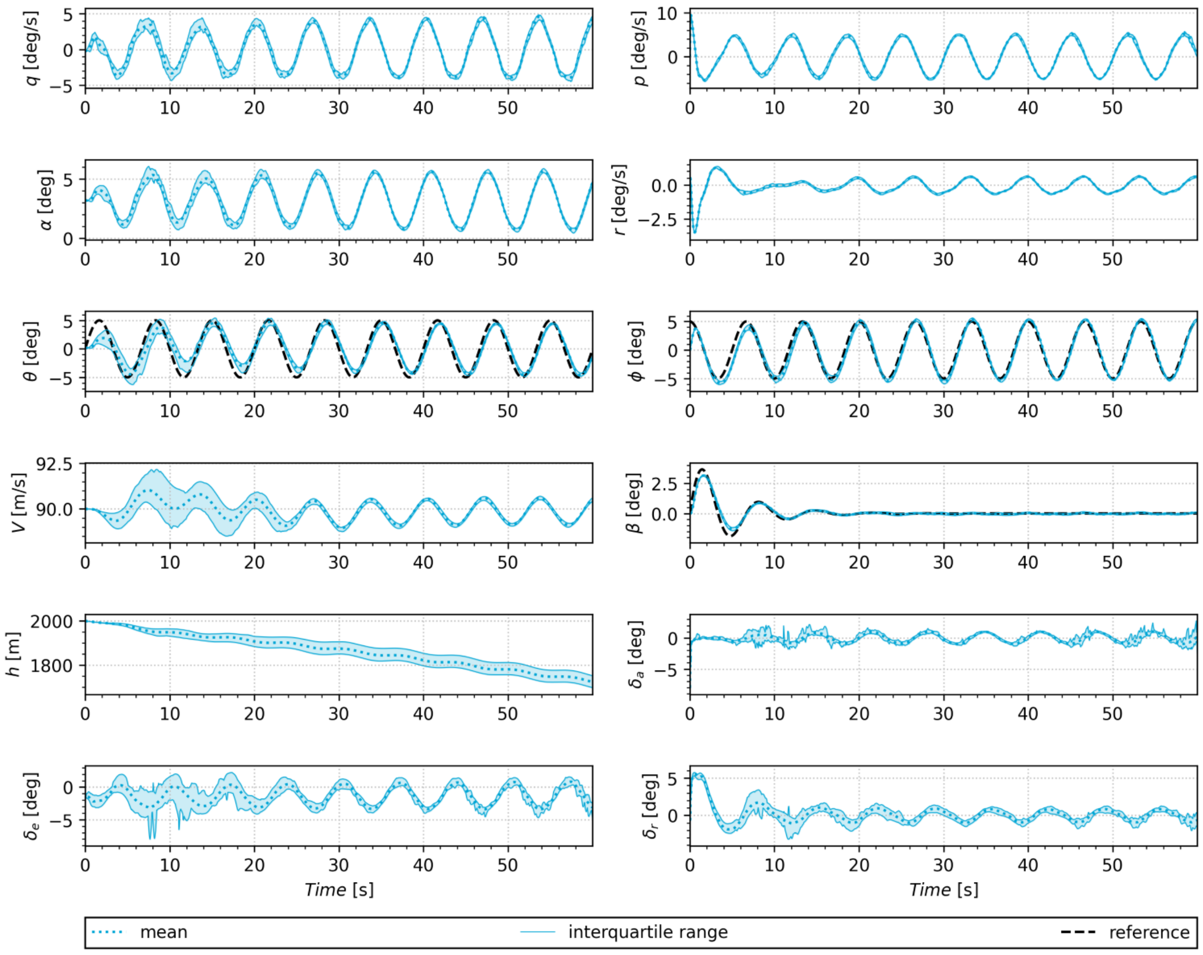


Fig. 12 SAC-IDHP response on attitude training task with $\sigma_c = 0.05$ over 36 successful runs.

V. Conclusion

It is demonstrated that a hybrid SAC-IDHP offline-online learning controller can be successfully implemented and provide coupled-dynamics fault-tolerant flight control in a complete RL-based cascaded altitude controller. It is shown that compared to SAC-only, the online learning of the hybrid policy architecture provides more adaptive control with lower tracking error across all tested nominal and failure cases. An improvement in nMAE of 0.74%, 5.46% and 0.82% is demonstrated for nominal case, longitudinal and lateral failure cases respectively. Compared to IDHP-only, random initialization of the actor is removed by the hybrid policy. It is proposed that this provides the ability to revert to a robust response only and reset online IDHP learning safely in flight due to the presence of robust pre-trained policy layers. Additionally, the hybrid architecture provides increased confidence in including IDHP into a fully coupled-dynamics 6-degree-of-freedom control loop.

The SAC-IDHP agent however exhibits increased oscillations mainly in the longitudinal states, most noticeable when adding biased sensor noise, but still providing lower tracking error compared to SAC-only. Challenges with offline SAC learning remain, including inconsistent training performance due to the many stochastic factors, but is improved by the use of CAPS regularization. Further research is recommended into comparison with IDHP-only and the safety of SAC and IDHP in covering the entire flight envelope before executing flight tests on the PH-LAB aircraft.

References

[1] Cokorilo, O., "Urban Air Mobility: Safety Challenges," *Transportation Research Procedia*, Vol. 45, 2020, pp. 21–29. https://doi.org/10.1016/j.trpro.2020.02.058.

[2] "Loss of Control In-Flight Accident Analysis Report 2019 Edition," *International Air Transport Association*, 2019, p. 44.

[3] Balas, G. J., "Flight Control Law Design: An Industry Perspective," *European Journal of Control*, Vol. 9, No. 2, 2003, pp. 207–226. https://doi.org/10.3166/ejc.9.207-226.

[4] Richard S. Sutton, and Andrew G. Barto, *Reinforcement Learning, Second Edition : An Introduction*, Adaptive Computation and Machine Learning, Vol. Second edition, Bradford Books, Cambridge, Massachusetts, 2018.

[5] Si, J., Barto, A. G., Powell, W. B., and Wunsch, D., *Handbook of Learning and Approximate Dynamic Programming*, Wiley-IEEE Press, 2004. https://doi.org/10.1109/9780470544785.

[6] Wang, F., Zhang, H., and Liu, D., "Adaptive Dynamic Programming: An Introduction," *IEEE Computational Intelligence Magazine*, 2009. https://doi.org/10.1109/MCI.2009.932261.

[7] Liu, D., Xue, S., Zhao, B., Luo, B., and Wei, Q., "Adaptive Dynamic Programming for Control: A Survey and Recent Advances," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 51, No. 1, 2021, pp. 142–160. https://doi.org/10.1109/TSMC.2020.3042876.

[8] 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. https://doi.org/10.2514/1.12597.

[9] 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. https://doi.org/10.1109/TNN.2003.813839.

[10] Zhou, Y., Van Kampen, E.-J., and Chu, Q., "Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control," *IMAV 2016*, Delft University of Technology, 2016.

[11] 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. https://doi.org/10.1016/j.conengprac.2017.12.011.

[12] 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*, AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, 2020. https://doi.org/10.2514/6.2020-1844.

[13] Kroezen, D., "Online Reinforcement Learning for Flight Control: An Adaptive Critic Design without Prior Model Knowledge," Master's thesis, Delft Unifersity of Technology, 2019.

[14] Lee, J., "Longitudinal Flight Control by Reinforcement Learning: Online Adaptive Critic Design Approach to Altitude Control," Master Thesis, Delft University of Technology, Delft, 2019.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-Level Control through Deep Reinforcement Learning," *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533. https://doi.org/10.1038/nature14236.

[16] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D., "Mastering the Game of Go without Human Knowledge," *Nature*, Vol. 550, No. 7676, 2017, pp. 354–359. https://doi.org/10.1038/nature24270.

[17] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P., "Trust Region Policy Optimization," *arXiv:1502.05477 [cs]*, 2015.

[18] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous Control with Deep Reinforcement Learning," *arXiv:1509.02971 [cs, stat]*, 2019.

[19] 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)*, 2019, pp. 523–533. https://doi.org/10.1109/ICUAS.2019.8798254.

[20] Wang, Z., Li, H., Wu, Z., and Wu, H., "A Pretrained Proximal Policy Optimization Algorithm with Reward Shaping for Aircraft Guidance to a Moving Destination in Three-Dimensional Continuous Space," *International Journal of Advanced Robotic Systems*, Vol. 18, No. 1, 2021, p. 1729881421989546. https://doi.org/10.1177/1729881421989546.

[21] Shehab, M., Zaghloul, A., and El-Badawy, A., "Low-Level Control of a Quadrotor Using Twin Delayed Deep Deterministic Policy Gradient (TD3)," *2021 18th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, 2021, pp. 1–6. https://doi.org/10.1109/CCE53527.2021.9633086.

[22] He, L., Aouf, N., Whidborne, J. F., and Song, B., "Deep Reinforcement Learning Based Local Planner for UAV Obstacle Avoidance Using Demonstration Data," *arXiv:2008.02521 [cs]*, 2020.

[23] Cheng, Y., and Song, Y., "Autonomous Decision-Making Generation of UAV Based on Soft Actor-Critic Algorithm," *2020 39th Chinese Control Conference (CCC)*, 2020, pp. 7350–7355. https://doi.org/10.23919/CCC50068.2020.9188886.

[24] Lee, M. H., and Moon, J., "Deep Reinforcement Learning-based UAV Navigation and Control: A Soft Actor-Critic with Hindsight Experience Replay Approach," *arXiv:2106.01016 [cs, eess]*, 2021.

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

[26] Dally, K., and Van Kampen, E.-J., "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control," *AIAA SCITECH 2022 Forum*, AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, 2021. https://doi.org/10.2514/6.2022-2078.

[27] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *arXiv:1801.01290 [cs, stat]*, 2018.

[28] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S., "Soft Actor-Critic Algorithms and Applications," *arXiv:1812.05905 [cs, stat]*, 2019.

[29] Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K., "Regularizing Action Policies for Smooth Control with Reinforcement Learning," , May 2021. https://doi.org/10.48550/arXiv.2012.06644.

[30] Sun, B., and van Kampen, E.-J., "Incremental Model-Based Global Dual Heuristic Programming with Explicit Analytical Calculations Applied to Flight Control," *Engineering Applications of Artificial Intelligence*, Vol. 89, 2020, p. 103425. https://doi.org/10.1016/j.engappai.2019.103425.

[31] 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, 2017, pp. 493–496. https://doi.org/10.2514/1.G001762.

[32] 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*, American Institute of Aeronautics and Astronautics, Kissimmee, Florida, 2018. https://doi.org/10.2514/6.2018-0385.

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

[34] Ba, J. L., Kiros, J. R., and Hinton, G. E., “Layer Normalization,” , Jul. 2016. https://doi.org/10.48550/arXiv.1607.06450.

[35] Kubota, S., Hayashi, H., Hayase, T., and Uchida, S., “Layer-Wise Interpretation of Deep Neural Networks Using Identity Initialization,” *arXiv:2102.13333 [cs]*, 2021.

[36] EASA, “CS-25 Amendment 27 - Review of Aeroplane Performance Requirements for Air Operations and Regular Update of CS-25,” https://www.easa.europa.eu/document-library/certification-specifications/cs-25-amendment-27, Jun. 2021.

II

Preliminary Research

This Part has previously been graded for AE4020

3 Literature Review

This chapter goes over the fundamental topics from literature which form the basis for understanding this research. Additionally, state-of-the-art applications are discussed. In section 3.1, the fundamentals of Reinforcement Learning (RL) are given including the theory necessary to understand and evaluate different RL frameworks and their applicability to the PH-LAB flight control task. Next, section 3.2 goes over methods to use RL in continuous state and action spaces like the PH-LAB flight control problem. This section discusses extensions of the fundamental methods in order to cope with the so called curse of dimensionality [62] that comes with using RL in continuous spaces. Two classes of continuous space RL algorithms are discussed in more detail in the next two sections. First, section 3.3 goes over Approximate Dynamic Programming and section 3.4 details Deep Reinforcement Learning. Both are considered state-of-the-art approaches in continuous RL control. The methods are discussed in detail and state-of-the-art applications on these methods are documented to help answer research question *RQ1.1*. Finally, section 3.5 is dedicated to specific challenges with closing the reality-gap and answers research question *RQ1.2*.

3.1. Reinforcement Learning Fundamentals

First of all, an overview of the fundamentals of RL is given. This section serves as an introduction to RL and presents the relevant terminology, key concepts and theoretical background of the RL problem and goes over commonly used methods to solve it in discrete spaces.

3.1.1. The Agent–Environment Interface

The essence of RL is learning from interaction, inspired by biological brains that can learn by trial and error. There are usually two main entities and their interaction that represent an RL framework, of which a general depiction can be seen in Figure 3.1. The *agent* (also called *controller*) is the learner and decision-maker and interacts with the second entity called the *environment* (also called *plant*). The interface through which the continual interaction between agent and environment happens can be defined by *actions* (also called *control signal*), *states* or *observations* and *rewards*. The action a is a decision made by the agent on the basis of the observation s of the environment. Note the distinction between the environment state and observation which are often used interchangeably in the literature. The state vector is defined as the complete state of the environment used to define its dynamics. While the observation vector can equal the state, it often contains only a subset of the state or is augmented with task-dependent elements like error vectors. In this chapter, state and observation are used interchangeably. In chapter 4 however, the distinction is important to make when discussing lower-level implementation details. Every iteration, the agent is then given feedback by the environment in the form of a reward r as a consequence of the action taken. This feedback plays a large role in the learning process of an RL framework as it traditionally aims to maximize the reward over time to achieve a certain goal.

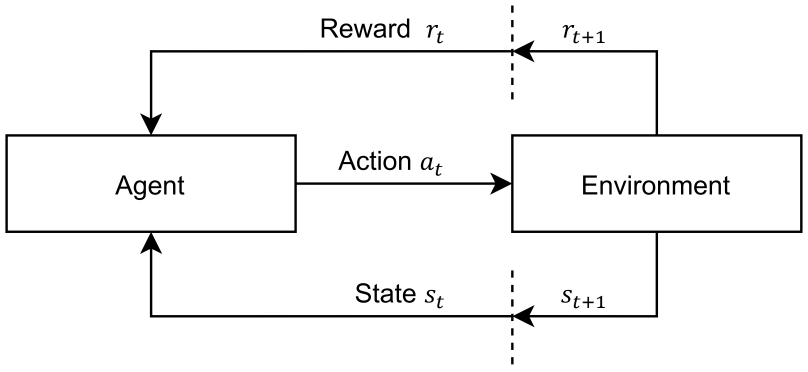


Figure 3.1: The fundamental agent–environment interface of RL frameworks

3.1.2. Markov Decision Processes

The finite Markov Decision Process (MDP) is the mathematically idealised framework that describes a stochastic sequential decision-making process and thus represents the idealised form of a reinforcement learning problem. In finite MDPs, the actions, states and rewards are part of finite sets, $\mathcal{A}(s)$, $\mathcal{S}$ and $\mathcal{R}$ respectively and the agent interacts with the environment in a sequence of discrete time steps $t = 0, 1, 2, \dots$. At time step t, the agent receives the environment's state s_t which leads to the agent selecting an action a_t. The next time step, the agent receives feedback from the environment in the form of a real-valued reward $r_{t+1} \in \mathbb{R}$ and observes the new state s_{t+1}. The resulting sequence or *trajectory* of state, action and reward then looks as follows: $s_0, a_0, r_1, s_1, a_1, \dots$. The probability of a certain state and reward happening as the result of the previous state and action taken can be expressed by Equation 3.1 where the function p essentially defines the dynamics of the MDP. [62]

$$p(s', r \mid s, a) \doteq \mathcal{P}\{s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a\} \qquad \begin{matrix} \forall s, s' \in \mathcal{S} \\ \forall a \in \mathcal{A}(s) \\ \forall r \in \mathcal{R} \end{matrix} \tag{3.1}$$

The MDP has the Markov Property, meaning that the current state and action carry all the information to predict the next state. Formally this property is defined by Equation 3.2 and allows for the probability distribution functions to be written only as a function of the previous states instead of all preceding states up to that point. Note that for an RL solution to an MDP to be feasible, enough relevant state variables need to be available to the agent. [62]

$$\mathcal{P}\{s_{t+1}, r_{t+1} \mid s_t, a_t\} = \mathcal{P}\{s_{t+1}, r_{t+1} \mid s_t, a_t, \dots, s_0, a_0\} \tag{3.2}$$

In addition to the dynamics function p which fully describes the environment, any other relevant probability can be defined. The state-transition function is defined by Equation 3.3 and the expected reward function is expressed by either the two-argument function as in Equation 3.4 or the three-argument function as in Equation 3.5. [62]

$$p(s' \mid s, a) \doteq \mathcal{P}\{s_{t+1} = s' \mid s_t = s, a_t = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{3.3}$$

$$r(s, a) \doteq \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{3.4}$$

$$r(s, a, s') \doteq \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)} \tag{3.5}$$

3.1.3. Rewards and Returns

The purpose of the agent is to maximize the reward over time. This does not mean maximizing the immediate reward on every time step, but the cumulative reward over the entire duration of the process. This cumulative reward can be defined formally by the return G_t as seen in Equation 3.6 which includes the entire sequence of rewards received after time step t. The return contains the discount rate $\gamma \in [0, 1]$ which determines the present value of future rewards and takes future rewards more into account if $\gamma \rightarrow 1$. When $\gamma = 0$, the agent is called "myopic" as it is only concerned with immediate reward. As opposed to periodic tasks, continuous tasks are infinite in time hence $T \rightarrow \infty$. This also means $\gamma < 1$ is required to get a finite value for the discounted return. [62]

$$G_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{3.6}$$

In order to understand coming derivations like the Bellman equation, it is important to see that the discounted return can easily be written in recursive form relating the current return to the future return as seen in Equation 3.7. [62]

$$G_t = r_{t+1} + \gamma G_{t+1} \tag{3.7}$$

3.1.4. Policy and Value Functions

The strategy that is deployed by the agent is called the *policy* and is formally defined by Equation 3.8. The policy is the law that decides what action to take in every state and can be seen as a mapping from state to the probability of selecting an action [62].

Note that the policy $\pi(a \mid s)$ as described by Equation 3.8 represents a *stochastic policy* as the action is sampled from a probability distribution. When reducing the variance of this distribution to the limit, the distribution becomes a Dirac delta function and the policy becomes a *deterministic policy* $\pi(s)$. A deterministic policy has only one unique action per state and is a direct mapping from state to action. This distinction is important when discussing policy-based optimization methods in section 3.2. [13]

$$\pi(a \mid s) \doteq \mathcal{P}\{a_t = a \mid s_t = s\} \tag{3.8}$$

Most RL algorithms involve estimating *value functions*. The state-value function v_π estimates how good it is to be in a given state or the value of being in a certain state s. This is expressed in terms of the expected future return as this is the metric that the agent aims to maximize. Hence the state-value function is defined as the expectation of the return under a certain policy π as can be seen in Equation 3.9.

The action-value function q_π defined by Equation 3.10 is similar to the state-value function but it assumes a state-action pair as starting position. This means the action-value function gives the value of taking the action a starting from an initial state s following the policy π. Note that as seen in Equation 3.11, the state-value function can be written as the sum of all action-value functions for every action taken from the state s. [62]

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid s_t = s] \tag{3.9}$$

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] \tag{3.10}$$

$$v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s, a) \tag{3.11}$$

The role and positions of the dynamics equation, policy, reward and value functions are visualised in the backup diagram in Figure 3.2. This represents the probabilities and steps taken by the actor (π) and environment (p) starting from the current state s in order to end up at the next state s'.

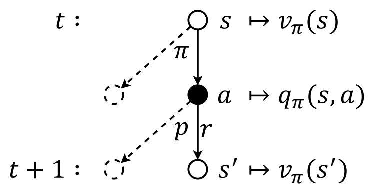


Figure 3.2: A backup diagram showing state, action, policy and reward for two time-steps, adapted from [62]

3.1.5. Bellman Equations

The value functions can be written in a recursive form which is useful for understanding update algorithms. The recursive formulation or Bellman equation for the state-value fiction can be seen in Equation 3.12 with a number of derivation steps. The recursive formulation is achieved by utilising the recursive formulation of G_t, the law of iterated expectation [13] and the linear properties of the expectation operator. Finally, the expectation operator can be expanded as the sum over all values of a, s' and r times the probability of the triple (a, s', r) occurring, which gives the last line of the Bellman equation for v_π. [62]

$$\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid s_t = s] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1} \mid s_t = s] && \textit{Equation 3.7} \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma \mathbb{E}_\pi[G_{t+1} \mid s_{t+1} = s'] \mid s_t = s] && \textit{law of iterated expectation} \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1}) \mid s_t = s] && \textit{Equation 3.9} \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]
\end{aligned} \tag{3.12}$$

3.1.6. Optimality Equations

To solve an RL problem involves finding the right policy so the rewards are maximised over time. This problem can be formally defined for finite MDPs using the definitions of the value functions. A policy π can be said to be better than another policy π' if $v_\pi(s) \geq v_{\pi'}(s)\ \forall s \in \mathcal{S}$. There will always be at least one policy that satisfies this and is called the *optimal policy(s)* π_*. The optimal policies share the same optimal state-value function and optimal action-value function which can be seen in Equation 3.13 and Equation 3.14 respectively. Both optimal value functions give the expected return when following the optimal policy. The last two lines for each equation represent the *Bellman optimality equations* for the value functions. [62]

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

$$\begin{aligned}
q_*(s,a) &\doteq \max_\pi q_\pi(s,a) \\
&= \mathbb{E}[r_{t+1} + \gamma \max_{a_{t+1}} q_*(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a] \\
&= \sum_{s',r} p(s', r, \mid s, a)[r + \gamma \max_{a'} q_*(s', a')]
\end{aligned} \tag{3.14}$$

3.1.7. High-Level Concepts

The following high-level concepts can be useful in categorising different RL frameworks and identifying their advantages and disadvantages.

Model-Dependence
A model of the environment is not always given, sufficiently accurate or identifiable for the RL framework to use. When a representative model of the environment is required beforehand, the RL algorithm is called model-based. An example of this is dynamic programming approaches, discussed in more detail in subsection 3.1.8. When no model is given to the RL agent, it is called model-free. Model-free RL methods can generally be separated into two categories.

The computed probabilities used in model-based methods can be replaced by sample-based probability approximations. These methods use the interaction with the real environment to learn the desired probability distributions and do not need them in explicit form [62].

Other approaches perform incremental model identification as part of the RL task. A class of particular interest for this research is Incremental Approximate Dynamic Programming (iADP) discussed more in section 3.3.

Note that the definition of model-free can vary across literature with respect to methods of implementing online model identification. These methods formally do include a model of the environment but do not require it to be known beforehand. As part of the research objective, the aim of this research is to advance developments in *model-independent* RL algorithms. During this research, the term model-independence is used to also include the online system identification methods and consider them to be model-free.

On-Policy vs. Off-Policy (Exploitation vs. Exploration)
In RL, there is always a trade-off between exploitation and exploration. Methods that use *exploitation* select the *greedy* action, which is the action with the highest value at a given time step. In contrast, a *non-greedy* action does not necessarily maximise the value function at a given time, but taking that action enables *exploration* of the state-action space.

Exploitation will always result in the highest reward at a given time step, however, exploration is also necessary to increase the chance of receiving a higher cumulative return in the long run. Finding a balance between exploitation and exploration is an important part of RL design. A simple way to do this is to use the greedy action most of the time and randomly select actions that are non-greedy with a small probability. This approach is also called near-greedy or *ε-greedy*, with ε the probability that the non-greedy action is selected.

On-policy and off-policy methods are closely related to the concepts of exploitation and exploration. In general, on-policy means that the *behavioural policy* is the same as the *target policy*, while with off-policy methods they are separate. The behavioural policy is the policy that selects the action, while the target policy is the policy that is being evaluated and improved by the agent. Off-policy methods can thus be more powerful in general as they use more exploration, but they often have greater variance and are slower to converge than on-policy methods. [62]

Online vs. Offline Learning (Adaptive vs. Robust Control)
With online learning, the learning happens during the interaction with the actual environment, so while performing the tasks. In the case of flight control, this means the RL framework is learning during flight and adapting to changes in the environment, which is also called *adaptive control*. Fault tolerance could be achieved through means of online learning and adaptive control. This technique of obtaining fault tolerance is mainly relevant for algorithms that have high sample efficiency so that the convergence time online is practical and safe.

Offline learning represents a fully offline learning process with no updates to the parameters being performed during the task itself. In the context of flight control, this can also achieve fault tolerance, in this case, through generalization, and is called *robust* control. This technique is most relevant to methods that might have a higher sample complexity like the Deep Reinforcement Learning (DRL) methods discussed in section 3.4 which have the generalization power to take advantage of the offline learning process.

The advantage of offline learning is that there are no real-time or safety constraints. Online learning in contrast is not allowed to explore safety-critical states but is expected to more easily adapt to faults during flight than a generalised robust control approach. Another option is a hybrid approach where there is an initial offline training phase and also online learning during flight, potentially combining the advantages of both approaches.

Sample Efficiency
Sample efficiency or data efficiency in RL relates to how efficiently an algorithm can make use of training samples. The term sample-complexity means the opposite, a high sample complexity means low sample efficiency. A sample-efficient RL algorithm makes better use of the training samples and can improve the policy faster. Generally, the performance of a sample-efficient method is higher than that of a sample-inefficient method when taking the same amount of training time. The problem of sample efficiency is of great importance when looking to build real-world applications of RL algorithms. The cost of the interaction between the agent and the environment plays a major role in this. In simulation, it can be the time and computational complexity available based on available computational resources that bottleneck the real-world applicability of a sample-inefficient algorithm. The same holds for real-world interactions where constraints on time, equipment degradation and safety of exploration require sample-efficient RL controllers. Sample efficiency is in general higher for model-based methods as opposed to model-free methods. In this case, the (learned) model provides additional information to the agent, resulting in faster learning of the optimal policy. [13]

Learning Stability
Reinforcement learning can suffer from an unstable training process, meaning a large variety in learning performance over time. Unstable learning is mostly caused by variance and bias of the value function or policy estimation and their gradients. These effects are often visible as large variances in the learning curves. Other stochastic effects like the variance of the bias itself, the randomness of the exploration process, randomness of the environment or numerical randomness all contribute to a less stable learning process.

DRL can particularly suffer from unstable learning due to the unpredictable properties of deep neural networks. Having Independent and identically distributed (i.i.d) (i.d.d.) data is a requirement for the effective training of neural networks. In supervised learning, this requirement poses less of a problem due to prepared data sets. Reinforcement learning, however, learns from samples that are generally highly correlated as they are generated by consecutive interactions with the environment. This reduces the effectiveness of learning with neural networks.

There are notable drawbacks to having less stable learning behaviour. Firstly, there are cases where the learning performance can even trend downwards after a long training time [20]. Secondly, the randomness of performance makes it difficult to accurately compare different algorithms, hence it is important to use multiple random seeds and averaged results. Multiple features of DRL state-of-the-art reinforcement algorithms like target networks, delayed update, replay buffer and smoothing regularization help with learning stability and are further discussed in section 3.4. [13]

Function Approximation
Function Approximation refers to the way the value function or policy is represented. In discrete state and action spaces, a tabular representation of the value function is used. However, when moving to continuous state and action spaces, function approximation techniques have to be used to generalize the value functions and policies to the full state and action domains. Using function approximation is a way to deal with the ”curse of dimensionality” associated with tabular methods, which have their computational requirements grow exponentially with the number of state variables. A further discussion of function approximation is given in subsection 3.2.1.

Value-Based vs. Policy-Based
The distinction between value-based and policy-based depends on the role of the agent in estimating either value functions or policies directly. The common discrete solution methods discussed in the next section are all value-based, as they use the value function to discover an optimal policy. The distinction mainly comes into play in continuous-space problems where function approximation can also be used to parameterise the policy as well as the value function. Value-based and policy-based actor structures are called critic and actor respectively and are discussed in subsection 3.2.2.

3.1.8. Common Discrete Solution Methods
RL frameworks can be categorised according to their update rule method. Three commonly used classes of RL algorithms are discussed here, Dynamic Programming (DP), Monte-Carlo (MC) and Temporal Difference (TD) learning. Many more update rules exist these three can be considered as

fundamental frameworks for discrete state and action spaces. Even though these methods are not feasible for the continuous-space flight control problem of the PH-LAB, they do contain the theoretical foundation of continuous-space methods discussed in section 3.2. In Figure 3.3, an overview of these methods can be seen, separated by model-free and model-based with rounded boxes representing these categories, sharp boxes algorithm classes and specific algorithms in text. These methods and their algorithms are discussed in more detail below. Note that the model-independence required for this research can also derive from the model-based category in this fundamental taxonomy as further discussed in section 3.3 as the incremental system identification implemented in ADP methods requires no prior known system model.

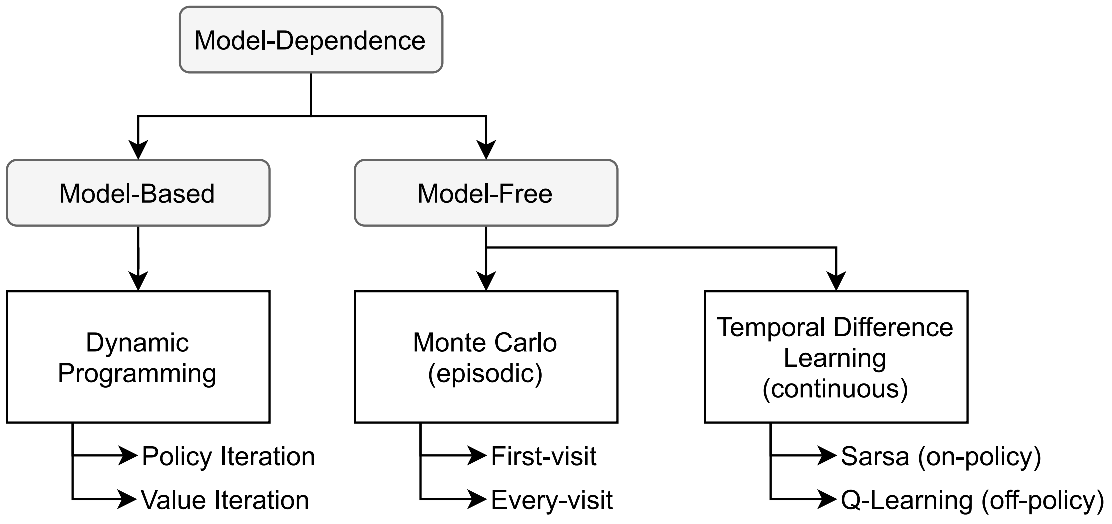


Figure 3.3: RL common solution methods for discrete state and action spaces

Dynamic Programming
Dynamic Programming (DP) is a fundamental framework used to solve MDPs iteratively. It assumes a finite MDP as the environment process and assumes perfect knowledge of the environment such as the reward and state transition functions. DP algorithms are thus model-based and are limited in real-world utility, but are nonetheless theoretically significant, also to continuous state and action spaces. DP algorithms work by using the Bellman optimality equations to create update rules and consist of mainly two types:

- *Policy Iteration* uses a policy evaluation and policy improvement step, where the general interaction between these steps is called Generalised Policy Iteration (GPI). The general principle of GPI is that both an approximate policy and approximate value function are maintained. The value function is repeatedly evaluated to correspond more closely with the current policy, and the policy is repeatedly improved using the current value function. Using the Bellman equation for v_π, an update rule can be stated as in Equation 3.15, where full knowledge of the environment is needed. This is the (iterative) *policy evaluation* or *prediction* step and is executed for every state $s \in \mathcal{S}$.

  The *policy improvement* step can be expressed by Equation 3.16. This selects the *greedy policy* with respect to the value function of the previous policy and will result in the action that looks best in the short term according to v_π. The initial v_0 and $\pi(s)$ are set arbitrarily and the resulting sequence of monotonically improving policies and value function can be seen in Equation 3.17 with E evaluation and I improvement. [62]

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \quad \forall s \in \mathcal{S} \tag{3.15}$$

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \sum_{s',r} p(s',r,|\, s,a)[r + \gamma v_\pi(s')] \tag{3.16}$$

$$\pi_0 \xrightarrow{\text{E}} v_0 \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_1 \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \dots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_* \tag{3.17}$$

- *Value Iteration* can decrease the number of iterations per time step by truncating the policy evaluation step from policy iteration. This is achieved by stopping the policy evaluation after one update of each state. Equation 3.18 combines the policy improvement and (truncated) policy evaluation steps in one simple update step. [62]

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r,|\, s,a)[r + \gamma v_k(s')] \quad \forall s \in \mathcal{S} \tag{3.18}$$

Monte Carlo
Monte-Carlo (MC) methods are different from DP in the way that they do not require any prior knowledge of the environment and require only experience from actual interaction with the environment. MC as opposed to DP also does not bootstrap, meaning using a previous estimate in the current estimate. MC prediction methods consist of *first-visit* and *every-visit* MC. First-visit MC uses only the average of the returns of first occurrences (visit) of a state s in an episode, while every-visit MC also uses repeated visits to a state s in estimating the value function. MC prediction methods can be used to both estimate state-value or action-value functions using sampled probabilities instead of computed probabilities from model knowledge. When an environment model is known like in DP, estimating the state-value function is sufficient in determining an optimal policy and executing the policy improvement step.

For MC control tasks, however, estimating the action-value function is more useful since, without an environment model, the action-value is needed in order to compute an optimal policy as can also be seen from Equation 3.16. MC control also utilises the GPI principle for the interaction between the prediction and improvement steps, the main difference with DP then lies in the policy prediction or evaluation step.

The value function update process using MC happens on an episode-to-episode basis, the general form of this incremental update rule can be written as in Equation 3.19. Here the $Estimate$ is for a state-value or action-value function, and $StepSize$ is a parameter that can be used to control the speed of the update. The term $[Target - OldEstimate]$ is often referred to as the error. In Equation 3.20 a simple MC prediction update rule for the action-value function is shown were $Q(s,a)$ is the estimate for $q_\pi(s,a)$, the $Target$ is the actual return G_t known only after an episode, and the $StepSize$ is represented by α. [62]

$$NewEstimate \leftarrow OldEstimate + StepSize\,[Target - OldEstimate] \tag{3.19}$$

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\,[G_t - Q(s_t,a_t)] \tag{3.20}$$

Referring back to the Bellman equation for the state-value function in Equation 3.12, MC methods utilize the first row as the target, with the expectation estimated after the termination of an episode using samples.

A major limitation for MC methods is that they can only learn after an episode is finished, meaning it is only available for episodic tasks and not continuous tasks like the flight control task considered for this research. [62]

Temporal Difference Learning

Temporal Difference (TD) learning combines ideas from both DP and MC. Like MC, TD does not require full knowledge of the environment and uses a sampling-based learning approach, while it also employs the bootstrapping feature from DP. The difference between DP and TD methods again lies in the policy evaluation phase, while the control problem or policy improvement step is largely the same following the GPI principle. With TD estimation, the value function can be updated after every time step which is possible due to the $Target$ consisting of the observed reward and value function estimate $V(s_{t+1})$ instead of the complete return as in MC, this also makes TD a bootstrapping method.

When using TD for control tasks, the action-value function is used for the estimation step to allow model-free policy improvement. The pattern of GPI is also utilised like for DP and MC in order to converge to an optimal value function and policy. The update rule for the on-policy TD control algorithm called Sarsa [65] can be seen in Equation 3.21. For off-policy TD control, Q-learning can be used using the update rule as in Equation 3.22 [62]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\,[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.21}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \tag{3.22}$$

Referring back to the Bellman equation for the state-value function in Equation 3.12, TD methods utilize the fourth row as the target, with the expectation estimated by sampling. In contrast, DP methods solve this same equation through the model of the environment being known instead of sampling the environment. [62]

3.2. Reinforcement Learning in Continuous Space

The previously discussed solution methods all assume finite MDP with finite and discrete state and action spaces. This however does not reflect the flight control tasks that are the subject of this research, which have continuous state and action spaces. This section goes over methods used to implement RL in continuous spaces starting with the concept of function approximation and optimization.

3.2.1. Function Approximation and Optimization

Policy prediction or evaluation methods estimate the value function, the value for a given state (or state-action pair). Up till now, the prediction step update rule caused the value for a given state/action to update incrementally towards the update target. This adjusts the value prediction for only that state and does not estimate an actual function that generalises to other states. For continuous state and action spaces, however, it is not feasible to maintain the value function in this tabular fashion. This would either increase discretization errors too much for complex tasks such as flight control or increase computational and memory requirements beyond practical use. Instead of directly maintaining this discrete mapping from state to value, these mappings can be used as input for *function approximation* methods from the field of Machine Learning (ML) which can estimate the underlying value function and generalise it to other states. This treats every update step as a training step for the function approximator. [62]

Functional Form

The structure of the function approximation can differ on mainly two levels, parameterization and linearity. On the level of parameterization, there is a trade-off between flexibility and computational efficiency. Non-parametric methods like kernel-based methods [56] make weaker assumptions about the form of the function and are generally more flexible. Non-parametric methods are however often memory-based and require significantly more data and computation than parametric methods [82]. In the continuous control problem of the PH-LAB, parametric methods are the more feasible option for any sort of online learning. They do compromise on generalization power by assuming a given class of functions but are much more computationally efficient for larger state spaces. Parametric methods have a set of parameters that control the shape of a predefined function class. The value function estimates $v_\pi(s) \approx V_w(s)$ or $q_\pi(s,a) \approx Q_w(s,a)$ are parameterized by the weight vector w and the policy π_θ

by the parameter vector θ. Tuning these parameters towards an optimal approximation is referred to as the optimization method which can be either *value-based optimization* or *policy-based optimization* depending on the agent structure, discussed more in subsection 3.2.2.

Linear function approximation methods are linear in the parameters, meaning the approximated function is a linear combination of the parameters and the feature vector. Linear methods can have a convergence guarantee for approximating value functions [79], however, this is highly dependent on the specific feature representation [13]. Examples of feature representations for linear methods are polynomials, Fourier series, coarse coding, tile coding and radial basis functions. The need for informative, hand-picked features that require sufficient domain knowledge is a major drawback of linear function approximation methods [90]. Linear methods are generally better understood and have a stronger guarantee of convergence. Convergence to a local optimum can be given in certain cases [45], but in general, there are fewer theoretical guarantees for non-linear methods. The stronger generalisation power of non-linear methods is however an advantage for the non-linear control problems considered in this research. Empirical results also demonstrate the usability of combining reinforcement-learning algorithms with non-linear function approximators like Artificial Neural Networks (ANNs) [90].

Like biological neural networks, ANNs are made up of multiple interconnected neurons. Generally, ANNs consist of at least three layers of neurons where an input layer is connected to an output layer through one or more hidden layers. When more than one hidden layer is present it is usually referred to as a Deep Neural Network (DNN). The output of a neuron is calculated as the weighted sum of its input plus a bias and passed through an activation function. The neuron connections each carry the weight value. The activation function is usually non-linear and controls the shape and amplitude of the neuron output. These weights and biases form the parameter vector for an ANN function approximator and have to be optimised using function optimization methods discussed in the next paragraph.

Function Optimization

Optimization in this case refers to how the parameters of a function approximator like an ANN are trained. Linear function approximators can use a closed-form computation of the parameters, for example for TD with linear function approximation, Least Squares TD Learning (LSTD) can be used[8] [7]. For non-linear function forms, however, other methods are required. A distinction can be made between gradient-free and gradient-based function optimization methods. Gradient-free methods can be used when the function to be optimized is not differentiable. Examples of gradient-free optimization methods are evolutionary methods and Cross-Entropy (CE). Evolutionary methods are usually very fast for simpler cases, but will not be considered for this high-dimensional flight control case due to the high variance [90]. Similarly, CE methods are fast but have limited applicability to the reinforcement learning problem at hand as it often converges to sub-optimal policies [13]. For the remainder of this research, only gradient-based optimization methods will be considered.

Gradient-based methods are also called Gradient Descent (GD) or ascend, depending on whether a loss function or value function is used as the objective. Gradient-based methods require the differentiability of the function that is to be optimized, which is no problem for structures like the non-linear ANNs. GD updates the parameters by taking small steps towards a maximum or minimum using the gradient. The term back-propagation [64] is also mentioned in the rest of this research and is a term used to describe a method to compute this partial derivative of the loss function with respect to the parameters of the function model. GD usually works by computing the loss of all samples after an iteration. A variation of GD called Stochastic Gradient Descent (SGD) [6] is more practical for the continuous control problem. SGD is a commonly used method that performs GD on a randomly sampled mini-batch of samples instead of on a sample-by-sample basis like with normal GD. This makes the computation of the loss function more efficient and allows it to be applied online.

3.2.2. Agent Structures

With function approximation in the picture, there are three main agent structures that are commonly used depending on what exactly is being approximated. Value functions and policies can both be learned by the agent. For example, the discrete solution methods discussed in subsection 3.1.8 all learn the value function which is then used to estimate the optimal policy, usually using the action-value function for a control problem. These structures that learn the value function are called the *critic*, while structures that learn a parameterised policy function directly are called the *actor*. Agents can consist of a critic, an actor or both, creating three main agent structures discussed in more detail below.

Critic-Only
Critic-Only or *Value-Based* agents estimate the value function which is then used to derive an optimal policy. Like the discrete-space solution methods without prior model knowledge, the action-value function estimate $Q(s,a)$ is maintained by the actor and used to select a greedy action by directly maximising the Q-function. In continuous spaces with function approximation, usually, gradient-based methods as discussed in subsection 3.2.1 are used with ANNs to optimise the value function estimates. A major limitation of purely value-based methods appears when working in continuous action spaces like the continuous control of the PH-LAB aircraft. Finding the greedy action based on a value-function estimate would require a search through the entire action space, which is practical for discrete action spaces, but impractical for infinite action spaces.

Actor-Only
Actor-Only or *Policy-Based* agents directly estimate the policy without estimating a value function. An advantage over value-based methods is that, with function approximation and parameterizing the policy $\pi_\theta(a_t \mid s_t)$, the optimal action can be found in continuous action space. Since a generalised formulation of the policy function is available in policy-based methods, function optimization methods can be used to effectively find an optimal action in infinite action space. Most practical to the flight control task at hand are gradient-based optimization methods as discussed in subsection 3.2.1, which with policy-based agents are generally referred to as *policy gradient* methods. The policy gradient theorem originally proposed by [77] describes how the parameters of the parameterized policy are updated and can be seen in Equation 3.23. Here the value function V_π is used as the objective function. This represents Stochastic Policy Gradient (SPG) applied to a stochastic policy. Note that T can extend to infinity when the discount factor of the discounted return G_t is smaller than one $\gamma < 1$. The policy gradient theorem has been proven to also hold for deterministic policies where a deterministic policy is a limit case of a stochastic policy with the variance parameter going to zero $\lim_{\sigma \to 0}$ [72]. This approach is called Deterministic Policy Gradient (DPG) and is further discussed as an actor-critic implementation in the deep reinforcement learning method Deep Deterministic Policy Gradient (DDPG) in subsection 3.4.4.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta V_\pi(s_t) \quad \text{with} \quad \nabla_\theta V_\pi(s_t) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t\right] \tag{3.23}$$

Since the term $\nabla_\theta V_\pi(s_t)$ requires the discounted return G_t, all future rewards of a task need to be known before an update can be performed. The vanilla policy gradient method is thus an MC method and is only suitable for episodic tasks. The MC stochastic policy gradient forms the basis of the REINFORCE algorithm [91]. Despite the simplicity of the REINFORCE algorithm, it has been shown to suffer a large variance of the discounted return G_t, increasing exponentially when the task length increases [13]. Baseline REINFORCE [92] [40] has been developed that aims to reduce the variance of the vanilla policy gradient method. A version of baseline REINFORCE with the value function as a baseline is an actor-critic framework discussed in the next section.

Actor-Critic
Actor-critic algorithms[33] [77], as the name suggests, combine both value-based and policy-based approximation. Instead of MC estimating the discounted return in the policy gradient method, a TD approximation critic can be used to estimate the value function. This reduces the variance present with the MC return estimation and enables continuous or non-episodic tasks due to the TD update. In the actor-critic framework, the critic receives the state and reward. It then uses TD to estimate the value function and outputs the TD error. The TD error is used for training the critic itself and is also sent to the actor. The actor uses the TD error from the critic to update its policy parameters. The actor still selects the action like in actor-only methods by accepting the state and outputting the action to the environment. Also see Figure 3.4 for the interaction between the actor, critic and environment.

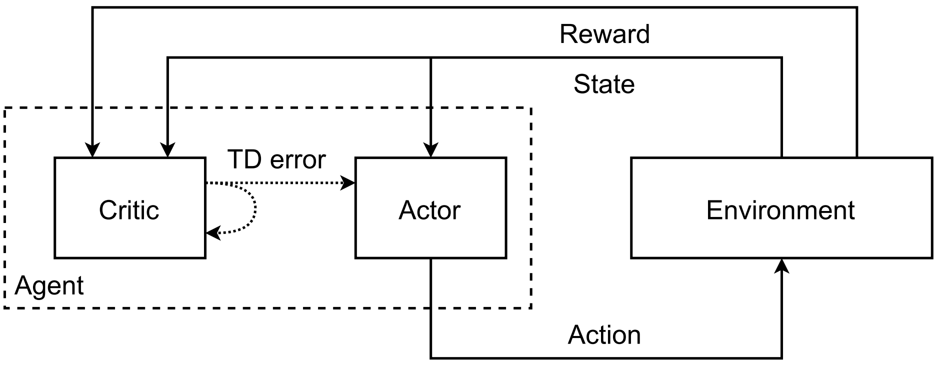


Figure 3.4: Actor-Critic framework interaction between actor, critic and environment

Due to the ability to deal with continuous action spaces because of the actor, and continuous-time tasks because of the TD critic, the actor-critic approaches will be considered a suitable candidate for the RL controller for the PH-LAB. Two main groups of actor-critic RL algorithms will be considered for further discussion. The Adaptive Critic Designs (ACDs) from the branch of Approximate Dynamic Programming (ADP) are an extension of Dynamic Programming into the continuous domain. These RL frameworks contain current state-of-the-art research on online adaptive flight control and are discussed in section 3.3. The second category is Deep Reinforcement Learning (DRL), combining the recent advances in Machine Learning with the function approximation of model-free RL methods. These methods are also the subject of state-of-the-art research into robust flight control and are further discussed in section 3.4.

3.3. Approximate Dynamic Programming

Approximate Dynamic Programming (ADP) [86] [60] is a class of RL algorithms and an extension of the model-dependent Dynamic Programming approach. ADP, as the name suggests, uses function approximation to tackle problems in continuous state and action spaces and thus deal with the curse of dimensionality that limits DP. The scope of this research limits the ADP methods considered to actor-critic designs using ANNs as function approximators.

To further categorise these actor-critic ADP methods, the distinction is made in the design of the critic. The actor always has the same structure and maps state to action. The design of the critic can vary and results in a number of different strategies as seen Figure 3.5 which shows a breakdown of the ADP methods discussed in the following sections. These on-policy ADP algorithms are called Adaptive Critic Designs (ACDs) and are discussed in more detail in the following section. Note that this is not an exhaustive breakdown of ADP and ACDs are only a part of ADP which are considered here for the flight control problem.

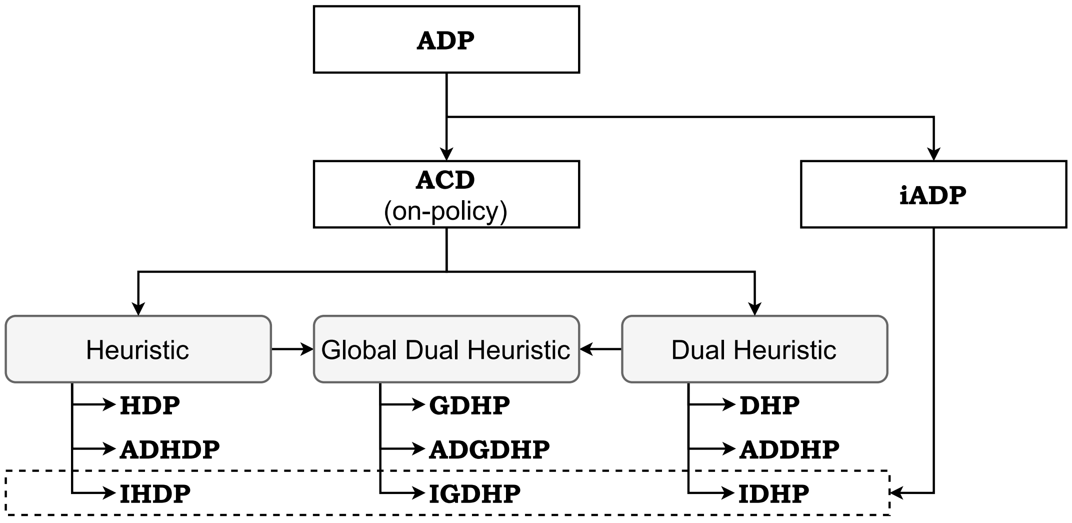


Figure 3.5: Approximate Dynamic Programming Adaptive Critic Designs

3.3.1. Adaptive Critic Designs

Adaptive Critic Designs (ACDs) [43] generally come in one of three categories: *Heuristic*, *Dual Heuristic* and *Global Dual Heuristic*. The basic frameworks under these categories are Heuristic Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP) and Global Dual Heuristic Programming (GDHP) respectively. Each ACD framework discussed has an Action-Dependent (AD) and incremental variant where the incremental versions of the frameworks are commonly referred to under the term Incremental Approximate Dynamic Programming (iADP).

iADP aims to reduce the model dependence of the general DP approach in ADP through the use of incremental model identification. The identification process of the environment model is implemented in the RL framework. This also makes the RL controller adaptive as opposed to the non-incremental methods which require full apriori knowledge of the plant dynamics. In the incremental methods, the environment model is approximated by a linearized, time-varying incremental model which still allows the computational complexity to stay manageable. To accurately identify the system dynamics online, a sufficiently high sampling rate is assumed for discretization accuracy [76]. This incremental model is identified at each time step by using the conditions of the system of the previous time step, hence the term incremental[99] [74]. Note that the incremental versions are only applied to the non-action-dependent frameworks. This is because the model-dependence reduction of iADP outweighs the partially reduced model dependence of AD structures [11]. Below, the three main categories of ACDs are discussed in more detail. A summary of the critic structure and model dependence can be seen in Table 3.1.

Heuristic Dynamic Programming

With HDP [71] the simplest form of the ACDs, the critic estimates the state-value function. This means the critic can be trained by receiving the state s_t and then outputs an estimate of the state-value function $V(s_t)$. As this is a state-value method, the actor requires the derivative of the state-value function w.r.t. the action $\frac{\delta V(s_t)}{\delta a_t} = \frac{\delta V(s_t)}{\delta s_t}\frac{\delta s_t}{\delta a_t}$ which requires backpropagation and also the state-transition model to be known. Hence HDP is model-based.

The model dependence can be addressed by letting the critic estimate the action-value function instead. This is what the action-dependent ADHDP framework does. With ADHDP the critic receives both the state s_t and action a_t and can be trained to estimate the action-value function $Q(s_t, a_t)$. The derivative of the value function $\frac{\delta Q(s_t,a_t)}{\delta a_t}$ can then be computed with just backpropagation through the critic. It has been shown that ADHDP has an almost halved convergence success ratio compared to HDP but does handle measurement noise better than a baseline HDP controller [84]. Note that from all the ACDs considered here, ADHDP is the only model-free, non-incremental method. The critic of ADHDP is essentially equivalent to Q-learning [43].

The incremental variant IHDP avoids the need for an offline learning phase because of the linearized local model identification at the beginning, which has been shown to be fast and accurate [96]. IHDP has been successfully applied for angle-of-attack, pitch rate and roll rate control [12] [97].

Dual Heuristic Programming

The critic in DHP outputs the gradient of the state-value function $\frac{\delta V(s_t)}{\delta s_t}$ instead of the value function itself like with HDP. This simplifies the actor training since $\frac{\delta V(s_t)}{\delta s_t}$ can now directly be used in calculating the partial derivative of the value-function w.r.t the action, instead of using backpropagation like with the actor of HDP. A state-transition model is still required for the actor, while the critic now also requires the environment model to calculate the derivatives. Compared to HDP, DHP can be smoother and more accurate due to the elimination of backpropagation to find the derivative and instead of training the critic to directly estimate the derivative [85] [61].

The critic of the action dependent variant ADDHP outputs the partial derivatives of the action-value function, which are both $\frac{\delta Q(s_t,a_t)}{\delta s_t}$ and $\frac{\delta Q(s_t,a_t)}{\delta a_t}$. This removes the model-dependence of the actor which now has access to $\frac{\delta Q(s_t,a_t)}{\delta a_t}$ directly from the critic. The critic itself however does still require the state-transition model.

IDHP is again an extension of the non-AD DHP framework where the a priori model knowledge is eliminated by including incremental plant identification in the learning process. IDHP was applied on a longitudinal angle of attack and pitch rate controller on a tracking task [98] where it showed to outperform DHP in fully online learning, efficiency, accuracy and robustness. Using non-linear NN's it was able to identify the incremental state-transition model fully online and showed it was fast enough to control unstable systems before the state converged.

Global Dual Heuristic Programming

GDHP combines both HDP and DHP and its critic outputs both the value function estimate $V(s_t)$ and the derivative $\frac{\delta V(s_t)}{\delta s_t}$. A critic similar to the HDP critic estimates the value function and passes that on to a second critic network or dual network. This DHP-type dual network also receives the states of all hidden neurons of the critic and outputs the value function derivative $\frac{\delta V(s_t)}{\delta s_t}$. Like with HDP and DHP, the actor is model-dependent, while the DHP-type dual network also makes the critic require a state transition model. GDHP is expected to outperform DHP which in turn outperforms HDP on the accuracy of the estimation. However, it is significantly more complex to implement and requires more computational complexity due to the inclusion of second derivative terms, which in most practical cases results in DHP frameworks being preferred over GDHP [43] [61].

ADGDHP [61] estimates the action-value function and its partial derivatives which are then used by the actor to select an action. IGDHP [76] adds incremental model identification to the GDHP framework and was tested on simple longitudinal control tasks. Like with the comparison between GDHP and DHP, the performance increase of IGDHP usually does not justify the increased complexity over IDHP.

Table 3.1: Summary of ADC's with their critic structure and model dependence, adapted from [11], [28]

Agent Structure	**Critic**		**Model-Dependence**	
	Input	Output	Critic	Actor
HDP	s_t	$V(s_t)$		×
ADHDP	s_t, a_t	$Q(s_t, a_t)$		
IHDP	s_t	$V(s_t)$		
DHP	s_t	$\frac{\delta V(s_t)}{\delta s_t}$	×	×
ADDHP	s_t, a_t	$\frac{\delta Q(s_t,a_t)}{\delta s_t}$, $\frac{\delta Q(s_t,a_t)}{\delta a_t}$	×	
IDHP	s_t	$\frac{\delta V(s_t)}{\delta s_t}$		
GDHP	s_t	$V(s_t)$, $\frac{\delta V(s_t)}{\delta s_t}$	×	×
ADGDHP	s_t, a_t	$Q(s_t, a_t)$, $\frac{\delta Q(s_t,a_t)}{\delta s_t}$, $\frac{\delta Q(s_t,a_t)}{\delta a_t}$	×	
IGDHP	s_t	$V(s_t)$, $\frac{\delta V(s_t)}{\delta s_t}$		

Based on the characteristic features of the ACDs discussed above, IDHP appears to be the preferred framework for this research's flight control problem of the PH-LAB. DHP is a clear improvement over the simpler HDP method due to the specialised training for the derivatives of the value functions. The recently developed incremental version of IDHP adds the adaptive control abilities in order to react to system changes making it more fault-tolerant and enabling full online learning [98]. A number of other related studies [97] [12] have also come to the conclusion that IDHP is currently a state-of-the-art RL framework for continuous adaptive flight control and implemented it as such. Hence, the next sections will mainly consider IDHP as a viable ADP method for the flight control problem.

3.3.2. State-of-the-art ADP Applications

This section goes over some state-of-the-art ADP flight control applications in more detail, specifically model-free iADP methods. These particular applications are of the most interest for the continuation of this research and further elaborate on the benefits and drawbacks of the ADP algorithms and recommendations for further improvement in flight control applications.

IDHP | Reinforcement Learning for Flight Control: Learning to Fly the PH-LAB

In [28], an ADC IDHP framework is derived and used for adaptive flight control of the PH-LAB aircraft. The framework utilises ANNs as function approximators and includes an additional network structure to improve learning stability. The resulting RL controller was tested in simulation using the Delft University of Technology Aircraft Simulation Model and Analysis Tool (DASMAT) high-fidelity simulation model of the Cessna 500 Citation I, developed by the Delft University of Technology. Other work on the same aircraft model already focused on longitudinal control with a manual outer loop [32].

The DASMAT model includes the engine dynamics, while actuator dynamics are modelled using a simplified first-order model with deflection saturation. Similarly, the sensor models are disregarded as clean sensor measurements are assumed.

The IDHP learning framework proposed in this paper is derived from [98]. It consists of the three main parametric structures characteristic of iADP frameworks: the actor, critic and the incremental plant model. Additionally, a target critic network is implemented in order to improve learning stability, as introduced by Q-learning discussed in subsection 3.4.1. The dual heuristic approach taken here means the critic is tasked with estimating the partial derivative of the state-value function with respect to the state as discussed in subsection 3.3.1. The incremental plant model is defined as a linear approximation of the plant by a first-order Taylor series expansion around a certain operating point. This operating point is can be set to the previous state and action resulting in the incremental form. The identification of the incremental model estimates the state and input matrices using an online recursive least squares (RLS) approach based on [25].

The flight controller is implemented to provide angular rate control of pitch and roll, as seen in Figure 3.6. The flight controller decouples the lateral and longitudinal control of the aircraft with individual IDHP controllers for each. The outer control loop consists of conventional PID controllers which are used for setting the reference signals of altitude and roll angle. A more recent paper [37] attempts to implement cascaded IDHP control loops to eliminate the PID loops and provide complete adaptive control. This implementation has shown to be too sensitive to measurement noise.

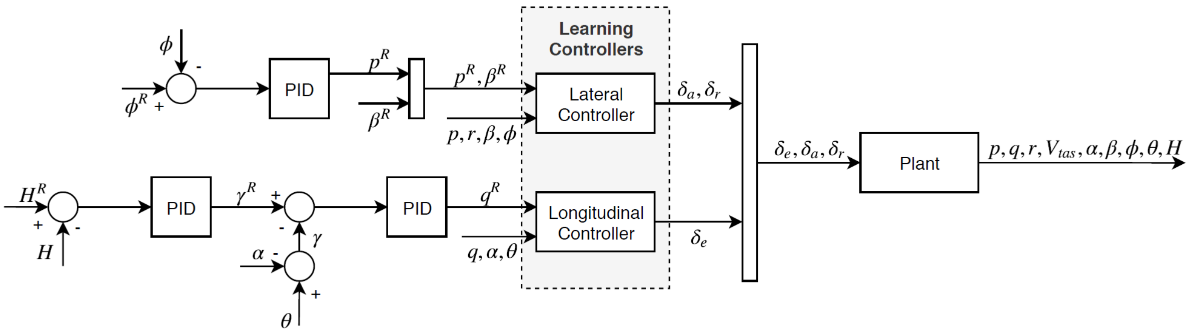


Figure 3.6: Schematic of the decoupled lateral and longitudinal IDHP flight controller design from [28]

The results from simulation testing showed that the IDHP controller was able to learn a near-optimal control policy with no prior system knowledge and with no offline training phase. The additional target critic managed to eliminate failure in online runs and fault tolerance was demonstrated on simple actuator failure and reduction of actuator effectiveness scenarios. A similar control framework was developed in [34] which a shared critic for the longitudinal and lateral controllers were used. This introduced the ability to take into account some coupling effects. For this application, however, a higher failure rate is observed of up to 27% and similarly to [28], a disadvantage of the continual online learning is identified. Since the continuous flight control task has an unknown varying duration in practice, the authors recognised an issue where increasing network parameters cause increasingly unstable or aggressive behaviour over time. Hence, the addition of parameter regularization [54], or an action penalty term to the reward function [19] is recommended.

Other recommendations from the authors include further research into the RL controller design before flight tests. Further performance analysis on the effect of measurement noise and delay is recommended due to the assumptions of clean measurements taken by the authors. The use of the high-fidelity sensor models available in the DASMAT model of the PH-LAB is thus also recommended. Additionally, in order to reach more areas of the flight envelope and better handle coupled effects, it is recommended that the decoupled architecture of the flight controller should be compared with a single coupled lateral and longitudinal controller design.

IDHP | Online reinforcement learning for fixed-wing aircraft longitudinal control
In [37], a cascaded IDHP controller is implemented in order to replace the outer PID control loops used in previous implementations. The performance is then compared with a baseline IDHP controller without the cascaded design. In Figure 3.7, the schematic of the cascaded altitude tracking controller architecture can be seen. As opposed to [28], an adaptive, error-based learning rate is implemented to combat the sensitivity to this hyperparameter. This has been shown to increase the learning stability [55] [98].

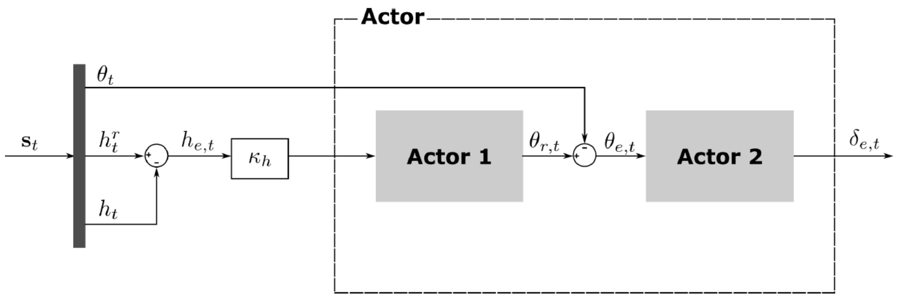


Figure 3.7: Schematic of the cascaded IDHP flight controller design for altitude tracking control from [37]

The baseline and cascaded controller were compared in three different experiments on an altitude tracking task. Firstly, a perfect scenario without measurement noise or disturbances showed good tracking performance for both controllers. The baseline controller had a significantly lower success ratio and the cascaded controller correctly exploited the extra knowledge of the outer loop, while having a longer convergence time than the baseline controller. On scenarios with measurement noise and disturbances, the cascaded controller showed an increased susceptibility to measurement noise and unique failure modes due to incorrect reference pitch angle generation by the outer loop. The overall conclusion was that the baseline IDHP controller was favoured over the cascaded IDHP controller and more in-depth analysis on the effects of measurement noise and disturbances is recommended.

3.4. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines RL with function approximation methods from Deep Learning (DL), hence DRL is also seen as a sub-field of Machine Learning (ML). This means the value function and policies are parameterized by variables in Deep Neural Networks (DNNs), which then also enables the use of gradient-based optimization methods. One of the advantages that DL brings to RL is the scalability of DNN for high-dimensional spaces. DNNs also feature a high level of generalization power. A major development by DeepMind in 2015 used DRL to achieve human-level performance on a number of classic Atari games using a deep Q-network [48], which helped bring DRL more into the forefront of RL research. With a similar achievement in 2017, DeepMind's AlphaGo [73] also demonstrated the ability of DRL algorithms to outperform a human by defeating a world-class GO player.

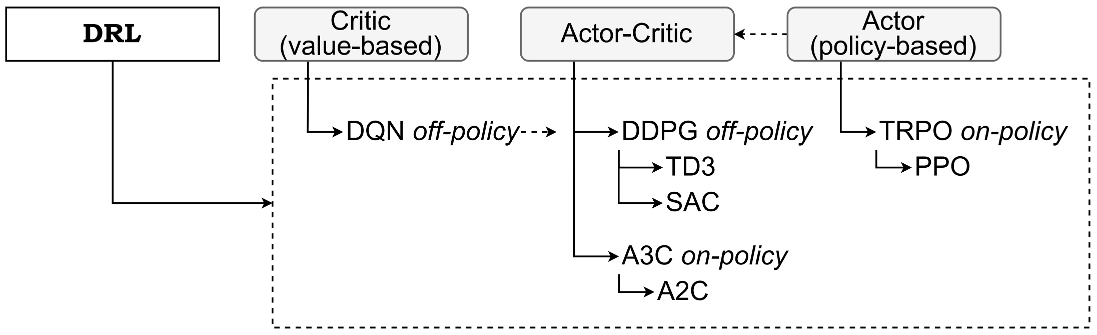


Figure 3.8: Deep Reinforcement Learning Algorithms

3.4.1. Deep Q-Network

Q-Learning is a tabular or linear function approximation method that has shown to be stable and convergent, however, when working with non-linear function approximators, Q-leaning becomes unstable or divergent [80]. With advancing research in the training of deep neural networks, a variant of Q-leaning was developed called a Deep Q-Network (DQN) [47]. DQN combines Q-learning with deep learning and introduces two features that are important to understand for related DRL methods as well. Just like Q-learning, DQNs are value-based and do not handle continuous action spaces like the PH-LAB control problem, however, the key ideas from DQN are still relevant to discuss. The two features introduced by DQNs are *replay buffer* and *target network* which are discussed in more detail below.

Replay Buffer

The replay buffer refers to the biologically inspired mechanism called *experience replay* [13]. The current state, action, reward and next state at time step t, $(s_t, a_t, r_{t+1}, s_{t+1})$, can be referred to as the experience of the agent at that time step t. The DQN then stores the experience of the agent into the replay buffer for every time step, until the memory limit is reached after which old experience samples are discarded. The Q-function is then trained using a sampled mini-batch from this replay buffer instead of a sample from only the current time step. Normally in Q-learning, experience samples are obtained consecutively and are thus highly correlated. Using the replay buffer addresses this issue and thus decreases the variance of the Q-function updates and smooths out the learning process reducing oscillations [13]. Another advantage is that experience samples can be reused, increasing the sample efficiency. Because experience replay uses stored transitions, actions are selected partly on previous policies, making DQN an off-policy method like Q-learning [47].

An improved version of experience replay called Prioritized Experience Replay (PER) [66] introduces a better sampling strategy. This technique prioritizes experiences based on the TD error. A higher TD error can be seen as a sample that contains more information for the learning process to work with. PER abandons the purely random sampling method of normal experience replay and instead uses importance sampling using a probability based on the TD error to ultimately select more information-rich samples.

Target Network

The stability of the neural networks can be further improved by using a separate target network. The target of the update rule refers to the estimate of the optimal value function, defined by the Bellman optimality equation Equation 3.14 in the case of TD control. Note that the Q-function TD update rule is dependent on the Q-function itself. A separate target network can be used instead to generate the Q-learning targets which are synchronised periodically (every x time step) with the Q-network. This synchronisation between the Q-network and the target network can happen with a hard update (direct parameter copy), or a soft update (exponentially decaying average) [13]. This essentially delays the Q-learning target and smooths out the update process, reducing oscillations and divergence.

3.4.2. Trust Region Policy Optimization

TRPO [68] is an on-policy policy gradient method. It is usually implemented in an actor-critic structure to allow for continuous-time tasks, but TRPO specifically refers to the policy optimization method used. Previously discussed policy-gradient methods all use an estimation of the value function for the objective function. TRPO on the other hand is characterised by making use of the advantage function instead. The advantage function is defined in Equation 3.24 which represents the difference, or advantage, in value between taking a certain action a_t and taking the action from the policy, starting from state s_t. The objective function $J(\theta)$ used for the policy gradient update can be seen in Equation 3.25. Instead of just using the advantage function, the Conservative Policy Iteration (CPI) scheme first proposed by [29] is used as an alternative policy updating method, which uses the probability ratio between the old policy π_θ and the improved policy $\pi_{\theta'}$. By maximizing the ”surrogate” objective function $J(\theta)$, an optimal policy can be estimated and monotonic improvements can be guaranteed.

The TRPO algorithm is aims to better handle the step size of the policy gradient. A problem with the vanilla policy gradient methods is that no guidance is given on the size of the step size. To improve learning stability the update step of the CPI can be theoretically bounded by using the Kullback–Leibler (KL) divergence as seen in Equation 3.26. This is a statistical measurement between two distributions, in this case, the policies, and when bounded by scalar δ keep the new and old policies close together ultimately resulting in better learning stability.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{3.24}$$

$$\max_\theta J(\theta) \quad \text{with} \quad J(\theta) = \mathbb{E}\left[\frac{\pi_{\theta'}(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} A(s_t, a_t)\right] \tag{3.25}$$

$$\mathbb{E}\left[D_{KL}(\pi_\theta || \pi_{\theta'})\right] \leq \delta \tag{3.26}$$

Proximal Policy Optimization (PPO)

TRPO suffers from high complexity in solving the constrained optimization problem. The policy gradient method PPO [69] tries to simplify the computational complexity of TRPO and provide better sample efficiency. Similarly to TRPO, PPO tries to keep the policy update bound for better learning stability. The surrogate objective function from TRPO is simplified by removing the KL-Divergence constraint. Instead, the step size is guided by clipping the probability ratio $\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ and taking the minimum with the unclipped objective as seen in Equation 3.27 with ϵ a hyperparameter.

$$J^{CLIP}(\theta) = \mathbb{E}\left[min\left(\frac{\pi_{\theta'}(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} A(s_t, a_t)\right), clip\left(\frac{\pi_{\theta'}(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon\right) A(s_t, a_t)\right] \tag{3.27}$$

PPO has been successfully demonstrated on attitude control of a fixed-wing UAV where it had better convergence than traditional PID controller with comparable performance [5]. In [88], a pre-trained PPO agent was implemented in an RL framework to enable the generation of a reference trajectory for an aircraft guidance task. For another quadcopter flight control task, PPO has been shown to outperform PID controllers and DDPG while using a PID outer loop [31].

3.4.3. Asynchronous Advantage Actor-Critic

A3C [49] was recently proposed by Google’s DeepMind as a state-of-the-art algorithm within an asynchronous RL framework. It is an actor-critic approach that focuses on parallel training. Previous RL frameworks have all consisted of a single actor-critic structure with their respective parameters and possible target networks. A3C on the other hand introduces multiple nodes each containing an actor and critic structure. A master node contains the global agent with the actor and critic responsible for the final policy. Multiple worker nodes are actor-critic agents that interact individually in parallel with a copy of the environment. The worker nodes send their experience to the master nodes which then use the TD error from the workers to drive the update steps for the global actor and critic. This method is also called asynchronous gradient descent. Also as the name suggests, A3C uses the advantage function for objectives like TD3 and PPO. In general, the worker nodes are only responsible for interaction with the environment and the master nodes are only responsible for updating the actor and critic. After the master agent has performed an update step, the worker agents synchronise with the master node. In the Atari games environments like DQN, the parallel approach of A3C has been shown to improve the scores while also halving the training time.

Improvements have been made to A3C by [93] which empirically showed that removing the asynchronous part of A3C actually resulted in a more efficient algorithm. This method called A2C introduces a coordinator that waits for every worker node and collects their experience. The global actor and critic are then updated only when all worker nodes are done. [13]

3.4.4. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [42] is a combination of Deterministic Policy Gradient (DPG) and DQN's learning method. Compared to DQN, DDPG is designed to handle large or continuous action spaces due to the actor-critic design from DPG.

DPG [72] was proposed in an actor-critic framework and results in a more efficient gradient calculation compared to Stochastic Policy Gradient. SPG has to integrate over the entire state-action space in order to ensure exploration, this integration can be problematic when requiring high computational efficiency. DPG is able to be more efficient than SPG by eliminating this integral over the action space. This will result in a deterministic policy meaning the policy maps a state to a single action instead of a distribution. To ensure exploration with DPG it is implemented off-policy and actions are selected according to a stochastic behaviour policy. This stochastic behaviour policy essentially adds noise to the deterministic policy which helps with exploration. DPG will also result in a deterministic policy meaning the policy directly maps a state to a single action. The performance increase of DPG compared to SPG is significant by several orders of magnitude by exploiting the efficiency of deterministic policies. Since DPG is implemented with linear function approximators, an extension of the DPG method is necessary to make use of the advantages of non-linear function approximators.

DDPG [42] extends DPG by using DNNs as function approximators. The learning methods for the networks is then taken from DQN, making DPDG combine the efficiency of deterministic policies with the generalisation power of DNN. The improvement over DQN is the ability to deal with continuous action spaces.

The critic in DDPG maintains an action-value Q-function. This is the same as with DQN where the TD method is used to update the value function. The replay buffer and target network of the DQN critic are also present where the target network is adapted to the actor-critic structure of DDPG. The difference with the DQN target network is that there now is a target network for both the critic and the actor. This increases the chance of convergence and is usually implemented with a soft update. The actor uses the policy gradient theorem using the critic's TD estimate, using the DPG method and resulting in a deterministic policy $\pi(s)$. Like with DPG, exploration is ensured by making the algorithm off-policy and adding a stochastic behavioural policy. DDPG can however be hard to use in practice due to high sensitivity to hyperparameters [15] [27].

DDPG was implemented using a novel learning strategy for the control of Distributed Stream Data Processing Systems with the objective of minimising processing time. [41]. Both offline and online learning was applied where first an offline learning phase on a dataset of samples was performed. The agents were then further trained online whereas the offline pre-trained agent ensured exploration and faster online convergence. These results showed that it is possible to run DRL algorithms online, with an initial offline training phase recommended. For increasing state and action spaces, however, the convergence time would increase too much and cause safety-critical concerns for a high-dimensional flight control task.

DDPG has also been used for flight control of a 6-DOF UAS model in a tracking task [81]. The resulting policy was able to track roll, pitch and yaw angles, but with notable steady-state errors on single attitude control. On multiple coupled attitude control, however, the agent did not reach desired performance and improvements on DDPG like SAC have produced more promising results on a coupled flight control task [11].

Twin-Delayed Deep Deterministic Policy Gradient (TD3)
TD3 [20] improves on DDPG using three additional techniques. A problem that both DDPG and DQN have is an overestimation of the value function. This is due to the $\max$ operation on the Q-function function approximation and its sensitivity to noise in the value function estimation. This can result in slow convergence or even divergence of the parameters [78]. Double Q-Learning [24] tries to reduce the overestimation by having two Q-function networks and using the minimum of the two in computing the Bellman equation. TD3 implements this double (twin) Q-learning into DDPG. This might introduce underestimation bias as opposed to however this is preferred [20] over overestimation.

A second technique that TD3 uses is delayed policy updates. This updates the actor's policy and target network less frequently than the critic's Q-function. Since policy updates on high-error states cause divergent behaviour, updating the policy network at a lower frequency than the value network minimizes the error before performing a policy update and improves stability and convergence [13]. The third technique called target policy smoothing regularization tries to combat the over-fitting of the deterministic policy [20]. This is done by adding noise to the target action which smooths out the Q-function.

TD3 has successfully been applied to low-level control of a quadrotor in a point tracking task [70], and also for UAV path planning and obstacle avoidance [26]. A novel variant named Meta-TD3 was developed for a UAV target tracking task in [39], showing improved performance over TD3 and DDPG.

Soft Actor-Critic (SAC)
SAC [22] employs the same off-policy implementation of DDPG but uses a stochastic policy instead. SAC makes use of the maximum entropy reinforcement learning framework, in which the actor not only aims to maximize the expected return but also the entropy. The soft policy iteration principle is used by SAC which is a policy iteration method that includes an entropy term in the objective function. The expected return in the objective function is scaled against the entropy using the *temperature* hyperparameter. Soft policy iteration uses the same two steps of policy iteration: soft policy evaluation and soft policy improvement while trying to converge to an optimal policy. This means the actor chooses optimal actions while acting as randomly as possible ensuring exploration and avoiding convergence onto sub-optimal solutions. SAC then adds function approximation to soft policy iteration to increase the applicability to continuous-space problems. The SAC framework uses a double Q-network (and double target networks) just like TD3 to mitigate the overestimation bias. Unlike DDPG and TD3, no target policy is needed as the stochastic policy has a smoothing effect by itself.

The TD error δ_i for the critic can be seen in Equation 3.29. The double Q-function critics have weights w_1 and w_2, with w_1' and w_2' being the weights for the target critics Q'. Note that the TD error contains $V(s_{t+1})$. This is the so-called soft value function which consists of the minimum target Q-function, weighted against the entropy with the temperature coefficient η. The expression for the entropy follows from the definition of entropy for a given probability distribution P as seen in Equation 3.28. The final objective function to be minimised to train the critic parameters is defined as the soft Bellman residual in Equation 3.30. Here the expectation operator is taken over the current batch $\mathcal{B}$, sampled from the replay buffer $\mathcal{D}$.

$$\mathcal{H}(P) = \mathop{\mathbb{E}}_{x\sim P}[-\log P(x)] \rightarrow \mathcal{H}(\pi_\theta(\cdot \mid s_{t+1})) = \mathop{\mathbb{E}}_{a\sim\pi}[-\log \pi_\theta(a \mid s_t)] \tag{3.28}$$

$$\delta_i = r_{t+1} + \gamma V(s_{t+1}) - Q_{w_i}(s_t, a_t) \quad \text{with} \quad V(s_{t+1}) = \mathop{\mathbb{E}}_{a\sim\pi}\left[\min_{i=1,2} Q'_{w_i'}(s_{t+1}, a) - \eta \log \pi_\theta(a \mid s_{t+1})\right] \tag{3.29}$$

$$J_Q(w_i) = \frac{1}{2} \mathop{\mathbb{E}}_{(s_t, s_{t+1}, a_t)\sim\mathcal{B}}\left[(-\delta_i)^2\right] \tag{3.30}$$

For the policy improvement step, it was chosen by the authors to update the policy toward the exponential of the soft Q-value using the KL-divergence as the objective function to be minimised as seen in Equation 3.31. In this objective function, Z_w represents a normalizing function which can be ignored in the surrogate objective function that will actually be used to update the policy. This surrogate objective function as seen in Equation 3.32 is derived to have the same gradient with respect to θ. A final modification to the policy objective function is made by applying a re-parameterization trick on the action: $a_t = f_\theta(\xi, s_t)$ with ξ random noise sampled from distribution $\mathcal{N}$. This is done to ensure that sampling from the policy is differentiable and the gradient of the objective function can be calculated. This results in the final objective function for the policy in Equation 3.33.

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{s_t \sim \mathcal{D}} D_{KL}\left(\pi_\theta(\cdot \mid s_t) \middle\| \frac{\exp\left(\frac{1}{\eta} Q_w(s_t, \cdot)\right)}{Z_w(s_t)}\right) \tag{3.31}$$

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{s_t \sim \mathcal{D}}\left[\mathop{\mathbb{E}}_{a_t \sim \pi}\left[\eta \log \pi_\theta(a_t \mid s_t) - \min_{i=1,2} Q'_{w'_i}(s_t, a_t)\right]\right] \tag{3.32}$$

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{s_t \sim \mathcal{D}}\left[\mathop{\mathbb{E}}_{\xi \sim \mathcal{N}}\left[\eta \log \pi_\theta(f_\theta(\xi, s_t) \mid s_t) - \min_{i=1,2} Q'_{w'_i}(s_t, f_\theta(\xi, s_t))\right]\right] \tag{3.33}$$

Earlier implementations of SAC have modelled the soft value function as a separate learnable network. This was done to improve learning stability. Later iterations however introduced a number of improvements like the re-parameterization trick for the policy and a separate learning process for the temperature coefficient [23]. The learning of the temperature coefficient greatly improves the usability of SAC as manually optimizing this hyperparameter is non-trivial. The extra learning network for the soft value function was subsequently omitted for the sake of simplicity as it empirically showed to have little effect on learning stability.

SAC was already successfully applied to UAV navigation and control problems. UAV path planning in [9] used a SAC agent Another UAV path planning application in [38] introduced the SACHER algorithm (soft actor-critic with hindsight experience replay). This method implements a modified reward function that also depends on a goal, helpful for increasing learning speed and success rate in path planning problems. Furthermore, low-level control of a quadrotor was achieved using a SAC agent in a go-to-target task [4].

3.4.5. State-of-the-art DRL Applications

This section goes over some state-of-the-art DRL flight control applications in more detail. These particular applications are of the most interest for the continuation of this research and further elaborate on the benefits and drawbacks of the DRL algorithms and recommendations for further improvement in flight control applications.

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

The author of [11] implemented a coupled, cascaded SAC controller for control of the Cessna Citation 500 aircraft. The SAC algorithm was directly adapted from the author's implementation [23]. The observation vector available to the agent was chosen to enable the 6-DOF tracking tasks and contains the weighted tracking error vector, the three-body rates and the current control input since the agent only controls the control increments. The reason for the latter was that it was found that control inputs appeared noisy when they corresponded directly to the agent, so the agent was implemented to provide the control increments instead as can also be seen from the controller diagram in Figure 3.9.

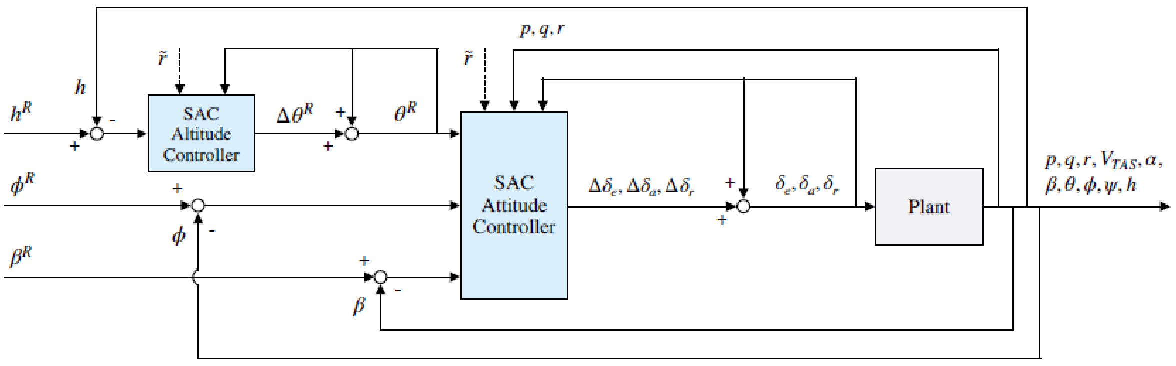


Figure 3.9: Schematic of the cascaded, coupled SAC flight controller design from [11]

Evaluation of several coordinated manoeuvres and tracking tasks demonstrated the ability of the SAC controller to provide robust online control. Fault tolerance was also demonstrated on a number of failure cases such as jammed rudder at $-15°$, reduced aileron effectiveness by 70%, sudden c.g. shift and simulated icing effects. The response of the c.g. shift failure case can be seen in Figure 3.10 where the SAC agent was also trained on the failed system to evaluate possible improvements compared to a fully robust response. In general it was concluded that training on the normal system should be sufficient to provide robust fault-tolerance. The resulting high robustness of control is attributed to the high exploration power of the stochastic policy and the high generalisation power of the DNNs. Although the online performance of the cascaded SAC controller was shown to be performant and stable, the offline training process could be unreliable due to the increased stochastic effects of the soft policy iteration and the stochastic policy.

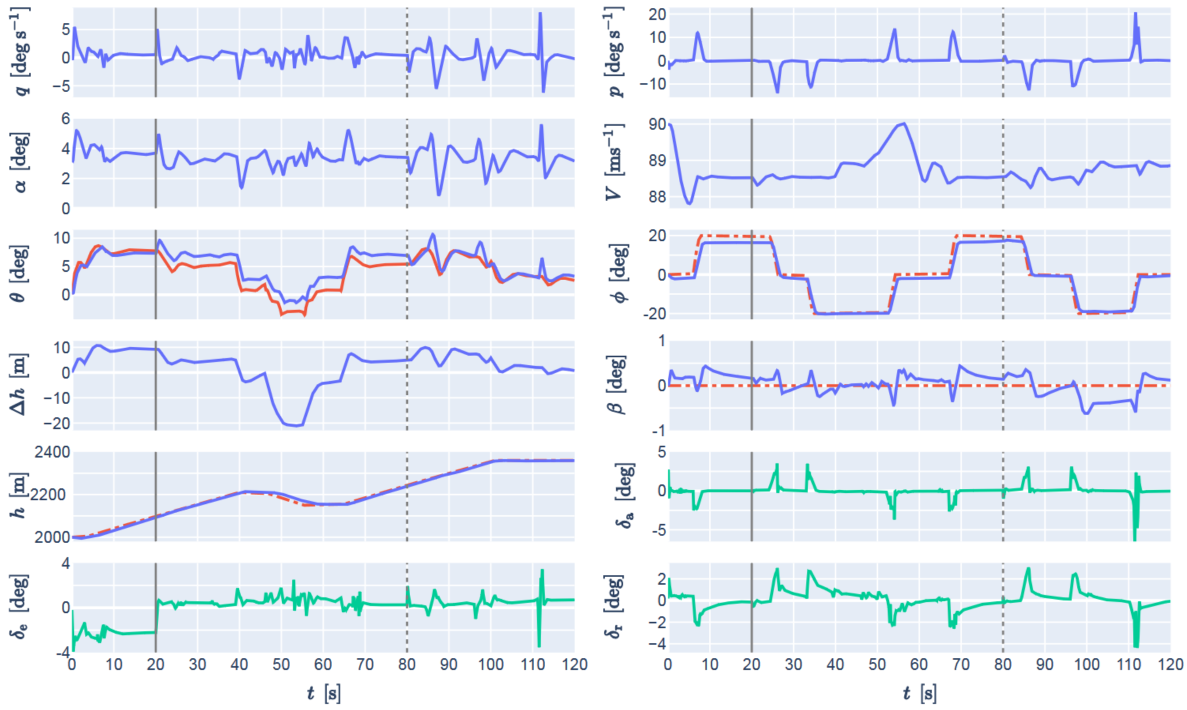


Figure 3.10: Altitude tracking response of cascaded SAC controller with sudden c.g. shift at 20s. Response of controller trained on normal system until 80s, afterwards response of controller trained on failed system [11]

Recommendations from this paper include further analysis of realistic effects such as transport and sensor delays, continuous atmospheric disturbances and reference signal disturbances. Furthermore, the authors recommend looking into increasing training reliability by further hyperparameter tuning or considering deterministic on-policy alternatives such as TD3 and PPO. An interesting recommendation by [11] on increasing the adaptiveness of control is to investigate a hybrid framework. It was proposed that incorporating iADP into the SAC framework would enable online learning using a state-of-the-art iADP algorithm like IDHP. This hybrid offline-online learning approach could make the cascaded SAC controller online adaptive, increasing the applicability to autonomous systems. Cascaded pure IDHP controllers on the other hand have shown to have high sensitivity to measurement noise and reduced adaptivity compared to an IDHP with a PID outer loop [37].

SAC, TD3 | Self-learned suppression of roll oscillations based on model-free reinforcement learning

In [14], a SAC and TD3 agent were implemented to learn policies for controlling the high angle of attack roll oscillations on a flying-wing model. The training was performed offline in simulation using a widely used wing-rock mathematical model. The paper also compares the performance with a real-world wind-tunnel test on a flying wing model subjected to spanwise blowing. As can be seen from the training reward curves in Figure 3.11, both TD3 and SAC algorithms eventually converged into a near-perfect policy. The difference in exploration strategy and training stability can also be seen here where TD3 has a stabler training curve. The stochastic policy and entropy terms of SAC cause more variance in the reward curve.

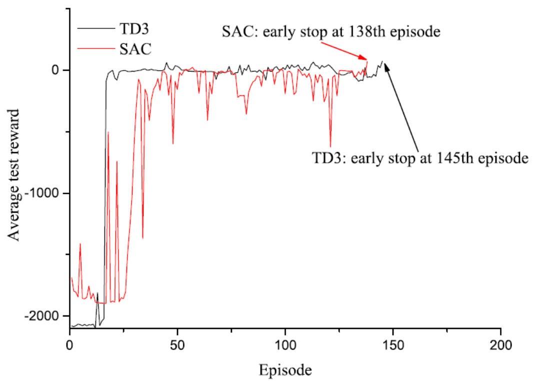


Figure 3.11: Reward curves comparing TD3 and SAC on suppression of roll oscillations task [14]

The wind tunnel experiments performed on the flying-wing model showed that time delay introduced non-Markovian effects into the environment with degraded performance on both agents. Attempting to improve this, the authors trained both agents on the experimental setup with different observation vectors. The memory size of the observation vector was increased from only the current time-step observation like in simulation, to three time-steps and six time-steps. The reward curves of the experimental training can be seen in Figure 3.12 for TD3 and Figure 3.13 for SAC with $m = 1$, $m = 3$, $m = 6$ for the three different observation vector sizes. These results showed that increasing the observation memory size improves the reward on both algorithms, both reaching a similarly high reward with three and six time steps per observation. Again, the increased stochastic effects of SAC are demonstrated by the reward curves. Note however that more unstable training performance of SAC does not translate when deployed on the system as the best agent from training is always used.

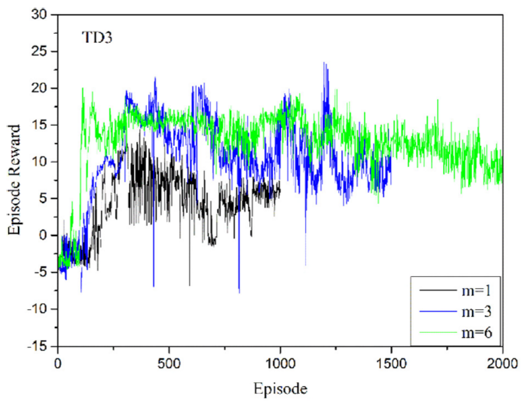


Figure 3.12: Reward curves for TD3 on suppression of roll oscillations task for increasing observation vector sizes [14]

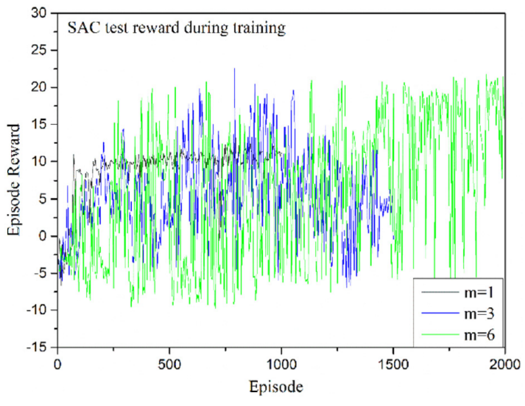


Figure 3.13: Reward curves for SAC on a suppression of roll oscillations task for increasing observation vector sizes [14]

3.5. Simulation Reality-Gap in Reinforcement Learning Flight Control

In order to progress towards enabling flight tests for RL controllers on the PH-LAB research aircraft, the main challenges in reducing the simulation reality-gap need to be identified. Reality-gap is not a concept exclusive to reinforcement learning but to any control technique. It generally points to the challenges associated with transferring simulated experience to the real world as there will always be a discrepancy between the simulated world and the real world. Applying reinforcement learning methods to the real world remains a major challenge. Especially in flight control applications, the requirements for safety are high. A simple solution to the problem of reality-gap and domain transfer might be to fully train the agent on the real system. This is however not feasible as this gets exceedingly expensive and unsafe due to high sample complexity and unsafe exploration, especially for complex systems like aircraft. In practice, learning from the simulation is thus necessary before deploying in production. The exploration process of reinforcement learning methods and learning instability and unpredictability already pose a challenge for safety-critical design. Being aware and addressing the reality-gap in the performance and stability assessments is thus of great importance when developing control applications for real-world tasks.

The current state-of-the-art implementations for CS-25 class aircraft RL controllers [32] [28] [11] all recommend further research into the effects of less simplified sensors, actuators and plant dynamics on the performance and stability of the RL controller. In developing and testing the proposed RL controller of this research, the reality-gap challenges that previously were already identified should be investigated and evaluated in order to judge the readiness of the RL controller for flight tests, compare the RL framework candidate to other state-of-the-art implementations and identify possible causes of any performance degradation.

This section contains the answer to research question *RQ1.2* and further decides on the steps that have to be taken in the implementation phase in order to attempt to develop a practically improved RL controller compared to existing implementations.

3.5.1. Common Causes and Solutions to Simulation Reality-Gap in Reinforcement Learning

Reality-gap in reinforcement learning is caused by a variety of factors. One of the factors causing a discrepancy between the simulation and the real world is differences in the system dynamics. The model of the environment used in simulation often contains assumptions and errors compared to the real-world system. A bias to the simulation model can develop a bias called the Simulation Optimization Bias [52] in which the optimizer exploits these modelling errors of the simulator. A bias to the simulation model can potentially result in damaging behaviour when deployed on the real system. Note that the offline learning DRL methods considered in this research focus on robustness of control and are expected to learn from a model that is not exactly the real system, but a bias to the simulation model is thus always present. In these cases it is also important to be aware of over-fitting the agent to the simulation model, decreasing the robustness when translated to the real work system.

A second major factor that causes the reality-gap is the time delays present in a real-world MDP, as illustrated by Figure 3.14. The MDP that is assumed in a traditional RL problem assumes the capturing of the state and the policy inference are instant processes. In reality, time delays are present for the state observation in the form of sensor time delays. This essentially results in the agent making decisions based on lagged observations with a policy of the form $\pi(a_t \mid s_{t-\delta})$ with δ the observation time delay. Also, the feed-forward process of the NNs can take more time when deployed on real hardware causing a delay in the policy inference. As discussed in subsection 3.5.2, time delay has already been identified as a significant cause of performance degradation and is important to include in the simulation model.

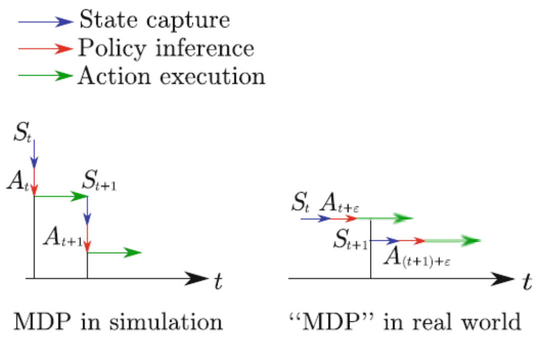


Figure 3.14: Difference of MDP in simulation and reality due to time delays of state capture and policy inference [13]

A number of techniques can be used to close the reality-gap. The techniques that are discussed here fall into three main categories of domain adaptation, domain randomization and online system identification.

Domain adaptation refers to adapting the policies online when transferring a pre-trained agent to a new, real-world target domain. Meta-learning [2] [53] [67] is an example of a method described by ”learning to learn” that achieves online learning by learning and tuning hyperparameters of the RL framework online. Meta-learning has been applied to a number of DRL algorithms like Meta-TD3 [39] and Meta-SAC [87] where the latter improves on learning stability of SAC by replacing the temperature coefficient optimizer usually implemented in SAC by a meta-learning process.

Domain adaptation techniques can be seen as separate from fully online learning techniques such as iADP. In iADP frameworks, the process of online system identification provides a safeguard against the model bias of pre-trained agents but comes with its own challenges due to the continual learning process [28] [34]. These techniques as discussed in section 3.3 can then adapt their policies online by using the incremental model to predict the state transition.

Domain randomization is an alternative technique to domain adaptation in which no online learning is performed, also called a zero-shot transfer [13]. Domain randomization makes sure the discrepancies between the simulation and target domain are modelled during the offline training phase. This is performed by implementing randomization and variance onto characteristics of the source domain. The agent can for instance be trained on a set of different simulation models, combating the over-fitting to one specific environment model and resulting in a policy that has more generalization power and robustness. This case can also be called dynamics randomization [58].

3.5.2. Modelling Requirements

The DASMAT tool by the Delft University of Technology for the Cessna 500 Citation I [21] can be used in the development of an RL flight controller candidate in the context of this research. This high fidelity, nonlinear, 6-degrees-of-freedom model can be considered equivalent to the Cessna 550 Citation II PH-LAB aircraft, despite the difference in fuselage size, engine power and wing size [83]. The DASMAT model includes flight dynamics as well as engine dynamics in high-fidelity versions of the actuator and sensor models.

The following state-of-the-art research has been discussed in more detail in subsection 3.3.2 and subsection 3.4.5 and all implement RL flight control focusing on the CS-25 class of aircraft.

In the field of iADP, a number of these recent studies have performed analyses on the reality-gap. The IDHP decoupled 6-degree-of-freedom flight controller developed in [28] assumed completely clean measurements and used a relatively low-fidelity actuator model, modelled as a first-order lag system with saturation limits but no transport delays. Also, no external disturbances such as atmospheric turbulence were tested. The IDHP partially coupled 6-degree-of-freedom flight controller from [34] similarly assumed completely clean measurements and no external disturbances. In [32], an IDHP inner loop was developed and analysis was done using a high fidelity sensor model which included sensor bias, noise, delays, resolution and sampling rate. The actuator model also modelled transport delays and no atmospheric disturbances were implemented. In the implementation in [37], the IDHP cascaded longitudinal controller only models the sensor noise, while the effect of external disturbances is analysed using a longitudinal Dryden gust model. In the field of DRL, the coupled cascaded 6-degree-of-freedom SAC controller from [11] takes into account sensor noise and bias, but no sensor delay. A simple low-pass filter actuator model with saturation limits is used with no transport delays. Additionally, the effects of a simple, non-continuous vertical gust model are reported.

As per the research objective, the proposed RL flight controller design should be tested using as many of the available high-fidelity dynamic models as possible. The main three elements of the environment model, sensor models, actuator models and external disturbances, should be implemented to the following degree.

Sensor Model
A high fidelity sensor model for the PH-LAB Cessna Citation II is available in the DASMAT analysis tool, based on flight test data [21]. This sensor model takes into account many of the practical phenomena that are neglected by most of the previous work mentioned and suspected to have a measurable impact on performance and learning stability. As per the recommendation of the authors of the previous works, these effects should be further investigated before attempting real-world flight tests. The phenomena include sensor noise, bias, delay, resolution and sampling rate. As mentioned before, these phenomena have previously been investigated in [32] on an inner loop IDHP longitudinal controller. This work concluded that the discretization process and sensor bias did not have any significant effect on the controller performance or on the incremental model identification. Sensor noise and time delays however were found to cause controller performance degradation due to incorrect estimation of the incremental model parameters. The negative effects of sensor noise were largely mitigated by appropriately filtering the sensor signals using a first-order low-pass filter. Sensor delays however are recommended to be studied further.

Actuator Model
As mentioned before, a high-fidelity actuator model for the PH-LAB Cessna Citation II is available in the DASMAT analysis tool [59]. This model provides a better estimation for the elevator and ailerons, including the saturation limits and transport delays. In [32], the transport delays were modelled using a simple first-order model and concluded that it did cause any significant performance degradation. As this was implemented only on an inner loop IDHP longitudinal controller, it is still desired to not neglect any of these effects to extend this analysis to a 6-degree-of-freedom environment.

3.6. Conclusion Literature Study

Before commencing a preliminary analysis and fully answering *RQ1*, conclusions have to be made on the type of RL framework to further investigate. The state-of-the-art in online and offline learning RL frameworks have been identified by the preceding literature survey to be in the field of iADP and DRL. In subsection 3.6.1, the advantages and disadvantages of algorithms in these two families are compared. A decision is made on which framework will be considered for the rest of this thesis and that could contribute the most to the further development of adaptive and robust control inside the scope of this thesis. Conclusions based on the reality-gap challenges can be found in subsection 3.6.2

Continuing the preliminary research, the next steps include a preliminary analysis by implementing RL framework candidates and assessing their performance on simple systems according to the points discussed in subsection 3.6.1, answering *RQ3* and *RQ4* respectively.

3.6.1. RL Framework Candidate

Following the literature study, the two main families considered for the RL framework candidate are iADP and DRL, detailed in section 3.3 and section 3.4 respectively which provide an answer to *RQ1.1*. These two families of RL algorithms contain state-of-the-art implementations that enable model-free learning, both online in the case of iADP and offline in the case of DRL. First, both families will be considered as they both have significant advantages and disadvantages. Then the most promising implementations can be compared and a choice can be made on how to continue in the preliminary implementation phase. The state-of-the-art implementations that are appropriate for the continuous flight control task at hand have been identified to be IDHP in the category of iADP, and SAC and TD3 in the field of DRL.

IDHP is the more recent development in ADP and provides fully model-free online learning with better performance and less complex implementation than other model-free options like ADHDP, IHDP and IGDHP. As discussed in subsection 3.3.2, IDHP has recently been studied in applications for flight control and on the Cessna Citation type aircraft, most notably the decoupled 6-degree-of-freedom flight controllers from [28] and [34]. In DRL, fewer examples exist of specific, modern aerospace applications. Mainly the cascaded SAC controller of [11] stands out as a complete 6-degree-of-freedom fault tolerant flight control application, with promising results. The choice to also consider TD3 comes from the promise of a more stable and consistent learning experience.

TD3 as a state-of-the-art DRL algorithm shows promising characteristics for flight control applications. Theoretically, the less aggressive exploration and deterministic policy form the basis for a stable, consistent and possible even safe online training process. The more favourable training stability compared to SAC has also been demonstrated in a study on roll oscillations [14]. The lower exploration of TD3, as part of the exploration-exploitation trade-off, has the drawback of resulting in a less robust controller. This means most likely a lower fault tolerance, a larger reality-gap and increased bias to the training model. Despite the attractive training behaviour and efficiency, TD3 has, after a thorough search of the relevant literature, not yet been found to have demonstrated a full, autonomous, fault-tolerant flight controller like SAC has in [11]. The presence of the latter study using SAC makes it a more attractive candidate following the authors promising results and recommendations.

SAC has due to its off-policy stochastic policy and soft policy iteration a high level of exploration. Together with the high generalization power of Deep Neural Networks (DNNs), SAC has been shown to provide robust, fault-tolerant control [11] in a cascaded inner and outer loop flight controller. A large area of improvement for SAC the training stability. Due to the stochastic nature of the learning process, training runs can be inconsistent and hard to compare. Note that this drawback pertains mainly to the development phase of a controller, while deployed in production, the best policy from training is used, usually made deterministic by using the mean of the distribution. The low sample efficiency however requires that realistically the training has to be conducted offline. Online learning with SAC also poses a safety concern due to the stochastic policy and unsafe exploration coupled with the longer converge time. For that, incremental methods like IDHP are much better suited. Even though the high robustness has been demonstrated by [11] to provide fault tolerance, severe failure cases cause instability.

IDHP has the advantage of not needing any offline training phase. It also has a higher sample efficiency than any DRL method and converges fast, hence its applicability to online learning. While SAC or other DRL methods attempt to achieve fault tolerance through their generalization power, and robust control, the fault tolerance of iADP methods like IDHP is assumed to always be higher due to the online adaptive approach. The online system identification and lack of an offline training phase also reduce the problems that are present with deploying pre-trained policies, like over-fitting and the policy possible exploiting errors in the training model. The continual online learning approach has however problems of its own, main pertaining to a monotonic increase in the magnitude of network parameters and estimator windup, as acknowledged by [28]. Also as discussed in subsection 3.3.2, a cascaded IDHP controller design has, unlike the cascaded SAC design, demonstrated high sensitivity and degraded performance due to real-world effects like sensor noise. Using some form of a manually tuned outer loop like PID has until now been used to complement an IDHP inner loop where results were promising.

SAC and IDHP so far seem to be most promising in the context of this research. The state-of-the-art implementations of both can benefit from further iteration and in-depth comparison, including gathering more data on the effects of noise, delays and disturbances. Additionally, as recommended by [11], the benefits of SAC could potentially be combined with the online learning of an IDHP actor-critic. A framework that utilizes both the generalization power of DNNs and the adaptive power of incremental system identification could result in a robust and highly fault tolerance design. Attempts by [11] to implement online learning using the SAC learning process have had a stability success rate of only 6%. The successful implementation of the cascaded SAC controller on a complex system indicates that a hybrid form is an option worth investigating when cascaded IDHP-only controllers have had less satisfactory results [37]. Hence, an offline-learning SAC and online-learning IDHP training strategy can be developed as part of the preliminary implementation in order to explore the possibility of a hybrid form further.

The discussion on a hybrid training strategy by [11] describes the IDHP agent as an add-on agent to the pre-trained SAC agent during flight. As soon as a severe fault or change in the environment dynamics is detected, the IDHP agent is expected to use its high sample efficiency to regain control by building the incremental model with updated dynamics. The question of how a mechanism to detect the failure and switch the agent would be implemented is still to be answered.

Policy fine-tuning as discussed in [94] is another technique that combines offline with online learning. Instead of a binary approach between the online and offline agent, policy fine-tuning uses the online agent continually, with access to a reference policy that is close to the optimal policy. In this case, the robust policy from the SAC agent can be a reference policy for the online IDHP agent. Whether this is possible for the SAC-IDHP combo or applicable to high-dimensional coupled systems remains to be seen.

In conclusion, both the SAC and IDHP RL frameworks are a candidate for preliminary analysis and are subject to further improvement and comparison. Additionally, an attempt at a hybrid offline-IDHP online-IDHP form is expected to provide insight into the possibility of combining the advantages of both.

3.6.2. Reality-Gap Challenges

In section 3.5 the main challenges concerning the simulation reality-gap are identified, answering *RQ1.2*. Based on this and using the insights of previous papers [28] [34] [32] [37] [11] it is concluded that using of the high fidelity sensor and actuator models available for the PH-LAB [21] have to be used to progress towards enabling flight tests of RL controllers as per the research objective. Special attention has to be given to sensor time delays [32] as discussed in subsection 3.5.1. Additionally, a continuous atmospheric disturbance model can be used to further assess robustness to different flight scenarios. Also testing a wider range of flight configurations of the PH-LAB simulation model can be beneficial to increase the flight envelope of the controller. In implementing the RL framework candidate, a number of domain adaptation techniques can be used to further close the reality gap. It should for instance be considered to investigate the effect of replacing the temperature coefficient optimizer with meta-learning in SAC [87] which can also increase learning stability.

4 Preliminary Analysis

Following from the conclusions of chapter 3, a preliminary analysis is performed in this chapter which aims to answer *RQ1.3* and *RQ1.4*, fully answering *RQ1* together with the answers to *RQ1.1* and *RQ1.2* given in chapter 3. Two main RL framework candidates have been identified from the fields of Incremental Approximate Dynamic Programming and Deep Reinforcement Learning: Incremental Dual Heuristic Programming (IDHP) and Soft Actor-Critic (SAC) respectively. The initial implementation of both these frameworks is documented here, plus the implementation of a combined IDHP-SAC framework. The analysis that follows using each of the three frameworks is performed on a simple dynamic system modeling the short-period mode of the Cessna Citation aircraft. This analysis aims to compare the performance of the different frameworks and should confirm that the RL agents can successfully control a simple system and exhibit the adaptive and/or robust qualities needed for fault-tolerance. Note that analysis on the reality-gap challenges including measurement noise, time delays and high fidelity dynamics modeling is reserved for the main phase of this thesis research.

In section 4.1, the short-period environment model is discussed, which is used for the subsequent simulations. Next, the implementation details of the RL frameworks are given in section 4.2 for the IDHP agent, section 4.3 for the SAC agent and section 4.4 discusses the implementation of a combined approach. The results of several simulation runs on the short-period environment model are given in section 4.5 including a discussion on how the different agents compare. Finally, the conclusions are given in section 4.6.

4.1. Environment

The environment in the setting of RL control needs to define the dynamics of the controlled system, usually called the plant in engineering terms. Since the development of the RL controller is fully performed in simulation, a mathematical model of the environment needs to be defined. This section presents the environment model that is used during the preliminary analysis. The control task that the RL agents need to optimize is defined as part of the reward function design, which in this case is considered part of the environment model.

4.1.1. State-Transition Model

The state-transition model defines the dynamics of the system. For the preliminary analysis, a Linear Time-Invariant (LTI) short-period model is selected which models the short-period mode of an aircraft and can be parameterized to approximate the dynamics of the PH-LAB Cessna Citation aircraft. This simplified model provides a relevant system to perform the initial analysis on while keeping the complexity low due to the smaller state and action spaces compared to a full aircraft model.

The mathematical short-period model is derived from [51] and represented as a state-space system. The dynamics equation for the state-space system can be seen in Equation 4.1 with the discrete form being executed every time-step Δt of the simulation. The state vector $\mathbf{x}$ and action vector $\mathbf{a}$ are defined as in Equation 4.2 where the state only contains the angle of attack α and the pitch rate q. The action or input vector contains the elevator deflection δ_e.

The state matrix A and input matrix B are defined as in Equation 4.3 where the values contain a number of aerodynamic coefficients. The parameters of A and B are available for the Cessna Ce500 aircraft, which can be considered similar to the PH-LAB aircraft for this simplified model. In Table 4.1 the aerodynamic coefficients of the Ce500 in cruise conditions can be seen. This parameterized form of the state-space equation can later be used to test the robustness of the offline-trained SAC controller by altering the aerodynamic coefficients. Note that the RL agents have no prior knowledge of these system matrices, including the incremental model of the IDHP framework which forms its own approximation of these matrices as part of the learning framework.

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{a} \quad \rightarrow \quad \mathbf{x}_{t+1} = \mathbf{x}_t + (A\mathbf{x}_t + B\mathbf{a}_t)\,\Delta t \tag{4.1}$$

$$\mathbf{x} = \begin{bmatrix} \alpha \\ q \end{bmatrix} \quad \mathbf{a} = [\delta_e] \tag{4.2}$$

$$A = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{Z_\alpha}}{2\mu_c - C_{Z_{\dot{\alpha}}}} & \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V^2}{\bar{c}^2} \frac{C_{m_\alpha} + C_{Z_\alpha} \frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} & \frac{V}{\bar{c}} \frac{C_{m_q} + C_{m_{\dot{\alpha}}} \frac{2\mu_c + C_{Z_q}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix} \quad B = \begin{bmatrix} \frac{V}{\bar{c}} \frac{C_{Z_{\delta_e}}}{2\mu_c - C_{Z_{\dot{\alpha}}}} \\ \frac{V^2}{\bar{c}^2} \frac{C_{m_{\delta_e}} + C_{Z_{\delta_e}} \frac{C_{m_{\dot{\alpha}}}}{2\mu_c - C_{Z_{\dot{\alpha}}}}}{2\mu_c K_Y^2} \end{bmatrix} \tag{4.3}$$

Table 4.1: Short-period stability and control derivatives for the Cessna Ce500 in cruise [51]

V	=	$59.9\ m/s$	C_{Z_α}	=	-5.16	C_{m_α}	=	-0.43
$\bar{c}$	=	$2.022\ m$	$C_{Z_{\dot{\alpha}}}$	=	-1.43	$C_{m_{\dot{\alpha}}}$	=	-3.7
μ_c	=	102.7	C_{Z_q}	=	-3.86	C_{m_q}	=	-7.04
K_Y^2	=	0.98	$C_{Z_{\delta_e}}$	=	-0.6238	$C_{m_{\delta_e}}$	=	-1.553

4.1.2. Reward and Observation Model

The reward function defines the type of task that the agent aims to optimize. In this case, the task is defined as a reference tracking task, more specifically, an angle of attack reference tracking task. An alternative choice can be to define a pitch rate tracking task instead of tracking the angle of attack. The latter has increased learning difficulty because the elevator input δ_e has a more direct relation to the pitch rate than to the angle of attack. It is thus expected that the tracking performance would be better on a q-tracking task. It is however preferred to demonstrate that the proposed RL controller can handle a more indirect dynamic relation, hence the choice for an angle of attack tracking task.

The reference tracking signal is chosen to be a simple combined sinusoidal as seen in Equation 4.4. The lowest frequency signal corresponds to a sine wave with a period of 20s, while a second sine wave is added to the first with a period of 10s. The amplitude of both signals is set to $5°$, resulting in a maximum amplitude of $10°$ for the combined reference signal.

$$\alpha_{ref} = \frac{5\pi}{180} \sin\left(\frac{2\pi}{20} t\right) + \frac{5\pi}{180} \sin\left(\frac{2\pi}{10} t\right) \tag{4.4}$$

The reward function is then defined as the negative squared tracking error as can be seen in Equation 4.5. The scaling constant κ can be considered a hyperparameter in order to tune the aggressiveness of the control policy. In DRL frameworks like SAC this reward value is used by the agent, in DHP however the update rules require the state derivative of the reward $\frac{\partial r_{t+1}}{\partial x_{t+1}}$. Since the state vector and reward function are already defined, the state derivative of the reward can be calculated as in Equation 4.6.

$$r_{t+1} = -(\alpha_{ref,t} - \alpha_t)^2 \cdot \kappa \tag{4.5}$$

$$\frac{\partial r_{t+1}}{\partial x_{t+1}} = \begin{bmatrix} \frac{\partial r_{t+1}}{\partial \alpha} & \frac{\partial r_{t+1}}{\partial q} \end{bmatrix} = \begin{bmatrix} -2(\alpha_{ref,t} - \alpha_t) & 0 \end{bmatrix} \cdot \kappa \tag{4.6}$$

The observation vector s_{t+1} of the short period model can be seen in Equation 4.7. This observation vector acts as the input for the actor and critic networks and on control tasks, it is usually defined using a combination of the states and reference tracking errors. In this case, the observation vector only contains the tracking error as it is expected to contain sufficient information in the context of the simplified short-period model [36] [34].

$$s_{t+1} = \begin{bmatrix} (\alpha_{ref,t} - \alpha_t) \end{bmatrix} \tag{4.7}$$

4.2. IDHP Agent

This section discussed the implementation of the IDHP framework. The IDHP agent is implemented in TensorFlow 2 using a number of previous implementations as reference [36] [34] [28].

In Figure 4.1, a detailed high-level overview of the flow of the IDHP framework can be seen. The solid lines represent the data flow between the elements on every time step, while the dashed arrows represent an update operation. The corresponding pseudo-code of the implemented IDHP algorithm can be seen in Algorithm 1. Note the environment in this case does not output the reward value, but the reward gradient with respect to the state. This is to accommodate the update rules of the actor and critic where a critic outputs the gradient of the value function as per the definition of DHP. Also note the distinction between the environment *state* x_t and the *observation* s_t. The state x is the vector that defines the state used in the state-transition model. The incremental model requires the state (increment) as input as it aims to approximate the state-space matrices. The policy and critic networks on the other hand accept the observation vector as input which can equal the state vector, but often contains additional elements such as the reference tracking errors. For the experiments conducted here, the observation vector is not equal to the state and consists of the reference tracking error as discussed in section 4.1. The following sections go over the three main elements of the IDHP framework, the actor, critic and incremental model, and discuss how they are implemented. Additional details on the neural network architectures of the actor and critic are given as well. Finally, the training strategy used in the preliminary analysis simulation runs is discussed. For the results and comparison of the simulations, see section 4.5.

Algorithm 1 IDHP framework, adapted from [28], [36]

```
Initialize:
    Hyperparameters as defined in Table 4.2
    Actor and critic parameters $\theta$, $w$ and $w' \leftarrow w$
    Incremental model parameters $\Theta$, $F \leftarrow \Theta$, $G \leftarrow \Theta$, $\Lambda$, $X$
    Environment state and observation $x$, $s$
for each time step t do
    Sample and execute action $a_t \leftarrow \pi_\theta(s_t)$
    Observe next observation, state and reward derivative $s_{t+1}$, $x_{t+1}$, $\frac{\partial r_{t+1}}{\partial x_{t+1}}$
    if $RMSE > RMSE_{thresh}$ then
        Update learning rates $\eta_a \leftarrow \eta_{a,high}$, $\eta_c \leftarrow \eta_{c,high}$
    else
        Update learning rates $\eta_a \leftarrow \eta_{a,low}$, $\eta_c \leftarrow \eta_{c,low}$
    end if

    Compute actor loss gradient w.r.t $\theta$ as in Equation 4.15: $\nabla_\theta L_\pi(\frac{\partial a_t}{\partial \theta}, \frac{\partial r_{t+1}}{\partial x_{t+1}}, \lambda'_{t+1}, G_{t-1})$
    Compute critic loss gradient w.r.t $w$ as in Equation 4.21: $\nabla_w L_\lambda(\frac{\partial a_t}{\partial x_t}, \frac{\partial r_{t+1}}{\partial x_{t+1}}, \lambda_t, \lambda'_{t+1}, F_{t-1}, G_{t-1})$
    Update actor weights $\theta \leftarrow \theta - \eta_a \nabla_\theta L_\pi$
    Update critic weights $w \leftarrow w - \eta_c \nabla_w L_\lambda$
    Update target critic weights $w' \leftarrow \tau w + (1-\tau) w'$

    Update incremental model according to Equation 4.9 and $F_{t-1} \leftarrow \Theta$, $G_{t-1} \leftarrow \Theta$
    if $\epsilon > \epsilon_{thresh}$ then
        Covariance reset $\Lambda \leftarrow \Lambda_0$
    end if
end for
```

In the preliminary analysis tracking task, $RMSE$ and $RMSE_{thresh}$ are equivalent to α_{RMSE} and α_{thresh} respectively.

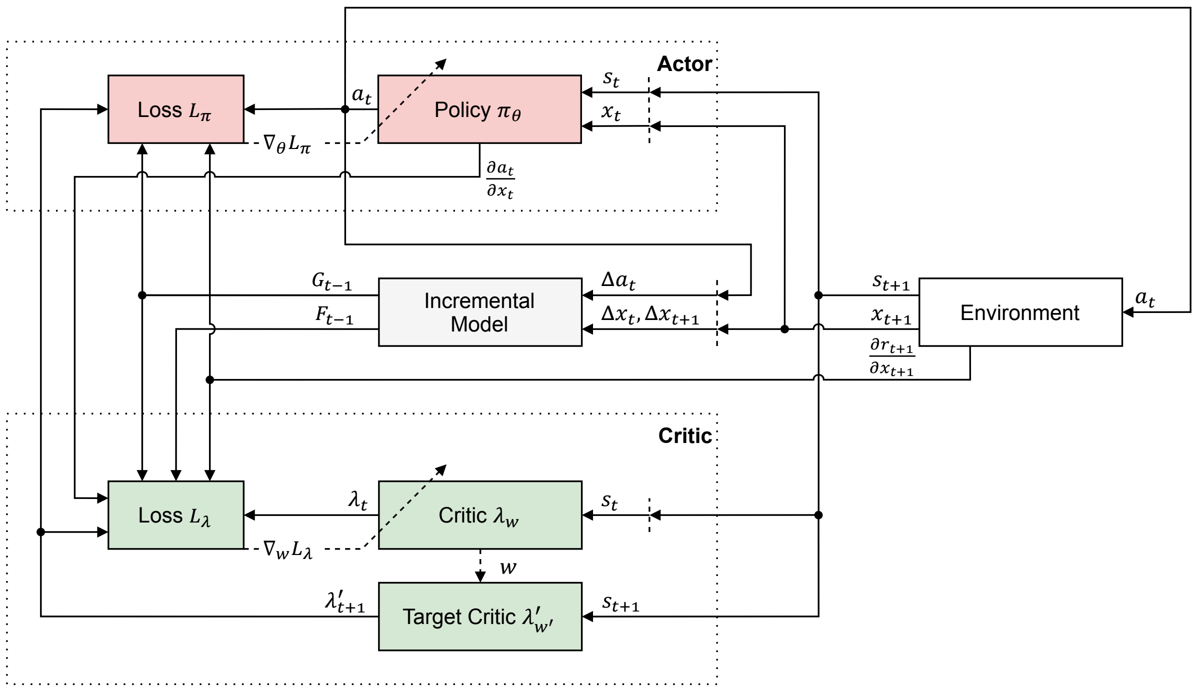


Figure 4.1: IDHP Framework, online on-policy

4.2.1. Incremental Model

The incremental model provides a future estimate of the environment state to be used in the update rules for the actor and critic. This model is derived from a first-order Taylor series expansion [97] and can be seen in Equation 4.8. Here the state matrix F_{t-1} and input matrix G_{t-1} are time-varying and are updated every time-step using a RLS estimator.

$$\Delta x_{t+1} = F_{t-1}\Delta x_t + G_{t-1}\Delta a_t \tag{4.8}$$

The RLS update rule of the incremental model can be seen in Equation 4.9 with Θ the parameter matrix as defined in Equation 4.10 and $\gamma_{RLS} \in [0,1]$ the forgetting factor. The measurement matrix X contains the increments of the previous state and action as seen in Equation 4.11. The error or innovation ϵ is defined in Equation 4.12 and represents the prediction error between the actual state and the predicted state. Finally, the covariance matrix Λ estimates a measure of the covariance of the parameter estimates and is updated according to Equation 4.13. Both the parameter matrix and covariance matrix are expressed recursively and thus need an initial value. In this case, the parameter matrix is initialized as zero's and the covariance matrix as an identity matrix of magnitude Λ_0 as no prior knowledge of the parameter covariances is assumed. The magnitude $\Lambda_0 = 1e8$ is set to a large value as the uncertainty of the parameters is high at the initial stage. The state and input matrices of the incremental model are used in the update rules of both the actor and the critic.

$$\Theta_t = \Theta_{t-1} + \frac{\Lambda_{t-1}X_t}{\gamma_{RLS} + X_t^T\Lambda_{t-1}X_t}\epsilon_t \tag{4.9}$$

$$\Theta_{t-1} = \begin{bmatrix} F_{t-1}^T \\ G_{t-1}^T \end{bmatrix} \tag{4.10}$$

$$X_t = \begin{bmatrix} \Delta x_t \\ \Delta a_t \end{bmatrix} \tag{4.11}$$

$$\epsilon_t = \Delta x_{t+1}^T - \Delta\hat{x}_{t+1}^T = \Delta x_{t+1}^T - X_t^T\Theta_{t-1} \tag{4.12}$$

$$\Lambda_t = \frac{1}{\gamma_{RLS}}\left[\Lambda_{t-1} - \frac{\Lambda_{t-1}X_tX_t^T\Lambda_{t-1}}{\gamma_{RLS} + X_t^T\Lambda_{t-1}X_t}\right] \tag{4.13}$$

4.2.2. IDHP Actor

The actor in the IDHP framework consists of the parameterized policy π_θ with parameters θ and a loss function L_π which calculates the gradients used to update the policy network. The policy receives the current observation from the environment s_t as input and outputs an action a_t. This action is used to act on the environment, as well as to update both the policy itself and the incremental model. The loss function for the policy can be seen in Equation 4.14 and consists of the next Bellman value estimate with γ the discount factor. In the IDHP framework, the value function is not available, instead the state derivative of the value function $\frac{\partial V(s_t)}{\partial x_t} = \lambda_w(s_t)$ is the output of the critic. This means the gradient of L_π does not need backpropagation through the critic network and can use the output of the critic directly in the gradient. The gradient of the loss function can then be derived as seen in Equation 4.15 where the critic value comes from the target critic $\lambda'_{w'}$. The term $\frac{\partial x_{t+1}}{\partial a_t}$ can be replaced by the incremental model input matrix G_{t-1} as per definition of the input matrix. The term $\frac{\partial a_t}{\partial \theta}$ is calculated using backpropagation on the actor. The policy can then be updated using the gradient by performing gradient descent as seen in Equation 4.16 with η_a the learning rate of the actor.

$$L_\pi = -V(s_t) = -\left[r_{t+1} + \gamma V(s_{t+1})\right] \tag{4.14}$$

$$\begin{aligned}\nabla_\theta L_\pi = \frac{\partial L_\pi}{\partial \theta} &= -\left[\frac{\partial r_{t+1}}{\partial x_{t+1}} + \gamma \lambda'_{w'}(s_{t+1})\right]\frac{\partial x_{t+1}}{\partial a_t}\frac{\partial a_t}{\partial \theta} \\ &= -\left[\frac{\partial r_{t+1}}{\partial x_{t+1}} + \gamma \lambda'_{w'}(s_{t+1})\right] G_{t-1}\frac{\partial a_t}{\partial \theta}\end{aligned} \tag{4.15}$$

$$\theta_{t+1} = \theta_t - \eta_a \nabla_\theta L_\pi \tag{4.16}$$

4.2.3. IDHP Critic

The IDHP critic approximates the state derivative of the state-value function and is defined as $\lambda_w(s_t) = \frac{\partial V(s_t)}{\partial x_t}$ with w the parameters of the critic network. The critic loss function L_λ provides the gradients used to update the critic parameters. The loss is defined as the mean squared error of the state derivative of the TD error $\frac{\partial \delta_t}{\partial x_t}$ as seen in Equation 4.17. In Equation 4.18, the formulation of the TD error for the critic can be seen which corresponds to the next value function estimate called the TD target minus the current value estimate $V(s_t)$. Taking the state partial derivative of the TD error results in Equation 4.19 where the TD target is calculated using the target critic value $\lambda'_{w'}$. The state derivative of the reward is provided by the environment, while the term $\frac{\partial x_{t+1}}{\partial x_t}$ can be computed by using the incremental model as seen in Equation 4.20. The term $\frac{\partial a_t}{\partial x_t}$ or $\frac{\partial \pi_\theta(a_t|s_t)}{\partial x_t}$ can be obtained by backpropagation trough the policy network.

$$L_\lambda = \frac{1}{2}\left(-\frac{\partial \delta_t}{\partial x_t}\right)\left(-\frac{\partial \delta_t}{\partial x_t}\right)^T \tag{4.17}$$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{4.18}$$

$$\frac{\partial \delta_t}{\partial x_t} = \left[\frac{\partial r_{t+1}}{\partial x_{t+1}} + \gamma \lambda'_{w'}(s_{t+1})\right]\frac{\partial x_{t+1}}{\partial x_t} - \lambda_w(s_t) \tag{4.19}$$

$$\frac{\partial x_{t+1}}{\partial x_t} = F_{t-1} + G_{t-1}\frac{\partial a_t}{\partial x_t} \tag{4.20}$$

Similar to the actor-network, the critic network is updated using gradient descent which needs the gradient of the loss function with respect to the critic parameters as seen in Equation 4.21. The update step in Equation 4.22 is then performed using η_c as the critic learning rate.

$$\nabla_w L_\lambda = \frac{\partial L_\lambda}{\partial w} = \frac{\partial L_\lambda}{\partial \lambda_w(s_t)}\frac{\partial \lambda_w(s_t)}{\partial w} = -\frac{\partial \delta_t}{\partial x_t}\frac{\partial \lambda_w(s_t)}{\partial w} \tag{4.21}$$

$$w_{t+1} = w_t - \eta_c \nabla_w L_\lambda \tag{4.22}$$

$$w'_{t+1} = \tau w_t + (1-\tau) w'_t \tag{4.23}$$

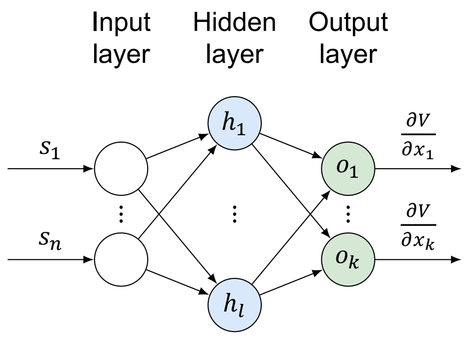
Generally, methods that suffer from the deadly triad [62] implement a target critic to slow down the learning process and consequently improve the learning stability. The IDHP framework is an on-policy method and technically does not appertain to the deadly triad. Previous research on the PH-LAB [28] has however concluded that any form of learning stability improvement is crucial to the safety-critical application of flight control and the addition of a target critic is proposed. Hence the choice to implement a target critic is made. The target critic is not updated using gradient descent, instead, a soft update rule in Equation 4.23 is applied which gradually copies the parameters from the critic to the target critic as introduced in [42], given a hyperparameter τ which sets the smoothness of the update.

4.2.4. IDHP Network Architectures

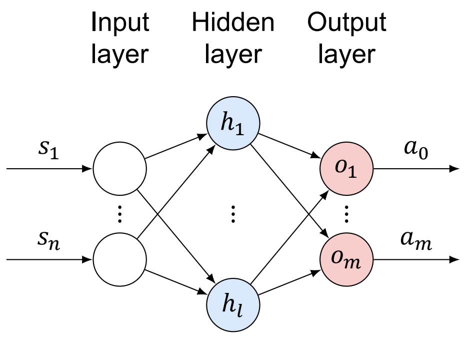
The policy and critic network are both defined as a feed-forward artificial neural network consisting only of fully-connected layers with an activation function and no bias terms. In Figure 4.2a a schematic of the critic network architecture can be seen and Figure 4.2b shows the policy network architecture. Since IDHP is not a DRL method and online learning needs fast inference and gradient calculation, the critic and policy networks are rather shallow using one hidden layer of limited size compared to Deep Learning ANNs. Both the critic and policy have the observation vector s with dimension n as input and a hidden layer with size l. The hidden layer uses tanh activation functions. The choice of activation function is taken from reference literature [36] [34] [28] and is preferred over other popular options like sigmoid and rectified linear unit (ReLU) functions as these mainly provide improvements when working with deep neural networks and larger batch sizes.

The difference between the policy and critic networks lies in the shape and activation function of the output layer. The critic outputs the state derivative of the state-value function with an output size k number of states. The output layer in the critic also uses a linear activation function. The actor on the other hand outputs m actions and uses a tanh activation function in order to keep the action output bounded. Since the action needs to translate to physical space, the tanh activation squashed the policy output to $[-1, 1]$ and a scaling function can then be applied to scale the action to the physical limits of the input of the environment. In the case of the short-period model the action is the elevator deflection which is limited to $[-20°, 20°]$ where the policy output is scaled accordingly before sending the action to the environment. Note that internally in the agent, the policy output used for update steps is the unscaled tanh bounded action. To initialize the network parameters a truncated normal distribution is chosen which are sampled to define the initial parameters. This means the network parameters w and θ are initialized using a standard deviation of 0.05 and kept inside two standard deviations, so $\in [-0.1, 0.1]$. The standard deviation is a common value obtained from literature and should be small enough to have a minimal random effect. The target critic is initialized as a copy of the critic.

The optimizers used to apply the gradient descend update rules are the same for the policy and critic networks. Improved optimizer algorithms exist like RMSprop and Adam [63], however, these provide benefits that mostly apply to deep DNNs and the simpler standard gradient descend optimizer is implemented here.


Input layer
Hidden layer
Output layer
s_1
s_n
h_1
h_l
o_1
o_k
$\frac{\partial V}{\partial x_1}$
$\frac{\partial V}{\partial x_k}$

(a) Critic



(b) Policy

Figure 4.2: IDHP network architectures

4.2.5. IDHP Training Strategy

The IDHP framework is designed to learn online without prior knowledge of the system dynamics. This means the training strategy is relatively straightforward. For this preliminary analysis, the short period model is used to simulate an angle-of-attack tracking task where the tracking error should be minimised. Each run is thus starting from random initialization which produces the results that are discussed in section 4.5.

The continual learning approach of IDHP can cause a number of adverse effects. The two effects of particular interest for this research are weight divergence and estimator covariance windup.

Weight divergence due to continued learning of the actor and critic networks is usually improved by scheduling the learning rate to decrease over time. In an online learning process, however, this makes the controller less adaptive over time. Error-based adaptive learning rates as implemented in [55] [98] [36] solve this by relating the learning rate to the tracking error which makes the adaptiveness of the agent not dependent on the time passed. Here, the high-low adaptive learning rate as is [36] is implemented where a threshold is set on the tracking error Root Mean Squared Error (RMSE). In this case, the RMSE is calculated on the angle-of-attack tracking error $\alpha_{ref} - \alpha$ over the last 50 time-steps. When the RMSE is above the empirically determined threshold α_{thresh}, the learning rate is set to η_{high} and decreases back to η_{low} when the RMSE falls to below the threshold. To ensure proper learning during the initial phase, it is decided to keep the learning rate at the high setting for the initial t_{warmup} time-steps. The high-low method is also used for the hybrid framework as discussed in section 4.4. A gradual adaptive learning rate can be explored in the future phase of this research.

Next, the estimator covariance windup pertains to the incremental model RLS estimator. During periods of low excitation, the covariance matrix of the incremental model increases exponentially which causes sudden changes in the parameters once the system is excited again as demonstrated by [28]. Setting the forgetting factor to $\gamma_{RLS} = 1.0$ ensures consistency of the estimator. This has the disadvantage of decreasing the adaptiveness of the agent over time. To circumvent this, the covariance matrix can be reset to Λ_0 once a change in system dynamics is detected. Similarly to [28], this is implemented here using a threshold on the innovation term ϵ_{thresh} as defined in Equation 4.12.

In Table 4.2, the hyperparameters used in generating the preliminary analysis results are presented. These values were obtained empirically with trial and error and from commonly used values in literature. A hyperparameter search should be conducted in a later phase of the research to obtain a more optimal configuration, but for this analysis, it is deemed sufficient to apply manual tuning.

Table 4.2: IDHP preliminary analysis hyperparameters

Parameter	Value	Unit	Description
γ	0.6	[-]	Discount factor
γ_{RLS}	1.0	[-]	Incremental model forgetting factor
τ	0.01	[-]	Target critic mixing factor
l	10	[-]	Actor and critic hidden layer size
t_{warmup}	100	[-]	Nr. of time-steps to hold initial high learning rate
κ	28	[-]	Reward scale
$\eta_{a,high}$	0.08	[-]	Actor high learning rate
$\eta_{a,low}$	0.005	[-]	Actor low learning rate
$\eta_{c,high}$	0.005	[-]	Critic high learning rate
$\eta_{c,low}$	0.0005	[-]	Critic low learning rate
α_{thresh}	1.0	[deg]	Threshold for the high-low adaptive learning rate
ϵ_{thresh}	$[0.0005, 0.001]$	[deg], [deg/s]	Threshold for incremental model covariance reset

4.3. SAC Agent

This section discussed the implementation of the SAC framework. Similarly to the IDHP implementation TensorFlow 2 was used using a number of references [22] [23] [11], Softlearning [1] and Stable Baselines [2].

In Figure 4.3, a detailed high-level overview of the flow of the IDHP framework can be seen. The corresponding pseudo-code of the implemented SAC framework can be found in 2. Compared with the

[1]`https://github.com/rail-berkeley/softlearning`

[2]`https://github.com/hill-a/stable-baselines`

IDHP framework from Figure 4.1, the main design differences can clearly be seen. The off-policy design of the SAC vs the on-policy design of IDHP is made clear by having essentially two kinds of forward passes through the policy, one where the environment observation generates a new action to take every time-step, and one where the replay buffer is sampled to perform the updates. This difference is also made in the notation of the replay buffer signals where $\{a_t\}$ is an action batch sampled from the replay buffer and $\{a_t\} \sim \pi$ is a batch of newly generated actions using the observations from the replay buffer. Furthermore, the twin Q-function critics and the entropy coefficient elements are other major differences with IDHP. The following sections go over the three main elements of the SAC framework, the actor, critic and the entropy element. Also, the neural network architectures of the actor and critic are discussed in more detail. Finally, the training strategy used to produce the simulation results is discussed.

Algorithm 2 SAC framework, adapted from [23], [11]

```
Initialize:
    Hyperparameters as defined in Table 4.3
    Actor and critic parameters $\theta$, $w_1$, $w_2$ and $w_1' \leftarrow w_1$, $w_2' \leftarrow w_2$
    Environment state and observation $x$, $s$
for each time step t do
    Sample and execute action $a_t \sim \pi_\theta(\cdot \mid s_t)$
    Observe next observation and reward $s_{t+1}$, $r_{t+1}$
    Store transition samples $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay buffer $\mathcal{D}$
    Sample mini-batch $\mathcal{B} = s_t, a_t, r_{t+1}, s_{t+1}$ from $\mathcal{D}$
    Update learning rates $\eta_a$ and $\eta_c$ from linear decay

    Compute actor loss $L_\pi$ as in Equation 4.26: $L_\pi(\min_i Q_t', \log\pi_t, \eta_t)$
    Compute twin critic losses $L_{Q_i}$ as in Equation 4.27: $L_{Q_i}(Q_{t_i}, \min_i Q_{t+1}', \log\pi_{t+1}, \eta_t, r_{t+1})$
    Compute temperature loss as in Equation 4.25: $L_\eta(\eta_t, \log\pi_t)$
    Update actor weights $\theta \leftarrow \theta - \eta_a \nabla_\theta L_\pi$
    Update twin critic weights $w_i \leftarrow w_i - \eta_c \nabla_{w_i} L_{Q_i}$
    Update temperature coefficient $\eta \leftarrow \eta - \eta_a \nabla_\eta L_\eta$
    Update twin target critic weights $w_i' \leftarrow \tau w_i + (1 - \tau) w_i'$
end for
```

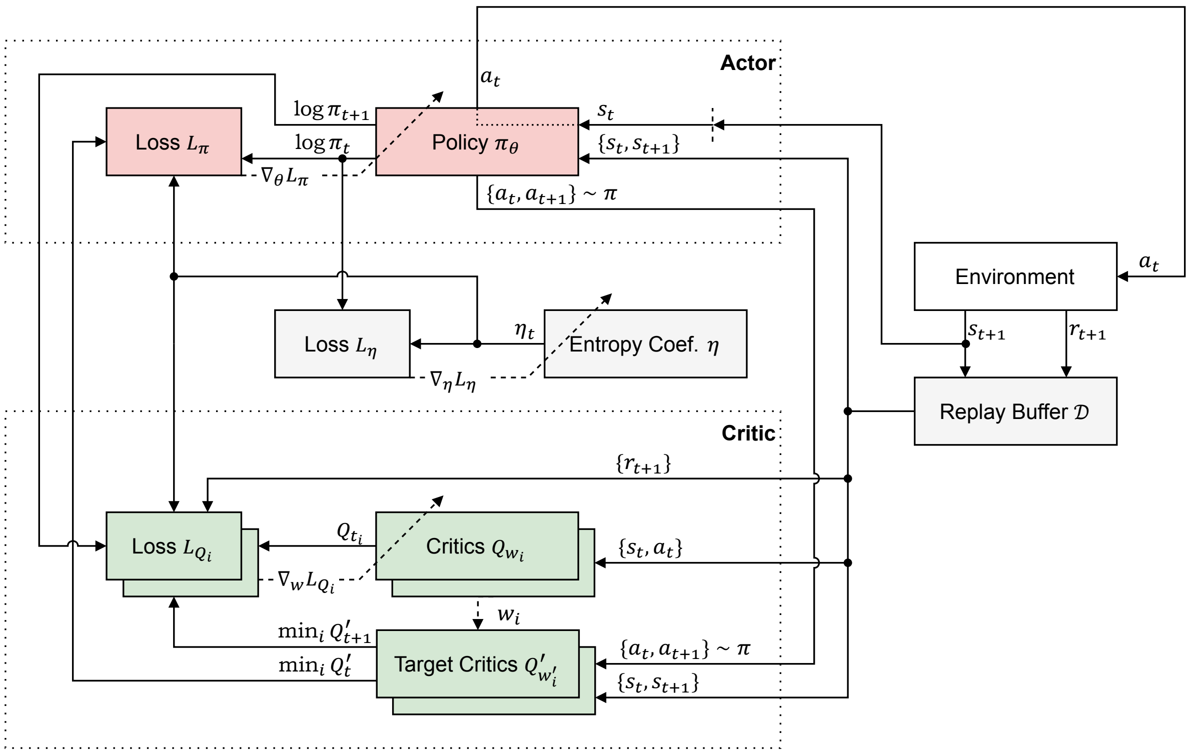


Figure 4.3: SAC Framework, offline off-policy

4.3.1. Entropy

In the SAC framework, entropy is used in the context of soft policy iteration and modifies the classic RL objective by trying to maximize the entropy in addition to the discounted return. The entropy of a distribution is defined as in Equation 4.24 with the distribution being the policy. This is possible as SAC uses a stochastic policy which is defined as a probability distribution.

$$\mathcal{H}(\pi_\theta(\cdot \mid s_{t+1})) = \mathop{\mathbb{E}}_{a\sim\pi}[-\log \pi_\theta(a \mid s_t)] \tag{4.24}$$

In the update rules of the actor and critic, the entropy term is weighted using the entropy coefficient or temperature coefficient η. The SAC algorithm has been shown to be highly sensitive to the temperature coefficient, thus it was proposed by [22] to learn η automatically. The loss function for this automatic learning process can be seen in Equation 4.25. The term $\bar{\mathcal{H}}$ is a constant entropy target and is set to the negative of action space size [23], in the case of the short period model $\bar{\mathcal{H}} = -1$. Using the loss function gradient $\nabla_\eta L_\eta$, the entropy coefficient is updated using SGD.

Note that the estimation operator in all the update rules of SAC is implemented by taking the average over the batch size. The update rules are computed on a mini-batch from the replay buffer randomly sampled on every iteration.

$$L_\eta = \mathop{\mathbb{E}}_{\substack{s_t\sim\mathcal{D}\\ a_t\sim\pi}}\left[-\eta \log \pi_\theta(a_t \mid s_t) - \eta\bar{\mathcal{H}}\right] \tag{4.25}$$

4.3.2. SAC Actor

The SAC policy π_θ is a stochastic policy parameterized with θ the parameter vector. This policy distribution is implemented by having the output of the policy be the standard deviation σ_θ and mean μ_θ of the distribution, more details about this in subsection 4.3.4. This is then used to sample an action from a normal distribution with σ_θ and μ_θ. Note that this sampling requires a ”reparameterization trick” to ensure differentiability of the sampled action which is necessary for the gradient calculations.

This is usually implemented using an input Gaussian noise vector ϵ_t and sampling an action using $a_t = f_\theta(\epsilon_t, s_t) = \mu_\theta(s_t) + \epsilon_t \cdot \sigma_\theta(s_t)$. In [23] this reparameterization is implemented manually, however, the implementation here defines the normal distribution using the TensorFlow probability library which ensures differentiability under the hood and calculates the log probabilities $\log \pi_\theta(a_t \mid s_t)$ used in the update rules of the actor and the critic.

The loss function for the policy can be seen in Equation 4.26. The minimum of the twin target Q-critics is used to improve the overestimation problem present with a single critic. Note that the observations s_t are sampled from the replay buffer, while the actions are newly generated from the current policy using the observations from the replay buffer. SGD is then used to update the policy weights using the gradient of the loss function.

$$L_\pi = \mathop{\mathbb{E}}_{\substack{s_t \sim \mathcal{D} \\ a_t \sim \pi}} \left[\min_{i=1,2} Q'_{w'_i}(s_t, a_t) - \eta \log \pi_\theta(a_t \mid s_t) \right] \tag{4.26}$$

4.3.3. SAC Critic

The SAC critic consists of twin Q-functions Q_w and twin target Q-functions $Q'_{w'}$ which unlike the value functions in IDHP take both the observation s_t and action a_t as input. Each Q-function is updated separately using its own loss function and the minimum of the two Q-values is used in the update rules to prevent overestimation. The loss function for the Q-function critics is seen in Equation 4.27 for the i-th Q-function and is defined as the mean squared error of the TD-error δ. The TD-error is defined as the TD-target minus the current Q-value estimation as seen in Equation 4.28. The TD-target in this case contains the entropy term originating from the soft value function term $V(s_{t+1})$ which is defined as in Equation 4.29. Again, the observation is sampled directly from the replay buffer while the actions are newly computed using the current policy. SGD is then used to update the critic weights for each of the twin critics separately using the gradient of their respective loss functions. The target Q-functions are updated using the same soft update rule used in IDHP seen in Equation 4.23.

$$L_{Q_i} = \frac{1}{2}\mathbb{E}\left[(-\delta_{t,i})^2\right] \tag{4.27}$$

$$\delta_{t,i} = r_{t+1} + \gamma V(s_{t+1}) - Q_{w_i}(s_t, a_t). \tag{4.28}$$

$$V(s_{t+1}) = \mathop{\mathbb{E}}_{\substack{s_{t+1} \sim \mathcal{D} \\ a_{t+1} \sim \pi}} \left[\min_{i=1,2} Q'_{w'_i}(s_{t+1}, a_{t+1}) - \eta \log \pi_\theta(a_{t+1} \mid s_{t+1}) \right] \tag{4.29}$$

4.3.4. SAC Network Architectures

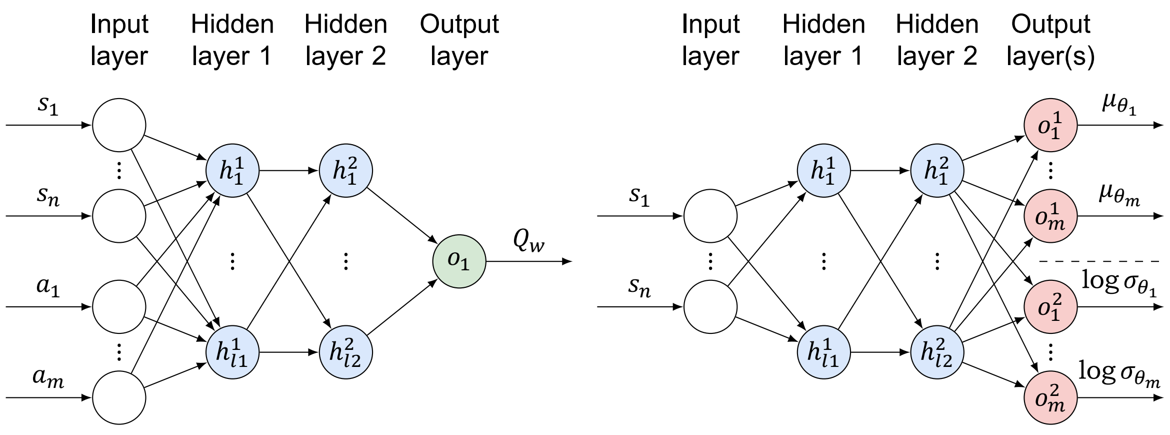
Since SAC is a DRL method, the network architectures are both wider and deeper than the IDHP networks. The critic network and policy network architectures can be seen in Figure 4.4a and Figure 4.4b respectively. The hidden layers are constructed the same for the policy and the critic networks. Two hidden layers are present with sizes $l1$ and $l2$. Each hidden layer has a ReLU activation function as suggested by the original implementation [22].

The critic Q-function takes both the observation vector s and the action vector a as a concatenated input. The output of the critic consists of a single scalar Q-value where the output layer has a linear activation function.

The policy has only the observation vector as input. The output of the policy consists of two output layers, one for the policy mean μ_θ and one for the log of the policy standard deviation σ_θ. Each of the two output layers connects to the second hidden layer and contains no activation function. The log of the standard deviation is used in order to estimate the parameter on $\mathbb{R}$ and the exponential of the network output is taken to be used in the policy distribution. The action however still needs to be bounded using a tanh function in order to properly scale to the physical domain of the action. The tanh squashing is performed on the action after the normal distribution has been sampled using the network outputs and is then scaled similarly to the IDHP implementation.

The hidden layers h^1 and h^2 also contain a normalization layer as per the original implementation [22]. This is to improve learning on input vectors with differing scales and non-zero means. Note that the normalization layer comes before the relu activation function of the hidden layers.

For the optimizer, an SGD update rule is used for the actor, critic and also the temperature coefficient updates. In the field of DL, a number of improved optimization algorithms are used that provide better results in larger networks and batch sizes compared to shallow IDHP networks. The Adam optimizer is implemented here as it can provide better learning stability than standard SGD [63] in the case of the SAC framework.



(a) Critic Q-function (b) Policy

Figure 4.4: SAC network architectures

4.3.5. SAC Training Strategy

The SAC framework is designed to be trained offline. Technically, online learning is possible given sufficient computational power, however amount of stochastic effects and high exploration raise significant safety concerns in flight control applications and are reasons to omit online learning from this analysis. The offline learning strategy consists of training the agent on multiple episodes of the angle-of-attack tracking task with a maximum number of time steps provided. Note that the episode length is finite in the training environment, but when deploying the offline trained policy online, the episode length is infinite because of the continual control application considered in this research.

After every training episode, the total return which is the sum of all rewards of that episode is used as a training metric. For more complicated tasks, the task that is used for training and the task that is used to evaluate the episode return metric can be different with for instance a different reference signal or episode length. In this case, however, a simplified training strategy is used where the training task is used to collect the return metric which saves time.

In Table 4.3, the hyperparameters used during training can be seen. Like the IDHP agent, the hyperparameters were manually tuned using trial and error and from commonly used values in literature. Further development on more environments and tasks requires a hyperparameter search to achieve a more optimal configuration.

The learning rates of the actor and critic are not constant but decay as a function of the maximum number of time steps. Reducing the learning rate is a common practice when training DNNs and has been empirically observed to improve learning stability [95]. For this implementation, the learning rate of both the actor and critic is linearly reduced from the initial learning rate to zero. This means the maximum number of time steps determine the slope of the learning rate decay.

Table 4.3: SAC preliminary analysis hyperparameters

Parameter	Value	Unit	Description
γ	0.99	[-]	Discount factor
τ	0.005	[-]	Target critic mixing factor
$l1, l2$	$[64, 64]$	[-]	Actor and critic hidden layer sizes
κ	1	[-]	Reward scale
η_a	$9.4e-4$	[-]	Actor initial learning rate
η_c	$9.4e-4$	[-]	Critic initial learning rate
max time steps	500000	[-]	Maximum number of training time-steps
$\|\mathcal{B}\|$	256	[-]	Replay buffer batch size
$\|\mathcal{D}\|$	50000	[-]	Replay buffer maximum size
η_0	1.0	[-]	Initial temperature coefficient

4.4. IDHP-SAC Hybrid Agent

As part of the search for a suiting RL framework candidate, the possibility of combining the IDHP and SAC framework is explored. For the flight control application relevant for this research, two possible methods of combining the advantages of IDHP and SAC are identified where one solution aims to share information from the critic and the other aims to share the policy.

A method of sharing information from the critic between SAC and IDHP would mean that the offline trained Q-function from SAC is used to augment the update rules of the IDHP actor and/or critic. Since this requires rethinking the update rules used in the IDHP framework, this approach is considered outside the scope of this research and only the other option of combining the policies is attempted for implementation.

A combined policy approach is more in line with the workflow used in transfer learning and fine-tuning from the field of machine learning [57] [100]. Recall that the offline-trained SAC framework has been shown to train a policy that achieves desirable qualities through robustness thanks to the high generalization power of DRL and the high exploration driven by the off-policy design and the entropy terms. Looking at fault tolerance, the robustness of the SAC policy can be insufficient to adapt to the changing system dynamics depending on the severity of the failure case. This is where the online learning and adaptive control approach of the IDHP framework are projected to increase the fault tolerance of the SAC policy. From the side of a pure IDHP framework, the addition of the pre-trained robust SAC policy might reduce or eliminate problems with continual learning. In this preliminary analysis phase, the goal is to investigate the feasibility of implementing the combined policy and testing it on the short-period model with the same angle-of-attack tracking task.

4.4.1. IDHP-SAC Network Architectures

In Figure 4.5, a schematic of the combined policy architecture can be seen. The input, output and hidden layer h are the same as for the pure IDHP policy. The difference lies in the addition of the three layers from the pre-trained SAC policy h^1, h^2 and o^1 that are added before the two IDHP layers. The last SAC layer o^1 is the mean output layer of the SAC policy, while the standard deviation of the SAC policy is not utilized as the combined policy is still deterministic. Note that the three SAC layers have their weights frozen during the online learning phase. This is in line with the approach of transfer learning which points to the process of transferring knowledge from what is learned in one domain to another domain. Re-training the pre-trained networks is usually referred to as fine-tuning where either a subset of the pre-trained weights are re-trained, or additional layers are added that are trained in the new domain. Since it is preferred to maintain the information of the robust SAC policy, all the SAC weights are kept frozen and only the two shallow IDHP layers are trained online. Regarding the initialization of the two IDHP layers, an identity initializer is used as discussed in subsection 4.4.2. Policy fine-tuning has been previously investigated in an RL framework where it is described as the online RL that has access to a reference policy [94].

Regarding the rest of the hybrid IDHP-SAC framework, the online-learning update rules of the IDHP framework as described in section 4.2 are used where only the policy architecture as described above is changed. The critic used in the online learning process is the same as the pure IDHP critic.

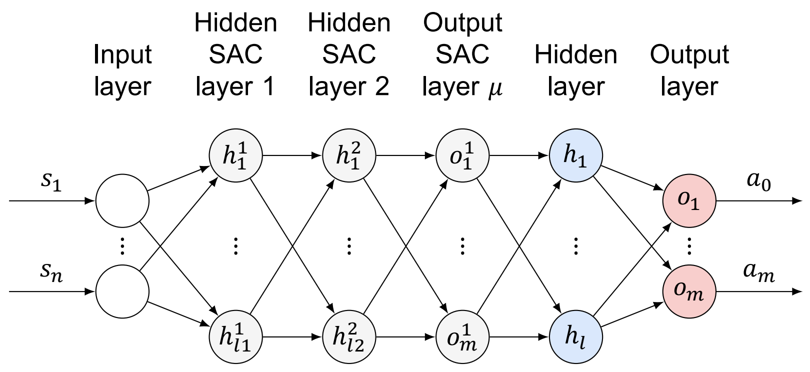


Figure 4.5: IDHP-SAC combined policy architecture

4.4.2. IDHP-SAC Training Strategy

The process of completely training the hybrid policy consists of first training the SAC framework as described in subsection 4.3.5, and then taking the SAC policy and constructing the combined policy architecture as described above. The IDHP framework then trains the combined policy, which can be seen as policy fine-tuning.

The high-low adaptive learning rate of the IDHP framework as discussed in subsection 4.2.5 is especially applicable to the current hybrid approach. It is desired to eliminate as many of the continual learning issues from IDHP, which could be achieved by setting the low learning rate of the high-low strategy to 0.0. This would mean that the IDHP layers in the combined policy only update when the adaptive learning rate threshold is reached. The goal of this is to utilize the robust SAC policy as much as possible and only adapt the policy when necessary.

Regarding the behaviour of the adaptive learning rate, the t_{warmup} as discussed in subsection 4.2.5 has the opposite function, in this case, holding the learning rate at 0.0 for the first t_{warmup} time-steps. This is to make the IDHP agent ignore the transient from the SAC policy. To enable this learning rate strategy, an additional change to the network initialization of the policy needs to be made. Random initialization would not allow the hold at 0.0 learning rate as the SAC policy is altered by the two randomly initialized IDHP layers h and o. Instead, an identity initialize is used which allows the SAC policy to pass through the IDHP layers during the initial phase. This does however mean that

For the offline SAC training, the same hyperparameters as in Table 4.3 are used. The hyperparameters used for the IDHP portion of the hybrid framework can be seen in Table 4.4. These values were obtained empirically and manually tuned with trial and error and from commonly used values in literature.

Table 4.4: IDHP preliminary analysis hyperparameters

Parameter	Value	Unit	Description
γ	0.6	[-]	Discount factor
γ_{RLS}	1.0	[-]	Incremental model forgetting factor
τ	0.01	[-]	Target critic mixing factor
l	10	[-]	Actor and critic hidden layer size
t_{warmup}	500	[-]	Nr. of time-steps to hold initial high learning rate
κ	1000	[-]	Reward scale
$\eta_{a,high}$	0.5	[-]	Actor high learning rate
$\eta_{a,low}$	0.0	[-]	Actor low learning rate
$\eta_{c,high}$	0.05	[-]	Critic high learning rate
$\eta_{c,low}$	0.0	[-]	Critic low learning rate
α_{thresh}	0.5	[deg]	Threshold for the high-low adaptive learning rate
ϵ_{thresh}	$[0.0005, 0.001]$	[deg], [deg/s]	Threshold for incremental model covariance reset

4.5. Results and Discussion

This section goes over the simulation results from the IDHP, SAC and hybrid implementations of the RL controller on the short-period model. The three frameworks are compared against each other and their individual results are discussed. These results mainly aim to demonstrate the advantages and disadvantages of iADP (IDHP) vs. DRL (SAC) and also demonstrate the feasibility of combining the advantages of both.

First of all, the offline training reward curve of the SAC agent can be seen in Figure 4.6. For the sake of simplicity and because of the simplified model and task, the return per episode is collected from the training task itself which is set to run for 60s at $100Hz$ using the combined sinusoidal tracking signal. A maximum of 500000 time steps have been set, with evaluation after every episode. Training took around 35 minutes on CPU with 4 cores at 2.8GHz. The same 60s task at $100Hz$ is used for all subsequent simulations in this chapter unless specified otherwise.

As can be seen from the reward curve, the SAC agent converges with a reward of ~ -0.005. As expected, the initial phase until around time-step 100000 is relatively unstable, but still quickly increases from the initial phase. After 100000 the training process is stable and only marginally increases until a return of ~ -0.005, with the exception of a large dropout at time-step 2200000. The stochastic effects on the learning process are expected to be more present in environments with larger state and action spaces. The agent with the best reward of -0.003429 is saved and used in the evaluation of the pure-SAC system response. The same SAC agent is used in the IDHP-SAC hybrid framework for constructing the combined policy.

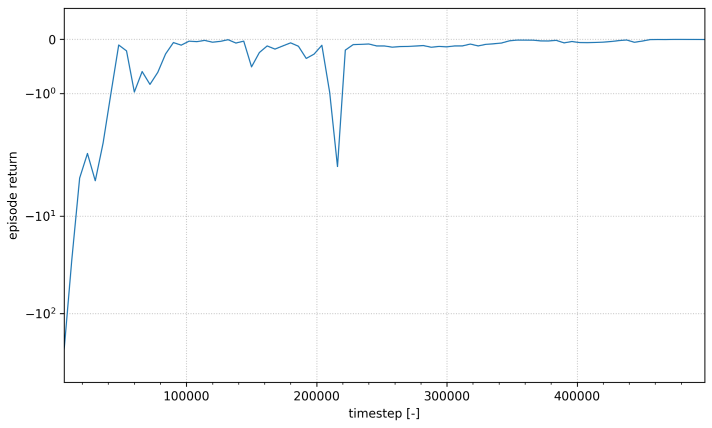


Figure 4.6: SAC offline training reward curve on the short-period model α tracking task

4.5.1. Normal System Dynamics

The response on the short period model as described in section 4.1 without any alterations is discussed here.

IDHP Controller

In Figure 4.7, the response of the IDHP controller can be seen with the elevator input signal. The bottom row of the figure displays the RMSE of the tracking error α^{RMSE} and the threshold which determines the learning rate of actor and critic. The vertical line in the RMSE plot indicates the t_{warmup} phase in which the learning rate is kept at the high setting regardless of the tracking error. The general response shows that the IDHP controller is able to successfully track the reference signal with a small steady-state error around the sinusoidal peaks. Some oscillations are detected at the initial stage indicating a somewhat aggressive policy. During manual hyperparameter tuning, the learning rate and reward scale have shown to affect the aggressiveness of the policy and further tuning might improve the response.

As can be seen from the RMSE plot, the learning rate is kept at the high setting after the warm-up period for another 100 time steps. After that, the response is stabilized under the RMSE threshold which means the agent is learning on the low learning rate setting for the majority of the simulation run. The same can also be seen from Figure 4.9 which displays the weights of the two layers from actor and critic.

It can be seen that the weights quickly diverge from the initialized $[-0.1, 0.1]$ values and stabilize once the learning rate is lowered. A possible disadvantage of the high-low adaptive learning rate strategy is also observed here at around $3s$. At this point, the RMSE touches the threshold again which results in an oscillation in the elevator input. Adjusting the threshold or learning rate gap can improve this, however, this provides a compelling case for a continual adaptive learning rate implementation.

SAC and Hybrid Controller

In Figure 4.8, the response of the hybrid IDHP-SAC controller can be seen. Looking at the tracking error threshold, the SAC policy keeps the error below the threshold for the entirety of the simulation. Recall that the hybrid training strategy holds the IDHP learning rates at zero during t_{warmup}, and when the RMSE is under the threshold. This means that the hybrid controller response equals the SAC controller response for normal system dynamics and they are discussed together here. Similarly to the IDHP response, the initial transient has oscillations, in this case, higher frequency oscillations than the IDHP controller, reaching about half the control limits. The rest of the response shows the SAC controller successfully tracks the reference signal with a near-zero steady-state error. This indicates the SAC policy is aggressive but still stable in this case. The strong oscillations could be improved by a hyperparameter search, or possibly evaluating the SAC at different a point on the reward curve as the aggressiveness might indicate over-fitting. Because of the offline training strategy, remodelling the evaluation task and/or the reward function might also lead to a more optimal result.

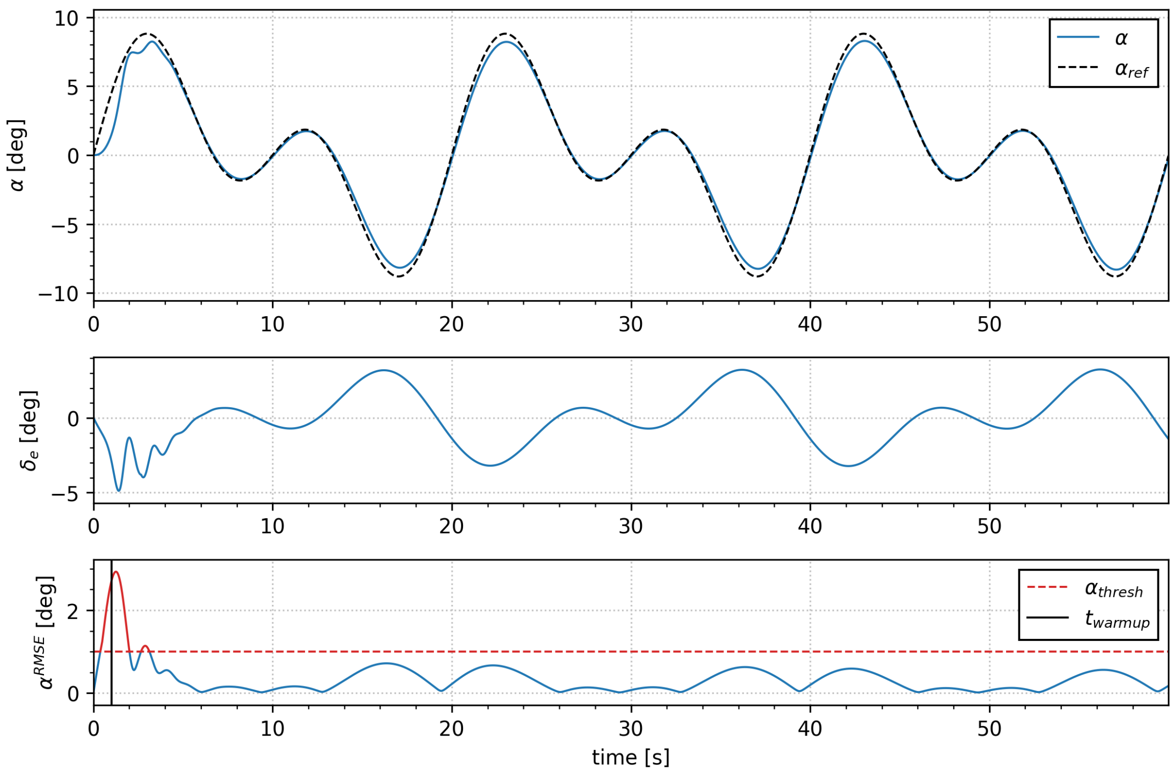


Figure 4.7: IDHP short-period response on normal system dynamics

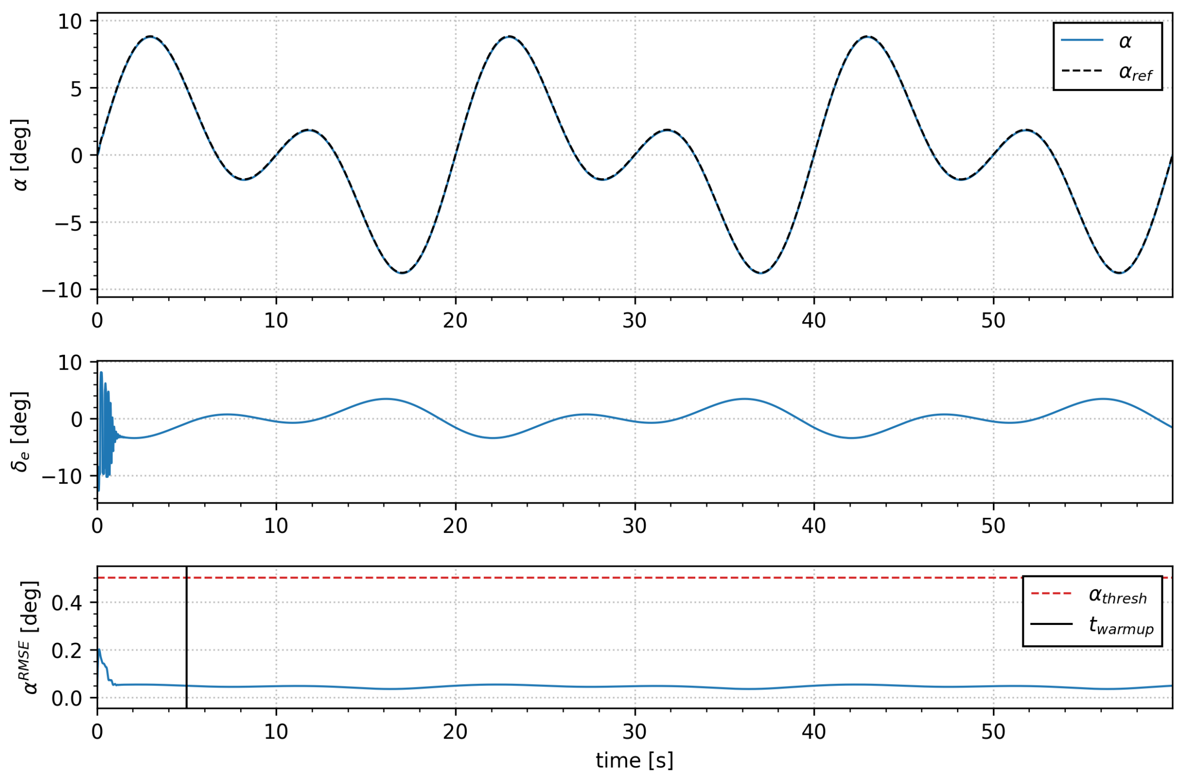


Figure 4.8: SAC and IDHP-SAC short-period response on normal system dynamics

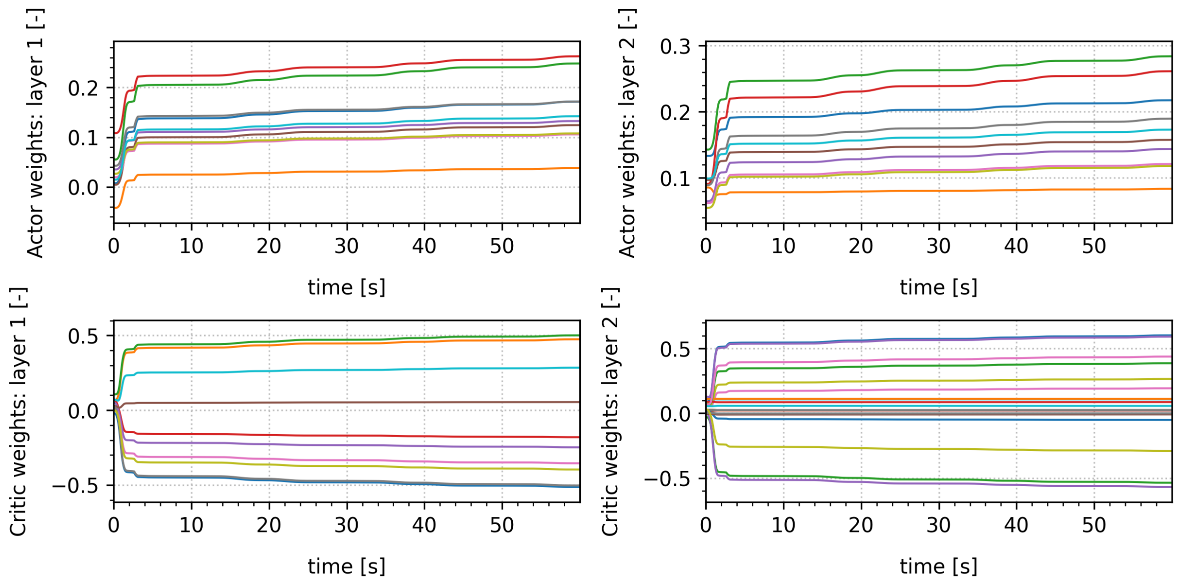


Figure 4.9: IDHP actor and critic weights on normal system dynamics

4.5.2. Untrimmed initialization

The previous simulations (and offline SAC training) have started from a trimmed state, in this case meaning the state vector of the short-period model was initialized at $\begin{bmatrix}0 & 0\end{bmatrix}^T$. To test the robustness and to show a generally more realistic scenario, a batch of 100 runs is performed with a random state initialization. The angle of attack α is initialized between $[-5, 5]$ deg, and the pitch rate q is initialized on the interval $[-3, 3]$ $\frac{deg}{s}$, randomly sampled on every repeated run. In order to analyze the results, the min-max bounds of the response are drawn, along with the mean response in between. Since for every controller the response stabilizes after one period of the reference signal, only the first 30s of the response is shown in order to better zoom in on the transient.

IDHP Controller

In Figure 4.10, the untrimmed response of the IDHP controller can be seen. From the state plot, it shows that the mean of the transient follows the reference signal, and the maximum transient length remains under 5s. The oscillatory behaviour in the input signal is also observed and shows that in the first ss, the control limits are reached indicating an aggressive response to the untrimmed states. The shape of the RMSE error is similar to the trimmed simulation run, with around 4s of high learning rate. In general, this shows the IDHP controller is able to control the system in an untrimmed state.

SAC and Hybrid Controller

Similarly to the previous trimmed simulation, the untrimmed response of the SAC and hybrid controllers are equal. Looking at the RMSE error in Figure 4.11, the threshold is not reached after the t_{warmup} period. Due to the untrimmed initialization, a larger transient is present, but after 4s, the SAC policy accurately tracks the reference signal and the IDHP learning is not engaged. Oscillations in the input signal are again visible with higher amplitude due to an aggressive response to the untrimmed initialization. This also shows that the SAC policy successfully tracks the reference signal for the untrimmed case.

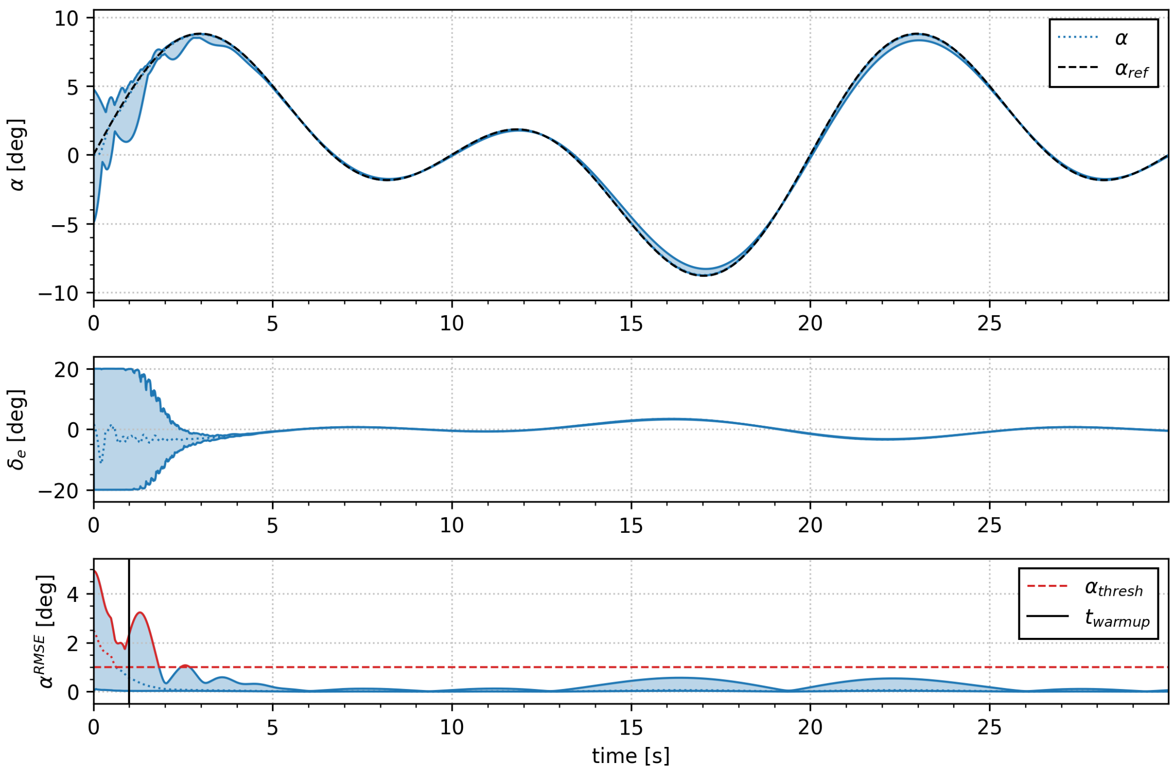


Figure 4.10: IDHP short-period response on 100 runs with untrimmed initialization

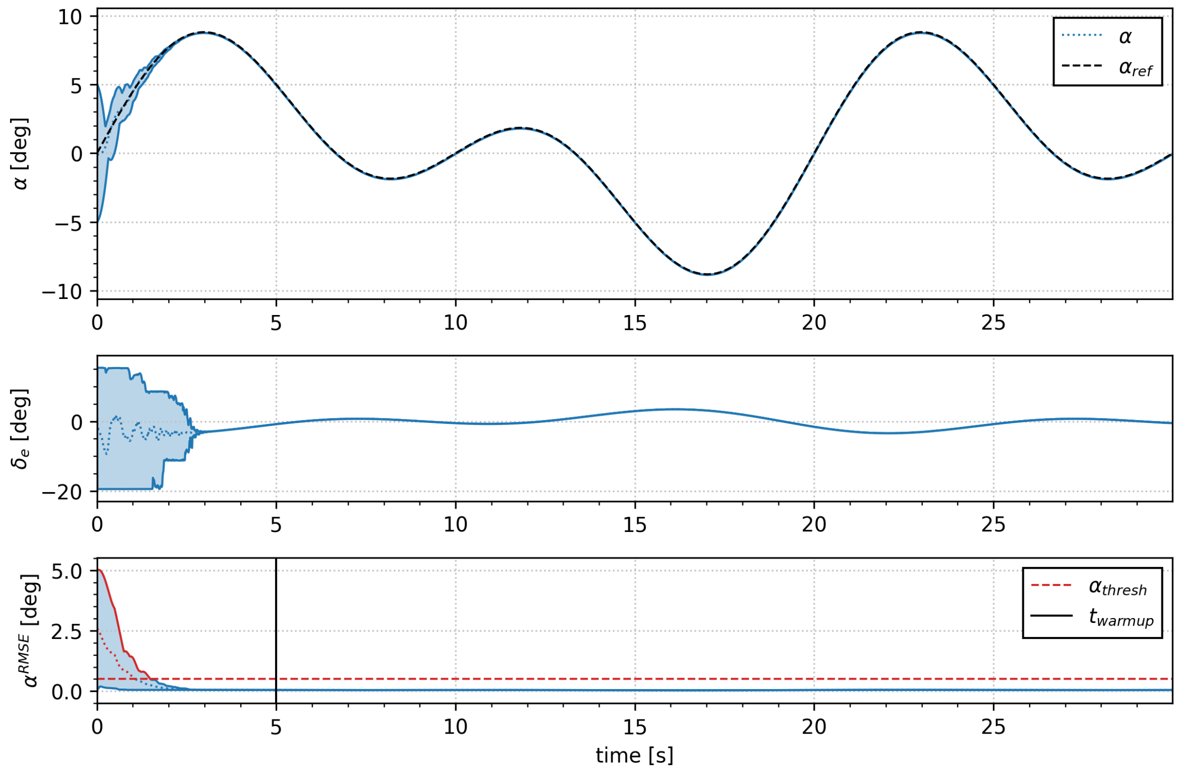


Figure 4.11: SAC and IDHP-SAC short-period response on 100 runs with untrimmed initialization

4.5.3. Robustness and Fault-Tolerance Analysis

A major point of interest in comparing these three frameworks is the fault-tolerance. Three cases are tested here which aim to demonstrate and evaluate the different levels of fault-tolerance. By altering the control and stability derivatives from Table 4.1 the dynamics of the short-period model are changed to simulate the failure cases. First, a mild failure is tested where the elevator effectiveness is reduced. Then, the effects of a sudden centre of gravity (c.g.) shift are evaluated. Lastly, a more extreme case is discussed where the effect of the elevator is reversed. All the alterations to the system are made at half the simulation time at $t = 30s$.

Decreased Elevator Effectiveness

The elevator effectiveness is reduced by 50% by multiplying the $C_{m_{\delta_e}}$ and $C_{Z_{\delta_e}}$ control derivatives by 0.5. Looking at the response for the IDHP controller in Figure 4.12, a successful tracking task is observed. The point made in subsection 4.5.1 regarding the tracking error threshold also appears in these results as oscillations are present when the tracking error only marginally reaches the threshold and the increase in learning rate is assumed to be too high for an optimal correction. nonetheless, the reduced elevator effectiveness at $t = 30s$ is barely detected in the state plot and shows only a minor oscillation in the elevator input. For this case, it is also interesting to observe the parameters of the RLS incremental model estimator as seen in Figure 4.15. This clearly shows the input matrix G reducing its pitch rate derivative by about 50% which corresponds with a halved elevator input effectiveness. Also, the covariance reset can be seen as the innovation term ϵ reaches the thresholds at the point of the change in system dynamics. This triggers the variances of the RLS parameters to be reset to the initial values. This step is necessary to ensure the adaptiveness over time when the forgetting factor is set to one [28]. The weights of the IDHP model as seen in Figure 4.14 also show the sudden increases at the points of reaching the tracking error threshold. In general, the IDHP performance is as expected as the incremental model is able to quickly reflect the change in system dynamics.

Similarly to the normal system dynamics simulation, the response of the SAC and hybrid controllers are again equal as seen in Figure 4.13. The SAC policy handles the change in elevator effectiveness without reaching the error threshold. In the state response, the change in dynamics at $30s$ goes unnoticed, while the elevator input plot shows very small oscillations at $30s$, $34s$, $38s$, $46s$, $55s$ and $58s$. This demonstrates the potential of the SAC policy to be robust to changes in the system dynamics.

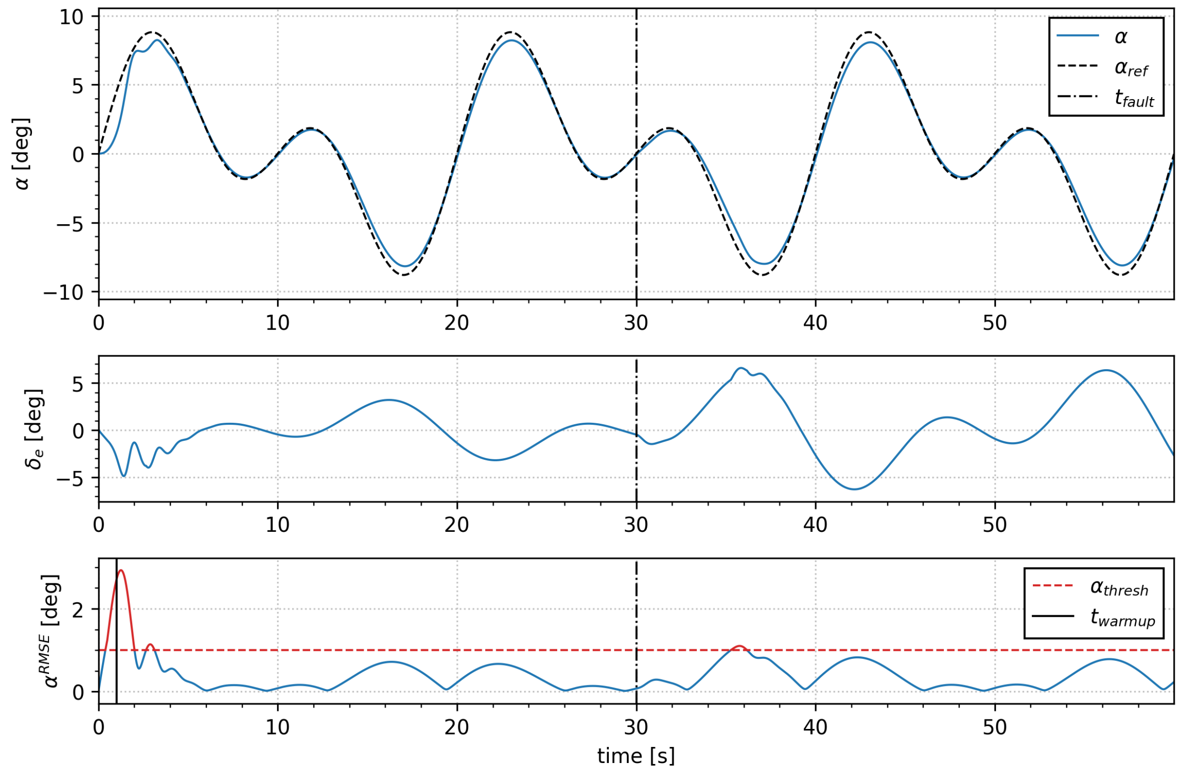


Figure 4.12: IDHP short-period response on 50% reduced elevator effectiveness

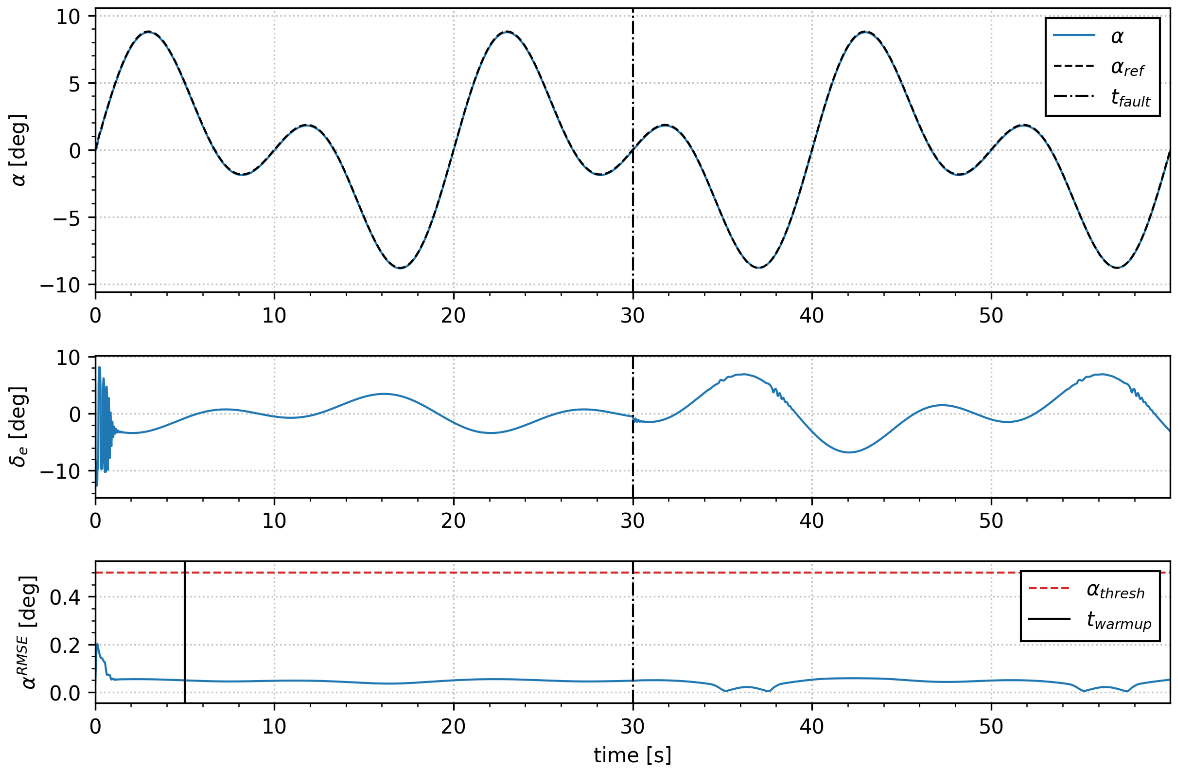


Figure 4.13: SAC and IDHP-SAC short-period response on 50% reduced elevator effectiveness

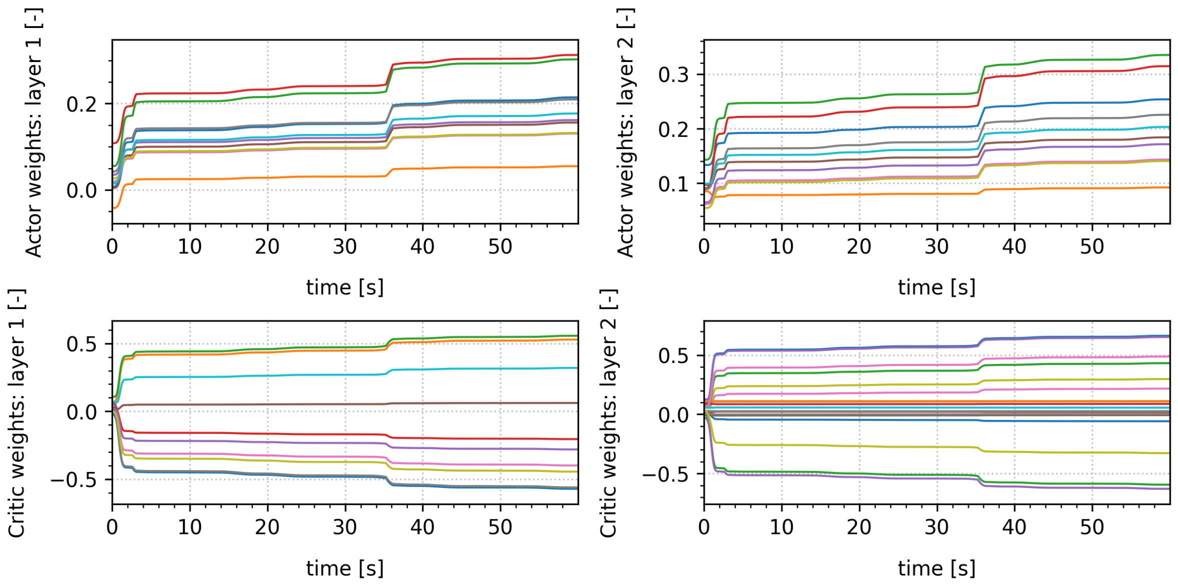


Figure 4.14: IDHP actor and critic weights on 50% reduced elevator effectiveness

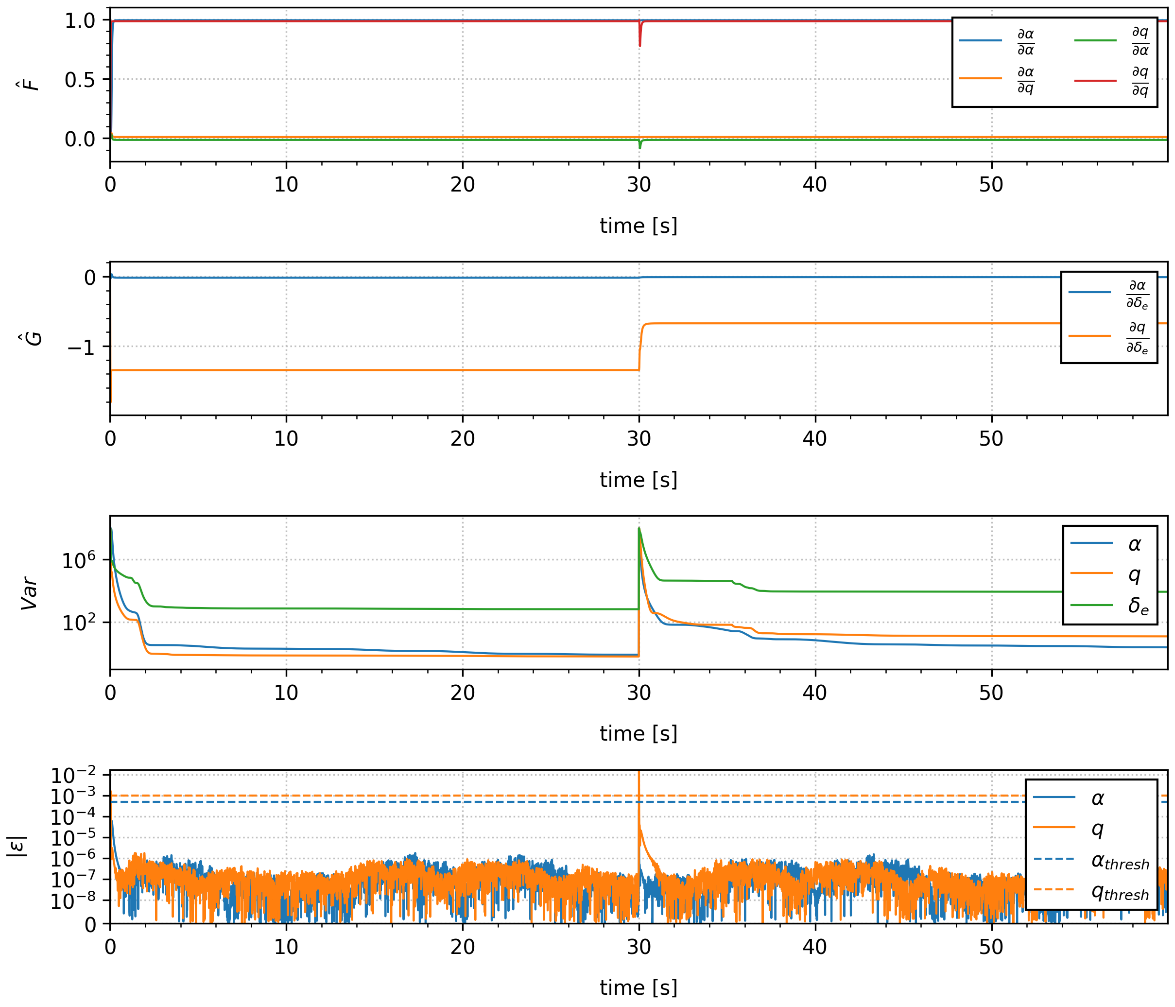


Figure 4.15: IDHP incremental model RLS estimator on 50% reduced elevator effectiveness

Sudden Center of Gravity Shift

To also test alterations in the stability derivatives, a shift in the c.g can be simulated which affects the C_{m_α}, C_{m_q}, $C_{m_{\dot\alpha}}$ and C_{Z_q} stability coefficients with their relation to the center of gravity given by Equation 4.30 [51]. In the following simulations, a c.g. shift of $\Delta x_{c.g.} = 1$ meaning $1m$ backwards is used.

$$\begin{aligned}
C_{m_{\alpha_2}} &= C_{m_{\alpha_1}} - C_{Z_\alpha}\frac{\Delta x_{c.g.}}{\bar{c}} \\
C_{Z_{q_2}} &= C_{Z_{q_1}} - C_{Z_\alpha}\frac{\Delta x_{c.g.}}{\bar{c}} \\
C_{m_{q_2}} &= C_{m_{q_1}} - (C_{Z_{q_1}} + C_{m_{\alpha_1}})\frac{\Delta x_{c.g.}}{\bar{c}} + C_{Z_\alpha}\left(\frac{\Delta x_{c.g.}}{\bar{c}}\right)^2 \\
C_{m_{\dot\alpha_2}} &= C_{m_{\dot\alpha_1}} - C_{Z_\alpha}\frac{\Delta x_{c.g.}}{\bar{c}}
\end{aligned} \tag{4.30}$$

In Figure 4.16, the response of the IDHP controller can be seen with the weights plotted in Figure 4.17. A successful tracking response can be seen, also after the expected pitch-up motion caused by the backwards c.g shift at $30s$. The progression of the weights also corresponds with the sudden learning rate increases due to the RMSE threshold. Notice that the learning rate threshold is not reached at the exact point of change, but only after, at around 37s. This can be explained by the larger tracking error at the maximum and minimum sinusoidal peaks from the normal response, which pushes the error above the threshold after the change in dynamics. A marginally higher steady-state error is observed after the transient response of the c.g. shift. Looking at the incremental model metrics in Figure 4.18, The threshold on the innovation is triggered at the point of the c.g. shift. Only the $\frac{\partial q}{\partial \alpha}$ element of the F matrix seems to adjust to accommodate the dynamics change.

For this simulation, the SAC and hybrid controllers have separate responses as the SAC controller has increased difficulty with this change in the dynamics as seen in Figure 4.19, but still produces a low tracking error. The state plot shows only a marginal steady-state error at the sinusoidal peaks with no observable transient in the state. The elevator input however shows string oscillations after the c.g. shift. This again indicates the policy might be too aggressive.

The response of the hybrid controller can be seen in Figure 4.20. From the state plot, the period before the c.g. shift is identical as the online learning is not engaged below the tracking error threshold. Looking at the elevator input, the SAC policy keeps the tracking error under the threshold until 42s. At that point, the tracking error at the sinusoidal peak reaches the threshold and the IDHP layers successfully adjust the policy which quickly results in a near-zero tracking error again. Strong oscillations are detected in the elevator input at the point of the IDHP adjustment. Compared to the pure SAC response, the steady-state error after 42s is slightly reduced, and the oscillations of the pure SAC elevator input are suppressed. The adjustment made by the IDHP layers is in this case desirable, but not necessary.

The weights of the IDHP portion of the hybrid controller as seen in Figure 4.21 show the identity initialization of the policy layers and the strong jump in the weights of the critic. Also, the incremental model metrics can be seen in Figure 4.22 with a similar response to the pure IDHP simulation.

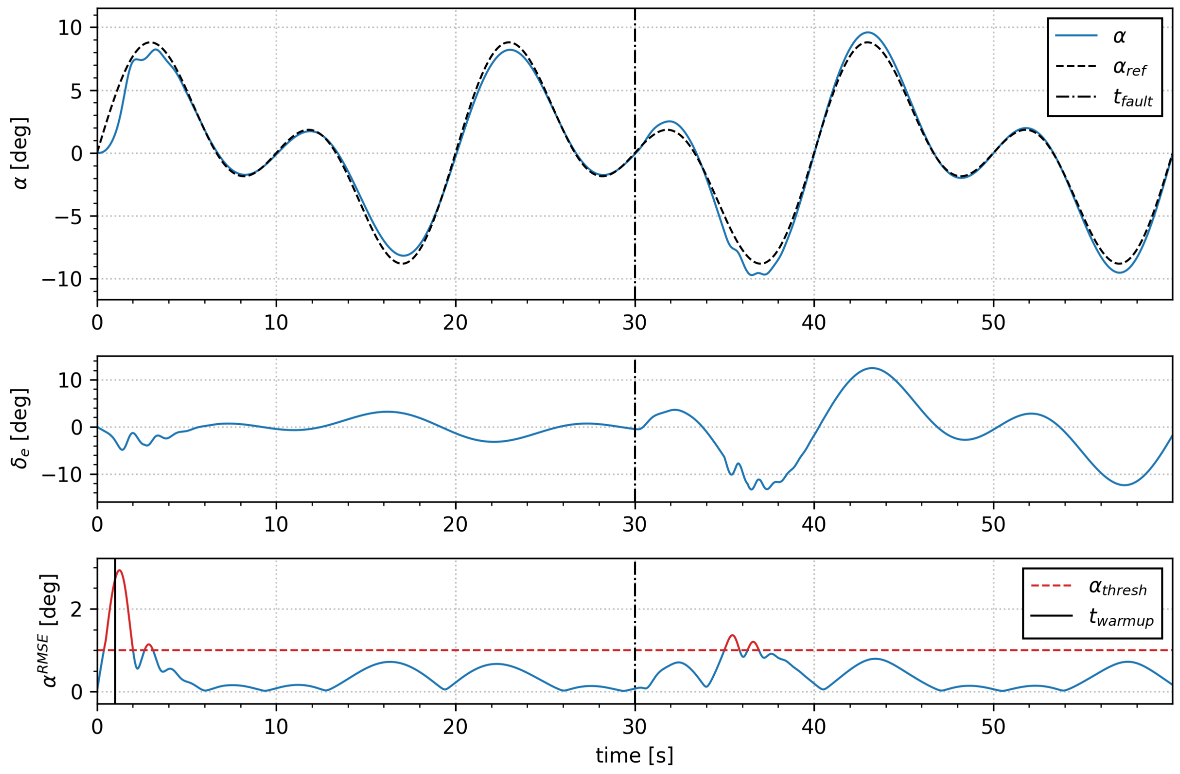


Figure 4.16: IDHP short-period response on c.g. shift

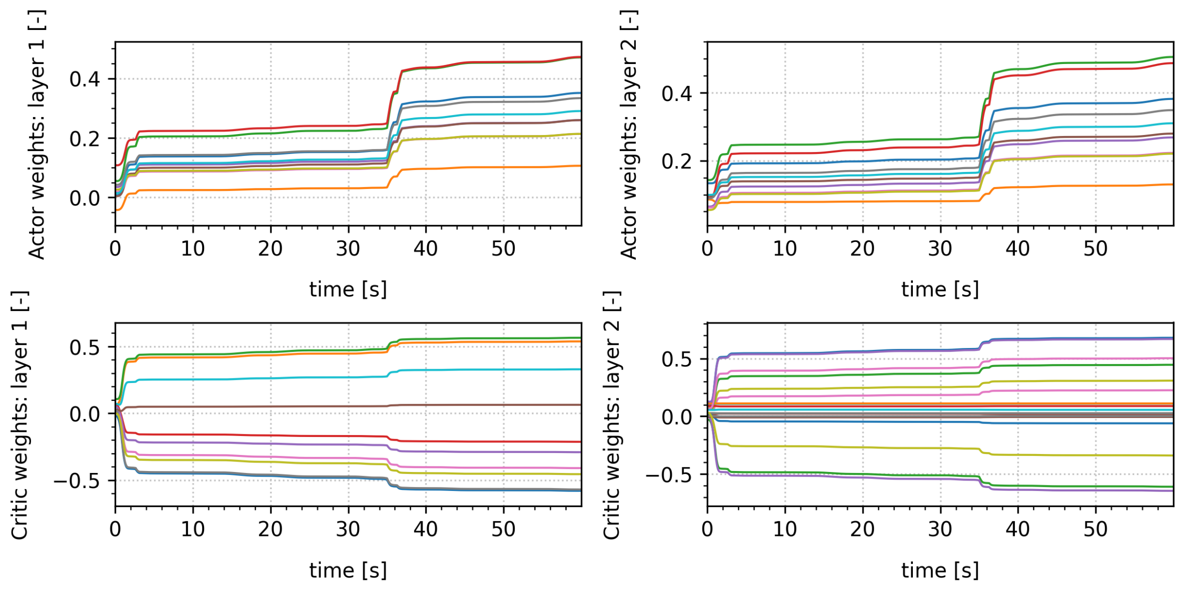


Figure 4.17: IDHP actor and critic weights on c.g. shift

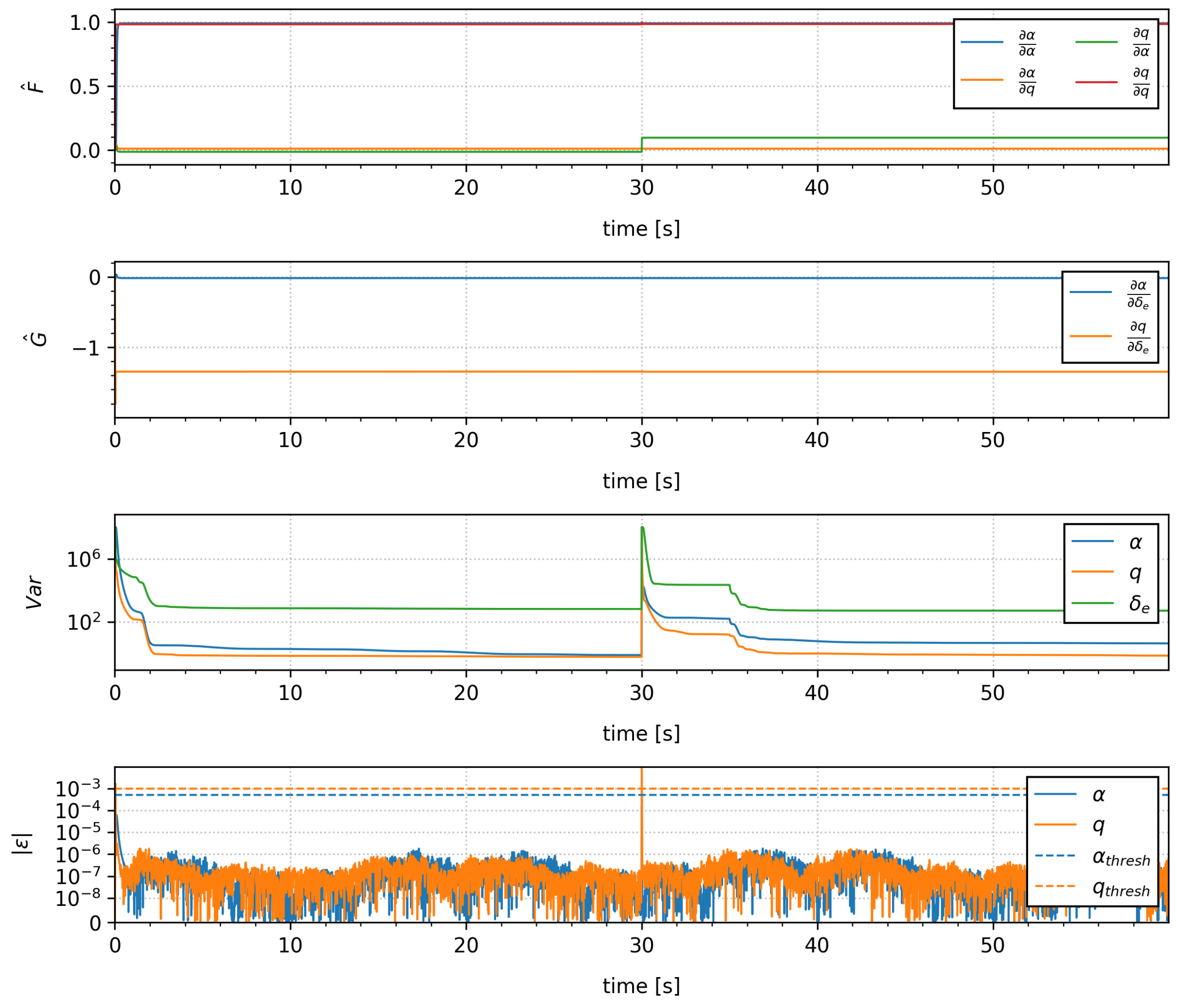


Figure 4.18: IDHP incremental model RLS estimator on c.g. shift

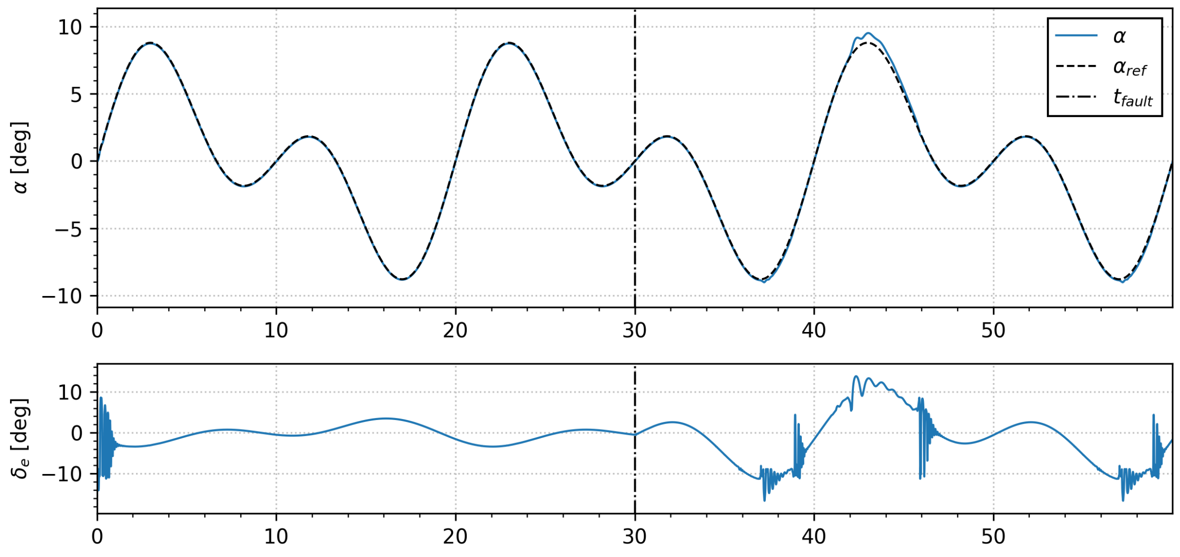


Figure 4.19: SAC short-period response on c.g. shift

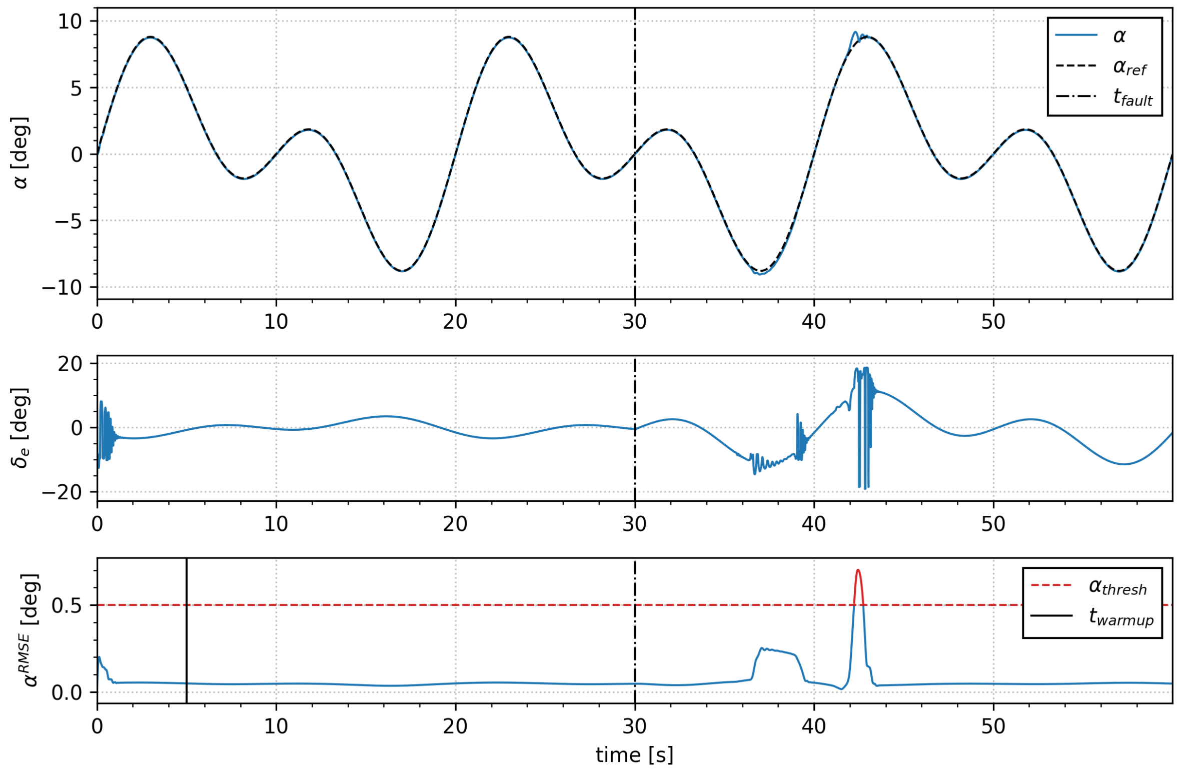


Figure 4.20: IDHP-SAC short-period response on c.g. shift

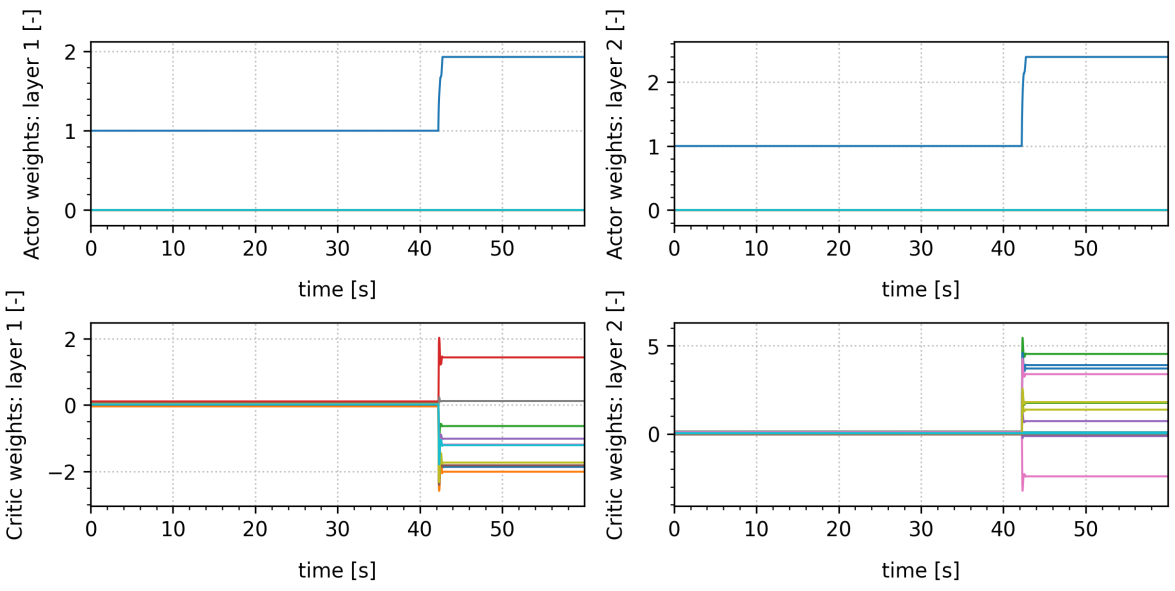


Figure 4.21: IDHP-SAC actor and critic weights on c.g. shift

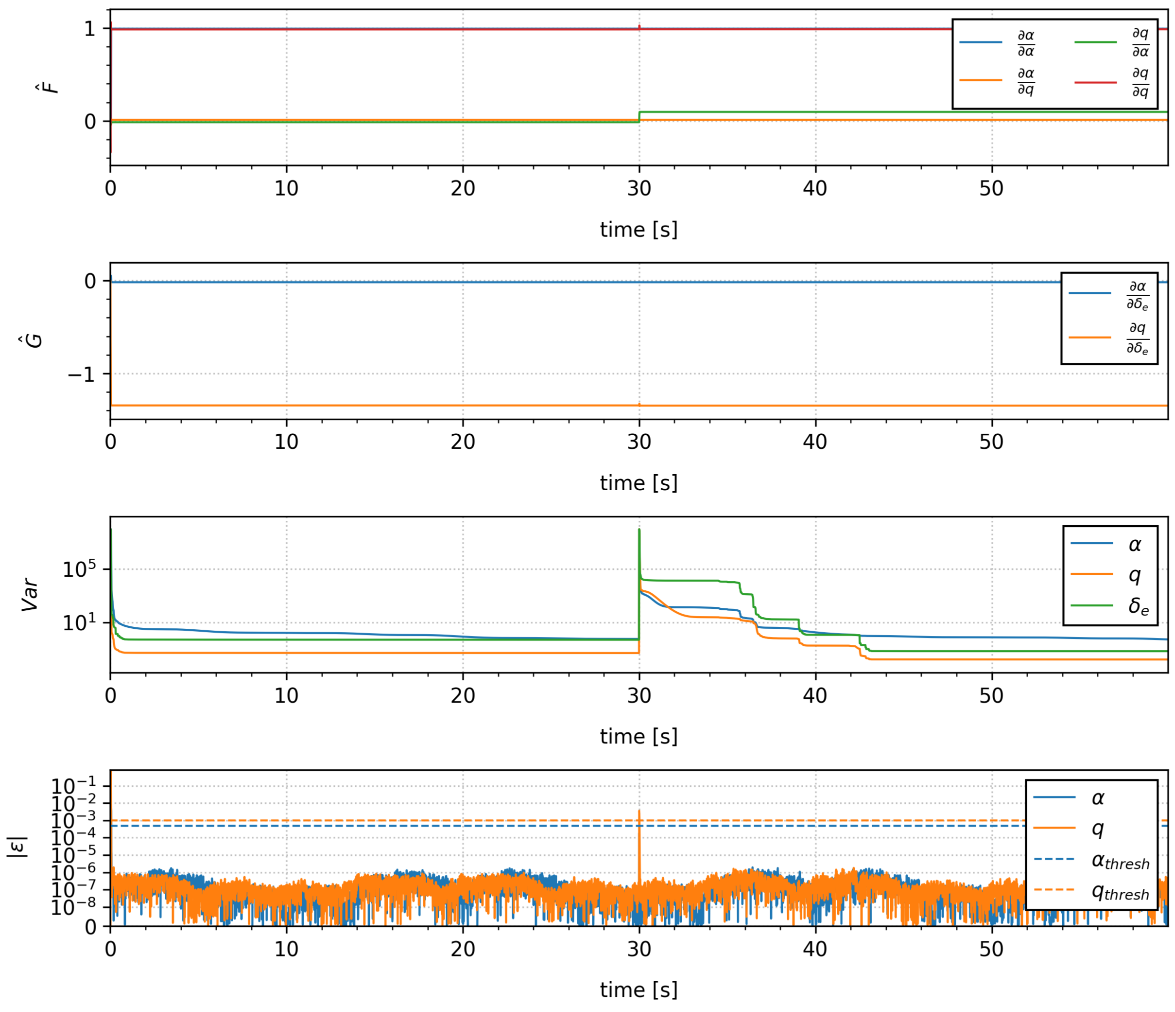


Figure 4.22: IDHP-SAC incremental model RLS estimator on c.g. shift

Inverted Elevator
A final more extreme failure mode is simulated by inverting the elevator input by multiplying the $C_{m_{\delta_e}}$ and $C_{Z_{\delta_e}}$ control derivatives with -1. For this case, the SAC controller is expected to fail, while the IDHP controller is expected to handle the extreme change by quickly adapting its incremental model. For this test, it is of special interest to see if the hybrid controller can utilize the online IDHP updates to overcome the extreme dynamics change better than the pure SAC controller.

First of all, the response of the IDHP controller can be seen in Figure 4.23. Successful handling of the inverted elevator can be seen as the RMSE threshold clearly getting reached right after the time of failure. In Figure 4.25, The input matrix is seen to adapt the pitch rate derivative correctly by changing the sign which is the expected change in the input matrix of the short-period model for an inverted elevator effect. The policy network weights as seen in Figure 4.24 also change sign. In the F matrix terms, a small peak is observed at the failure time, which is assumed to be a numerical issue. Again, some oscillations are present when reacting to the dynamics change. Also, the steady-state error seems to have decreased after the failure due to the period of higher learning rate.

For the pure SAC controller, the expected result can be seen in Figure 4.26. This dynamics change is too extreme for the SAC policy which fails to keep tracking the reference signal. The hybrid controller on the other hand successfully corrects for the inverted elevator as seen in Figure 4.27. Compared to the pure IDHP response, the hybrid controller reaches a smaller tracking error at the failure point. Oscillations in the input signal are however higher in frequency and magnitude than the pure IDHP controller. During manual hyperparameter tuning, it was observed that the rewards scaling factor can be increased substantially to reduce the tracking error on this failure mode, when on the pure IDHP controller this causes an unstable response. The increased aggressiveness set by the high reward scale does however seem to come at the cost of stronger oscillation in the input signal. Looking at the weight divergence in Figure 4.28, a similar pattern to the c.g. shift simulation can be seen where the shifts in mainly the critic weights are large and sudden, likely caused by the high reward scale. As seen from Figure 4.29, the incremental model behaves the same as in the pure IDHP simulation where the inverted elevator input is quickly updated in the G matrix.

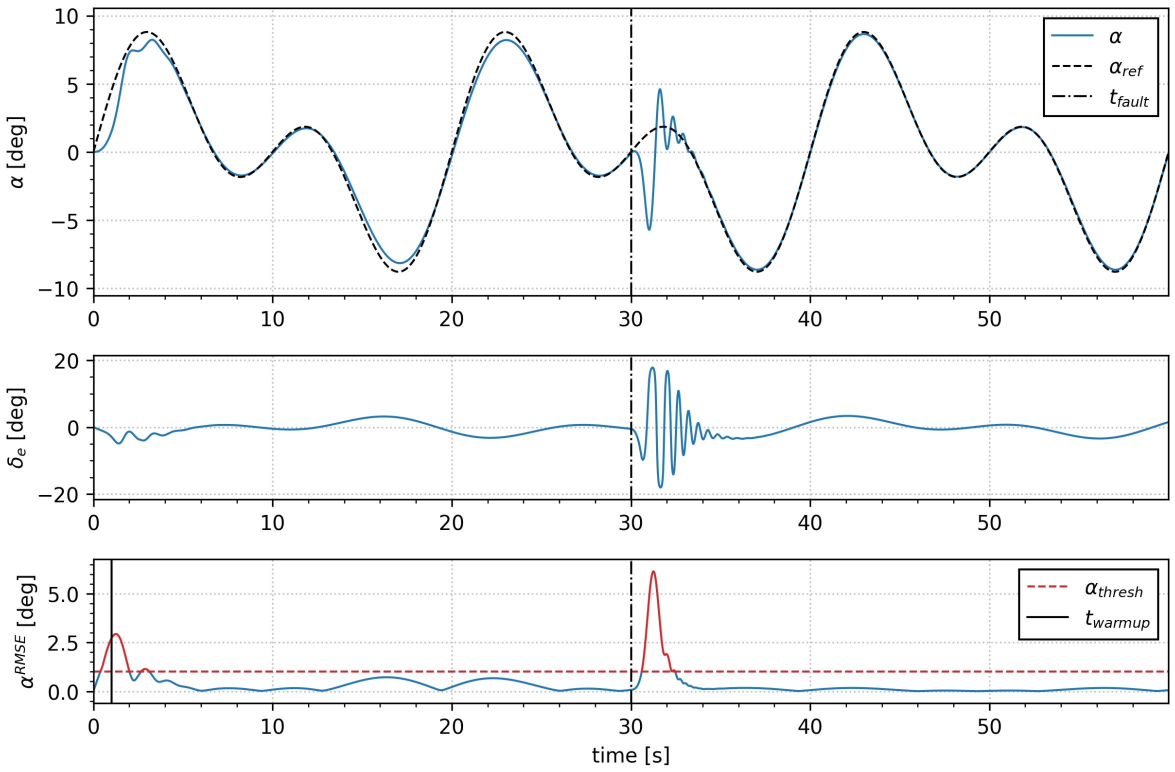


Figure 4.23: IDHP short-period response on inverted elevator

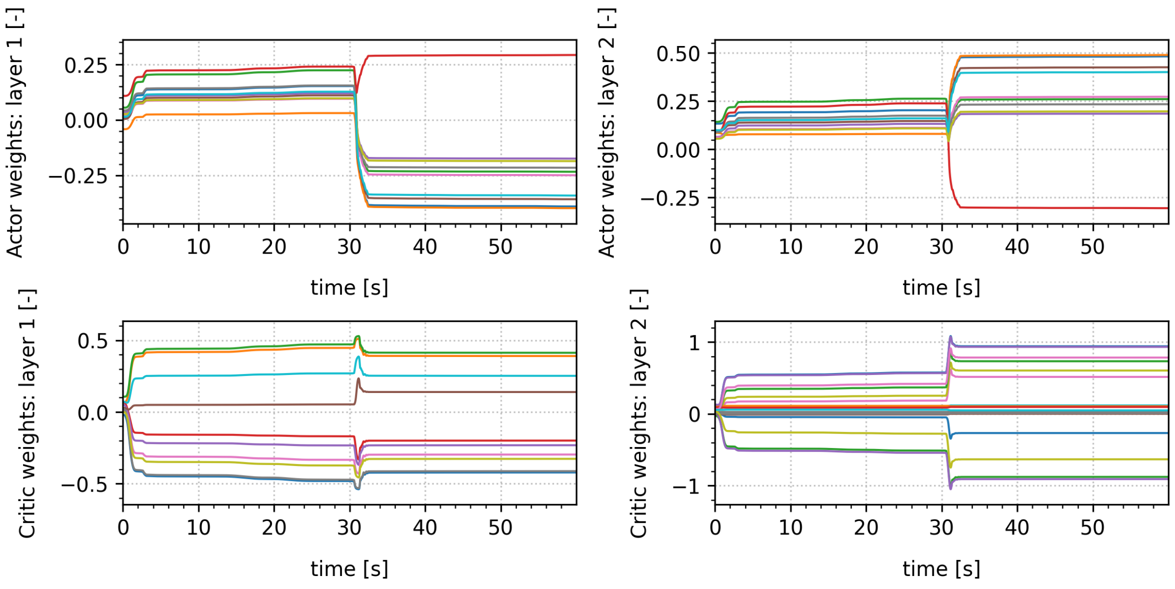


Figure 4.24: IDHP actor and critic weights on inverted elevator

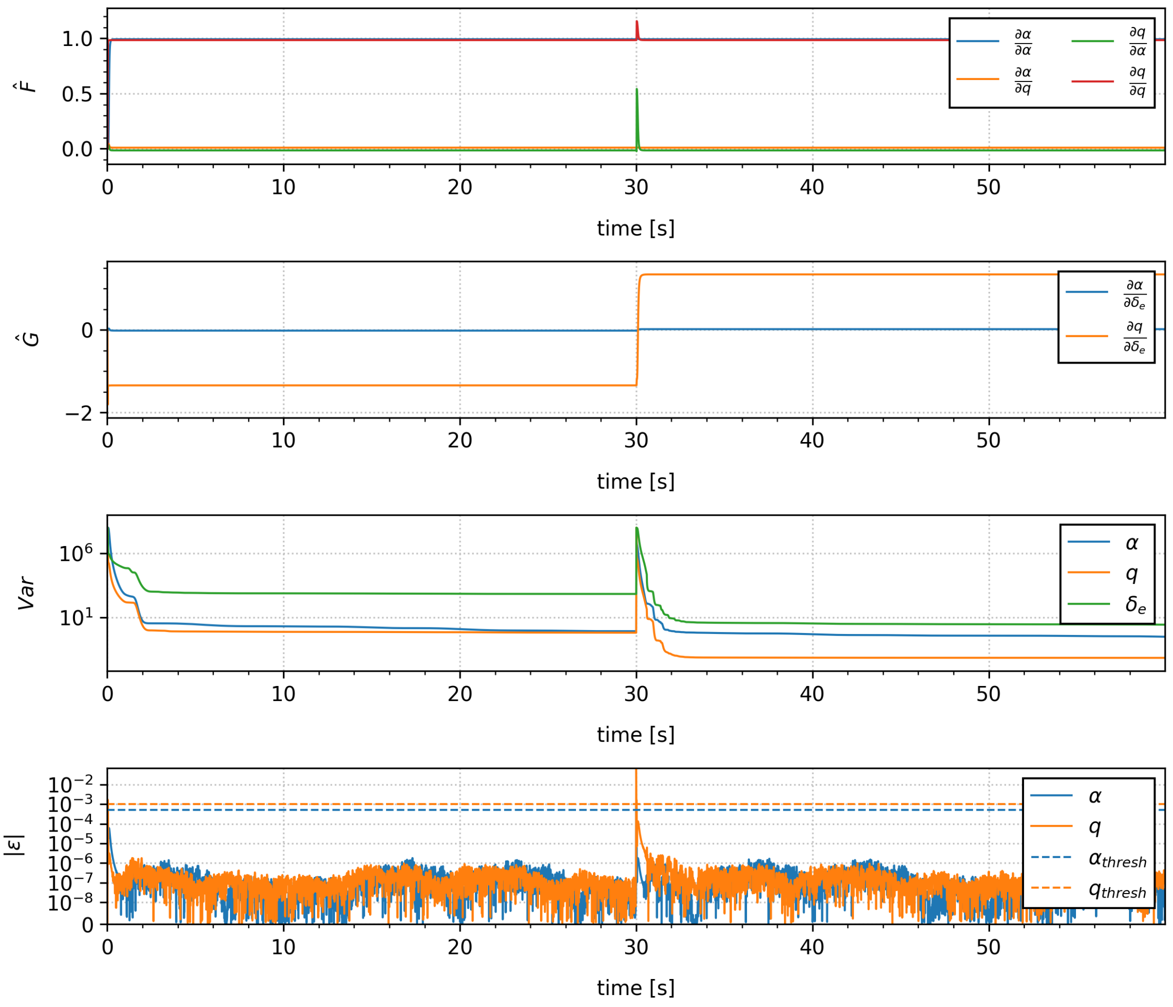


Figure 4.25: IDHP incremental model RLS estimator on inverted elevator

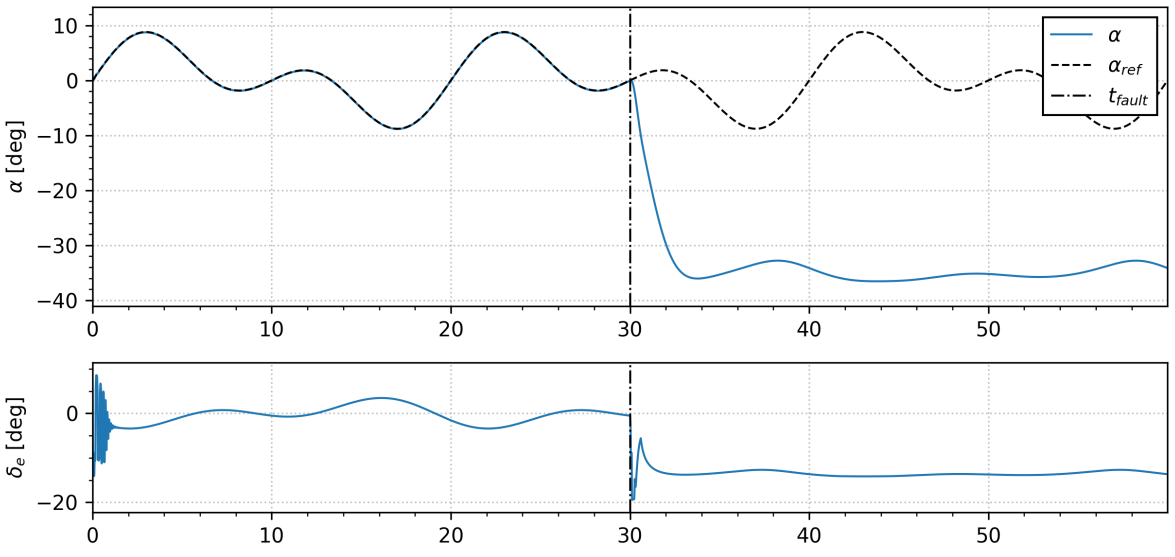


Figure 4.26: SAC (failed) short-period response on inverted elevator

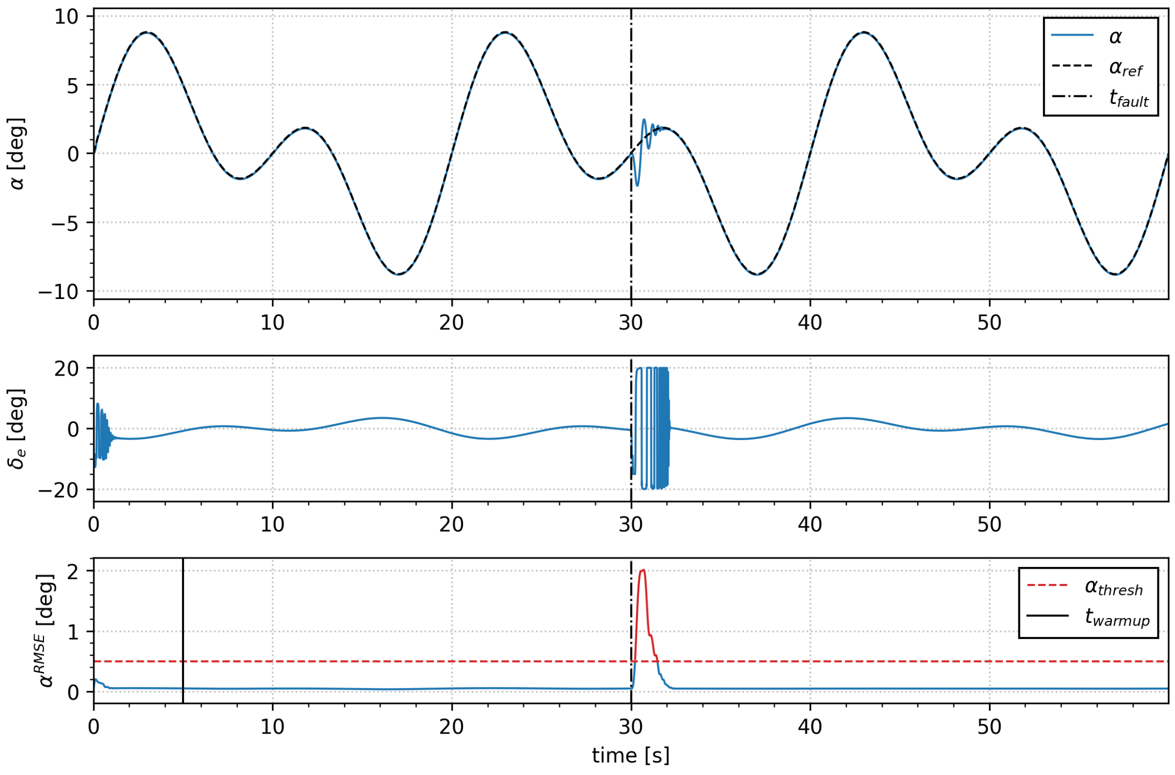


Figure 4.27: IDHP-SAC short-period response on inverted elevator

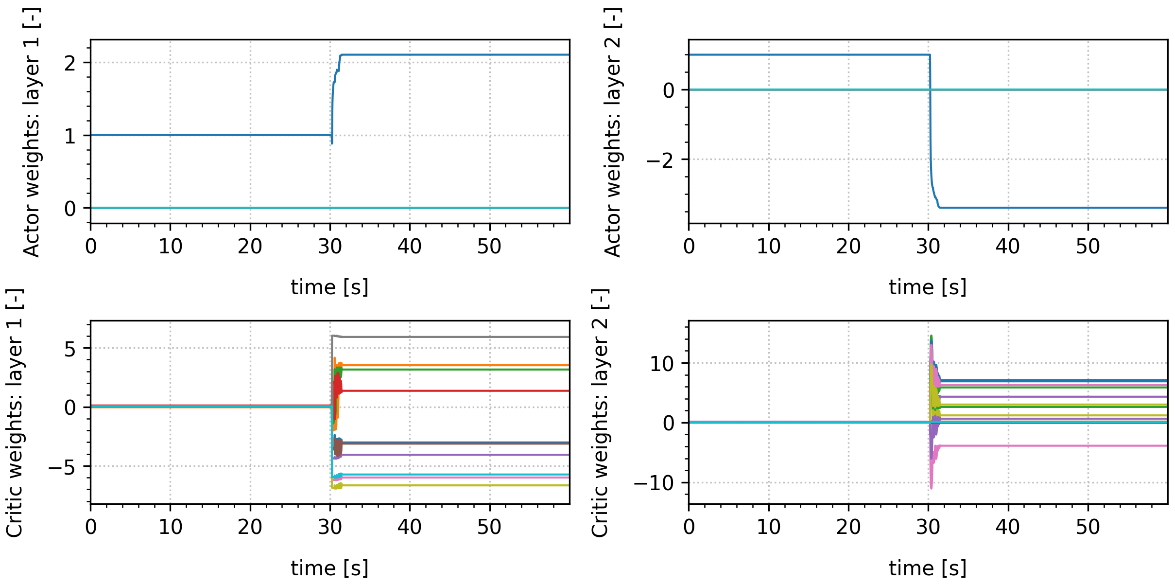


Figure 4.28: IDHP-SAC actor and critic weights on inverted elevator

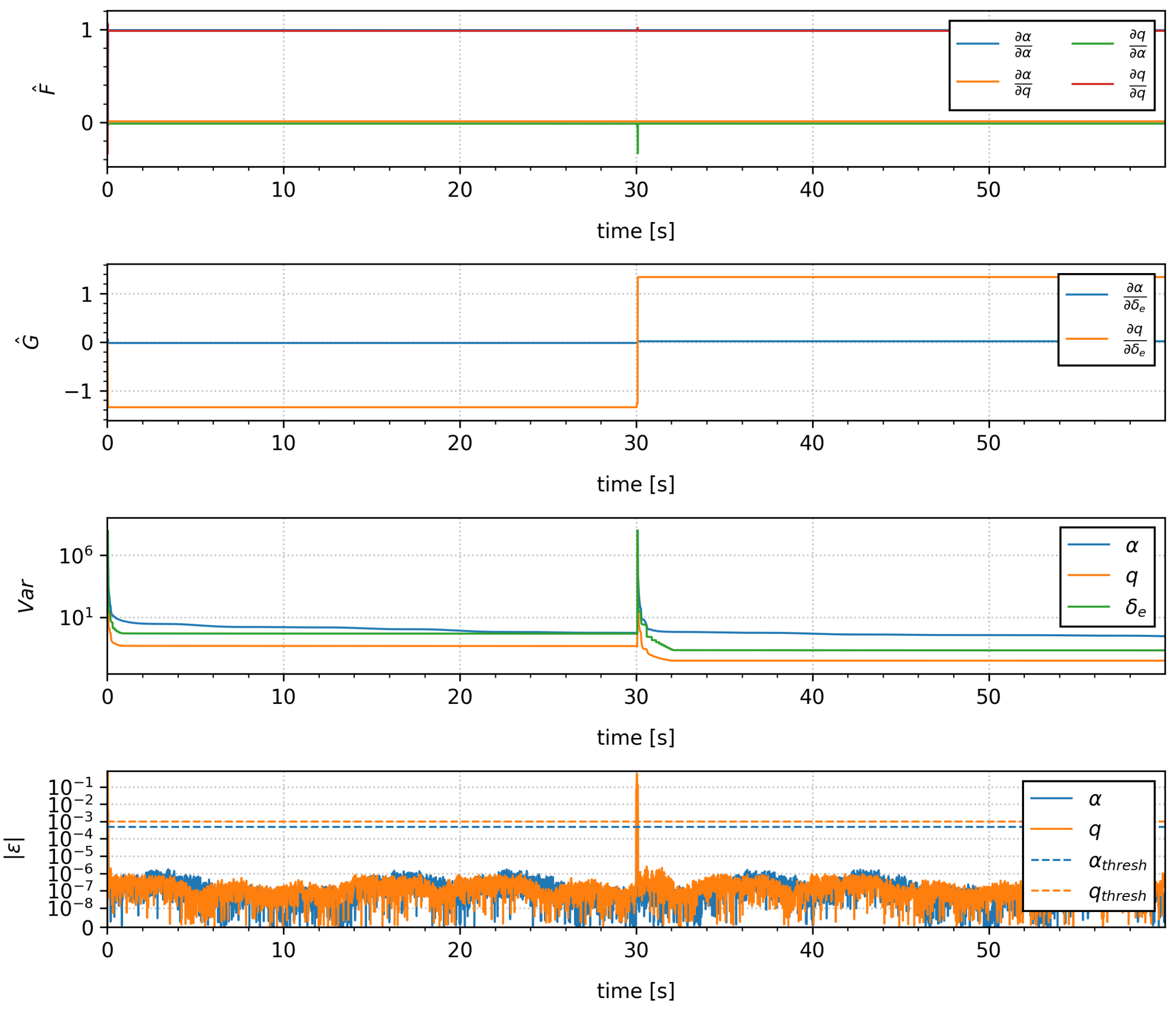


Figure 4.29: IDHP-SAC incremental model RLS estimator on inverted elevator

4.6. Conclusion Preliminary Analysis

During the preliminary implementation of the IDHP, SAC and hybrid frameworks as discussed in section 4.2, section 4.3 and section 4.4 respectively, *RQ1.3* has been answered. From the results of the short-period simulations, an answer to *RQ1.4* can be formulated. Note that the performance of the three RL framework candidates was tested on the angle-of-attack tracking task which has a less direct dynamic relationship to the elevator input than for example a pitch rate tracking task. This demonstrated the ability of the RL controllers to handle more challenging dynamics. This is likely also the cause for the oscillations present in all the simulation results. Adding more information in the observation vector for example the pitch rate and angle of attack states and adding a reference signal for the pitch rate is expected to improve this.

The advantages and disadvantages of IDHP and SAC have been demonstrated and are in line with expectations. Fault tolerance of the IDHP controller is high with fast convergence during online learning. On the other hand, the SAC agent can provide robustness against some level of failure while having no issues with continual learning. The SAC controller still fails to adapt to more extreme failure cases which the IDHP controller still manages to control. From this preliminary analysis, it seems feasible to implement a combined policy that effectively combines the advantages of IDHP and SAC. This was implemented here by using the offline-trained SAC policy as a reference for a policy fine-tuning process executed online by the IDHP framework. Performance on the previously tested simulations indicates the combined controller performs either as good as or better than either IDHP or SAC separately in terms of reducing tracking error. Additionally, the effect of the IDHP online learning can be scaled using the adaptive learning strategy where a zero low learning rate was deployed in the simulations. Implementing a continuously adaptive learning rate instead of the high-low strategy is a potential improvement as a large gap between the high and low learning rate, as is the case with a zero low learning rate, showed oscillatory behaviour. Generally, high sensitivity to hyperparameters was experienced during manual tuning. An automated hyperparameter search is to be implemented for the final controller design. With this, *RQ1.4* is answered. Additionally, oscillatory behaviour can be managed by implementing a rate limit on the input signal or using the policy output as control increments instead of directly controlling the elevator with the policy output as implemented in [11]. An implementation of a similar rate limit is planned for the final thesis phase.

For the continuation of this research, the three RL framework candidates of IDHP, SAC and the hybrid form remain. In developing a complete flight controller for the PH-LAB aircraft, an inner and outer loop controller has to be developed. In [11] a coupled SAC inner and outer loop have proven successful, providing the most promising starting point regarding controller design. Hence the possibilities exist to augment either the inner, outer or both loops of a SAC flight controller with the combined policy and increase the adaptability to failure.

III

Additional Results

5

Decoupled Hybrid Attitude Controller

The hybrid controller architecture presented in Part I consists of a single actor, critic and incremental model for the longitudinal and lateral states proving knowledge of any coupling effects to the IDHP algorithm. Previous IDHP implementations have either only focused on longitudinal control [36], utilized a split actor structure [34] or utilized a fully separate IDHP model for longitudinal and lateral states by splitting up the observations and having two actors, critics and incremental models [28]. Part of the implementation phase that resulted in the proposed coupled hybrid controller involved developing a decoupled version of the hybrid controller which is presented in this section.

In Figure 5.1, the inner attitude control loop can be seen with the longitudinal and lateral states split according to Equation 5.1, excluding the altitude state as it is only used by the outer altitude controller of the cascaded design. The RL-state and tracking reference vectors are also split up accordingly. Due to the coupled design of the pre-trained SAC policy, the hybrid policy is constructed like the coupled hybrid policy, but only the longitudinal or lateral outputs are used in their respective controllers. This is a limitation of the decoupled design when using the pre-trained SAC policy layers. Furthermore, each controller has its own critic and incremental model and the outputs are concatenated before being sent to the environment. By splitting the states this way, the learning load on the critic and incremental model are reduced, but computational complexity is increased by performing twice the amount of updates per time-step for longitudinal and lateral separately.

$$\mathbf{x}^{lon} = [q, \alpha, \theta] \qquad \mathbf{x}^{lat} = [p, r, \phi, \beta] \tag{5.1}$$

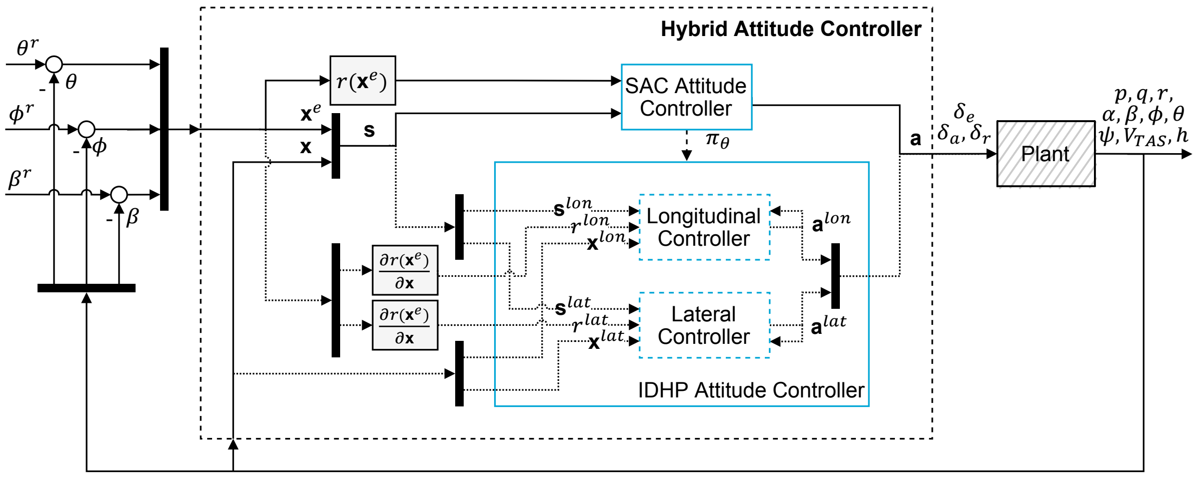


Figure 5.1: SAC-IDHP Decoupled Attitude Controller Structure

5.1. Decoupled Hybrid Online Training

The online training task for the decoupled hybrid controller is performed using the same reference signals and hyperparameters used for the nominal case in Part I. In Figure 5.2 the response to the online training task shows a successful converging hybrid policy on both the longitudinal and lateral states. Compared to the online training task presented in Part I, the initial convergence phase appears less aggressive and has no increased deviations from the reference signal during the initial time-steps. like the coupled design, the training task demonstrated the ability of the hybrid controller to correct for any existing state-state errors form the SAC policy with reduced errors at the peaks of the sinusoidals after approximately 15s on the pitch reference and 25s on the bank reference. The sideslip angle shows little improvement as the SAC policy alone achieves a low error. Comparing the performance, the SAC-only has an nMAE of 10.31% and the decoupled hybrid an nMAE of 9.03% which is 1.62% higher than the coupled version. This increased error compared to the coupled version is assumed to be because of the decoupling resulting in less aggressive gradient steps due to differing and/or smaller state spaces on the critic input vector and reward function inputs. Hence, a similar tracking performance is assumed to be achievable by performing an additional hyperparameter search for longitudinal and lateral controllers separately.

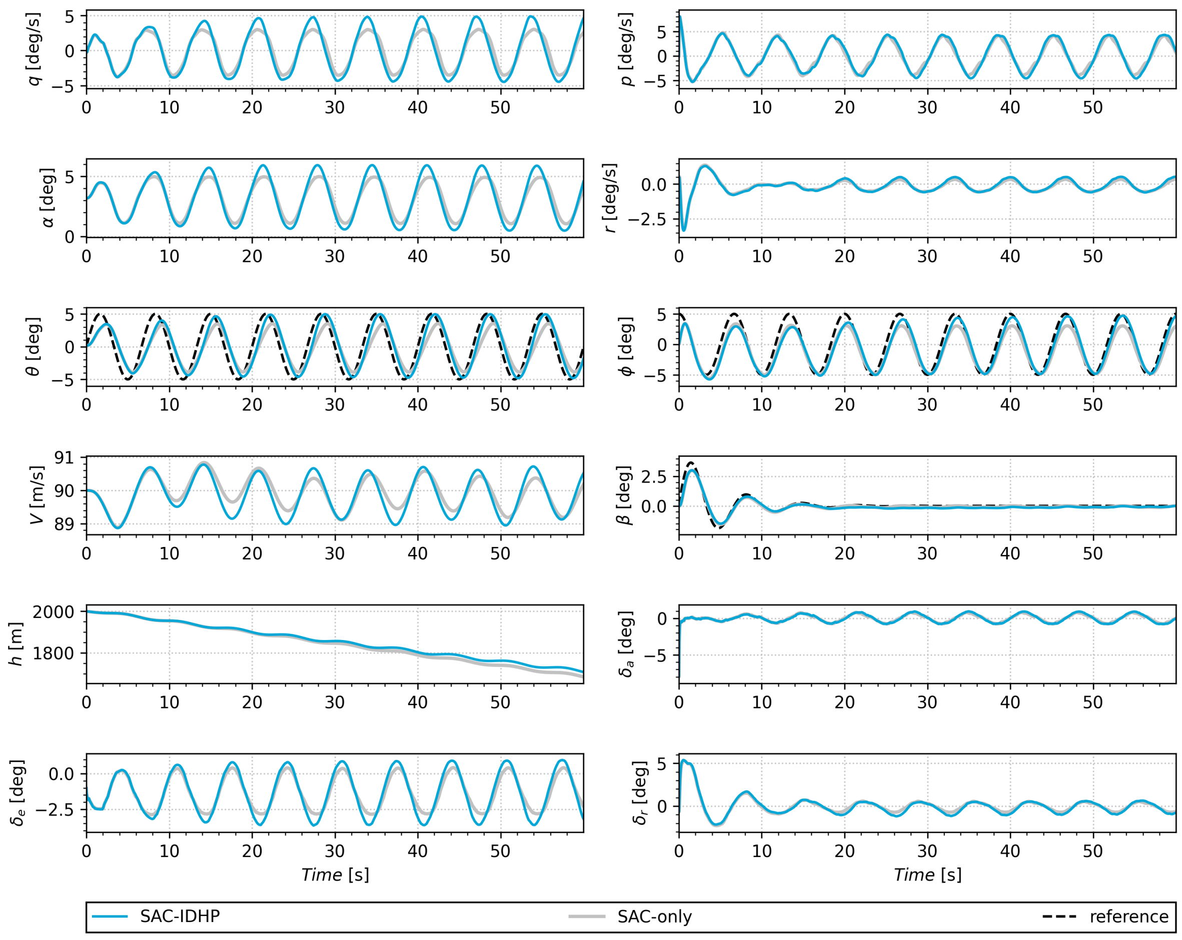


Figure 5.2: SAC-IDHP Decoupled response on Attitude training task.

The actor/critic weights and incremental model parameter evolution during the training task can be seen in Figure 5.3 for the longitudinal agent and in Figure 5.4 for the lateral agent. These plots confirm the earlier observations that both agents converged successfully, and that the learning is less aggressive than with the coupled version using the same hyperparameters indicated by the steadily increasing actor weights. The F and G matrices of the incremental model converge in under 5s for the longitudinal and under 1s for the lateral models which is an improvement from the coupled model. This demonstrates the expected lower learning complexity of the decoupled models compared to a single model estimating coupled dynamics.

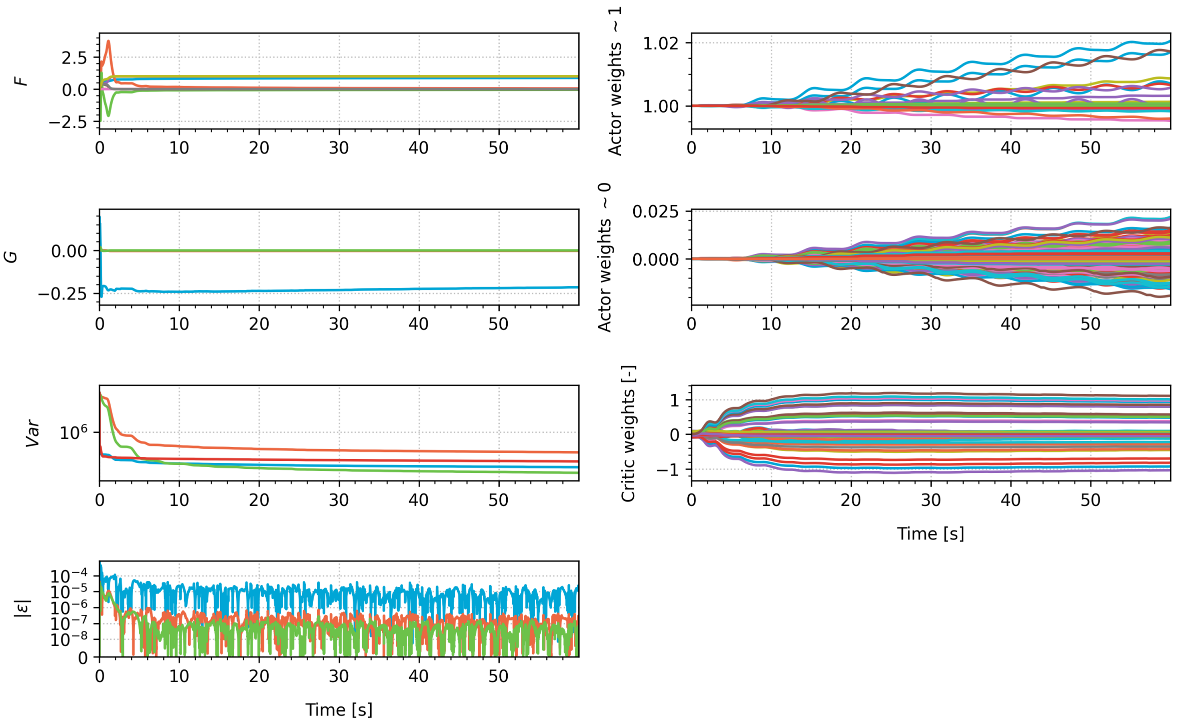


Figure 5.3: SAC-IDHP Decoupled response on Attitude training task, actor/critic weights and incremental model parameters longitudinal.

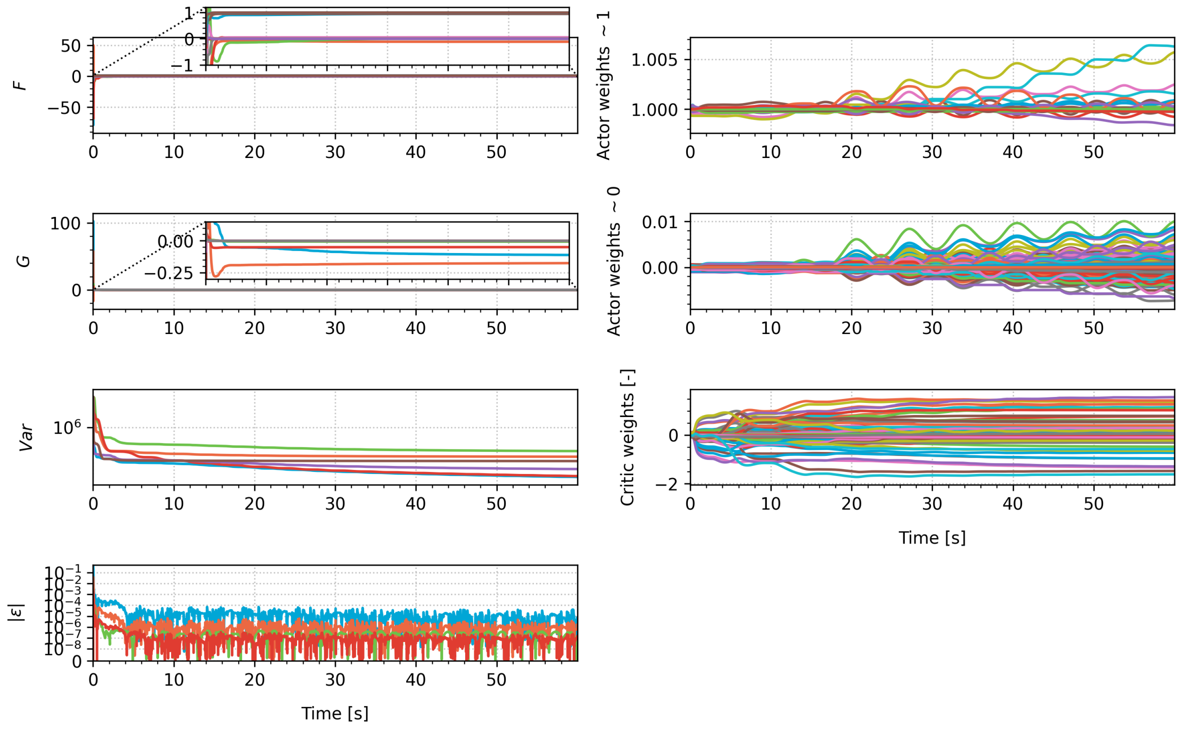


Figure 5.4: SAC-IDHP Decoupled response on Attitude training task, actor/critic weights and incremental model parameters lateral.

5.2. Comparison on Fault-Tolerance with Coupled Controller

To assess the effect of the decoupling on the adaptive response to failure cases, the partial loss of horizontal tail failure mode is chosen to focus on the longitudinal failure mode. The reference signals are set at a constant $8°$ for the pitch angle in order to simulate a steady climb, and an alternating bank angle of $20°$ with zero sideslip for coordinated turns. The same learning rates from Part I are used.

The response of the decoupled agent can be seen in Figure 5.5 and the response of the coupled controller in Figure 5.6. Both hybrid controllers provide a large correction to the elevator action, resulting in a lower tracking error compared to the SAC-only agent with an nMAE of 21.25% for the SAC-only controller and 5.21% and 6.91% for the decoupled and coupled hybrid controllers respectively. This shows the decoupled controller has a 1.7% lower tracking error compared to the coupled controller. This can be attributed to the reduced learning complexity of the decoupled controller, resulting in faster corrections. Also, the lateral states of the coupled controller are more affected by the adaptive response with a slightly larger sideslip error, contributing to the overall larger error.

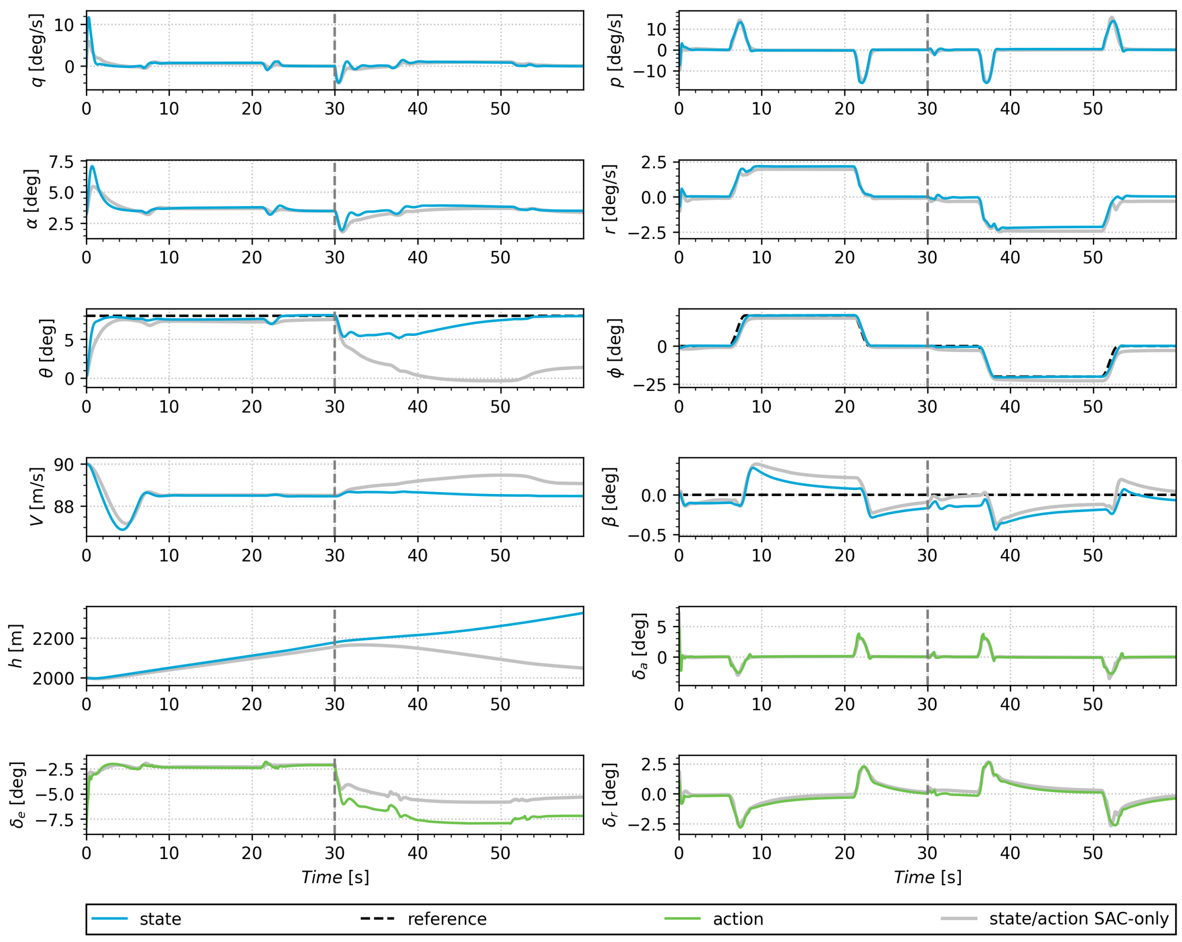


Figure 5.5: SAC-IDHP Decoupled response on Attitude task with partial loss of horizontal tail at t=30s.

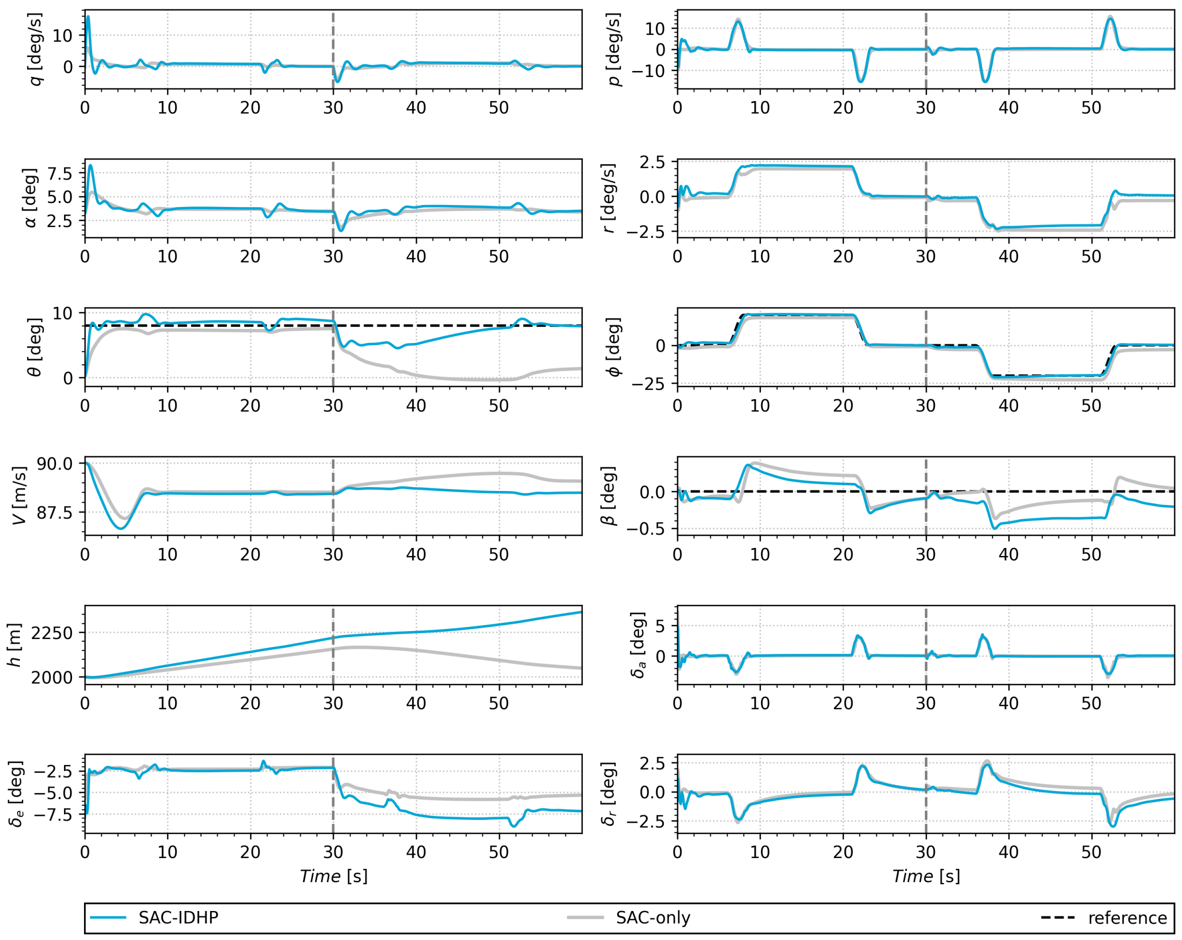


Figure 5.6: SAC-IDHP Coupled response on Attitude task with partial loss of horizontal tail at t=30s.

The actor/critic weights and incremental model parameters for the decoupled controller can be seen in Figure 5.7 and Figure 5.8 for longitudinal and lateral controllers respectively. The lateral parameters are expected to not change significantly, which is reflected by the results. The lateral parameters do appear to be affected by continuous learning, noted by small changes in policy parameters and model parameters around the times of excitation of the roll axis. The longitudinal parameters however adjust from the 30s mark when the system dynamics changes are detected by the jump in the innovation term. A small change in input matrix is seen accompanied by changing actor/critic weights. Note that because of the 1.0 forgetting term of the incremental model, the adaptiveness can be limited over lime without an additional fault detection system, as discussed in chapter 7. The parameter evolution of the coupled controller can be seen in Figure 5.9 and shows the changing actor/critic weights at the 30s point as expected with the same jump in the innovation term.

Overall, the decoupled controller has the advantage of lower learning complexity, but with considerably higher computational and implementation complexity. Also in the case of more severe coupled failure modes, it is expected that a coupled model can better model the system dynamics.

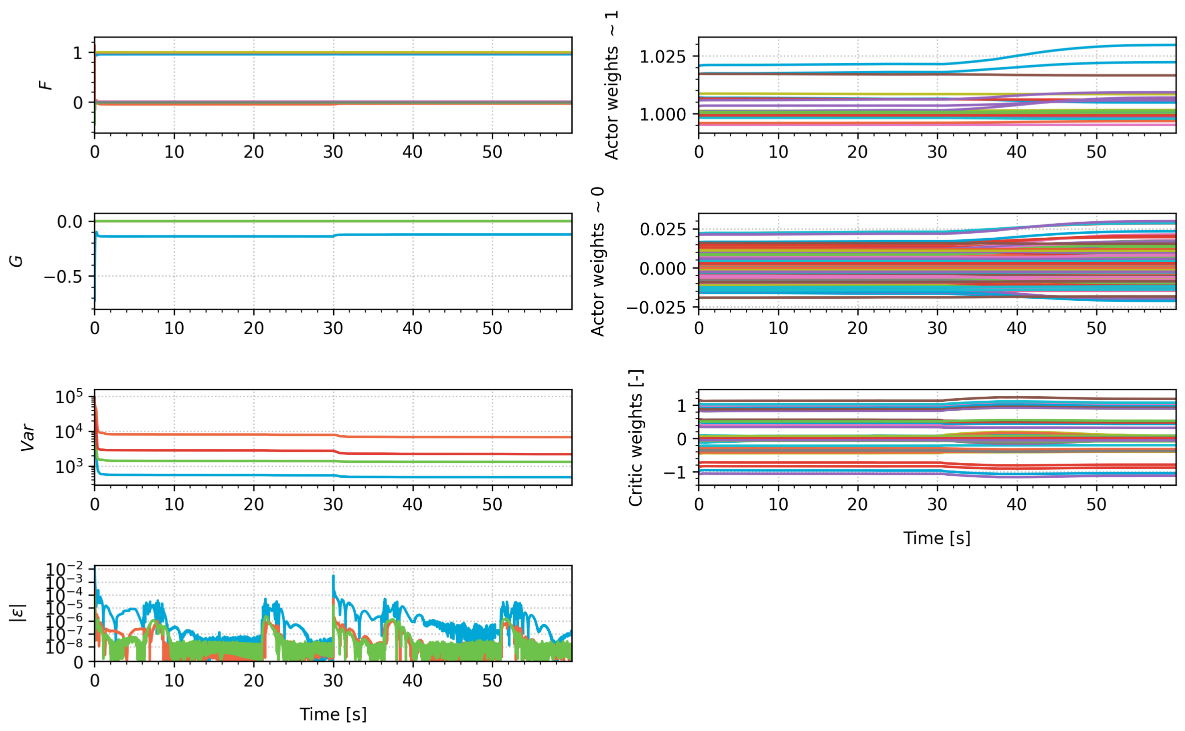


Figure 5.7: SAC-IDHP Decoupled Attitude Controller with partial loss of horizontal tail at t=30s, actor/critic weights and incremental model parameters longitudinal.

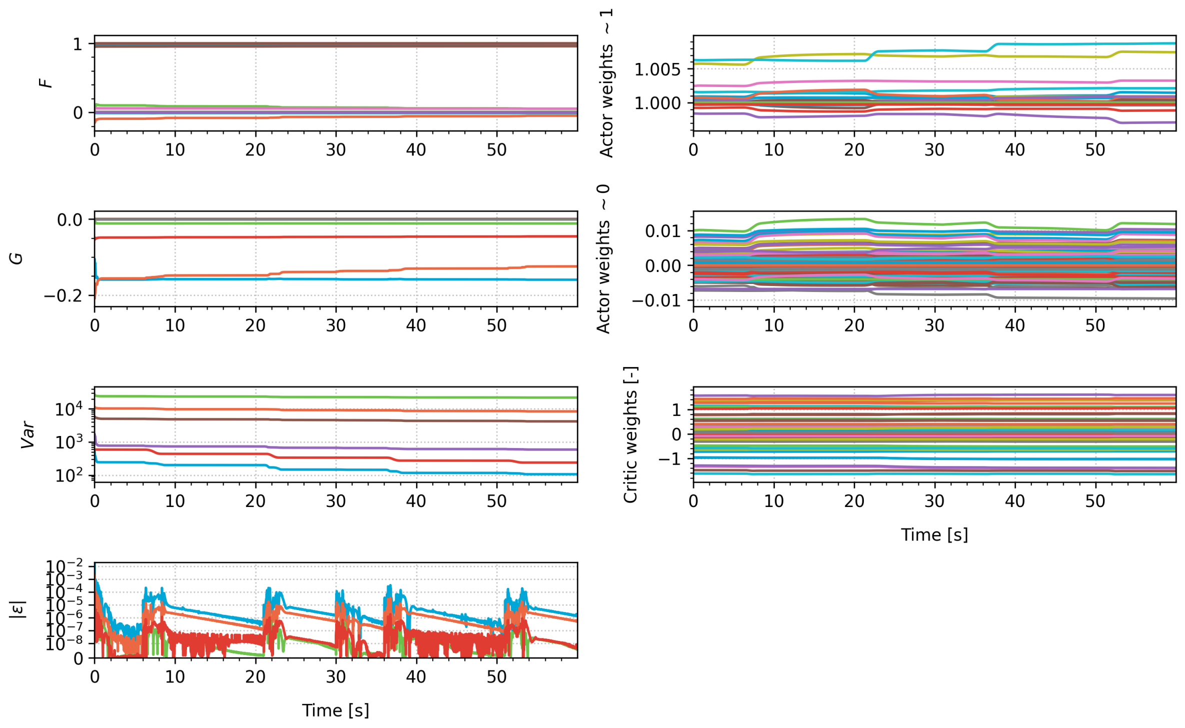


Figure 5.8: SAC-IDHP Decoupled Attitude Controller with partial loss of horizontal tail at t=30s, actor/critic weights and incremental model parameters lateral.

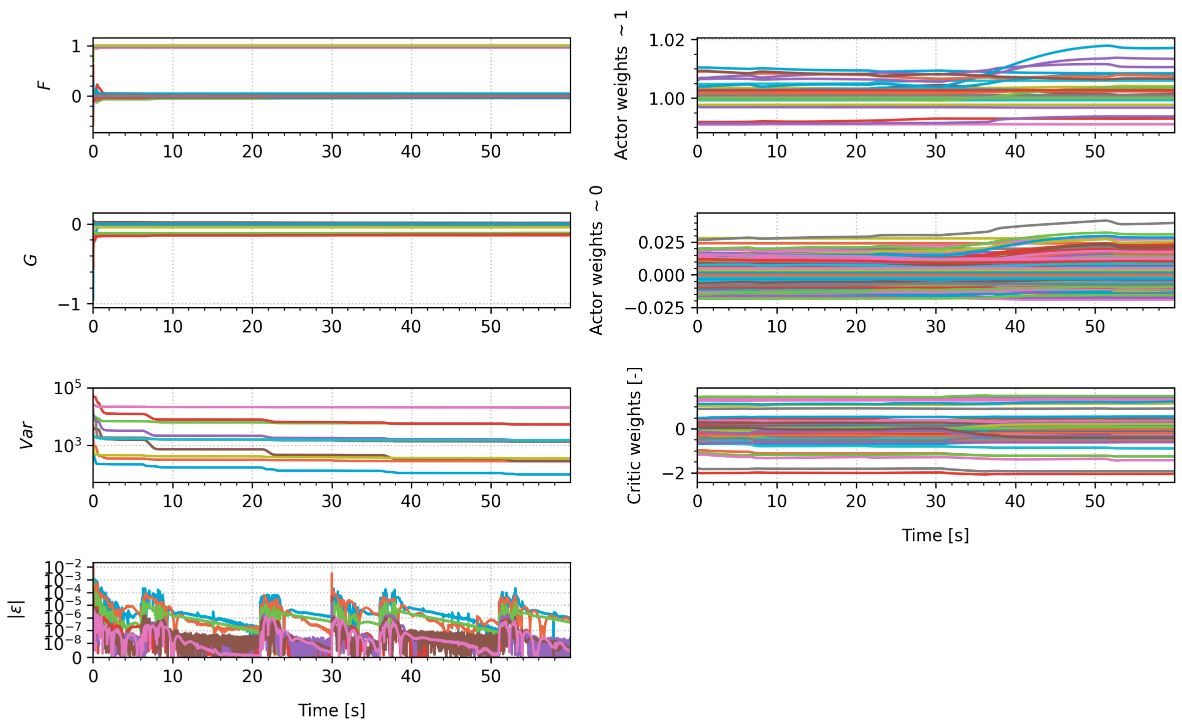


Figure 5.9: SAC-IDHP Coupled Attitude Controller with partial loss of horizontal tail at t=30s, actor/critic weights and incremental model parameters.

6 Robustness to Initial Flight Condition

As presented in Table 6.1 from Part I, the SAC and Hybrid cascaded controllers have been tested on three additional initial flight conditions differing from the initial flight condition used to train the SAC agents. This section discusses the response on each initial flight condition in more detail. The altitude task with a maximum of 20° bank angle is used. The flight conditions are ordered in order of increasing dynamic pressure.

Table 6.1: Robustness to initial flight conditions of cascaded altitude controllers.

Flight Condition	Initial Altitude [m]	Initial Airspeed [m/s]	nMAE SAC-only	nMAE SAC-IDHP
FC1	5000	90	4.71%	1.96%
FC2 (nominal)	2000	90	2.76%	2.02%
FC3	5000	140	2.05%	1.75%
FC4	2000	140	2.27%	2.01%

6.1. FC1

This flight condition has the lowest dynamic pressure which shows in the nMAE of the SAC controller which is 1.95% above the nominal flight condition. Looking at the response plot in Figure 6.1, the SAC controller has trouble maintaining the pitch up angle with increasing altitude when approaching closer to the stall flight regime. The larger error originates from the divergence from the pitch angle reference between 40s and 70s. The hybrid controller on the other hand is more consistent with the response of the nominal flight condition remaining within margin of error with the nMAE, but producing more noticeable oscillations compared to its nominal case.

6.2. FC3

This flight condition has a higher dynamic pressure compared to the nominal training condition and is expected to produce a lower tracking error compared to FC1 or even FC2. Both the SAC-only and the hybrid controllers achieve a nMAE 0.71% and 0.27% lower than the nominal condition respectively. The larger difference lies with the SAC-only controller, which exhibits considerably improved the pitch tracking compared to the lower dynamic pressure cases. The hybrid controller still outperforms SAC-only by a small 0.3% margin, while also showing increased oscillations.

6.3. FC4

The final tested flight condition has the highest dynamic pressure and is expected to perform equal or better than all the previous flight conditions. Looking at the results from Figure 6.3, the response is visually similar to FC3, but curiously with a slightly higher nMAE for both SAC-only and SAC-IDHP.

For the hybrid controller, this can be attributed to the more aggressive and oscillatory reaction having diminishing returns, with the SAC-only difference lying within margin of error.

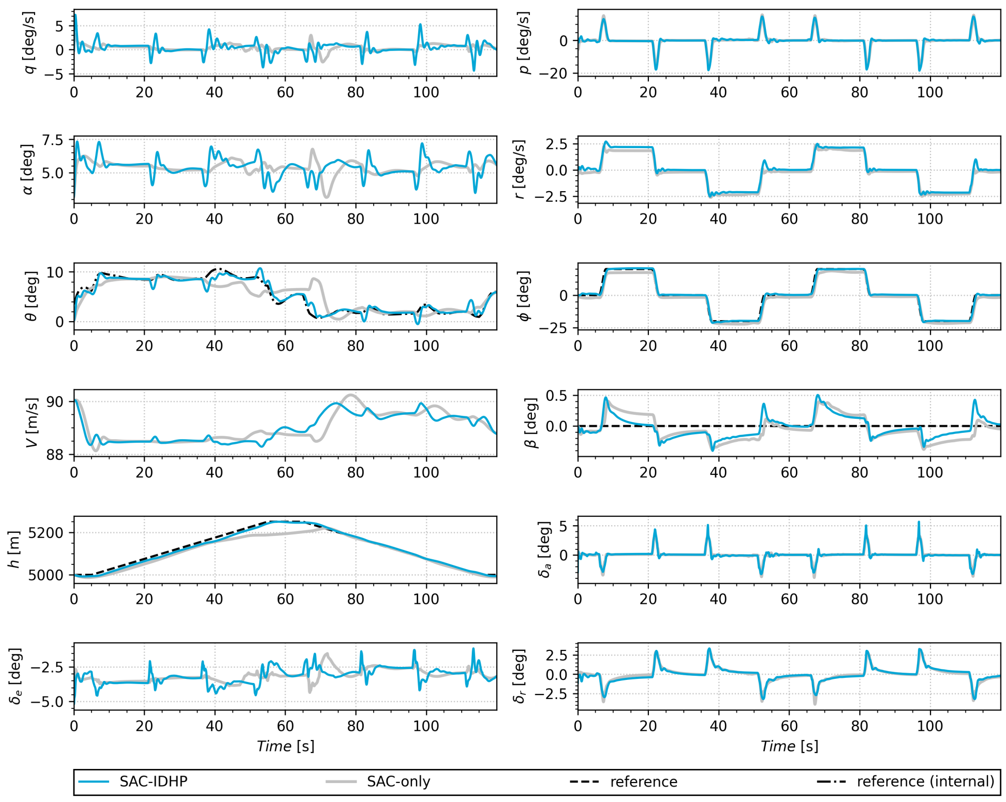


Figure 6.1: Altitude tracking response of SAC-only and SAC-IDHP controllers on FC1.

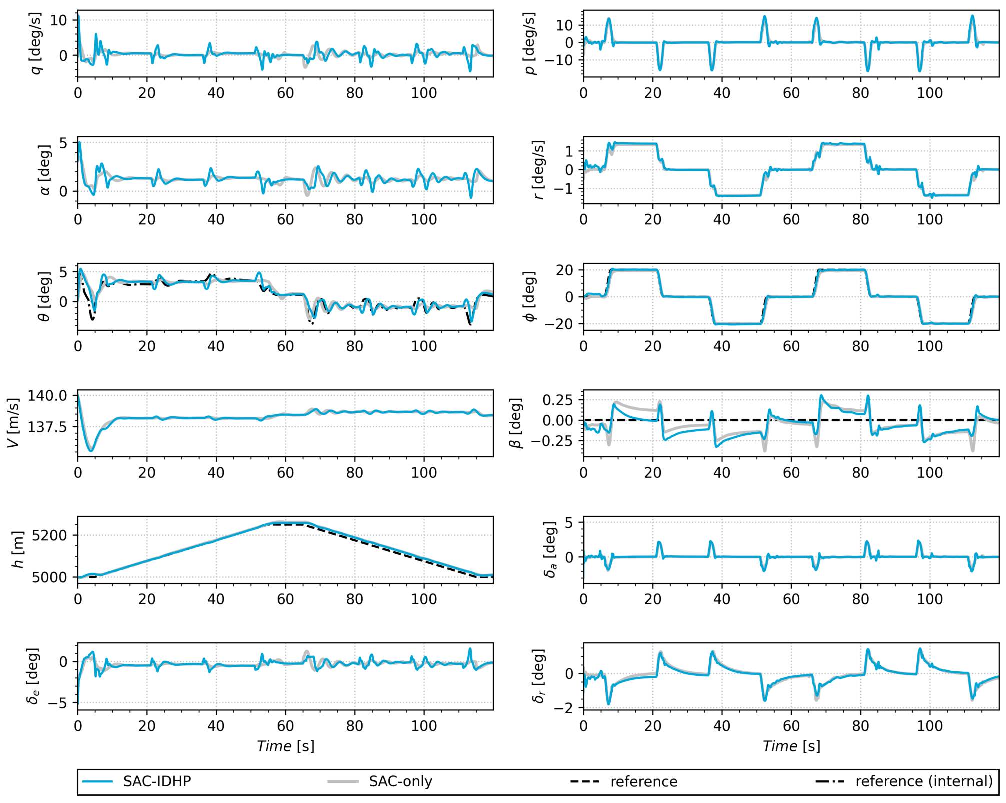


Figure 6.2: Altitude tracking response of SAC-only and SAC-IDHP controllers on FC3.

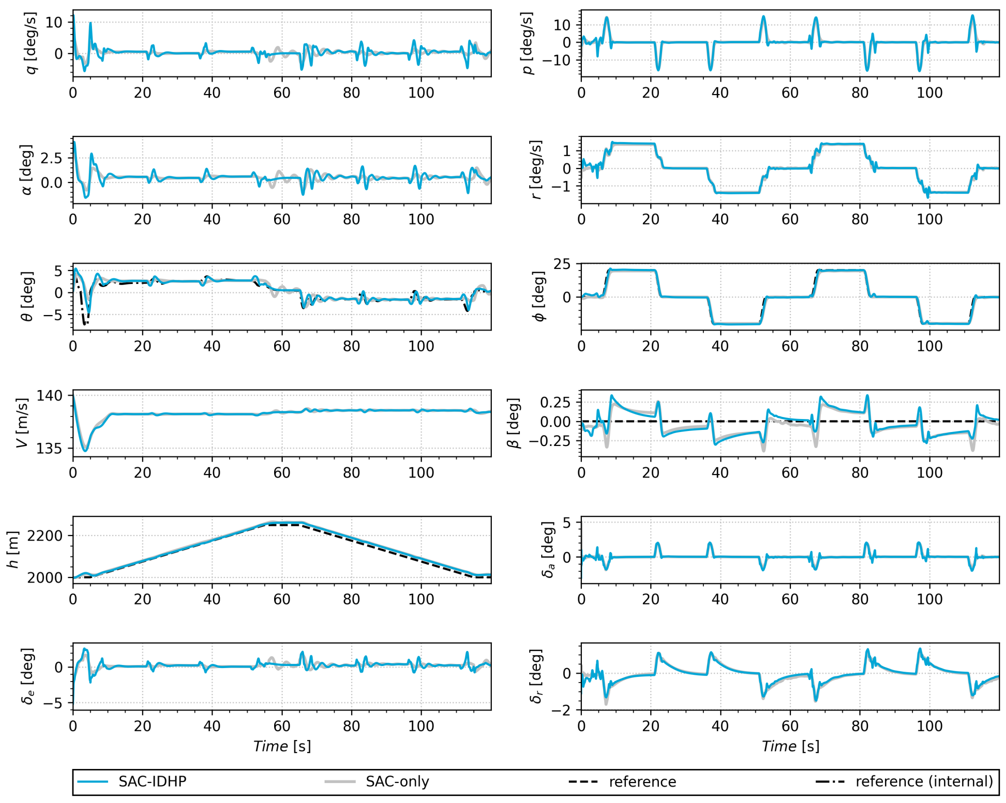


Figure 6.3: Altitude tracking response of SAC-only and SAC-IDHP controllers on FC4.

7 Effect of Covariance Reset

The forgetting factor of the incremental model has been set to $\kappa = 1.0$ in the results presented in Part I. This was done to prevent estimator covariance windup [28]. This appears during periods of no or poor excitation and involves an exponential increase of the covariance matrix when the system is exited again. A disadvantage of using a forgetting factor of 1.0 is the reduced adaptiveness over time. In the fault-tolerance results presented in Part I, the hybrid controller was successful in adapting to changes in system dynamics despite the forgetting factor. Nevertheless, a covariance reset can be used at the time of the system change in order to reinstate the uncertainty of the model parameters.

An implementation of the covariance reset is tested in this section by applying a threshold on the innovation factor ϵ [28]. This acts as a simplified failure detection system. For this experiment, the 70% reduced elevator effectiveness failure case is used. The innovation threshold is set for the pitch rate at $q_{thresh} = 0.001 rad$.

First, the baseline response corresponding with the results from Part I and without the covariance reset is shown again in Figure 7.2. This results in an nMAE of 7.99% for the SAC-only controller and an nMAE of 2.53% for the hybrid controller.

The response with the covariance reset can be seen in Figure 7.1 with an nMAE of 2.90%. The tracking error is a 0.37% increase from the case without covariance reset, showing that the covariance reset does not have a positive effect in this case, while also remaining within margin of error between the two cases. Despite the negligible difference in tracking performance, the effect of the covariance reset can clearly be seen in the weights and parameters plots in Figure 7.3. The threshold $q_{thresh} = 0.001 rad$ is indicated by the dashed line on the innovation term and visualizes the point where the pitch rate error overshoots the threshold and activating the covariance reset. The covariance is seen to increase back to its initial value of $1 \cdot 10^8$ and reduces back over time while the estimator becomes more confident of the parameters. At the point of covariance reset, the parameter matrices are excited with the input matrix correcting more heavily to the reduced elevator effectiveness than without covariance reset as seen in Figure 7.4. Despite the larger correction of the input matrix derivative, this is not reflected into stronger tracking performance. Additionally, the transient period in the F and G matrices after the covariance reset is likely responsible for the higher tracking error. It is expected that the covariance reset will provide a performance increase when longer flight scenarios are tested, where the covariance windup has had more time to manifest.

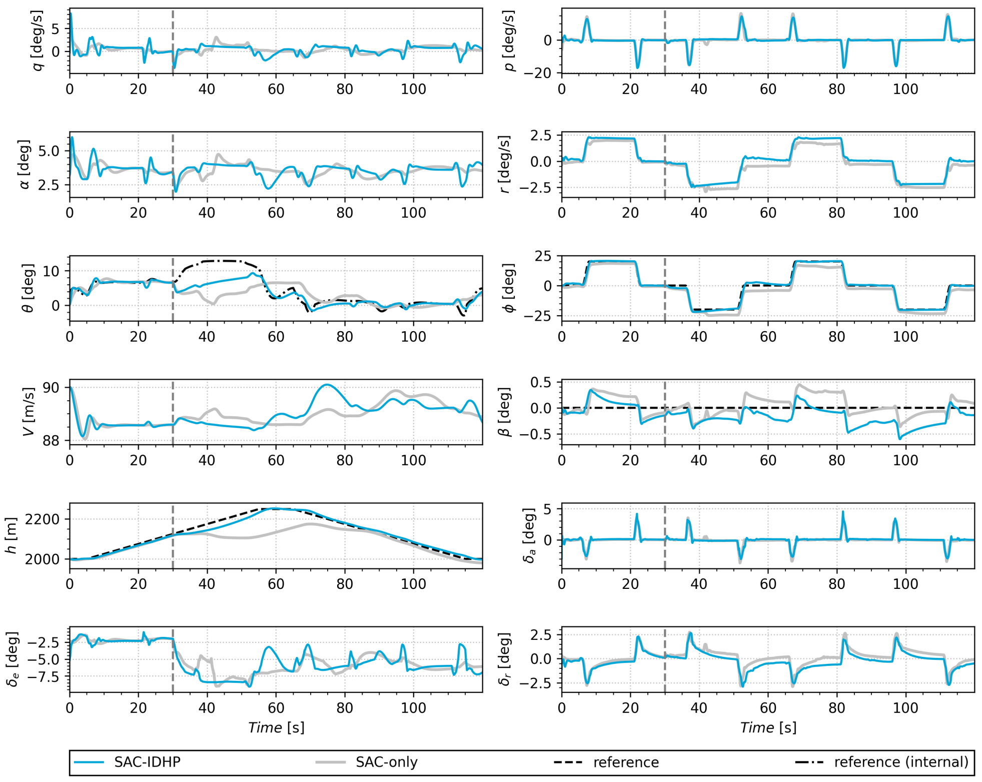


Figure 7.1: SAC-IDHP and SAC-only response on altitude task with 70% reduced elevator effectiveness from t=30s. With covariance reset.

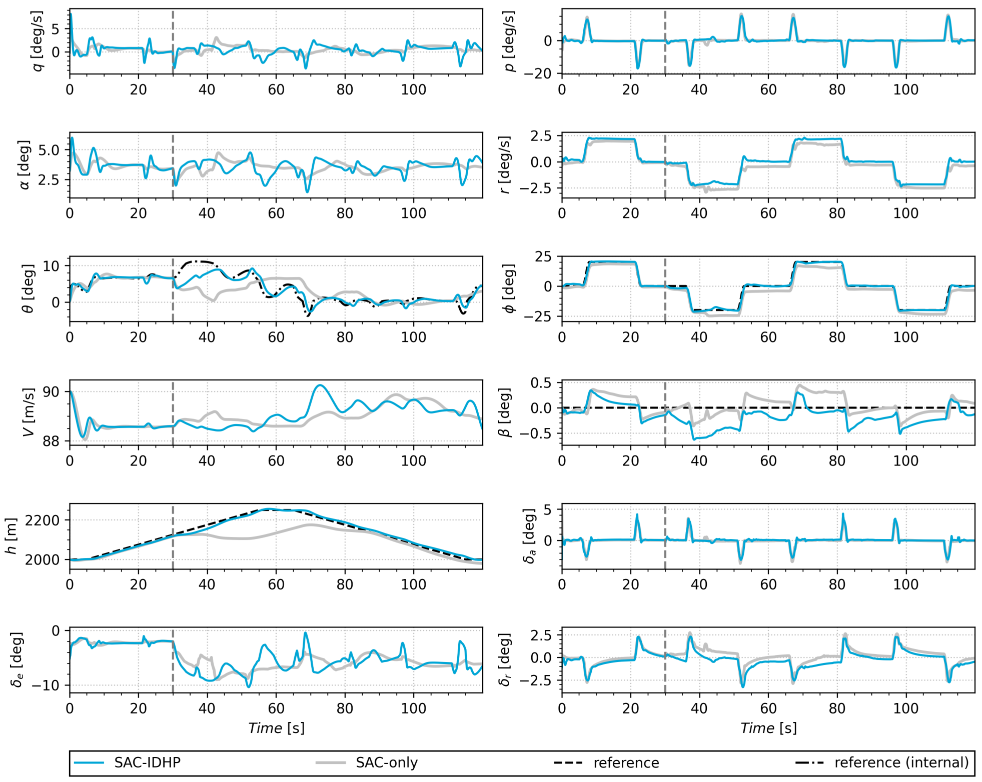


Figure 7.2: SAC-IDHP and SAC-only response on altitude task with 70% reduced elevator effectiveness from t=30s. Without covariance reset.

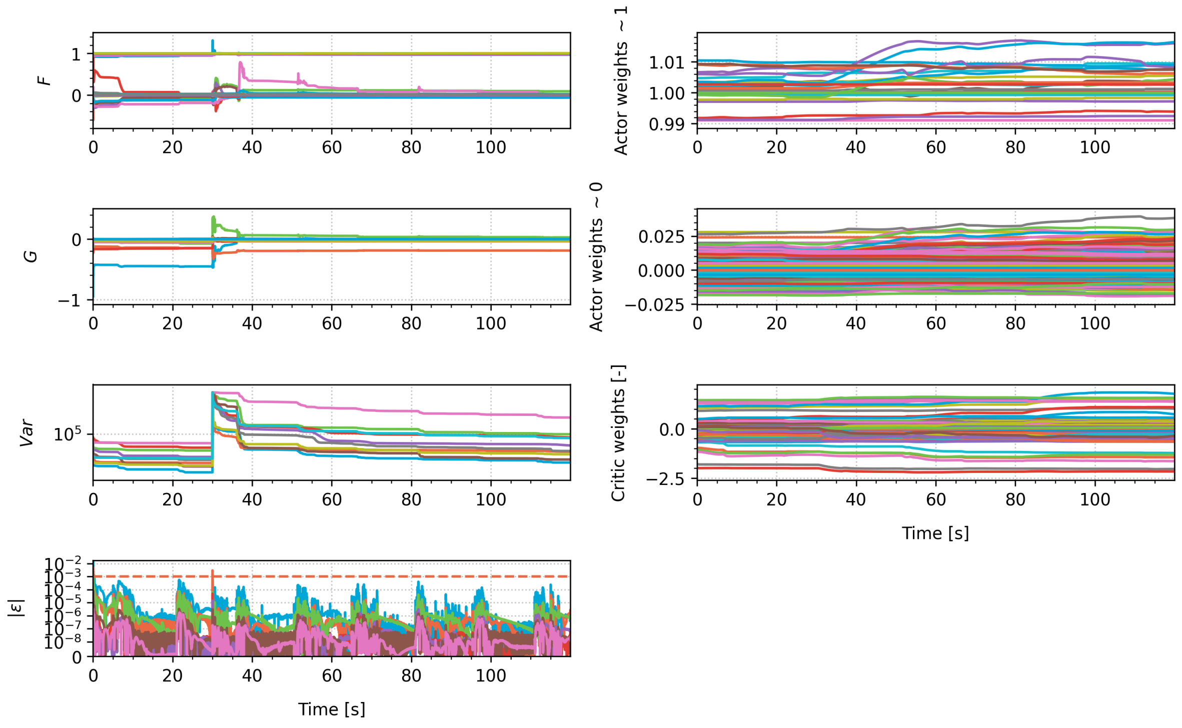


Figure 7.3: SAC-IDHP and SAC-only response on altitude task with 70% reduced elevator effectiveness from t=30s. Actor/critic weights and incremental model parameters with covariance reset.

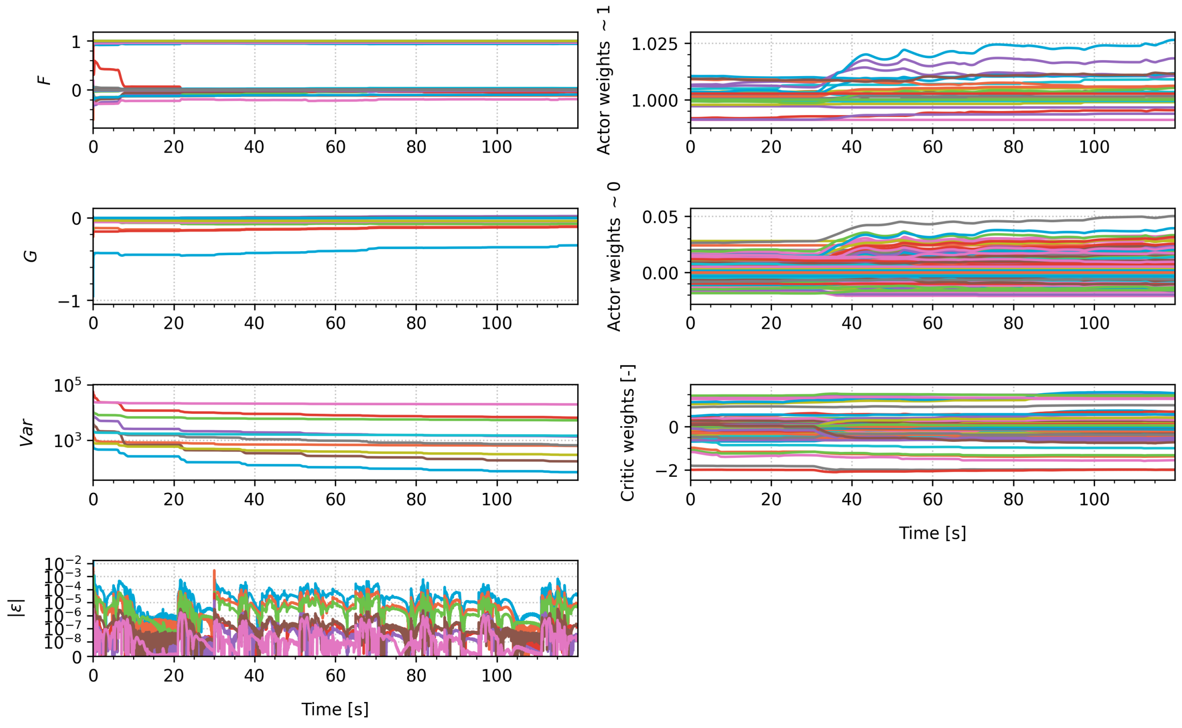


Figure 7.4: SAC-IDHP and SAC-only response on altitude task with 70% reduced elevator effectiveness from t=30s. Actor/critic weights and incremental model parameters without covariance reset.

8 Atmospheric and Control Disturbances

This chapter elaborates on the effect of atmospheric and control disturbances and expands the discussion on the effect of biased sensor noise on the tracking performance presented in Part I.

8.1. Biased Sensor Noise

The additional results in Part I present the effect of biased sensor noise and concludes a stable response for both SAC-only and SAC-IDHP with satisfactory tracking performance, but increased oscillations for the hybrid controller. This section elaborates on the response seen in case of the nominal altitude tracking task. The values used for the noise can be seen in Table 8.1.

Note that it was noticed during initial testing that the incremental model identification of the IDHP framework produces inconsistent results when high frequency oscillation are present in the states. Hence, a low-pass filter with $\omega_0 = 40deg$ is applied to the observation for the IDHP update rule, with an equivalent filter applied to the SAC-only update rules for a fair comparison.

In Figure 8.1 the effect of the biased sensor noise can be seen. The increased oscillations of the hybrid response is noticeable in both longitudinal and lateral states which is in contrast with a case without noise where the hybrid controller mostly showed oscillations in the longitudinal states. To recall the findings from Part I, the nMAE of 2.69% for SAC-only and 2.00% for the hybrid agent are determined when biased sensor noise is present. This corresponds to a respective 0.08% and 0.03% reduction in nMAE compared to the case without noise, attributed to the bias having a positive effect on the error, but also indicating both controllers maintain performance in the presence of sensor noise.

Table 8.1: Cessna Citation PH-LAB sensor noise characteristics [21]

State	**Bias μ**	**Variance σ^2**
$p, q, r\ [rad/s]$	$3.0 \cdot 10^{-5}$	$4.0 \cdot 10^{-7}$
$\theta, \phi\ [rad]$	$4.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-9}$
$\beta\ [rad]$	$1.8 \cdot 10^{-3}$	$7.5 \cdot 10^{-8}$
$h\ [m]$	$8.0 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$

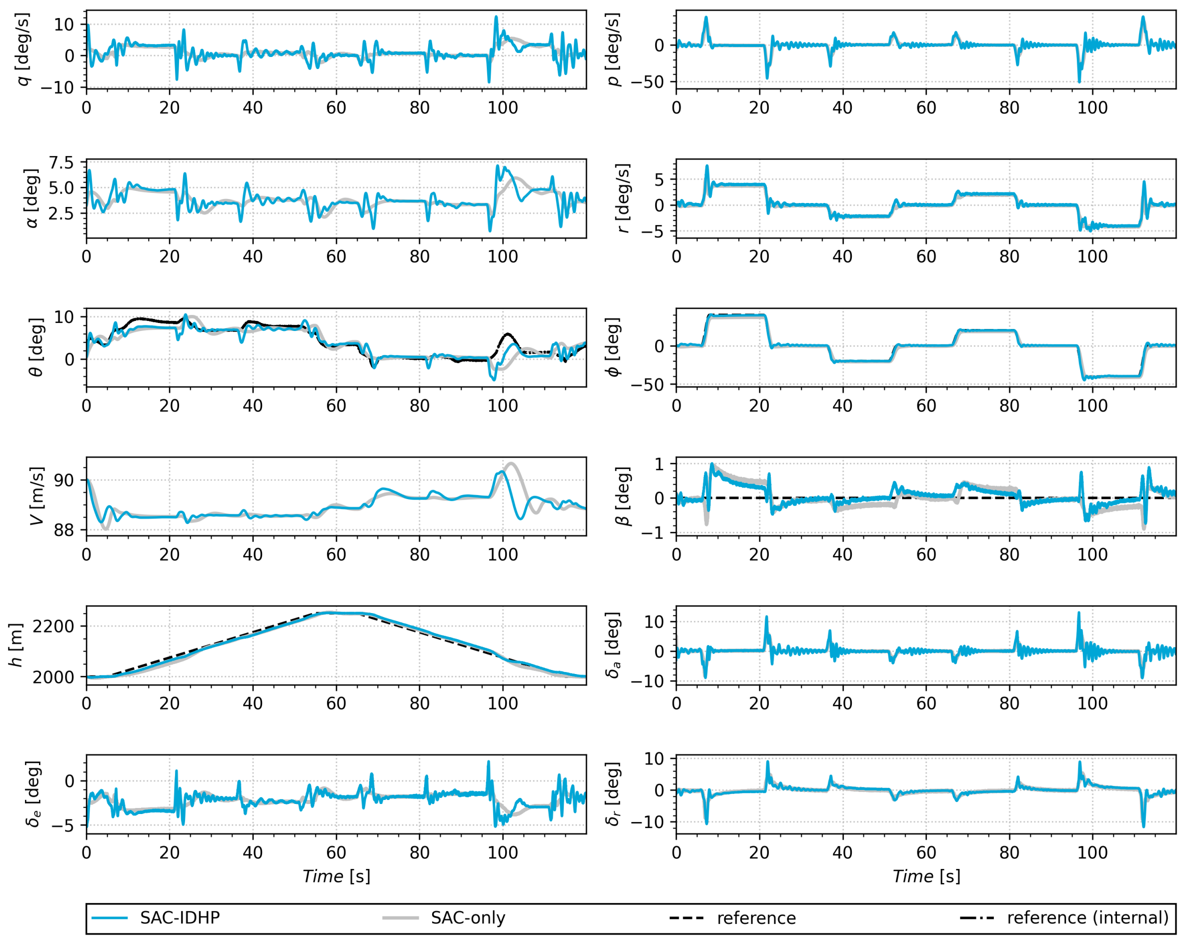


Figure 8.1: Altitude tracking response on system with biased sensor noise. SAC-IDHP and SAC-only compared.

8.2. Atmospheric Disturbance

The atmospheric disturbance is implemented similarly to [11] as discrete vertical gusts of 15ft/s according to MIL-F-8785C [50] and implemented as a $2.5deg$ to $-2.5deg$ disturbance over 3s on the angle of attack. This is applied at $t = 20s$ and $t = 80s$ on the altitude tracking task.

The response can be seen in Figure 8.2 with an nMAE of 2.76% for the SAC-only controller and 2.04% for the hybrid controller. Both controllers are not noticeably affected by the disturbances on the angle of attack and the improvement margin of the hybrid controller is maintained. To increase the number of realistic effects, the same experiment is performed with the addition of the biased sensor noise defined in section 8.1. The response can be seen in Figure 8.3 with an nMAE of 2.61% for SAC-only and 1.87% for the hybrid controller. Like discussed in the previous section, the lower tracking errors compared to the experiment without noise can be attributed to the bias having a positive effect on the error. This shows both controllers maintain their tracking performance and their relative comparison in the presence of noise and atmospheric disturbances compared to the ideal case.

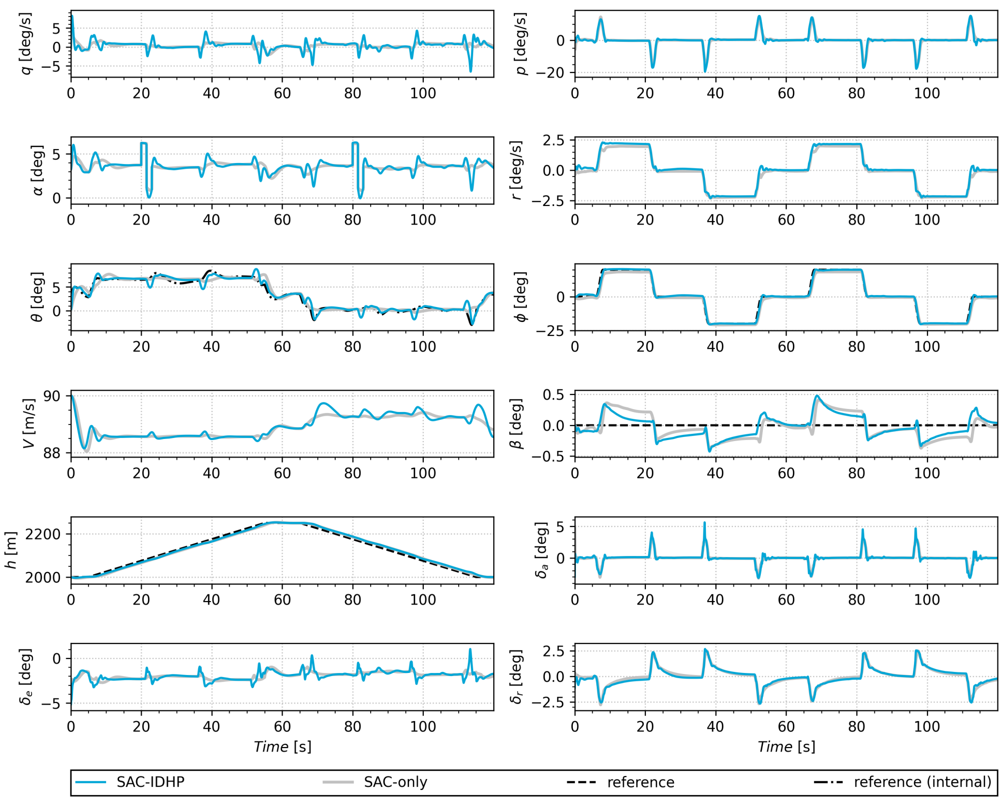


Figure 8.2: SAC-IDHP and SAC-only response on altitude task with atmospheric disturbances at t=20s and t=80s.

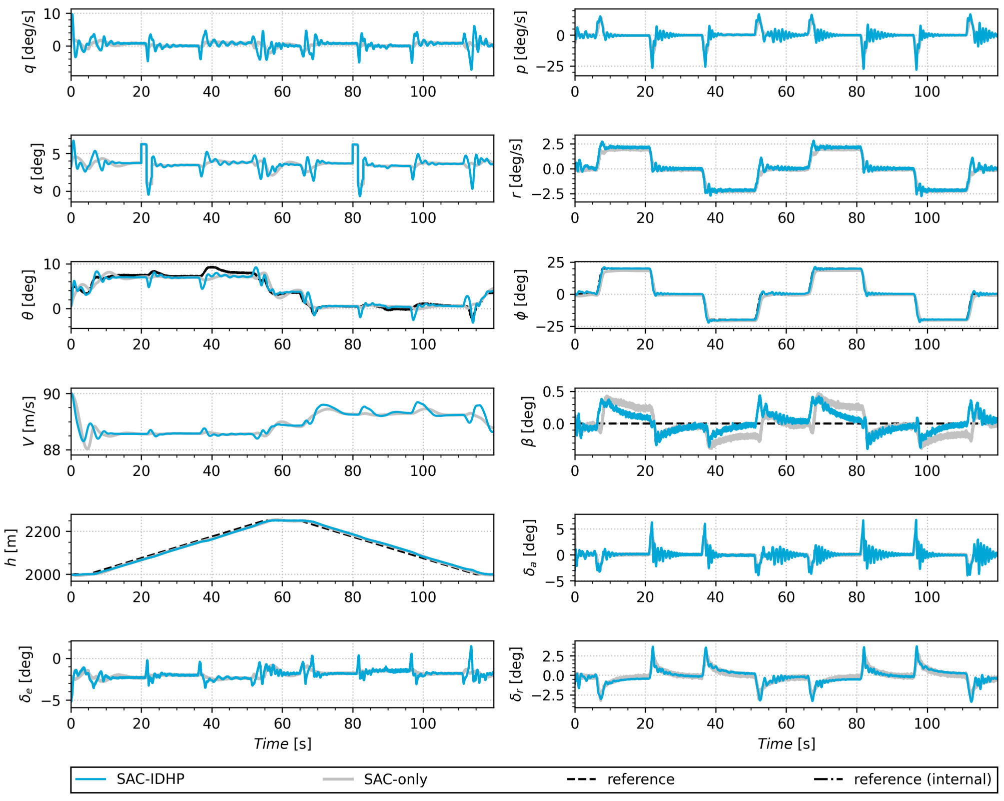


Figure 8.3: SAC-IDHP and SAC-only response on altitude task with atmospheric disturbances at t=20s and t=80s and biased sensor noise.

8.3. Control Disturbance

A control disturbance allows for a more direct analysis of the ability for disturbance rejection. This section presents an experiment using a 3211 disturbance signal added to the elevator, aileron and rudder channels. The reference signals are set at an initial altitude hold, zero roll angle hold and zero sideslip angle to provide a clear visual of the effect of the control disturbance.

Looking at the response in Figure 8.4, both controllers remain stable in the presence of control disturbance. Again, more oscillations are present in the response of the hybrid controller, both in the longitudinal and lateral states, especially the roll rate. The disturbance rejection on the sideslip is most similar between the hybrid and SAC-only, while the hybrid controller does improve on tracking error on the altitude hold signal and little improvement on the roll angle. The addition of biased sensor noise to this experiment case can be seen in Figure 8.5, This case shows a weaker response from the hybrid controller with severe oscillations on all states. The tracking performance is still improved on the altitude hold signal, but overall, the SAC-only controller shows a more robust response to control disturbances and biased sensor noise combined.

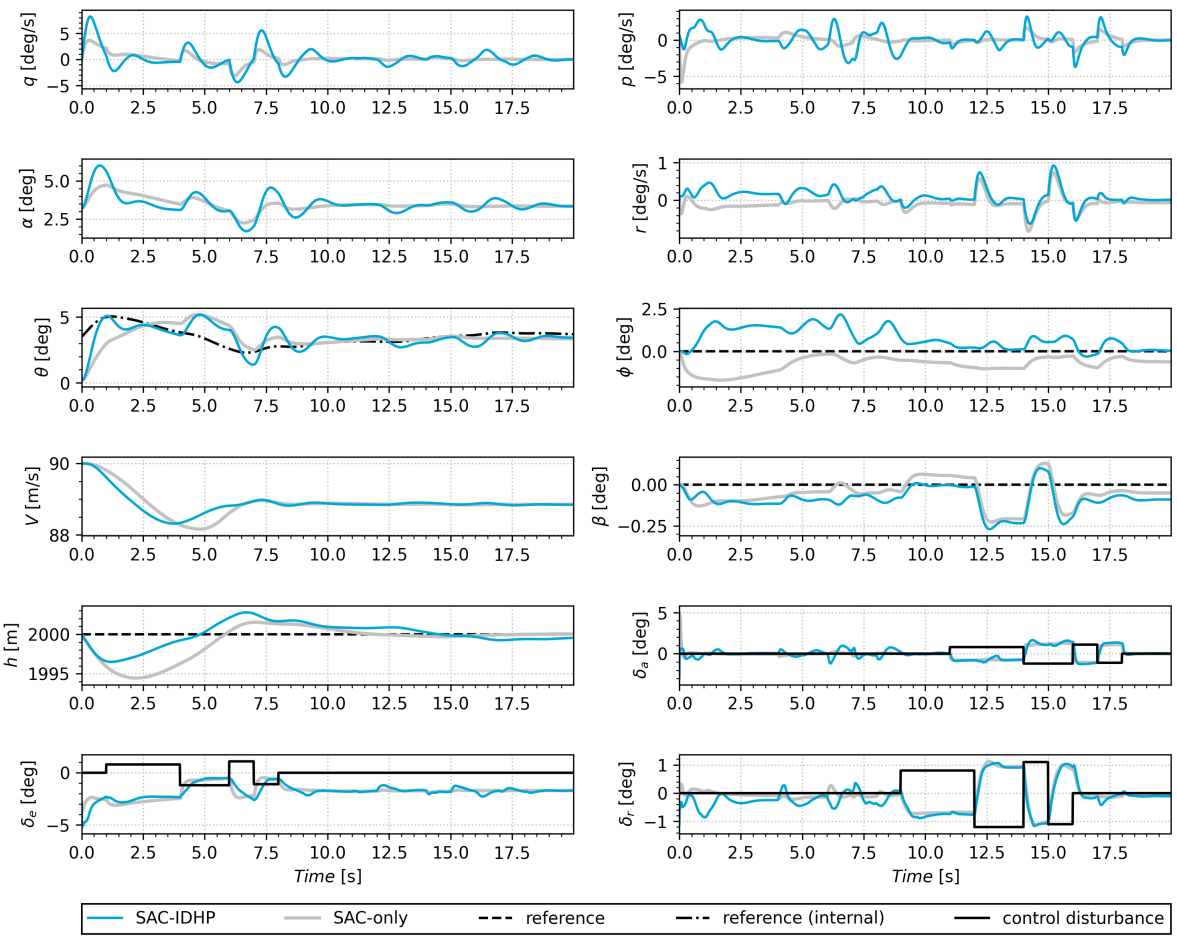


Figure 8.4: SAC-IDHP and SAC-only response on altitude task with control disturbances as 3211 signals on elevator, aileron and rudder.

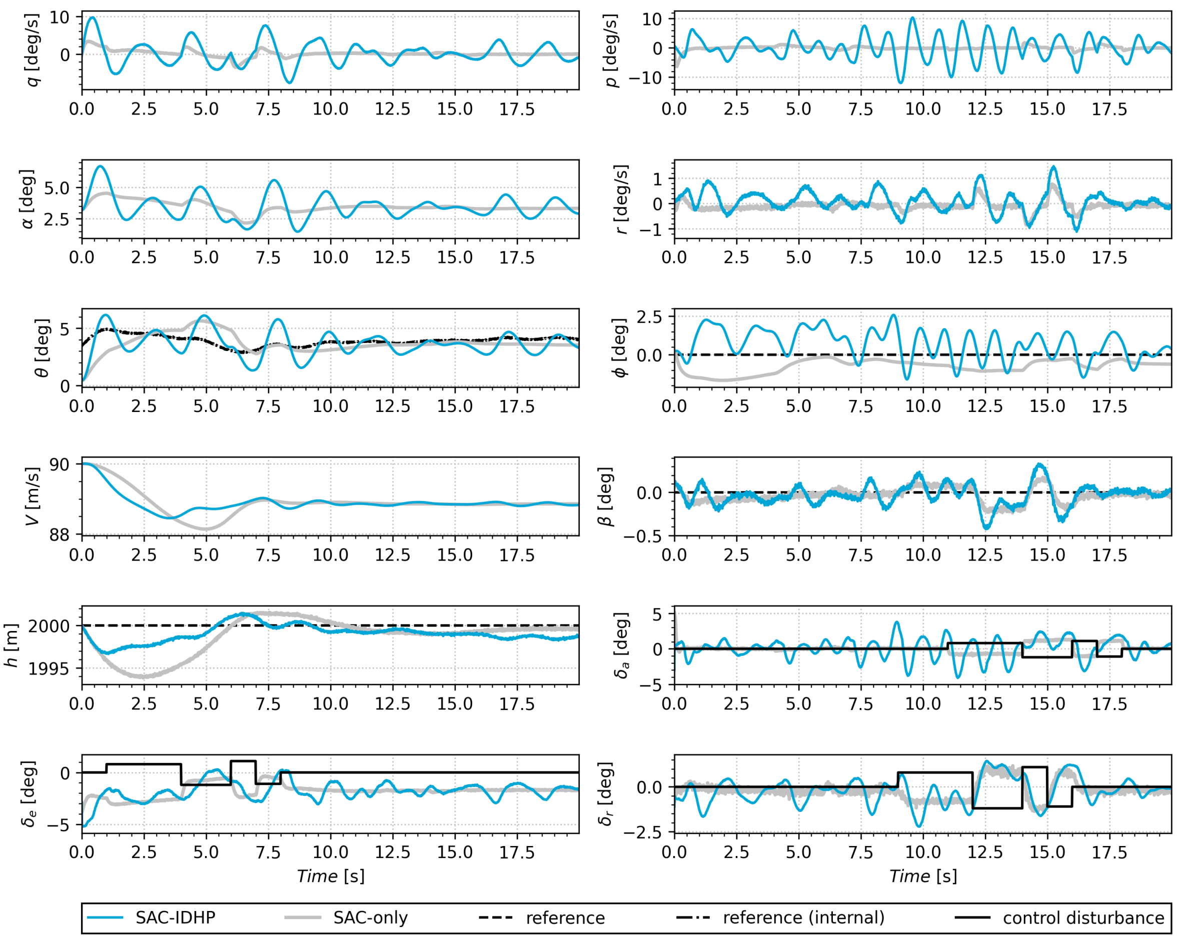


Figure 8.5: SAC-IDHP and SAC-only response on altitude task with control disturbances as 3211 signals on elevator, aileron and rudder and biased sensor noise.

9 Additional Failure Modes

As an addition to the two failure cases presented in Part I, this section goes over three additional longitudinal failure cases further demonstrating and validating the fault-tolerance of the both the SAC-only and hybrid altitude controllers.

9.1. Center-of-Gravity Shift

This failure mode is simulated by a sudden c.g. shift of 0.25m backwards implemented into the Citation simulation model. The response can be seen in Figure 9.1 with the c.g. shift set at t=30s. Both the SAC-only and SAC-IDHP controllers handle the c.g. shift equally well with no obvious visual deviation from the reference signals. Looking at the elevator deflection, the SAC policy is able to compensate by increasing the deflection by approximately $2.5°$. With an nMAE of 2.54% for SAC-only and 1.99% for SAC-IDHP, the hybrid controller still provides marginally closer tracking, most noticeable on the sideslip angle, but with increased oscillations on the longitudinal states.

9.2. Icing Effects

A common effect experienced by aircraft is ice accumulation on the wings. This effect can be simulated by reducing the maximum lift coefficient $C_{L_{max}}$ by 30% and increasing the drag coefficient C_D by 0.06 for mid-range Reynolds numbers [44] [11].

The response in Figure 9.2 shows the reaction of the outer SAC controller by increasing the pitch reference due to the reduced lift and increased drag. The SAC-only controller however struggles to adapt to the increased pitch reference, while the hybrid controller has a smaller pitch tracking error. This is reflected by a nMAE of 4.67% for SAC-only and 1.92% for SAC-IDHP.

9.3. Partial Loss of Horizontal Tail

Another variation on a longitudinal failure mode is a simulated partial loss of the horizontal tail. This affects elevator effectiveness, but also pitch damping. This failure mode is implemented by a 70% reduction of the $C_{L_{\delta_e}}$, $C_{D_{\delta_e}}$, $C_{m_{\delta_e}}$ and C_{m_q} coefficients [11] at t=30s. The response in Figure 9.3 shows again the reaction of the outer controller increasing the pitch reference, but the inner SAC policy reaches a large error to this reference. The hybrid controller is more successful in tracking the pitch reference. Compared to the icing effect, a larger steady state error appears on the roll angle of the SAC policy which is corrected by the hybrid policy. The nMAE of 7.65% for SAC-only and 2.41% for SAC-IDHP reflect the improvement in the altitude tracking.

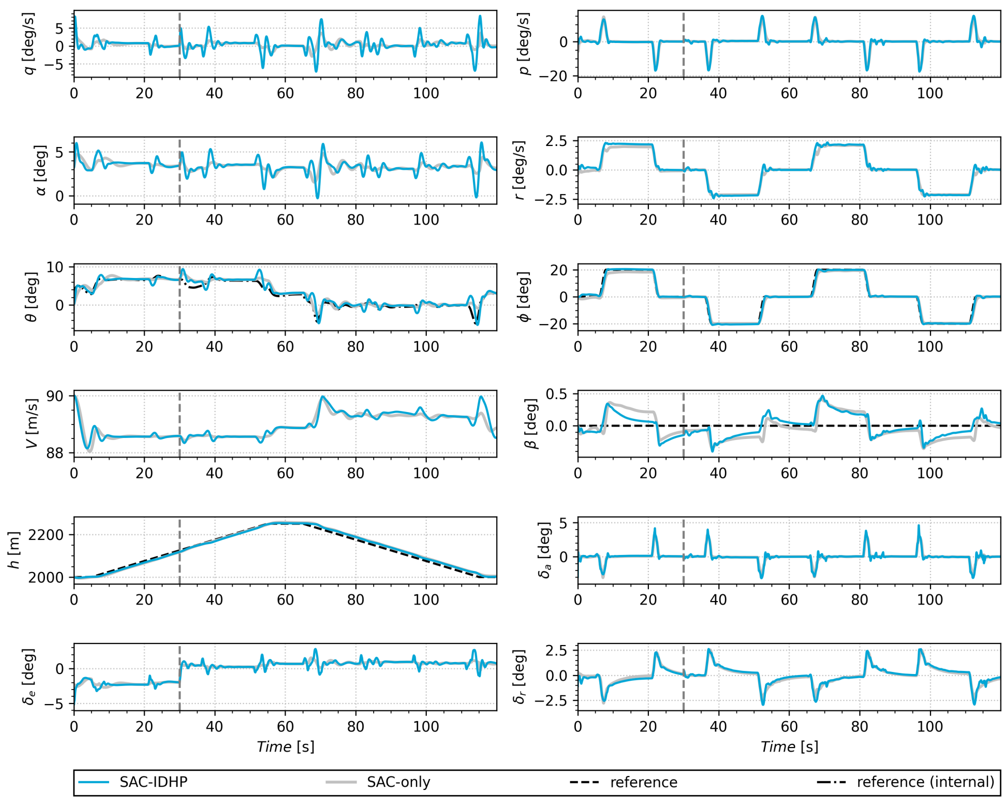


Figure 9.1: Altitude tracking response on system with 0.25m cg-shift from t=30s. SAC-IDHP and SAC-only compared.

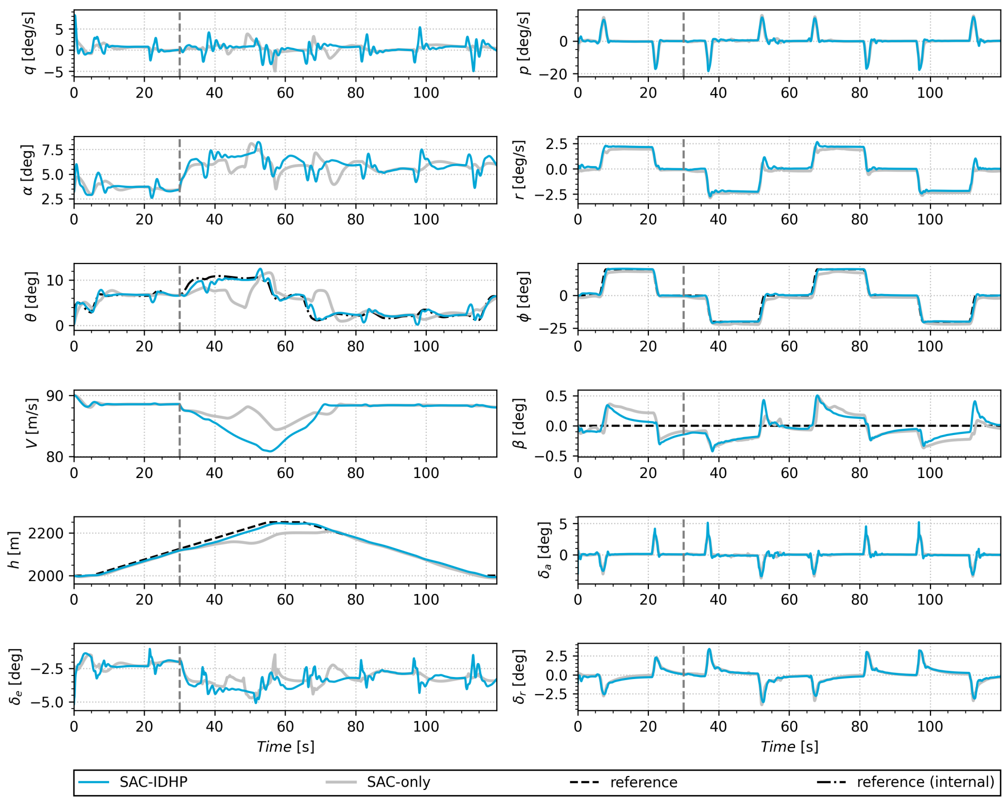


Figure 9.2: Altitude tracking response on system with icing effects from t=30s. SAC-IDHP and SAC-only compared.

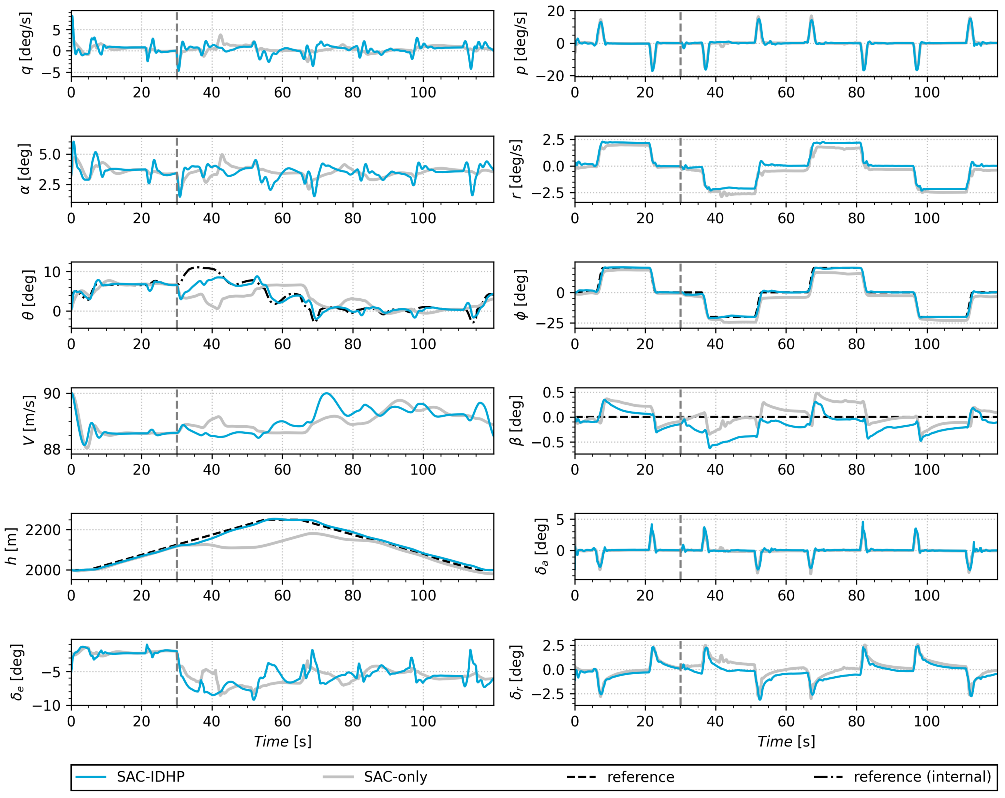


Figure 9.3: Altitude tracking response on system with partial loss of horizontal tail from $t = 30$s. SAC-IDHP and SAC-only compared.

10 Verification & Validation

The verification and validation process is crucial in providing confidence in the implementation of the proposed reinforcement learning controller. Verification and validation is performed for the high fidelity simulation model of the Cessna Citation, and for the RL-controllers developed during this research.

10.1. Verification

The simulation model of the Cessna Citation 500 used for all simulations is build by the Delft University of Technology using the Delft University of Technology Aircraft Simulation Model and Analysis Tool (DASMAT) [21]. This model is delivered in Simulink and compiled to a python executable for use on this research. The compilation to python is verified by performing a 3211 input analysis using the Simulink run-time and the python run-time and comparing the response. In Figure 10.1 the response and input signals on the elevator and ailerons can be seen. As seen from the states, the responses are visually identical, which is supported by a RMSE of 0.31% normalized over the maximum range per state and averaged over all states.

The SAC controller was developed using [11] and [23] as reference. In order to verify a correct implementation of the SAC algorithm, a positive learning curve is observed in Part I for the altitude and attitude agents. This indicates the RL objective is being solved. The tracking performance is able to be compared to [11] due to the use of the same environment model, same nMAE tracking error metric and similar altitude tasks. It is verified that the nMAE is within margin of error on nominal tasks. with an nMAE of for SAC 2.77% on the nominal altitude task with 40° turns which compares within margin of error to an nMAE of 2.64% obtained in the reference implementation. On longitudinal failure cases and the lowest dynamic pressure initial flight condition FC1, the tracking performance is slightly worse with noticeable larger tracking errors on the altitude. Note that in this comparison, the SAC controller developed in this research uses direct control with the addition of temporal and spacial regularization, as opposed to a rate control approach. A comprehensive hyperparameter search and training strategy revision is expected to bring the robust tracking performance of the SAC controller on the same level.

The IDHP algorithm used in the online learning of the hybrid policy is developed with [28] and [34] as reference. The correct implementation of the IDHP algorithm is verified during the preliminary analysis in section 4.5 by obtaining similar tracking performance to the reference material on a short-period model of the PH-LAB aircraft. The eventual hybrid policy employed in the attitude controller is verified by observing gradual changes of the actor/critic weights accompanied by a decreasing tracking error compared to the baseline policy, as demonstrated in Part I.

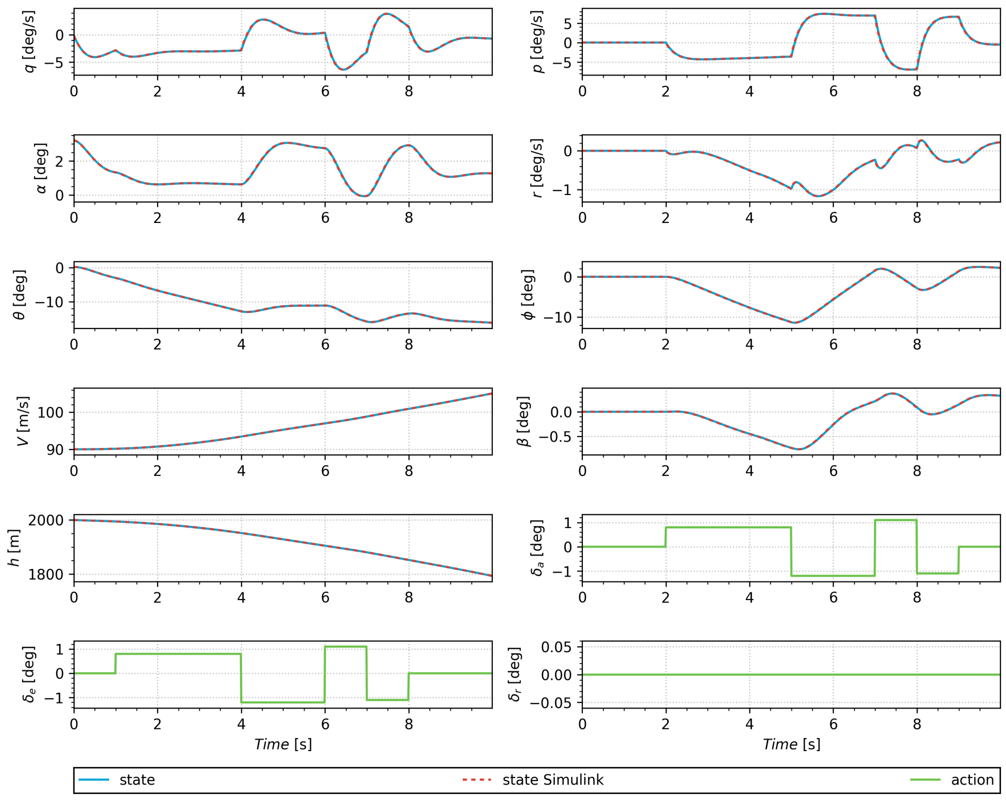


Figure 10.1: Response to 3211 input comparing the DASMAT Simulink model with the executable used in the python simulations.

10.2. Validation

The DASMAT simulation model has been validated by [83] where flight data is compared to the simulation model. A relative RMSE of 8.38% and 12.65% for longitudinal force and moment coefficients and 7.34% and 8.58% for the lateral force and moment coefficients validate the representational power of this model. This assumes to remain inside the normal pre-stall envelope in which all the simulations in this research are conducted.

The tracking performance of the SAC and SAC-IDHP controllers can be compared to the expected standards by requiring sufficiently low nMAE while covering multiple scenarios, which are presented in Part I and Part III. Referring to Table 6.1, the nMAE for several initial flight conditions remains under 5% with only the least favourable FC1 producing an nMAE above 3%. Additionally, tests performed including more realistic phenomenon such as biased sensor noise and atmospheric disturbances show a stable response for any case presented in Part III and tracking error remains small with both controllers for all non-failure flight conditions. The hybrid agent also remains below 3% nMAE for all tested failure cases.

Despite a number of realistic effects being implemented, some limitations remain and part of the flight envelope remains unexplored in this research. Assumptions made include a clean aircraft configuration at all times, no actuator transport delay, no sensor delay, simplified actuator models using saturated first-order low-pass filters, fully observable environment, simplified atmospheric disturbance using direct angle of attack steps and a constant simulation frequency of 100Hz.

A comprehensive exploration of the flight envelope is still recommended, including higher fidelity sensor and actuator models. Also, the sensitivity of the IDHP updates on the hybrid policy to produce oscillations is to be further investigated to not compromise the validity of the hybrid RL-framework.

IV

Closure

11 Conclusion

Fault-tolerant control plays a vital role in automation, especially in the aerospace industry. Both the safety and economics of autonomous control can benefit from developing novel adaptive and robust controllers. This research proposes a RL-based framework for controlling CS-25 class aircraft in the context of the Cessna Citation II PH-LAB research aircraft.

The preliminary research presented in Part II aims to answer the first research question.

Research Question

RQ1 What RL framework is best suited for implementation on the Cessna Citation II?

RQ1.1 What is the current state-of-the-art for continuous adaptive and robust flight control?

RQ1.2 What are the main challenges in reducing the simulation reality-gap of current implementations?

RQ1.3 How can the proposed RL framework be implemented for a simple dynamic system?

RQ1.4 How does the proposed RL framework perform for a simple dynamic system?

In the literature study in chapter 3, the general categories of Incremental Approximate Dynamic Programming (iADP) and Deep Reinforcement Learning (DRL) are further investigated, with a focus on the actor-critic methods. These methods enable continuous control in space and time with the help of function approximation methods, more specifically Artificial Neural Networks (ANNs). Both iADP and DRL methods are also model-independent, meaning they require no prior knowledge or an accurate model of the environment, which is a requirement of the research objective. The main differentiating factor between iADP and DRL methods is the training strategy. iADP methods are best suited for online learning and are thus highly adaptive with a high sample efficiency. DRL methods on the other hand realistically have to learn offline because of lower sample efficiency, but have higher generalization power thanks to the use of Deep Neural Networks (DNNs) and can achieve a level of fault tolerance through robustness. From iADP, the framework of Incremental Dual Heuristic Programming (IDHP) has been selected as the best candidate. Utilizing an incremental model Recursive Least-Squares (RLS) estimator, it can achieve fast convergence and fills the spot of adaptive control. From DRL, the framework of Soft Actor-Critic (SAC) is selected as a state-of-the-art method thanks to its strong exploration and generalization, filling the spot of robust control. Furthermore, it is proposed to evaluate the possibility of a hybrid framework that aims to combine the advantages of both. Hence, *RQ1.1* is answered. The literature study also answers *RQ1.2* and identifies the challenges in the simulation reality-gap noting the importance of sensor noise, delays and external disturbances.

To further evaluate the two frameworks, in the preliminary analysis in chapter 4 both IDHP and SAC are implemented and evaluated on a simplified short-period system corresponding with the PH-LAB aircraft. Additionally, a hybrid form is implemented by combining the offline trained SAC policy in the online-learning IDHP framework. This implementation answers *RQ1.3*. The IDHP and SAC controllers were successfully able to track an angle of attack reference signal on the short-period model with low tracking error. The hybrid SAC-IDHP framework demonstrates its feasibility and ability to correct online for more severe failure cases where the SAC-only agent fails on an inverted elevator failure. These results then formulate an answer to *RQ1.4*, completing the answer to **RQ1**.

The second research question is answered during the further development of the hybrid SAC-IDHP controller presented as part of Part I and Part III.

Research Questions

RQ2 How can the proposed RL controller be integrated into the Cessna Citation II?

RQ2.1 At what control level should the RL controller be implemented?

RQ2.2 What are the architectural characteristics of the RL controller to ensure applicability to the control level?

RQ2.3 What are the characteristic system dynamics, sensor dynamics and actuator dynamics that need to be accounted for to ensure applicability to the real system?

In order to demonstrate a full RL-based flight controller, the high level 6-degree-of-freedom altitude-bank-sideslip tracking loop is chosen, which enables demonstration of coordinated turn manoeuvres and presents the possibility to interface with existing navigation algorithms. Additionally, an altitude control loop provides comparison with relevant tracking performance from literature. This answers *RQ2.1*.

The control loop proposed in Part I consists of a cascaded design, separating the altitude and attitude control motivated by their different dynamics. In order to limit the implementation complexity, the SAC-IDHP hybrid policy is applied to the inner attitude controller only as the attitude dynamics are disproportionately affected by the tested failure modes. Using principles from transfer learning, the architecture of the SAC-IDHP hybrid policy is designed as alternating pre-trained SAC layers and online learning identity initialized IDHP layers with the SAC layers frozen during online learning. This controller design answers *RQ2.2*. A decoupled attitude agent is also explored in chapter 5 which demonstrates a lower learning complexity for the incremental system identification, but with limited effect on tracking performance and requires higher computational and implementation complexity.

The flight scenarios tested in this research use the high fidelity Delft University of Technology Aircraft Simulation Model and Analysis Tool (DASMAT) aircraft model, providing validated flight dynamics that are applicable to the real system. A number of simplifications are made when testing sensor and actuator dynamics with the use of biased sensor noise and saturated low pass filters respectively. Measurement and transport delays are outside the scope of this research and recommended to be investigated further. Additionally, external disturbances are implemented using simplified atmospheric angle of attack step disturbances and control disturbances. The choices made here together with the findings from section 3.5 answers *RQ2.3*. This concludes the answers to **RQ2**.

The last research question concerns the performance and stability of the proposed RL controller, which is evaluated as part of Part I and Part III.

Research Questions

RQ3 What is the overall performance and stability of the proposed RL controller on the Cessna Citation II?

RQ3.1 How are the performance and stability metrics defined and what are their requirements to ensure applicability to the real system?

RQ3.2 How does it perform in standard manoeuvres?

RQ3.3 How does it perform on fault tolerance?

RQ3.4 How does it compare to baseline?

RQ3.5 What is the effect of sensor dynamics and external disturbances on performance and stability?

The performance metric used to quantitatively compare tracking performance is the normalized Mean Absolute Error (nMAE) described in Part I. This represents the overall normalized tracking error averaged over all the externally tracked states. A threshold of 5% can be used to ensure satisfactory performance compared to existing controllers. Additionally, a stability metric is used in the form of a temporal loss which tracks the severity of oscillations with an acceptable threshold of 0.01. This answers *RQ3.1*.

The hybrid controller is tested on a successive climb and descend tracking task with $40°$ coordinated turns involving full 6-degree-of-freedom control and coupling effects. Tracking performance is stable with an nMAE of 2.03% on the nominal system. The initial flight condition is varied from the nominal training case, where the tracking performance remains stable with a variance of only 0.02 in nMAE. Tracking performance is maintained when adding biased sensor noise and atmospheric disturbances. Robustness to online random initialization is demonstrated with a success rate of 100% using the stability threshold, given an initial critic weight standard deviation of 0.01. With this, *RQ3.2* is answered.

A total of five different failure modes are tested including 70% reduced elevator effectiveness, 90% reduced aileron effectiveness, 2.5m center-of-gravity shift, icing effects and 2.41% on partial loss of horizontal tail. The SAC-IDHP controller provides strong tracking performance over all the failure cases, with a maximum nMAE of 2.53% on the case of reduced elevator effectiveness. This answers *RQ3.3*.

Compared to the SAC-only baseline, an improvement in tracking performance is demonstrated across all the tested flight scenarios. The nominal altitude tracking task provides a marginal 0.74% improvement over SAC-only due to the competence of the SAC policy alone. A minimum nMAE improvement of 0.55% is observed in the case of a c.g. shift, while a maximum improvement of 5.46% is seen with reduced elevator effectiveness, demonstrating the ability of the hybrid policy in improving tracking performance. Increased oscillations are however noticeable on all failure cases compared to the SAC-only baseline. Most notable, adding a control disturbance with the addition of biased sensor noise increases the sensitivity of the hybrid controller to oscillations.

Compared to IDHP-only baseline, the presence of the pre-trained policy layers provides increased confidence in including the IDHP update rules into a fully coupled-dynamics 6-degree-of-freedom control loop. This makes use of the fast converging IDHP update rules into a deeper neural network with increased generalization power. Additionally, the random policy initialization from IDHP is eliminated by the hybrid policy architecture and the use of identity initialization, resulting in an argument for increased safety. With this, *RQ3.4* and *RQ3.5* are answered, completing **RQ3**.

Finally, the research objective is reviewed.

Research Objective

Contribute to the development of *model-independent*, *adaptive* and *robust* flight control with the purpose of progressing towards enabling flight tests **by** investigating the challenges in reducing the simulation reality-gap of current reinforcement learning implementations and developing a reinforcement learning-based flight controller for the PH-LAB (Cessna Citation II) research aircraft.

This research proposes a new hybrid offline-online model-independent RL framework providing both offline robust and online adaptive flight control.This controller shows to be fault-tolerant inside a complete RL-based 6-degree-of-freedom cascaded controller structure. The challenges in closing the reality-gap are investigated, and the proposed RL controller demonstrates robustness to biased sensor noise, external disturbances and varying initial flight conditions. This indicates a promising approach to RL flight control by using DRL and iADP with principles from transfer learning to have safe offline learning with more adaptive flexibility during online operation. The results of this research ultimately provide insight into the possibilities of increasing safety and autonomy in the aerospace industry.

12 Recommendations

The following recommendations are made based on the insights learned from this research.

- The robust tracking performance on longitudinal failure modes of the SAC agent developed during this research is slightly outperformed by previous attempts [11]. It is recommended to investigate the effect of different reference policies on the performance of the hybrid policy. Either different SAC training strategies or alternative state-of-the-art DRL frameworks such as TD3 and PPO could provide reference policies with more offline training stability. The latter can be less robust due to lower exploration compared to SAC. Since the hybrid policy provides adaptive fault-tolerance, the lower exploration of reference policies from TD3 and PPO could be offset by the online learning process.

- The CAPS regularization is used to suppress heavy oscillations in the offline learning of the SAC policy when using direct control of the action. A direct control approach was taken as opposed to the incremental control approach [11] as during initial testing, the IDHP framework appeared brittle when controlling a complex system using rate control. It is expected that with additional hyperparameter search or regularization on the IDHP update rules, a stable hybrid policy can be developed using incremental control.

- The hyperparameters used for both the offline SAC training and online SAC-IDHP learning are obtained as defaults from literature or by trial and error. A comprehensive hyperparameter search is expected to improve training stability in SAC and IDHP. The increased oscillations in the response of the hybrid policy is expected to be improved by further hyperparameter tuning.

- In the preliminary analysis, an adaptive learning rate is proposed for the hybrid policy. Due to problems with continual learning [28], adaptive learning rates have been used in IDHP frameworks to lower the learning rate when tracking errors are low [36]. Unique to the hybrid policy, an adaptive learning rate can take advantage of the identity initialization by having a zero learning rate when the robust SAC policy provides satisfactory tracking and thus maximally limiting problems with continual learning. Testing the adaptive learning rate on the cascaded controller could help with oscillations experienced on the hybrid policy.

- The realistic effects of sensor delays, actuator transport delays and a detailed Dryden gust model have been considered outside the scope and time constrains of this research. To improve confidence in the RL controller for use on the real system, these effects are recommended to be tested inside a larger flight envelope.

- Analysis on the effect of covariance reset with the use of a 1.0 forgetting factor on the incremental model can be expanded upon. It is expected that longer simulation times expose degradation in adaptiveness and a covariance reset could be necessary for effective online learning. More sophisticated fault detection mechanism might be necessary compared to innovation term thresholds when deploying to a real system.

Bibliography

[1] Loss of Control In-Flight Accident Analysis Report 2019 Edition. *International Air Transport Association*, page 44, 2019.

[2] Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. Meta Reinforcement Learning for Sim-to-real Domain Adaptation. *arXiv:1909.12906 [cs]*, September 2019.

[3] Gary J. Balas. Flight Control Law Design: An Industry Perspective. *European Journal of Control*, 9(2):207–226, January 2003. ISSN 0947-3580. doi: 10.3166/ejc.9.207-226.

[4] Gabriel Moraes Barros and Esther Luna Colombini. Using Soft Actor-Critic for Low-Level UAV Control. *arXiv:2010.02293 [cs]*, October 2020.

[5] Eivind Bøhn, Erlend M. Coates, Signe Moe, and Tor Arne Johansen. Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization. *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533, June 2019. doi: 10.1109/ICUAS.2019.8798254.

[6] Léon Bottou and Olivier Bousquet. The Tradeoffs of Large Scale Learning. In *Optimization for Machine Learning*, volume 20, January 2007.

[7] Justin A. Boyan. Technical Update: Least-Squares Temporal Difference Learning. *Machine Learning*, 49(2):233–246, November 2002. ISSN 1573-0565. doi: 10.1023/A:1017936530646.

[8] Steven J. Bradtke and Andrew G. Barto. Linear Least-Squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, March 1996. ISSN 1573-0565. doi: 10.1007/BF00114723.

[9] 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, July 2020. doi: 10.23919/CCC50068.2020.9188886.

[10] Olja Cokorilo. Urban Air Mobility: Safety Challenges. *Transportation Research Procedia*, 45: 21–29, January 2020. ISSN 2352-1465. doi: 10.1016/j.trpro.2020.02.058.

[11] Killian Dally and Erik-Jan Van Kampen. Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control. In *AIAA SCITECH 2022 Forum*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, December 2021. doi: 10.2514/6.2022-2078.

[12] 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*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, January 2019. doi: 10.2514/6.2019-2339.

[13] Hao Dong, Zihan Ding, and Shanghang Zhang, editors. *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Springer Singapore, Singapore, 2020. ISBN 9789811540943 9789811540950. doi: 10.1007/978-981-15-4095-0.

[14] Yizhang Dong, Zhiwei Shi, Kun Chen, and Zhangyi Yao. Self-learned suppression of roll oscillations based on model-free reinforcement learning. *Aerospace Science and Technology*, 116: 106850, September 2021. ISSN 12709638. doi: 10.1016/j.ast.2021.106850.

[15] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. *arXiv:1604.06778 [cs]*, May 2016.

[16] Dale Enns, Dan Bugajski, Russ Hendrick, and Gunter Stein. Dynamic inversion: An evolving methodology for flight control design. *International Journal of Control*, 59(1):71–91, January 1994. ISSN 0020-7179. doi: 10.1080/00207179408923070.

[17] R. Enns and Jennie Si. Helicopter trimming and tracking control using direct neural dynamic programming. *IEEE Transactions on Neural Networks*, 14(4):929–939, July 2003. ISSN 1941-0093. doi: 10.1109/TNN.2003.813839.

[18] Jay Farrell, Manu Sharma, and Marios Polycarpou. Backstepping-Based Flight Control with Adaptive Function Approximation. *Journal of Guidance, Control, and Dynamics*, 28(6):1089–1102, November 2005. doi: 10.2514/1.13030.

[19] Silvia Ferrari and Robert F. Stengel. Online Adaptive Critic Flight Control. *Journal of Guidance, Control, and Dynamics*, 27(5):777–786, September 2004. doi: 10.2514/1.12597.

[20] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv:1802.09477 [cs, stat]*, October 2018.

[21] 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*, Kissimmee, Florida, January 2018. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-526-5. doi: 10.2514/6.2018-0385.

[22] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, August 2018.

[23] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv:1812.05905 [cs, stat]*, January 2019.

[24] Hado Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[25] Simon S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 2002. ISBN 978-0-13-090126-2.

[26] Lei He, Nabil Aouf, James F. Whidborne, and Bifeng Song. Deep Reinforcement Learning based Local Planner for UAV Obstacle Avoidance using Demonstration Data. *arXiv:2008.02521 [cs]*, August 2020.

[27] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. *arXiv:1709.06560 [cs, stat]*, January 2019.

[28] 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*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, January 2020. doi: 10.2514/6.2020-1844.

[29] Sham Kakade and John Langford. Approximately Optimal Approximate Reinforcement Learning. In *In Proc. 19th International Conference on Machine Learning*, pages 267–274, 2002.

[30] 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*, San Diego, California, January 2019. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-578-4. doi: 10.2514/6.2019-0129.

[31] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement Learning for UAV Attitude Control. *arXiv:1804.04154 [cs]*, April 2018.

[32] 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*, VIRTUAL EVENT, January 2021. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-609-5. doi: 10.2514/6.2021-0883.

[33] Vijay Konda and John Tsitsiklis. Actor-Critic Algorithms. *Society for Industrial and Applied Mathematics*, 42, April 2001.

[34] Dave Kroezen. Online Reinforcement Learning for Flight Control: An Adaptive Critic Design without prior model knowledge. Master's thesis, Delft Unifersity of Technology, 2019.

[35] Stephen H. Lane and Robert F. Stengel. Flight control design using non-linear inverse dynamics. *Automatica*, 24(4):471–483, July 1988. ISSN 0005-1098. doi: 10.1016/0005-1098(88)90092-1.

[36] Jun Lee. Longitudinal Flight Control by Reinforcement Learning: Online Adaptive Critic Design Approach to Altitude Control. Master Thesis, Delft University of Technology, Delft, 2019.

[37] Jun H. Lee and Erik-Jan Van Kampen. Online reinforcement learning for fixed-wing aircraft longitudinal control. In *AIAA Scitech 2021 Forum*, VIRTUAL EVENT, January 2021. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-609-5. doi: 10.2514/6.2021-0392.

[38] Myoung Hoon Lee and Jun Moon. Deep Reinforcement Learning-based UAV Navigation and Control: A Soft Actor-Critic with Hindsight Experience Replay Approach. *arXiv:2106.01016 [cs, eess]*, June 2021.

[39] Bo Li, Zhigang Gan, Daqing Chen, and Dyachenko Sergey Aleksandrovich. UAV Maneuvering Target Tracking in Uncertain Environments Based on Deep Reinforcement Learning and Meta-Learning. *Remote Sensing*, 12(22):3789, January 2020. doi: 10.3390/rs12223789.

[40] Jiajin Li and Baoxiang Wang. Policy Optimization with Second-Order Advantage Information. *arXiv:1805.03586 [cs, stat]*, May 2019.

[41] 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*, 11(6):705–718, February 2018. ISSN 2150-8097. doi: 10.14778/3184470.3184474.

[42] 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. *arXiv:1509.02971 [cs, stat]*, July 2019.

[43] Derong Liu, Shan Xue, Bo Zhao, Biao Luo, and Qinglai Wei. Adaptive Dynamic Programming for Control: A Survey and Recent Advances. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1):142–160, January 2021. ISSN 2168-2232. doi: 10.1109/TSMC.2020.3042876.

[44] Frank T Lynch and Abdollah Khodadoust. Effects of ice accretions on aircraft aerodynamics. *Progress in Aerospace Sciences*, 37(8):669–767, November 2001. ISSN 0376-0421. doi: 10.1016/S0376-0421(01)00018-5.

[45] Hamid Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard Sutton. *Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation.* January 2009.

[46] Christopher Miller. Nonlinear Dynamic Inversion Baseline Control Law: Architecture and Performance Predictions. In *AIAA Guidance, Navigation, and Control Conference*, Portland, Oregon, August 2011. American Institute of Aeronautics and Astronautics. ISBN 978-1-60086-952-5. doi: 10.2514/6.2011-6467.

[47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013.

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236.

[49] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]*, June 2016.

[50] D.J. Moorhouse and R.J. Woodcock. Background Information and User Guide for MIL-F-8785C, Military SpecificationFlying Qualities of Piloted Airplanes. Technical report, Air Force Wright Aeronautical Labs Wright-Patterson AFB OH, 1982.

[51] J. A. Mulder, W. H. J. J. van Staveren, J. C. van der Vaart, E. de Weerdt, C. C. de Visser, A. C. in 't Veld, and E. Mooij. Flight Dynamics: Lecture Notes AE3202. Lecture Notes, Delft University of Technology, 2013.

[52] Fabio Muratore, Michael Gienger, and Jan Peters. Assessing Transferability from Simulation to Reality for Reinforcement Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(4):1172–1183, April 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2019.2952353.

[53] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning. *arXiv:1803.11347 [cs, stat]*, February 2019.

[54] A. Ng. Feature selection, L1 vs. L2 regularization, and rotational invariance. *ICML 2004*, 2004. doi: 10.1145/1015330.1015435.

[55] Zhen Ni, Haibo He, and Jinyu Wen. Adaptive Learning in Tracking Control Based on the Dual Critic Network Design. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6): 913–928, June 2013. ISSN 2162-2388. doi: 10.1109/TNNLS.2013.2247627.

[56] Dirk Ormoneit and Śaunak Sen. Kernel-Based Reinforcement Learning. *Machine Learning*, 49 (2):161–178, November 2002. ISSN 1573-0565. doi: 10.1023/A:1017928328829.

[57] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. ISSN 1558-2191. doi: 10.1109/TKDE.2009.191.

[58] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, May 2018. doi: 10.1109/ICRA.2018.8460528.

[59] Tijmen Pollack, Gertjan Looye, and Frans Van der Linden. Design and flight testing of flight control laws integrating incremental nonlinear dynamic inversion and servo current control. In *AIAA Scitech 2019 Forum*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, January 2019. doi: 10.2514/6.2019-0130.

[60] Warrren B. Powell. Approximate dynamic programming : Solving the curses of dimensionality. John Wiley & Sons, Inc., 2011. doi: 10.1002/9781118029176.

[61] D.V. Prokhorov and D.C. Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5):997–1007, September 1997. ISSN 1941-0093. doi: 10.1109/72.623201.

[62] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, Second Edition : An Introduction*, volume Second edition of *Adaptive Computation and Machine Learning*. Bradford Books, Cambridge, Massachusetts, 2018. ISBN 978-0-262-03924-6.

[63] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*, June 2017.

[64] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. ISSN 1476-4687. doi: 10.1038/323533a0.

[65] G. Rummery and Mahesan Niranjan. On-Line Q-Learning Using Connectionist Systems. *Technical Report CUED/F-INFENG/TR 166*, November 1994.

[66] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *arXiv:1511.05952 [cs]*, February 2016.

[67] Jurgen Schmidhuber. *Evolutionary Principles in Self-Referential Learning. On Learning Now to Learn: The Meta-Meta-Meta...-Hook*. Diploma Thesis, Technische Universitat Munchen, Germany, May 1987.

[68] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *arXiv:1502.05477 [cs]*, 2015.

[69] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017.

[70] Mazen Shehab, Ahmed Zaghloul, and Ayman El-Badawy. Low-Level Control of a Quadrotor using Twin Delayed Deep Deterministic Policy Gradient (TD3). In *2021 18th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pages 1–6, November 2021. doi: 10.1109/CCE53527.2021.9633086.

[71] Jennie Si, Andrew G. Barto, Warren Buckler Powell, and Don Wunsch. *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, 2004. ISBN 978-0-470-54478-5. doi: 10.1109/9780470544785.

[72] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395. PMLR, January 2014.

[73] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, October 2017. ISSN 1476-4687. doi: 10.1038/nature24270.

[74] P. Simplício, M. D. Pavel, E. van Kampen, and Q. P. Chu. An acceleration measurements-based approach for helicopter nonlinear flight control using Incremental Nonlinear Dynamic Inversion. *Control Engineering Practice*, 21(8):1065–1077, August 2013. ISSN 0967-0661. doi: 10.1016/j.conengprac.2013.03.009.

[75] Lars Sonneveldt, E. van Oort, Q. Chu, C. de Visser, J. Mulder, and J. Breeman. Lyapunov-based Fault Tolerant Flight Control Designs for a Modern Fighter Aircraft Model. In *AIAA Guidance, Navigation, and Control Conference*, Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, August 2009. doi: 10.2514/6.2009-6172.

[76] 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, March 2020. ISSN 0952-1976. doi: 10.1016/j.engappai.2019.103425.

[77] R. Sutton, David A. McAllester, Satinder Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *NIPS*, 1999.

[78] S. Thrun and Anton Schwartz. Issues in Using Function Approximation for Reinforcement Learning. *undefined*, 1999.

[79] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997. ISSN 1558-2523. doi: 10.1109/9.580874.

[80] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997. ISSN 1558-2523. doi: 10.1109/9.580874.

[81] 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, October 2019. doi: 10.1109/ICARES.2019.8914343.

[82] Daniel Urieli and Peter Stone. Model-Selection for Non-parametric Function Approximation in Continuous Control Problems: A Case Study in a Smart Energy System. volume 8188, pages 65–80, September 2013. doi: 10.1007/978-3-642-40988-2_5.

[83] M. A. van den Hoek, C. C. de Visser, and D. M. Pool. Identification of a Cessna Citation II Model Based on Flight Test Data. *4th CEAS Specialist Conference on Guidance, Navigation and Control*, 2017.

[84] Erik-Jan van Kampen, Q.p. Chu, and J.a. Mulder. Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, August 2006. doi: 10.2514/6.2006-6429.

[85] G.K. Venayagamoorthy, R.G. Harley, and D.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, May 2002. ISSN 1941-0093. doi: 10.1109/TNN.2002.1000146.

[86] F. Wang, Huaguang Zhang, and Derong Liu. Adaptive Dynamic Programming: An Introduction. *IEEE Computational Intelligence Magazine*, 2009. doi: 10.1109/MCI.2009.932261.

[87] Yufei Wang and Tianwei Ni. Meta-SAC: Auto-tune the Entropy Temperature of Soft Actor-Critic via Metagradient. *arXiv:2007.01932 [cs, stat]*, July 2020.

[88] Zhuang Wang, Hui Li, Zhaoxin Wu, and Haolin Wu. A pretrained proximal policy optimization algorithm with reward shaping for aircraft guidance to a moving destination in three-dimensional continuous space. *International Journal of Advanced Robotic Systems*, 18(1): 1729881421989546, January 2021. ISSN 1729-8806. doi: 10.1177/1729881421989546.

[89] Elwin Weerdt, Erik-Jan Kampen, Daan Gemert, Q. P. Chu, and J. A. Mulder. Adaptive Nonlinear Dynamic Inversion for Spacecraft Attitude Control with Fuel Sloshing. In *AIAA Guidance, Navigation and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics, 2008. doi: 10.2514/6.2008-7162.

[90] Marco Wiering and Martijn van Otterlo, editors. *Reinforcement Learning: State-of-the-Art*. Number 12 in Adaptation, Learning, and Optimization. Springer, Berlin Heidelberg, softcover reprint edition, 2012. ISBN 978-3-642-44685-6.

[91] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696.

[92] Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham Kakade, Igor Mordatch, and Pieter Abbeel. Variance Reduction for Policy Gradient with Action-Dependent Factorized Baselines. *arXiv:1803.07246 [cs, stat]*, March 2018.

[93] Yuhuai Wu, Elman Mansimov, Shun Liao, Alec Radford, and John Schulman. OpenAI Baselines: ACKTR & A2C. https://openai.com/blog/baselines-acktr-a2c/, August 2017.

[94] Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. Policy Finetuning: Bridging Sample-Efficient Offline and Online Reinforcement Learning. *arXiv:2106.04895 [cs, stat]*, June 2021.

[95] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I. Jordan. How Does Learning Rate Decay Help Modern Neural Networks? *arXiv:1908.01878 [cs, stat]*, September 2019.

[96] Ye Zhou, Erik-Jan Van Kampen, and Q. Chu. Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control. In *IMAV 2016*. Delft University of Technology, October 2016.

[97] 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, February 2017. doi: 10.2514/1.G001762.

[98] 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, April 2018. ISSN 0967-0661. doi: 10.1016/j.conengprac.2017.12.011.

[99] Ye Zhou, Erik-Jan van Kampen, and QiPing Chu. Incremental Approximate Dynamic Programming for Nonlinear Adaptive Tracking Control with Partial Observability. *Journal of Guidance, Control, and Dynamics*, 41(12):2554–2567, 2018. doi: 10.2514/1.G003472.

[100] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, 109(1): 43–76, January 2021. ISSN 1558-2256. doi: 10.1109/JPROC.2020.3004555.