Online Reinforcement Learning for Flight Control

An Adaptive Critic Design without prior model knowledge

D. Kroezen

TU Delft

Online Reinforcement Learning for Flight Control

An Adaptive Critic Design without prior model knowledge

by

D. Kroezen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday April 10, 2019 at 13:30.

Student number:	4097890	
Project duration:	April, 2018 – April, 2019	
Thesis committee:	Dr. G.C.H.E. de Croon,	TU Delft, chair
	Dr. ir. E. van Kampen,	TU Delft, daily supervisor
	Dr. Mihaela Mitici	TU Delft
	Dr. Wei Pan	TU Delft

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

TU Delft

Preface

Since being introduced to machine learning and its capabilities, I have always wondered why this method has not been put to use for actual control problems, instead focusing mostly on gaming problems. This curiosity resulted in an internship at the German Aerospace Center (DLR) where I implemented a simplified longitudinal flight controller using Deep Reinforcement Learning. In my thesis research I further advanced this field and show the potential of Reinforcement Learning for control purposes on a high fidelity dynamic simulation with a focus on online learning without prior model knowledge.

In addition to advancing the field of reinforcement learning for flight control, I want to show the bias in how current machine learning techniques are regarded. This especially applies to the computational intensity, reliance on large amounts of data and how this influences the ability to have an online and real-time controller design. Reinforcement Learning can be unforgiving and hard to interpret. To help others pursuing the same path of using Reinforcement Learning for online and continuous control, I will make the code used in this research publicly available on my GitHub repository[1].

I would like to thank Dr. ir. Erik-Jan van Kampen for his role in my thesis research as my daily supervisor. I greatly appreciate the meetings, support and guidance you provided throughout my project. In truth, this thesis research was not possible without the unconditional support and love of my family and all my friends. In particular I would like to thank my parents, Tanja and Marinus, who helped in all ways possible to pursue my dreams, finish my university career and never stopped believing in me. Furthermore, I would like to thank some of my closest friends, Remie, Raya and Michelle, for offering a listening ear, all the amazing and fun moments and being there when I need you most. This list of thanks is also not complete without mentioning Cherize, who has taught me the importance of love and made me aware of who I am and want to be.

D. Kroezen
Delft, March 2019

[1] `https://github.com/dave992/msc-thesis`

Contents

Nomenclature

$(a, b]$	the real interval between a and b including b but not including a
$\approx$	approximately equal
$\doteq$	equality relationship that is true by definition
$\leftarrow$	assignment
$\mathbb{E}[X]$	expectation of a random variable X
N_{trunc}	truncated Gaussian distribution
$\mathrm{P}(X = x)$	probability that a random variable X takes on the value x
$\mathcal{A}$	set of all actions
$\mathcal{R}$	set of all rewards
$\mathcal{S}$	set of all states
Δt	sample time
T	simulation duration
t	time
$\mathbf{a}$	action
$\mathbf{s}, \mathbf{s}', \mathbf{x}, \mathbf{x}'$	states
g	return
r	reward
$p(\mathbf{s}'\|\mathbf{s}, \mathbf{a})$	probability of transition to state $\mathbf{s}'$, from state $\mathbf{s}$ and action $\mathbf{a}$
$p(\mathbf{s}', r\|\mathbf{s}, \mathbf{a})$	probability of transition to state $\mathbf{s}'$ with reward r, from state $\mathbf{s}$ and action $\mathbf{a}$
$r(\mathbf{s}, \mathbf{a})$	expected immediate reward on transition from $\mathbf{s}$ under action $\mathbf{a}$
π	policy (control law)
$\pi(\mathbf{a} \mid \mathbf{s})$	probability of taking action $\mathbf{a}$ in state $\mathbf{s}$ under *stochastic* policy π
$\pi(\mathbf{s})$	deterministic policy as function of state vector
$\pi(\mathbf{s}, \mathbf{w}_c)$	parametric approximation of the policy

$\lambda(\mathbf{s})$	state-value partial derivative w.r.t. state vector
$\lambda(\mathbf{s}, \mathbf{w}_c)$	parametric approximation of state-value partial derivative w.r.t. state vector
$A_\pi(\mathbf{s}, \mathbf{a})$	advantage of taking action $\mathbf{a}$ in state $\mathbf{s}$ under policy π
$Q(\mathbf{s}, \mathbf{a})$	estimate of action-value function
$q^*(\mathbf{s}, \mathbf{a})$	value of taking action $\mathbf{a}$ in state $\mathbf{s}$ under the optimal policy
$Q_\pi(\mathbf{s}, \mathbf{a})$	value of taking action $\mathbf{a}$ in state $\mathbf{s}$ under policy π
$V(\mathbf{s})$	state-value as function of state vector
$V(\mathbf{s})$	estimate of state-value function
$v^*(\mathbf{s})$	value of state $\mathbf{s}$ under the optimal policy
$V_\pi(\mathbf{s})$	value of state $\mathbf{s}$ under policy π
$\boldsymbol{\Theta}$	parameter matrix
$\mathbf{A}$	state matrix for continuous system
$\mathbf{B}$	input matrix for continuous system
$\mathbf{F}$	state matrix for discrete-time system
$\mathbf{G}$	input matrix for discrete-time system
$\mathbf{P}$	State selection matrix
$\mathbf{Q}$	State weighting matrix
$\mathbf{w}, \mathbf{w}'$	parameter vector
$\mathbf{w}_a$	parameter vector actor
$\mathbf{w}_c$	parameter vector critic
η	learning rate
η_a	learning rate actor
η_c	learning rate critic
γ	discount factor
κ	forgetting factor
τ	target critic mixing factor
α	angle of attack
β	sideslip angle
δ_a	aileron deflection

δ_e	elevator deflection
δ_r	rudder deflection
ϕ	roll angle
θ	pitch angle
h_e	altitude
p	roll rate
q	pitch rate
r	yaw rate
V_{TAS}	true airspeed

List of Figures

1 Introduction

1.1. Background

The domains of automatic and autonomous control are rapidly advancing due to the development of new applications, such as self driving cars [54] and unmanned aerial vehicles [10], as well as the increasing complexity of current systems in search of better performance. Self driving cars face some major challenges due to the mixture of autonomous/automatic and manual controlled traffic and is being researched by several large artificial intelligence and automotive companies such as Waymo and Daimler [13]. Other challenges include having to work with partial observations and uncertain environments. Nevertheless, Waymo has as of July 2018 already driven 12.8 million kilometers with its fleet of autonomous cars [54]. To put this into perspective, the circumference of the earth is only forty-thousand kilometers. For unmanned aerial vehicles the current challenges are related to the unmanned aspect and the implications this has for the control system and air traffic control. The flight controllers have to detect and alter the control laws such that the vehicles can safely be controlled or landed even during failures, unanticipated changes to the environment or other faults. Another branch of research looks at the air traffic control aspect of the increased presence of unmanned aerial vehicles in the airspace, their communication and influence on conventional air traffic.

Most automatic control systems of current aerospace vehicles are designed for a specific operation point and as a result do not generalize well for flight and plant conditions outside the design space. They rely on multiple predefined linear flight controllers, which are combined with a scheduler to switch between the linear controllers [6]. The aforementioned challenges and advancements in control theory are therefore especially relevant for the control system design of aerospace systems, as it faces the exact same challenges without the possibility of experimentation and tests on the same scale. With multi-objective control tasks, growing complexity of the system dynamics and the increased number of system and model uncertainties, control systems have to learn to adapt and show a higher level of autonomy.

An adaptive and fault tolerant controller design reduces the impact of uncertainties, lessens the need for highly accurate model during the design phase and diminishes the impact of faults or failures during operation. This area has been extensively researched in [2, 3, 19, 22, 27, 40, 41, 44] and is primarily based on nonlinear control frameworks such as Backstepping [43] and Nonlinear Dynamic Inversion [25]. An alternative method for adaptive, fault tolerant and nonlinear control is Reinforcement Learning, an optimization method where interaction between agent and environment is used to improve toward an optimal policy. The ability to learn from past experiences, where each experience is composed of the state, action and reward, allows the controller to learn a policy and system model online and adapt without prior knowledge on the plant dynamics.

Up until now Reinforcement Learning has not been applied to flight control of a real passenger aircraft. Flight control applications are limited to UAVs [52] and simulated aerospace vehicles such as helicopters, missiles and military aircraft [15, 51, 60]. Furthermore, recent advancements from the subfields of Deep Reinforcement Learning [26, 36, 38] and Approximate Dynamic Programming [60] have not yet been explored for the purpose of flight control of a passenger aircraft. The Delft University of Technology has access to the PH-LAB research aircraft, a Cessna Citation 550 as shown in figure 1.1.

The unique fly-by-wire flight control system allows to flight test experimental controllers. This research will provide the first step in filling this knowledge gap by developing and demonstrating a novel online Reinforcement Learning framework for flight control on the Cessna Citation 550 aircraft model. Future research can use this knowledge to implement this framework on the PH-LAB research aircraft.

Figure 1.1: The PH-LAB research aircraft operated by Delft University of Technology. The Cessna Citation 550 is a CS-25 certified aircraft with a fly-by-wire control system certified for experimental flight controllers.

1.2. Research objective and questions

The design of an adaptive, fault tolerant and online flight controller on the real Cessna Citation 550 aircraft is not possible in the scope of this research. A research objective is therefore formulated to limit and focus the scope to a more realistic and achievable goal within the time span of nine months. This research objective is shown below and details the goal and means to successfully complete this thesis research. The framework developed in this research serves as a proof of concept and is a first step to future flight tests of online adaptive flight control using Reinforcement Learning. For this reason the framework is applied to the full dynamic model of the Cessna Citation 550.

Research Objective: *The objective of this research is to develop an online, adaptive and non-linear continuous Reinforcement Learning based flight controller for fixed wing aircraft by investigating and implementing a novel Reinforcement Learning framework on the Cessna Citation 550 model and by proposing and investigating possible solutions to improve performance and robustness to random initializations of the agent.*

The research objective is translated to a set of research questions. This set consists of a main research question and several supporting (sub-)questions. The four research questions and their sub-questions are formulated such that they collectively answer the main research question, which in turn fulfills the objective of this research. The questions serve the goal of guiding the research process by separating the large problem at hand in several more manageable smaller problems with a clear focus. The set of research questions are presented below.

Research Questions: *How can a novel online and continuous Reinforcement Learning framework be implemented on the Cessna Citation 550 model as a proof of concept for the purpose of adaptive fight control and be further improved to enhance performance and increase robustness to random agent initializations compared to the baseline framework?*

1. *What is the state-of-the-art of Reinforcement Learning for flight control of fixed-wing aircraft?*
2. *What is the proposed baseline framework for adaptive and online Reinforcement Learning flight control on the PH-LAB research aircraft model?*
   (a) *What are the characteristics of the PH-LAB model and is this environment fully observable?*

   (b) *What type of control should the framework provide?*
   (c) *What Reinforcement Learning framework is best suited for the purpose of this research and how does this compare against other promising methods?*
3. *How can the robustness to random agent initialization and tracking performance be improved for this framework?*
4. *How do the proposed methods compare to the baseline framework with respect to adaptability, online learning and tracking performance?*

The first research question provides the research with the needed information on the state-of-the-art and the knowledge to apply this. Using this knowledge the second question ensures the selection of a framework which is compatible with the PH-LAB model and is able to control this environment. This selection should take into account the type of control and provide motivation as in why this selection is made. This framework will be implemented and tested after which proposals are made to improve training time and sample complexity. The training speed captures several attributes of the online training process such as real-time computation time and how fast the framework converges to a stable controller. The sample complexity on the other hand captures the amount of data or experiences needed to train the controller. The final question makes a comparison between the baseline and enhanced framework and their characteristics.

1.3. Report structure

This report is split in two parts. Part I presents the scientific paper which details the developed framework and main results obtained in this research. The framework is analyzed based on the tracking performance and robustness to random initializations of the agent. It provides a standalone conclusion and recommendation on the provided work and serves as the main body of this thesis report. The paper answers research question 3 and 4. In part II the literature survey and preliminary analysis are elaborated on, followed by the recommendations, conclusion and additional results. For readers which are unfamiliar with Reinforcement Learning, the literature survey is a good starting point to get more information on the origin and basics of the Reinforcement Learning in respectively sections 2.1 and 2.2. The rest of chapter 2 focuses on the current state-of-the-art for control purposes in general and frameworks applied for flight control applications. This chapter answers research question 1. In chapter 3 the preliminary analysis is detailed. Using the information from the literature survey, a preliminary and simplified framework based on Heuristic Dynamic Programming is used to test the feasibility of the method for flight control. This chapter answers research question 2. The conclusion on the complete thesis research is given in chapter 4 and is followed by the recommendations for future research in chapter 5. In appendix A additional results outside the main research are presented. This chapter gives more in dept explanation and visualizations of the results and elaborates on insights obtained regarding the failed episodes and their causes.

I

Scientific Paper

Online Reinforcement Learning for flight control of a fixed-wing CS-25 aircraft using an Adaptive Critic Design without prior model knowledge

Dave Kroezen*
Delft University of Technology, Delft, The Netherlands

Online Reinforcement Learning is a possible solution for adaptive nonlinear flight control. In this research an Adaptive Critic Design (ACD) based on Dual Heuristic Dynamic Programming (DHP) is developed and implemented on a simulated Cessna Citation 550 aircraft. Using an online identified system model approximation, the method is independent of prior model knowledge. The agent consists of two Artificial Neural Networks (ANNs) which form the Adaptive Critic Design and is supplemented with a Recursive Least Squares (RLS) online model estimation. The implemented agent is demonstrated to learn a near optimal control policy for different operating points, which is capable of tracking pitch and roll rate while actively minimizing the sideslip angle in a faster than real-time simulation. Providing limited model knowledge is shown to increase the learning, performance and robustness of the controller.

Nomenclature

RL	=	Reinforcement Learning
ACD	=	Actor-Critic Design
RLS	=	Recursive Least Squares
DHP	=	Dual Heuristic Dynamic Programming
$\boldsymbol{s}$, $\boldsymbol{x}$	=	Reinforcement learning state vector and plant state vector
$\boldsymbol{a}$	=	Action vector
r	=	Reward
$\pi(\boldsymbol{s})$	=	Policy
$V(\boldsymbol{s})$, $\lambda(\boldsymbol{s})$	=	State value and state value derivative
$\boldsymbol{P}$, $\boldsymbol{Q}$	=	State selection matrix and tracking error weight matrix
$\boldsymbol{F}$, $\boldsymbol{G}$	=	Discrete state matrix and discrete input matrix
η	=	Learn rate
γ	=	Discount factor
τ	=	Target learn rate
κ	=	RLS forgetting factor
p, q, r	=	Body rotation rates
V_{TAS}, α, β	=	True airspeed and aerodynamic angles
x_e, y_e, h_e	=	Cartesian position relative to the earth

I. Introduction

The domains of automatic and autonomous control are rapidly advancing due to the development of new applications, such as self-driving cars [1] and unmanned aerial vehicles [2], as well as the increasing complexity of the controlled systems in search of better performance. One of the main challenges of these domains is to form adaptive and fault-tolerant control systems, which can alter their response in case of unforeseen changes in the environment or in the dynamic response of the vehicle. This is especially true for the aerospace industry, where in-flight loss of control (LOC-I) is still the largest contributor to fatalities in aircraft accidents [3].

*MSc. Student, Control and Operation, Aerospace Engineering, Delft University of Technology

Most current flight control systems are designed for specific operating points and as a result do not generalize well outside this region. They rely on multiple predefined linear flight controllers, which are combined with a scheduler to switch between the linear controllers [4]. For the design of these controllers high fidelity system models are required. As aerospace vehicles become more complex and nonlinear, the amount of uncertainties grows. This again results in an increase in resources needed to acquire high fidelity system models. Robust control techniques have been developed and applied to enhance the operating range of current flight control systems, reduce the sensitivity to this growing amount of uncertainties and require less accurate system models [5, 6]. Nevertheless, the current push toward an increase in autonomy resulted in a need for truly adaptive and even fault-tolerant control.

Inspired by the learning principles of animals, Reinforcement Learning is a form of Machine Learning which uses interaction and experience to adapt toward an optimal control policy [7]. The optimal control policy is defined to maximize the reward or objective function and is a form of optimal control. Reinforcement Learning has successfully been applied to complex control problems with no prior knowledge on the plant dynamics [8, 9]. However, these methods are often computationally expensive and require large amounts of data to arrive at this near optimal policy. Recent developments [10–12] have shown that Adaptive Critic Designs can function with the same amount of data as traditional feedback controllers, greatly reducing the computational requirements as a consequence. Furthermore, using an internal local model approximation the ACDs can achieve adaptive and model-free control for trajectory planning, regulation and reference tracking control problems. Up until now this method and RL in general has been applied to simulations of a general fixed wing aircraft [13], helicopters [14, 15] and fighter aircraft [16].

The next step to advance the field of adaptive, fault-tolerant and model-free flight control is to investigate the implementation and physical testing of reinforcement learning, more specifically a framework within ACDs named Dual Heuristic Dynamic Programming (DHP) [17]. Delft University of Technology has the unique capability to test experimental flight controllers on the PH-LAB research aircraft. The PH-LAB is a Cessna Citation 550, a CS-25 certified passenger airplane.

This research will integrate the current knowledge on reinforcement learning to a reference tracking flight control problem. The control system or agent is applied to the simulated research aircraft in anticipation of future flight tests based on the results of this research. This research aims to quantify the tracking performance of the framework and the robustness to random initialization of the controller, and investigate the effect of target networks, using limited model information and parameter initialization of the ACD function approximators.

II. Framework Design

The RL agent provides angular rate control using the elevator, ailerons and rudder control surfaces. The decision for this form of control allows for fast learning of the aircraft dynamics as the actuators have the most profound and direct effect on the angular body rates. Furthermore, using more conventional control techniques allows to use this rate controller to provide higher levels of navigational control, while retaining the learning and adaptive characteristics of the Reinforcement Learning agent.

A. Control Diagram

The agent is designed to accept two input vectors, namely the current system state $\boldsymbol{x}_t$ and the current references combined in $\boldsymbol{x}^{ref}$. The two main components of the reference vector are the pitch rate q^{ref} and the roll rate p^{ref}, which is combined with a zero sideslip reference β^{ref} to provide incentive to the agent to minimize sideslip throughout the flight. The values of the pitch rate and roll rate references are provided by conventional PID feedback controllers. An overview of the complete control framework is shown in figure 1.

A high level overview of the content of the agent is shown in figure 2. It consists of three main structures, namely the actor, critic and a system model. The actor is responsible for the current control law, while the critic is used in the optimization process of the actor and guides the control law to the optimal policy. Finally, the plant model supports the optimization process by providing the required model derivatives and state predictions. The information flow used in this update process is indicated with the dotted arrows passing through the actor and critic.

B. Reference tracking in reinforcement learning

Conventional RL problems often consider a stationary problem, where the goal or reference is static or predetermined. For example, in gaming environments like Go the goal is to win [18], while in maze environments the goal is to reach a location [19]. In control theory this translates to a regulation problem or trajectory planning task. In this research a

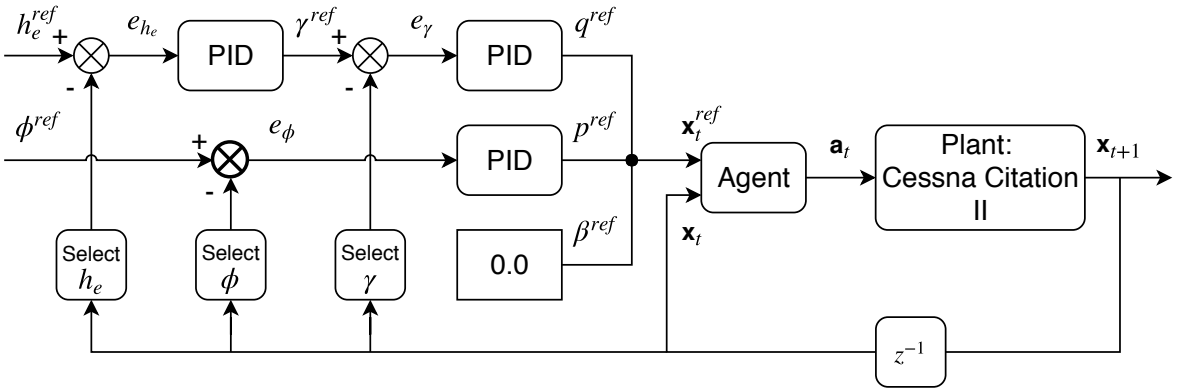


Fig. 1 General layout of the flight controller. The inner control loop consists of the reinforcement learning agent and provides body rate control. The outer loop consists of PID controllers converting the desired altitude and bank angle to body rate references for the inner control loop.

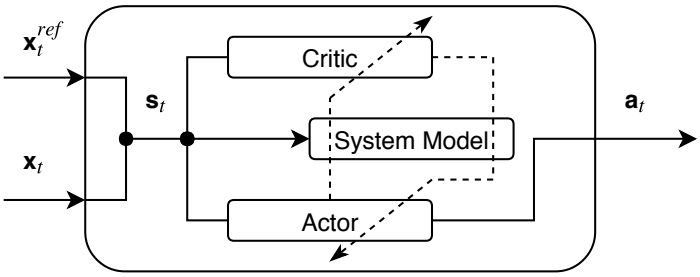


Fig. 2 High level overview of the agent block content and interactions as shown in the control diagram in figure 1.

reference tracking problem is considered, which adds an arbitrary changing reference signal $\boldsymbol{x}^{ref}$ to the state vector. As a result a distinction is made between the system state $\boldsymbol{x}_t$ and the reinforcement learning state $\boldsymbol{s}_t$, which contains additional information on the tracking task. The relation between both state vectors is shown in eq. (1). The actor and critic take the reinforcement learning state as input to receive information on the current system state and goal. This information is then used to calculate the action and update the agent.

$$\boldsymbol{s}_t = \begin{bmatrix} \boldsymbol{x}_t & \boldsymbol{x}_t^{ref} \end{bmatrix}^T \tag{1}$$

The agent update requires the prediction of the successive reinforcement learning state $\hat{\boldsymbol{s}}_{t+1}$ in addition to the current RL state $\boldsymbol{s}_t$, as will be explained in more detail in section IV. Following equation (1) this requires the prediction of the next reference $\hat{\boldsymbol{x}}_{t+1}^{ref}$, which is not available at time t for a tracking problem. Instead, the current reference $\boldsymbol{x}_t^{ref}$ is used, resulting in equation (2) to construct $\hat{\boldsymbol{s}}_{t+1}$. The goal of the agent is therefore to steer toward the current reference $\boldsymbol{x}_t^{ref}$ and maintain this value afterward. Each time step the goal is then adjusted to reflect the current desired state.

$$\hat{\boldsymbol{s}}_{t+1} = \begin{bmatrix} \hat{\boldsymbol{x}}_{t+1} & \boldsymbol{x}_t^{ref} \end{bmatrix}^T \tag{2}$$

III. Design and implementation of the environment

The environment is based on the high fidelity dynamic model of the Cessna Citation 550 aircraft. The configuration of this dynamic model is treated in section III.A. In reinforcement learning, the environment is also responsible for generating the reward. The design and implementation of the reward function is elaborated on in section III.B.

A. Cessna Citation II plant dynamics

The plant dynamics are implemented using the DASMAT tool and the non-linear dynamic coefficients of the Cessna Citation 550 [20]. The implementation is aimed at simulating the dynamics of a real aircraft, more specifically the PH-LAB research aircraft shown in figure 3, and therefore includes engine and actuator dynamics. The thrust levers are controlled by an internal proportional speed controller which regulates the initial airspeed, this functionality is specific to the simulation model and needs to be controlled by the pilots in the PH-LAB aircraft. Furthermore, the implemented yaw-damper is disabled to give the agent full authority over all aerodynamic control surfaces. The simulation does not include the effects of turbulence and is operated at 50 Hz. The obtained Simulink model is converted to C code and compiled for use as a Python library.

Fig. 3 The PH-LAB research aircraft operated by Delft University of Technology. The Cessna Citation 550 is a CS-25 certified aircraft.

The state vector of the Citation model $\boldsymbol{x}_{cit}$ is defined in eq. (3) and contains information on body rates, aerodynamic angles, airspeed, Euler angles and the position relative to the earth surface.

$$\text{Aircraft state: } \boldsymbol{x}^{cit} = \begin{bmatrix} p & q & r & V_{TAS} & \alpha & \beta & \phi & \theta & \psi & h_e & x_e & y_e \end{bmatrix}^T \tag{3}$$

For this research the aerodynamic control surfaces are considered as controllable inputs to the model. The resulting input vector contains the commanded elevator, aileron and rudder deflections, as shown in equation (4).

$$\text{Model input: } \boldsymbol{a} = \begin{bmatrix} \delta_e & \delta_a & \delta_r \end{bmatrix}^T \tag{4}$$

Each simulation step the Citation model outputs a subset of the aircraft state vector, which contains the state information needed by the agent. The returned state elements do not include measurement errors or other sensor dynamics. In eq. (5) the model output $\boldsymbol{y}$ is given. For the purpose of this research, the observed model output $\boldsymbol{y}$ is used as the system state $\boldsymbol{x}$.

$$\text{Model output: } \boldsymbol{y} = \begin{bmatrix} p & q & r & V_{TAS} & \alpha & \beta & \phi & \theta & h_e \end{bmatrix}^T \tag{5}$$

B. Reward function

The goal of the experimental setup is to track the reference states contained in $\boldsymbol{x}^{ref}$. The function is designed to return a negative reward for the weighted tracking error, resulting in an optimal reward of zero if all states are tracked perfectly. In eq (6) the ideal cost function for this task is displayed, where $\boldsymbol{P}$ and $\boldsymbol{Q}$ are diagonal matrices which respectively select and weight the error of the tracked system states. The weighting of the error is one for all states except for the side-slip angle β. A weight of 10^2 is used to bring the side-slip error contribution more in line with the error magnitude of the other tracked states.

$$r_t = -\left(\boldsymbol{P}\left(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t^{ref}\right)\right)^T \boldsymbol{Q}\boldsymbol{P}\left(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t^{ref}\right) \tag{6}$$

The next system state $\boldsymbol{x}_{t+1}$ is used to allow the agent to have one time step to adjust the action and steer toward the current reference. Unfortunately, since at time t the next system state $\boldsymbol{x}_{t+1}$ is not yet known, the state is estimated and replaced with $\hat{\boldsymbol{x}}_{t+1}$ resulting in eq. (7) to calculate the reward r_t. The state prediction is performed using a Recursive Least Squares model, which is detailed in section IV.C.

$$r_t \approx -\left(\boldsymbol{P}\left(\hat{\boldsymbol{x}}_{t+1} - \boldsymbol{x}_t^{ref}\right)\right)^T \boldsymbol{Q}\boldsymbol{P}\left(\hat{\boldsymbol{x}}_{t+1} - \boldsymbol{x}_t^{ref}\right) \tag{7}$$

The reward function and its derivative are used to update the actor-critic such that the return is maximized. As the reward function described in eq. (7) is dependent on the predicted state $\hat{\boldsymbol{x}}_{t+1}$, the reward and therefore the policy performance has an upper bound proportional to the accuracy of the prediction.

IV. Design and implementation of the agent

The actor-critic design is based on the Dual Heuristic Dynamic Programming framework [11, 13, 21]. For this research the actor and critic use function approximators in the form of fully connected artificial neural networks. The functions approximators parameterize the actor and critic with $\boldsymbol{w}_a$ and $\boldsymbol{w}_c$. The resulting actor and critic definition used is shown in equation (8). The policy is defined to be provided by the actor network, while the critic network approximates the real state value derivative $\lambda(\boldsymbol{s}_t)$. The actor-critic design updates rely on local derivatives of the system dynamics, together with a next state prediction. This information is identified by using an online model identification based on a Recursive Least Squares (RLS) parameter estimation.

$$\lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct}) \approx \lambda(\boldsymbol{s}_t) \qquad \text{and} \qquad \pi(\boldsymbol{s}_t, \boldsymbol{w}_{at}) := \pi(\boldsymbol{s}_t) \tag{8}$$

Gradient decent and ascent is used to improve the state value derivative approximation $\lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct})$ and in turn the policy $\pi(\boldsymbol{s}_t, \boldsymbol{w}_{at})$ of the agent. The general principle of the parameter update using a gradient based optimization is shown in equation (9).

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta\boldsymbol{w}_t \tag{9}$$

A. Actor update

The actor forms the policy $\pi(\boldsymbol{s}_t, \boldsymbol{w}_a)$ which maps the state $\boldsymbol{s}_t$ to an action $\boldsymbol{a}_t$ used as input to the Cessna Citation 550. The relation between actor and action is shown in equation (10).

$$\pi(\boldsymbol{s}_t, \boldsymbol{w}_{at}) = \boldsymbol{a}_t \tag{10}$$

The goal of the presented reinforcement learning problem is to improve the policy toward the optimal policy, which maximizes the state value function $V(\boldsymbol{s}_t)$ or alternatively the temporal decomposed equivalent $r_t + \gamma V(\boldsymbol{s}_{t+1})$ [7]. The optimization process of the actor therefore has to follow equation (11), where $\boldsymbol{a}_t^*$ is the optimal action and γ the discount factor [11].

$$\boldsymbol{a}_t^* = \arg\max_{\boldsymbol{a}_t} \left(r_t + \gamma V(\boldsymbol{s}_{t+1})\right) \tag{11}$$

Applying gradient ascent to the optimality equation and combining this with the general parameter update equation of (9) results in eq. (12) to update the actor toward the optimal policy.

$$\begin{aligned} \Delta\boldsymbol{w}_{at} &= \eta_a \frac{\partial\left(r_t + V(\boldsymbol{s}_{t+1})\right)}{\partial \boldsymbol{a}_t} \frac{\partial \boldsymbol{a}_t}{\partial \boldsymbol{w}_{at}} \\ &= \eta_a \left(\frac{\partial r_t}{\partial \hat{\boldsymbol{x}}_{t+1}} + \gamma\lambda(\hat{\boldsymbol{s}}_{t+1})\right) \frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial \boldsymbol{a}_t} \frac{\partial \boldsymbol{a}_t}{\partial \boldsymbol{w}_{at}} \end{aligned} \tag{12}$$

The resulting equation to update the actor parameters involves the learn rate η_a, the cost function derivative, the critic output and the system model. As the next state $\boldsymbol{x}_{t+1}$ is unavailable, the predicted system state $\hat{\boldsymbol{x}}_{t+1}$ is instead used for the actor update. The predicted reinforcement learning state $\hat{\boldsymbol{s}}_{t+1}$ is constructed using equation (2).

B. Critic update

The critic estimates the state value derivative $\lambda(\boldsymbol{s}, \boldsymbol{w}_c)$ using a function approximator with parameters $\boldsymbol{w}_c$ [21]. The relation between the critic output and the value derivative is shown in equation (13). The state value $V(\boldsymbol{s}_t)$ is equivalent to the expexted sum of discounted rewards, where the reward is defined by eq. (7).

$$\lambda(\boldsymbol{s}_t, \boldsymbol{w}_c) \approx \frac{\partial V(\boldsymbol{s}_t)}{\partial \boldsymbol{x}_t} \quad \text{with} \quad V(\boldsymbol{s}_t) = \mathbb{E}\left[\sum_{n=0}^{\infty} \gamma^n r_{t+n}\right] \tag{13}$$

The critic update is based on minimizing the temporal difference error, the error between the current and successive estimates of the state value. Taking the derivative of the temporal difference error with respect to the system state results in the critic error $\boldsymbol{e}_c$ to be minimized. The critic error $\boldsymbol{e}_c$ is shown in equation (14).

$$\boldsymbol{e}_c = \frac{\partial V(\boldsymbol{s}_t, \boldsymbol{w}) - r_t - \gamma V(\boldsymbol{s}_{t+1}, \boldsymbol{w})}{\partial \boldsymbol{x}_t} \tag{14}$$

The critic error is then rewritten to the form given by equation (15) to involve only terms related to the critic, cost function, system model and actor. The derivative of the next state with respect to the current state is expanded as it is both a function of the current system state and the action.

$$\begin{aligned} \boldsymbol{e}_t &= \lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct}) - \left(\frac{\partial r_t}{\partial \boldsymbol{x}_{t+1}} + \gamma \lambda(\boldsymbol{s}_{t+1}, \boldsymbol{w}_c)\right) \frac{\partial \boldsymbol{x}_{t+1}}{\partial \boldsymbol{x}_t} \\ \boldsymbol{e}_t &= \lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct}) - \left(\frac{\partial r_t}{\partial \boldsymbol{x}_{t+1}} + \gamma \lambda(\boldsymbol{s}_{t+1}, \boldsymbol{w}_c)\right) \left(\frac{\partial \boldsymbol{x}_{t+1}}{\partial \boldsymbol{x}_t} + \frac{\partial \boldsymbol{x}_{t+1}}{\partial \boldsymbol{a}_t} \frac{\partial \boldsymbol{a}_t}{\partial \boldsymbol{x}_t}\right) \end{aligned} \tag{15}$$

The temporal difference error is one of the key elements of reinforcement learning, but can suffer from numerical instability [22]. The instability is caused by the successive critic terms using the same set of parameters $\boldsymbol{w}_c$ for calculating the error [23]. A successful approach to reduce numerical issues is by utilizing targets, a copy of the critic or actor with a different set of parameters [22, 23]. Replacing the second critic term by a target critic and using the next state predictions $\hat{\boldsymbol{x}}_{t+1}$ and $\hat{\boldsymbol{s}}_{t+1}$ results in the final formula for the critic error as shown in equation (16).

$$\boldsymbol{e}_t = \lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct}) - \left(\frac{\partial r_t}{\partial \hat{\boldsymbol{x}}_{t+1}} + \gamma \lambda'(\hat{\boldsymbol{s}}_{t+1}, \boldsymbol{w}_c{}')\right) \left(\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial \boldsymbol{x}_t} + \frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial \boldsymbol{a}_t} \frac{\partial \boldsymbol{a}_t}{\partial \boldsymbol{x}_t}\right) \tag{16}$$

The critic loss is then defined as the square critic error, to which gradient decent is applied and combined with the general parameter update, resulting in equation (17).

$$\begin{aligned} \Delta \boldsymbol{w}_{ct} &= \frac{1}{2} \eta_c \frac{\partial \boldsymbol{e}_c^T \boldsymbol{e}_c}{\partial \lambda(\boldsymbol{s}_t, \boldsymbol{w}_c)} \frac{\partial \lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct})}{\partial \boldsymbol{w}_{ct}} \\ &= \eta_c \boldsymbol{e}_{ct} \frac{\partial \lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct})}{\partial \boldsymbol{w}_t^c} \end{aligned} \tag{17}$$

The addition of the target critic results in an additional update rule for the target parameters $\boldsymbol{w}_c{}'$. The update is based on a slow moving average where τ indicates the time constant. The update rule for the target parameters is shown in equation (18) [23].

$$\boldsymbol{w}_{c\,t+1}' = \tau\, \boldsymbol{w}_{ct} + (1 - \tau)\, \boldsymbol{w}_{c\,t}' \tag{18}$$

C. System Model

The actor and critic updates both rely on local gradients of the controlled plant dynamics. To deliver an algorithm which is independent on prior model knowledge this information is estimated online using a RLS parameter estimation. The model structure used for the parameter estimation is shown in eq. (19), with the current state $\boldsymbol{x}_t$ and control input $\boldsymbol{a}_t$ as inputs. The constant vector $\boldsymbol{c}_{t-1}$ is added to the model structure to compensate for the lack of a fixed linearization point.

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{F}_{t-1} \boldsymbol{x}_t + \boldsymbol{G}_{t-1} \boldsymbol{a}_t + \boldsymbol{c}_{t-1} \tag{19}$$

The RLS parameter estimation problem is often written in the form of eq. (20), where the state variables are combined in a vector $\boldsymbol{x}^{RLS}$ and all parameters to be estimated in matrix $\boldsymbol{\Theta}$.

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{x}_t^{RLS} \boldsymbol{\Theta}_{t-1}, \qquad \boldsymbol{x}_t^{RLS} = \begin{bmatrix} \boldsymbol{x}_t^T & \boldsymbol{a}_t^T & 1 \end{bmatrix}, \qquad \boldsymbol{\Theta}_{t-1} = \begin{pmatrix} \boldsymbol{F}_{t-1} & \boldsymbol{G}_{t-1} & \boldsymbol{c}_{t-1} \end{pmatrix}^T \tag{20}$$

It is possible to use other model structures, such as incremental models [24] or structures which use additional terms [25]. However, this specific structure allows to easily retrieve the model derivatives required by the actor-critic, as the derivatives correspond to the $\boldsymbol{F}$ and $\boldsymbol{G}$ matrices as shown in eq. (21).

$$\boldsymbol{F}_{t-1} = \frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial \boldsymbol{x}_t}, \qquad \boldsymbol{G}_{t-1} = \frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial \boldsymbol{a}_t} \tag{21}$$

D. Algorithm Implementation

The agent algorithm is implemented in Python 3 and consists of four steps, namely the retrieval and processing of the references, the iterative actor-critic update, applying the current policy to the citation simulation and finally the update to the RLS system model. In algorithm 1 the pseudo code of the algorithm is provided. The hyperparameters of the implementation are displayed in table 1.

The discount $\gamma \in [0, 1]$ factor is a measure of how future rewards are weighted with respect to current rewards. As the control surfaces have a direct effect on the controlled states, the agent does not need to take the far future into account resulting in a value in the lower range of possible values. The learn rates influence the magnitude of the parameter updates and are determined empirically. The final parameter, the number of updates per time step N, is chosen such that the simulation is faster than real-time. This parameter has the largest influence on the computational load of the algorithm.

Algorithm 1 Pseudocode of the implemented DHP framework

Require:
Environment: citation, pid_controller, $\frac{\partial r_t}{\partial \hat{\boldsymbol{x}}_{t+1}}$
Agent hyperparameters: γ, η_a, η_c, τ, N
Agent: $\pi(\boldsymbol{s}, \boldsymbol{w}_a)$, $\lambda(\boldsymbol{s}, \boldsymbol{w}_c)$, $\lambda'(\boldsymbol{s}, \boldsymbol{w}_c{}')$, model($\boldsymbol{\Theta}$)
Initialize:
citation, $\boldsymbol{w}_{a0}(0)$, $\boldsymbol{w}_{c0}(0)$, $\boldsymbol{s}_0$, $\boldsymbol{\Theta}_0$, $\boldsymbol{w}_{c_0}'(0) \leftarrow \boldsymbol{w}_{c0}(0)$
for $t \leftarrow 0,\ T$ **do**
  # Sample reference
  $h_{e_t}^{ref}, \phi_t^{ref} \leftarrow$ sample()
  $\boldsymbol{x}_t^{ref} \leftarrow$ pid_controller($h_{e_t}^{ref}, \phi_t^{ref}$)
  $\boldsymbol{s}_t \leftarrow \boldsymbol{x}_t,\ \boldsymbol{x}_t^{ref}$
  # Update Agent
  $\hat{\boldsymbol{F}}_{t-1}, \hat{\boldsymbol{G}}_{t-1} \leftarrow \boldsymbol{\Theta}_t$
  for $i \leftarrow 0,\ N$ **do**
    $\boldsymbol{a}_t(i) \leftarrow \pi(\boldsymbol{s}_t, \boldsymbol{w}_{at}(i))$
    $\hat{\boldsymbol{x}}_{t+1}(i) \leftarrow$ model.predict($\boldsymbol{x}_t, \boldsymbol{a}_t(i)$)
    $\hat{\boldsymbol{s}}_{t+1}(i) \leftarrow \hat{\boldsymbol{x}}_{t+1}(i),\ \boldsymbol{x}_t^{ref}$
    # Critic Update
    $\boldsymbol{\lambda}_c^t(i) \leftarrow \lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct}(i)),\ \boldsymbol{\lambda}_c^{t+1}(i) \leftarrow \lambda'\left(\hat{\boldsymbol{s}}_{t+1}(i), \boldsymbol{w}_{ct}'(i)\right)$
    $\boldsymbol{w}_{ct}(i+1) \leftarrow \boldsymbol{w}_{ct}(i) - \eta_c \left(\boldsymbol{\lambda}_c^t(i) - \left[\frac{\partial r_t}{\partial \hat{\boldsymbol{x}}_{t+1}} + \gamma \boldsymbol{\lambda}_c^{t+1}(i)\right]\left[\hat{\boldsymbol{F}}_{t-1} + \hat{\boldsymbol{G}}_{t-1}\frac{\partial \pi(\boldsymbol{s}_t, \boldsymbol{w}_{at}(i))}{\partial \boldsymbol{x}_t}\right]\right)\frac{\partial \lambda(\boldsymbol{s}_t, \boldsymbol{w}_{ct}(i))}{\partial \boldsymbol{w}_{ct}(i)}$
    $\boldsymbol{w}_{ct}'(i+1) \leftarrow \tau\, \boldsymbol{w}_{ct}(i+1) + (1-\tau)\, \boldsymbol{w}_{ct}'(i)$
    # Actor Update
    $\boldsymbol{\lambda}_a^{t+1}(i) \leftarrow \lambda(\hat{\boldsymbol{s}}_{t+1}(i), \boldsymbol{w}_{ct}(i+1))$
    $\boldsymbol{w}_{at}(i+1) \leftarrow \boldsymbol{w}_{at}(i) + \eta_a \left(\frac{\partial r_t}{\partial \hat{\boldsymbol{x}}_{t+1}} + \gamma \boldsymbol{\lambda}_a^{t+1}(i)\right)\hat{\boldsymbol{G}}_{t-1}\frac{\partial \pi(\boldsymbol{s}_t, \boldsymbol{w}_{at}(i))}{\partial \boldsymbol{w}_{at}(i)}$
  end for
  $\boldsymbol{w}_{at+1}(0) \leftarrow \boldsymbol{w}_{at}(N), \boldsymbol{w}_{ct+1}(0) \leftarrow \boldsymbol{w}_{ct}(N)$
  # Perform action and update system model
  $\boldsymbol{a}_t \leftarrow \pi(\boldsymbol{s}_t, \boldsymbol{w}_{at+1}(0))$
  $\boldsymbol{x}_{t+1} \leftarrow$ citation.step($\boldsymbol{a}_t$)
  $\boldsymbol{\Theta}_{t+1} \leftarrow$ model.update($\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t+1}$)
end for

Table 1 Hyperparameters used for the agent implementation as described by algorithm 1.

Hyperparameter	Symbol	Value
Discount factor	γ	0.40
Critic learn rate	η_c	0.10
Actor learn rate	η_a	0.05
Target critic learn rate	τ	0.001
Number of actor-critic update cycles	N	2
RLS forgetting factor	κ	0.9995

E. Actor-Critic network design

The algorithm described in section IV can be applied to any function approximator which allow gradient based updates. For the purpose of this paper artificial neural networks are used. The networks are implemented using the open-source TensorFlow library which provides straight-forward network construction, gradient calculation and parameter updates.

In figure 4 the design of the actor and critic networks is visualized. The critic and target network consists of a single fully connected neural network, mapping the input state $\boldsymbol{s}_t$ to the value derivative with respect to the system state $\lambda\left(\boldsymbol{s}_t, \boldsymbol{w}_{ct}\right)$. In contrast the actor uses a more complex internal structure to map the input state $\boldsymbol{s}_t$ to an action $\boldsymbol{a}_t$.

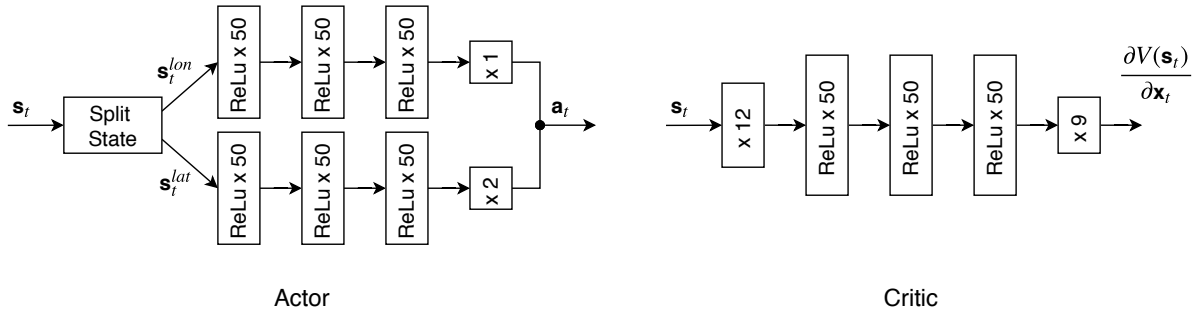


Fig. 4 Artificial neural network layout of the actor and critic.

The actor is internally split in two artificial neural networks separately controlling the longitudinal and lateral motions of the aircraft. The separation is implemented to reduce the initial influence of states and references on the uncoupled motion as the agent learns. To provide each actor with the correct inputs, the incoming state and reference vector is sliced to contain only the relevant elements resulting in the internal vectors shown in eq. (22). The sliced state vectors $\boldsymbol{s}_t^{lon}$ and $\boldsymbol{s}_t^{lat}$ pass through the two neural network structures, after which the resulting actions are concatenated to provide a single action vector $\boldsymbol{a}_t$ as output.

$$\begin{aligned} \boldsymbol{s}_t^{lon} &= \begin{bmatrix} q & V_{TAS} & \alpha & \theta & h_e & q^{ref} \end{bmatrix}^T \\ \boldsymbol{s}_t^{lat} &= \begin{bmatrix} p & r & V_{TAS} & \beta & \phi & h_e & p^{ref} & \beta^{ref} \end{bmatrix}^T \end{aligned} \tag{22}$$

As a result of the actor implementation, the derivative of the action with respect to some of the full state elements is assumed to be zero as shown by eq. (23). This also implies the longitudinal action elements are not a function of the lateral state elements and vice-versa.

$$\frac{\partial \boldsymbol{a}^{lon}}{\partial \boldsymbol{x}^{lat}} = \mathbf{0} \qquad \text{and} \qquad \frac{\partial \boldsymbol{a}^{lat}}{\partial \boldsymbol{x}^{lon}} = \mathbf{0} \tag{23}$$

These assumptions may give rise to errors if the coupling between symmetric and asymmetric motion is significant. However, the outer loop control adjusts the body rate references accordingly and the continued actor optimization should

minimize errors actively, reducing the impact. Furthermore, the critic can still communicate cross coupling effects to the actor through the update process.

Both network designs use rectified linear units (ReLu) as activation functions on the hidden layers. ReLus do not suffer from vanishing gradients and are computational light in use. A downside of the use of this activation function is the network has to be deeper and/or wider, as the discontinuity of the ReLu can otherwise become visible in the network output. This size requirement is already visible in this network layout, with three hidden layers of size fifty. The actor and critic can both be reduced to a single hidden layer of size 13 by using sigmoid based activation functions.

V. Experiment

A. Flight profile and intial conditions

The experiment uses a single flight profile, which is defined in terms of a desired altitude and bank angle. Each episode lasts five minutes and consists of climb, descent and turn segments, which are combined such that all combinations of these segments are encountered within the profile.

The first thirty seconds of each episode is labeled as the forced excitation phase. During the first twenty seconds of this phase an excitation signal is added to the elevator and aileron commands to improve the system identification and forming of the initial policy of the agent. The excitation consists of a frequency sweep signal for the elevator and a sine signal for the ailerons. These signals are chosen to be uncorrelated so the agent can separate the contribution of each control surface individually. The forced excitation is combined with a changing reference signal to provide the agent with rich state information to learn from and correlate with the reward function. The forced excitation phase is concluded with ten seconds of commanded steady straight and level flight to partially recover from the excitation and transient dynamics before the next phase starts.

The final four and a half minutes of flight does not contain an excitation signal and aims to simulate different flight maneuvers. This phase is labeled as the maneuver phase. In figure 5 the profile is shown, containing the relative altitude and the bank angle commands. The profile contains a prolonged climb to reach a different flight regime before descending. The bank angle alternates such that during this altitude profile a combination of climbing, level, descending and turning, straight flight is commanded.

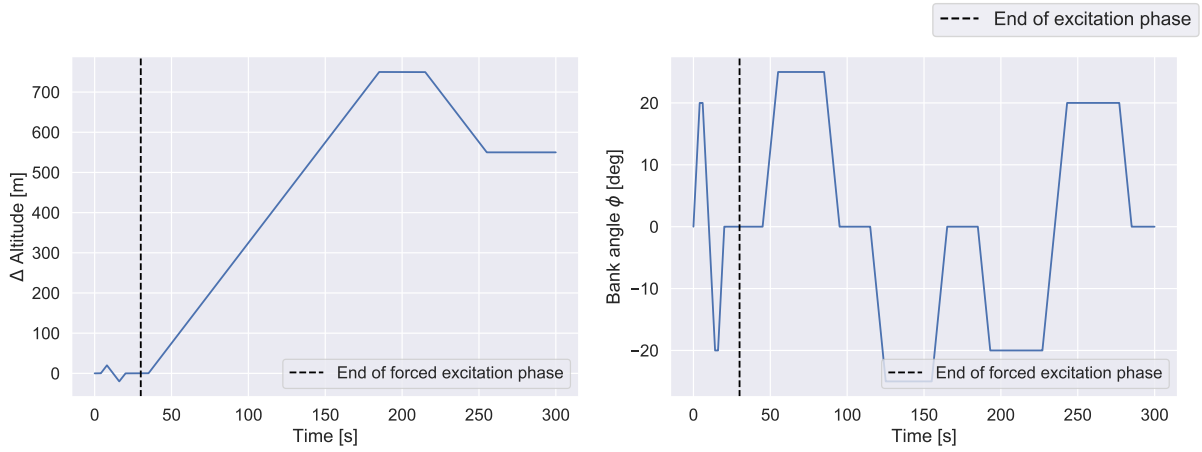


Fig. 5 The altitude and bank angle profile used for the episodic simulations as part of the experiment. The altitude is relative to the initial altitude of the aircraft.

The climb and decent rates of the shown flight profile are both 5 m/s, whereas the maximum climb rate of the Cessna Citation 550 under ideal circumstances is 14.7 m/s [26]. This number is chosen such that during the maneuver phase the angle of attack stays below $10deg$ during the combined climb and turns. The bank angle rate during the maneuver phase is 2.5 deg/s, with a maximum of 25 deg. This bank angle is based on the typical bank angle when performing a standard rate turn.

The experiment will use four initial flight conditions to showcase and quantify the performance and learning

capabilities of the controller under different starting circumstances. During the episode the altitude will change based on the flight profile shown in figure 5. In table 2 an overview of the flight conditions used in the experiment are shown. The flight conditions vary the airspeed and altitude to cover different points in the flight envelope of the Cessna Citation II.

Table 2 The four flight conditions used in the experiment, which vary the initial airspeed and altitude.

Flight Condition	Airspeed	Altitude
FC0	140 m/s	5000 m
FC1	90 m/s	5000 m
FC2	140 m/s	2000 m
FC3	90 m/s	2000 m

Combined with the flight profile, the four starting conditions are well within the bounds of the flight envelope of the Cessna Citation II. Flight condition FC1 and FC3 operate below the maneuver speed and are therefore limited by stall, while conditions FC0 and FC3 are above the maneuver speed and are therefore limited by the structural limits of the aircraft. The maximum lift or stall region occurs above 10 deg angle of attack for the implemented model, although it should be noted that the model is less accurate around the stall regions. This is a current area of research to further improve the dynamic model of the Cessna Citation II. Nevertheless, the true limiting factor during the maneuver phase is the excess power available, which is enough for the maneuver, but not enough to retain speed in case of large altitude corrections.

The experiment consists of five cases which are listed in table 3. Each case consists of 100 episodes for each flight condition, resulting in a total of 400 simulations for each case. Case A considers the proposed agent framework, without any alterations as described in section IV. This serves as a benchmark for the other cases and provides a quantification of performance and learning robustness of the proposed controller. Case B is repeated for an agent design without the use of a target critic network. All other settings and parameters are equal. The goal of this case is to quantify the influence of target networks on the performance and learning robustness and compare this with the proposed baseline agent. Case C uses fixed model gradients $\boldsymbol{F}$ and $\boldsymbol{G}$, which are obtained by linearizing the Cessna Citation II model around flight condition FC3. The RLS model is still in place and is used for the prediction of the next state.

Case D and E alter the initialization of the actor and critic network parameters. The original networks are initialized with a truncated normal distribution using a standard deviation of 0.1. For Case D and E the magnitude of the standard deviation used is respectively increased and decreased with respect to Case A, to investigate the effects of the network initialization on the learning robustness.

Table 3 The five experiment cases. Case A corresponds to the proposed agent framework as described in section IV.

Experiment Case	Target Critic	Fixed Model Gradients	Network Initialization[1]
A	Yes	No	$\sigma = 10^{-2}$
B	No	No	$\sigma = 10^{-2}$
C	Yes	Yes	$\sigma = 10^{-2}$
D	Yes	No	$\sigma = 10^{-1}$
E	Yes	No	$\sigma = 10^{-3}$

[1] Truncated normal distribution $N_{trunc}\left(\mu, \sigma^2\right)$, with $\mu = 0.0$

The learning robustness is defined as the probability of a random initialized agent to successfully control the Cessna Citation 550 aircraft during the provided flight profile. An agent is considered to be successful if the provided body rate references and as a result the flight profile is flown without large continued errors from which the agent cannot recover. The large continued errors cause the agent parameters to become unstable and overflow, which is used as the criteria to separate the failed and successful runs.

VI. Results

The experiment is performed using the proposed agent implementation and several variations to investigate the effects of the target networks, network initializations and learn rates on performance and learning robustness. Furthermore, the dependency on the system identification is highlighted.

A. Experiment: Case A

A random successful episode from experiment case A is shown in figure 6 and 7 for respectively the lateral and longitudinal states. The illustrated episode is performed using flight condition FC3, which starts at an altitude of 2000 m with an airspeed of 90 m/s. The vertical dashed line indicates the separation between the forced excitation phase and the maneuver phase of the experiment.

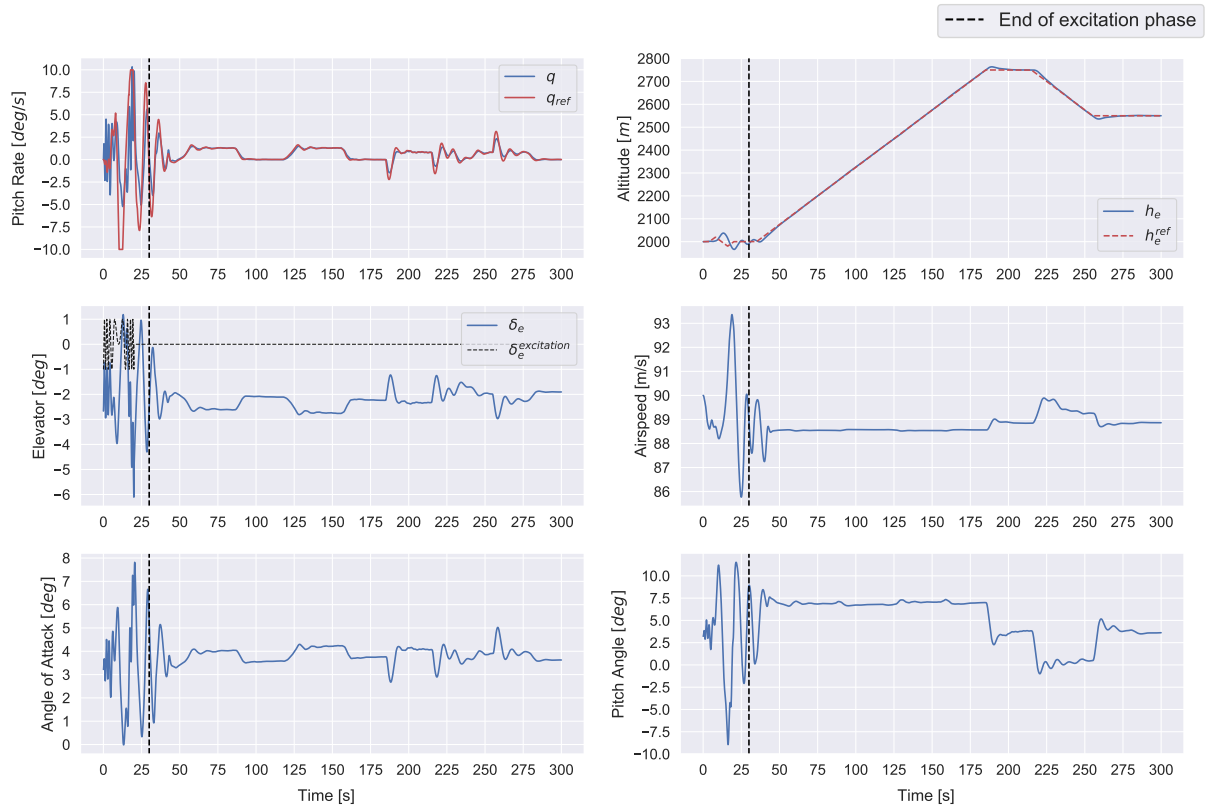


Fig. 6 The longitudinal subset of states of a single episode starting at flight condition FC3. The episode is divided by the black dotted line on $t = 30$ s between the forced excitation phase and the maneuver phase.

The forced excitation phase shows abrupt changes in the different longitudinal and lateral states as a result of the forced excitation signals added to the elevator and aileron commands. The outer loop controller tries to counteract the errors introduced by the excitation signal on top of following the flight profile. The agent is not able to follow the reference signals with rate errors between -5 to 5 deg/s. The large errors result from the fast changes in the reference signal of up to 10 deg/s in a time span of 5 seconds for both the pitch and roll axis of the aircraft. This is combined with the agent still forming a control policy.

The angle of attack ranges from 0 to 8 deg throughout the forced excitation phase, which may also impact safety for the real aircraft as a result of the structural limits. Furthermore, this may be uncomfortable for passengers. The load factor experienced by the aircraft is therefore calculated and found to be within the range of 0.26 to 2.08. This is within the limits of -1.0 to 3.1 specified by the CS-25 regulations.

The transition between the forced excitation phase en the maneuver phase shows damped oscillations as the agent is still recovering from the forced excitation and further learns a correct policy. Most agents are on the current target as the prolonged climb starts.

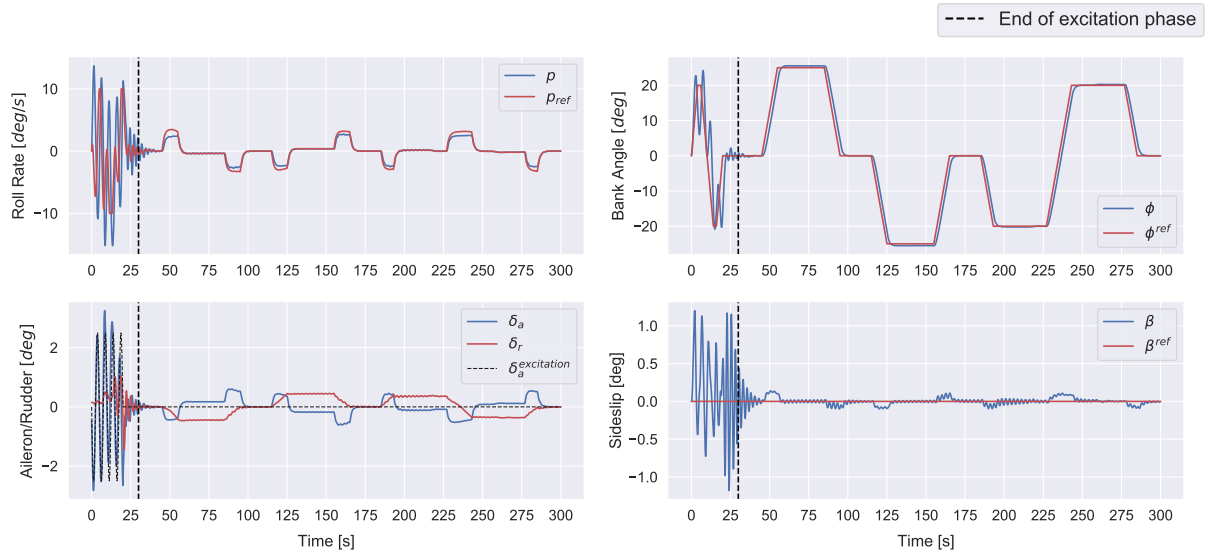


Fig. 7 The lateral subset of states of a single episode starting at flight condition FC3. The episode is divided by the black dotted line on $t = 30\ s$ between the forced excitation phase and the maneuver phase.

All successful episodes are able to follow the provided references in the maneuver phase. The tracking error depends on the specific run and is directly proportional to the cost and episode return. The episode displayed in figures 6 and 7 is not able to reach the roll rate reference when steering toward the desired bank angle. Also the pitch rate is seen to not reach the target fully when the reference changes fast due to the transition in desired climb rates. The final tracked state, the sideslip angle, is able to reach an error between -0.1 and 0.1 degrees, mostly due to the large weight of 10^2 in the cost weight matrix $\boldsymbol{Q}$.

One of the performance parameters is robustness of the proposed agent framework. For a online reinforcement learning controller, the success rate is a measure of its robustness. The success rate is a clear indication if the agent is able to maintain a stable policy throughout the episode. A side effect of an unstable policy or control law are the increasing cost or tracking errors over time. The unbounded growth in error combined with aggressive learn rates quickly result in float overflows of the actor-critic. In table 4 the succes rate of the experiment is shown for all flight conditions.

Table 4 Case A: Success rate of the episodes per flight condition

Flight Condition	Total Episodes	Success	Failed
FC0	100	87	13
FC1	100	73	27
FC2	100	100	0
FC3	100	100	0

All episodes at a starting altitude of 2000 m (FC2 and FC3) are successful, while the other flight conditions (FC0 and FC1) contain failed episodes. The number of failed episodes increases as the aerodynamic damping decreases with an increase in height or a decrease in airspeed. Investigating the flight cases with less aerodynamic damping showed a decrease in damping of the pitch rate and roll rate. In part this is due to the outer loop controller, which is designed on flight condition FC3 and produced the body rate references with less damping.

The increased oscillations can result in two conditions which cause an episode to fail. The first is an unstable pitch motion resulting in stall of the simulated Cessna Citation 550. The stall and rapid changing local system dynamics result in the plant model to rapidly deteriorate after which the episode fails. The second cause of failure are the prolonged errors resulting from the constantly changing references. The critic tries to adapt and can become numerical unstable as

the parameters grow without bound, causing the episode to fail.

The second performance measure of the agent is the episodic return, the sum of all rewards obtained during the episode. In figure 8 the distribution of the return magnitude is visualized using box-plots. The figure is split in the magnitude of the total and maneuver return, which respectively sums the rewards obtained from the complete episode and only the maneuver phase. As the magnitude of the return is visualized, a lower value indicates better performance. A first look indicates that both the total and maneuver return share the same distribution, with the largest difference being the magnitude of the returns. The total return is around a factor 10 larger than the maneuver phase return. To put this in perspective, the maneuver phase consists of the final 270 s compared to the total length of 300 s of a complete episode.

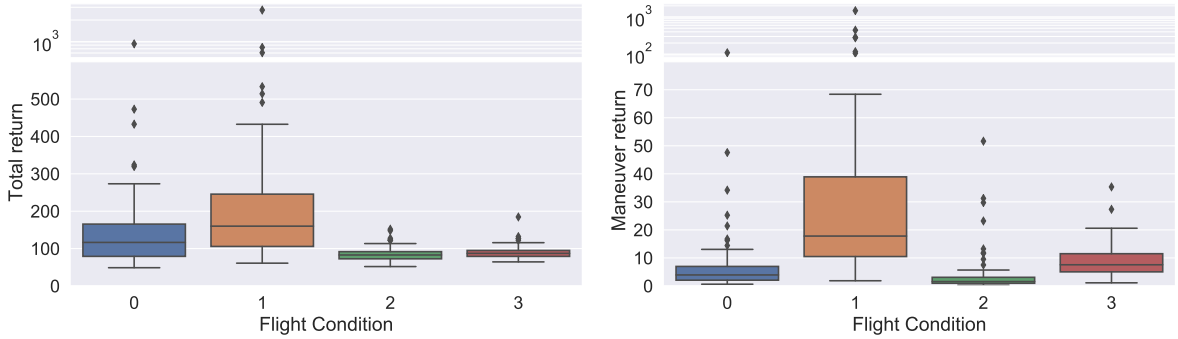


Fig. 8 The distribution of the magnitude of the returns for experiment Case A. The left figure visualizes the complete episode, and right the maneuver phase only. A return of zero indicates the agent is able to perfectly follow the reference signals.

From the box-plots it can be observed that the return increases as the aerodynamic damping diminishes. The spread of the total return is significant larger for the flight conditions at 5000 m. The maneuver return however is more sensitive to the airspeed, where the conditions at 140 m/s perform best in terms of the interquartile range and median.

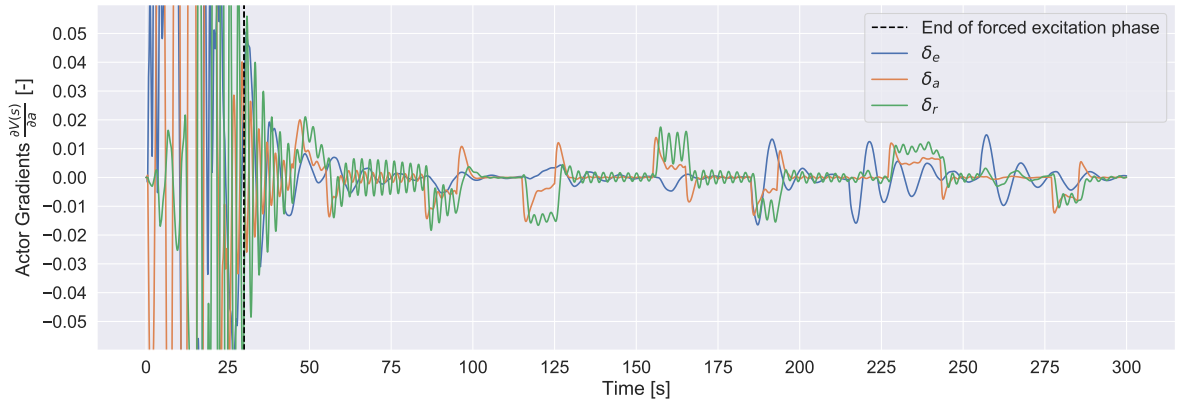


Fig. 9 The actor gradients of the final update cycle throughout the illustrated episode. In the forced excitation phase the gradient magnitude exceed the limits of the figure, and reach values within the range of -0.40 to 0.31.

In figure 9 the gradients used to update the actor are shown. The gradients correspond to the derivative of the state value with respect to the actions as part of equation (12). The actor gradients shape the control law and are a measure of how the actor adapts during the episode. In the forced excitation phase the largest gradients are obtained as a result of the excitation and the large error due to the random initialized agent. As the forced excitation phase ends, the actor is still shown to learn and recover, with the gradient magnitude rapidly decreasing as the tracking errors are minimized. After roughly 50 seconds, the actor is providing a near-optimal control policy, where the actor continuously adapts to

the changes in the commands and states. As expected the gradients closely resemble the tracking errors.

B. Experiment Case B: The influence of target networks

The goal of implementing the target critic network in the agent algorithm is to improve the numerical stability, resulting in less failed episodes and thereby increasing the learning robustness. The same experiment is repeated for agents with the target network disabled, to demonstrate the effects of the target network on both performance and learning robustness.

In table 5 the overview of the episode succesrate for each flight condition is shown for the agent implementations without a target critic network. As expected, the robustness of the algorithm decreased without the use of the target network. The differences are six and seven additional failures for respectively flight condition FC0 and FC1. Additional experiments are needed to conclude if this difference is significant. The flight conditions at 2000 m do not shown an increase in failure rate as all episodes were successful for both Case A and B.

Table 5 Case B: Outcome of the episodes per flight condition compared against the success rate of Case A

		Case A		Case B	
Flight Condition	Total Episodes	Success	Failed	Success	Failed
FC0	100	87	13	81	19
FC1	100	73	27	66	34
FC2	100	100	0	100	0
FC3	100	100	0	100	0

The return for the complete flight profile and only the maneuver phase is displayed in figure 10. Compared to experiment case A, no significant differences are observed in the distributions of the total and maneuver return.

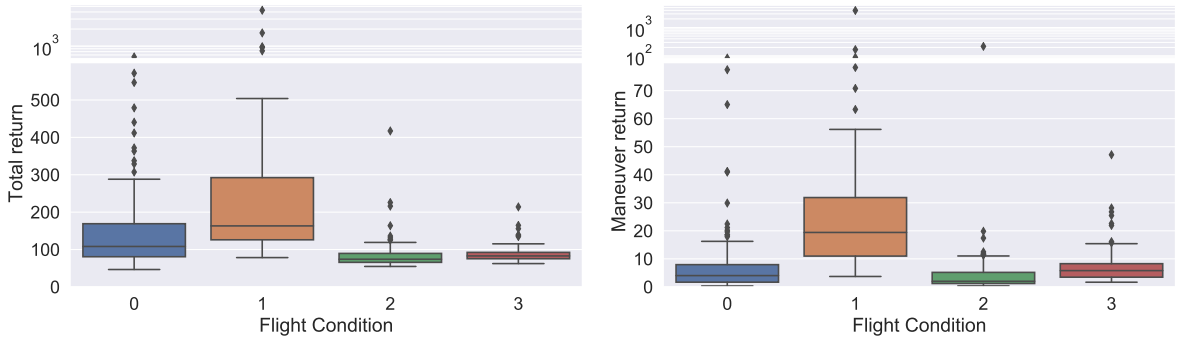


Fig. 10 The distribution of the magnitude of the returns for experiment Case B. The left figure visualizes the complete episode, and right the maneuver phase only. A return of zero indicates the agent is able to perfectly follow the reference signals.

C. Experiment Case C: The influence of fixed model gradients

The algorithm depends on the system model for a state prediction and local gradient approximation. The gradients are a fundamental part of the algorithm and are hypothesized to have significant influence on the robustness and performance of the agent. In experiment C the gradients obtained from the RLS system model are replaced by a fixed set of gradients obtained by linearizing flight condition FC3.

In table 6 the success rate of experiment C is given. Over all flight conditions, only one episode failed. The flight condition of the failed episode is the furthest away from the linearization point used for calculating the fixed set of gradients. Nevertheless, the increase in success rate is significant for flight condition FC0 and FC1.

Table 6 Case C: Outcome of the episodes per flight condition compared against the success rate of Case A

		Case A		Case C	
Flight Condition	Total Episodes	Success	Failed	Success	Failed
FC0	100	87	13	99	1
FC1	100	73	27	100	0
FC2	100	100	0	100	0
FC3	100	100	0	100	0

The fixed gradients prevent the policy from learning bad behavior while the system is still identified and an initial policy is learned. The result is a large increase in successful episodes with respectively the success rate increasing from 87% to 99% for FC0 and increasing from 73% to 100% for FC1. Using only a single set of gradients indicates the agent is robust to a margin of error on the local gradients. Further research is needed to quantify the margin of error on the gradient estimation.

The distribution of returns is again shown in figure 11. The box-plots show both different in scale of the return and in the distribution. The scale of the total return is decreased by a factor five and only has two outliers relatively close to the maximum value indicated by the box-plot. The returns are again ordered based on the aerodynamic damping, where a higher speed is favored. The maneuver return also has a lower value indicating an increase in tracking performance compared to experiment A.

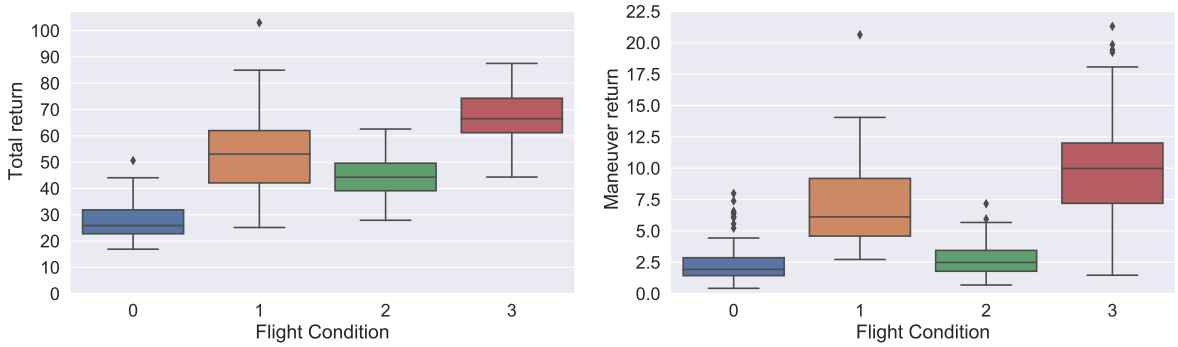


Fig. 11 The distribution of the magnitude of the returns for experiment Case C. The left figure visualizes the complete episode, and right the maneuver phase only. A return of zero indicates the agent is able to perfectly follow the reference signals.

D. Experiment Case D and E: The influence of neural network parameter initialization

The influence of the network initialization is investigated by changing the standard deviation parameter used to initialize the neural network parameters. For experiment A truncated normal distribution with a standard deviation of 0.1 is used to initialize the networks randomly. The experiment is repeated twice for standard deviations of 0.01 and 1.0.

Using a smaller standard deviation of 0.01 resulted in all runs failing. The learning capabilities of the actor-critic is greatly reduced by using parameters too close to zero and/or each other. The resulting agents did not alter any actions until at least 150 seconds had expired. If the agent did manage to change the action and recovered from the inability to learn, it quickly failed due to the large tracking error. The larger standard deviation of 1.0 also resulted in all runs failing. The larger spread in initial parameters results in oversensitive updates causing actuator saturation within a few time steps.

VII. Conclusion and recommendations

This research contributes to the development of an online and adaptive reinforcement learning framework for flight control. The framework is able to learn a near-optimal policy without any prior model knowledge. The proposed agent is able to reliably learn and control the body rate dynamics of a high fidelity Cessna Citation II aircraft model at a regulated airspeed in a faster than real-time simulation. Using forced and uncorrelated excitation added to the control input of the elevator and aileron, the proposed agent is able to learn a policy in thirty seconds starting from a random agent initialization. After the forced excitation to learn an initial policy, the agent is able to fly a combination of level, climbing and descending flight combined with coordinated turns.

The experiment flight conditions do have an effect on the robustness of the proposed controller. At higher altitudes and as a result less aerodynamic damping, the agent experiences failed episodes. The failures originate from prolonged error propagation through the critic to the actor and are in part due to the continued oscillation of the reference signal. As a result the critic parameters can grow unbounded resulting in failure of the agent. The addition of target networks increases the learning robustness of the proposed agent at higher altitudes. The slow moving target is able to stabilize the critic in a subset of the experiment runs and prevent failures when compared to an identical agent without a target critic network. Additional experiments are needed to indicate the significance of this result. As expected the learning robustness benefits from using prior model knowledge in the form of initial model gradient. The agent still adapts the policy to a form of near optimal control if the gradients are fixed throughout the episode and for different flight conditions. This results suggest the method only relies on correct signs and relative magnitude of the elements of the system model gradients. The standard deviation of the truncated normal distribution used to initialize the parameters of the actor-critic neural networks have a large influence on the robustness of the agent by altering the learning sensitivity of the function approximators. Using a smaller standard deviation to generate the initial parameter values results in an inability to learn and adapt the policy. Increasing the standard deviation increases the learning sensitivity resulting in saturation of commanded actuator deflection within several simulation time steps.

To further advance the field of reinforcement learning for flight control, additional research is needed. To produce the results of this research a fixed length episode of five minutes is used. In reality, the episode length will vary based on the duration of the flight. Continued learning can cause instability in the neural networks as the parameters can keep growing over time as it tries to minimize tracking errors. Possible solutions can be L2-regulation[27], parameter clipping or penalizing large commanded actuator rates. This research focuses on implementing the full plant dynamics of the Cessna Citation 550. Additional research is needed to quantify the effects and further develop the framework based on the influence of measurement errors due to sensor dynamics and the influence of environment disturbances such as turbulence.

Reinforcement learning is shown to be a feasible candidate for online adaptive flight control of a CS-25 aircraft and should be further researched by means of flight tests on the PH-LAB aircraft. Additional research is needed to improve the robustness of the framework and reduce the reliance on the online identification of the system model. In the future, reinforcement learning will be able to learn and adapt online to changing environments and provide an effective alternative for fault tolerant flight controllers, reducing the impact of failures and increasing the overall safety of automatic and autonomous controlled flight.

References

[1] Waymo, "On the road," , 2018. https://waymo.com/ontheroad/ [Accessed: 20-08-2018].

[2] Boeing, "Boeing Autonomous Passenger Air Vehicle Completes First Flight," , 2019. http://www.boeing.com/features/2019/01/pav-first-flight-01-19.page [Accessed: 02-03-2019].

[3] Boeing Aviation Safety, "Statistical Summary of Commercial Jet Airplane Accidents: Worldwide Operations 1959 – 2017," Tech. rep., Boeing Commercial Airplanes, 2018.

[4] Balas, G. J., "Flight control law design: An industry perspective," *European Journal of Control*, Vol. 9, No. 2-3, 2003, pp. 207–226.

[5] Acquatella, P., van Kampen, E., and Chu, Q. P., "Incremental backstepping for robust nonlinear flight control," *Proceedings of the EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation and Control*, 2013, pp. 1444–1463.

[6] Sieberling, S., Chu, Q., and Mulder, J., "Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction," *Journal of guidance, control, and dynamics*, Vol. 33, No. 6, 2010, pp. 1732–1742.

[7] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, 2nd ed., MIT Press Cambridge, 2018.

[8] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P., "Trust region policy optimization," *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[10] Zhou, Y., Kampen, E.-J. v., and Chu, Q., "Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2016, pp. 493–496.

[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.

[12] Zhou, Y., "Online reinforcement learning control for aerospace systems," Ph.D. thesis, Delft University of Technology, 2018.

[13] 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.

[14] Enns, R., and Si, J., "Apache helicopter stabilization using neural dynamic programming," *Journal of guidance, control, and dynamics*, Vol. 25, No. 1, 2002, pp. 19–25.

[15] 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.

[16] van Kampen, E.-J., Chu, Q., and Mulder, J., "Continuous adaptive critic flight control aided with approximated plant dynamics," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2006, p. 6429.

[17] Werbos, P. J., Miller, W., and Sutton, R., "A menu of designs for reinforcement learning over time," *Neural networks for control*, 1990, pp. 67–95.

[18] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al., "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, Vol. 362, No. 6419, 2018, pp. 1140–1144.

[19] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K., "Asynchronous methods for deep reinforcement learning," *International conference on machine learning*, 2016, pp. 1928–1937.

[20] Van der Linden, C., "DASMAT: Delft University aircraft simulation model and analysis tool," *Faculty of Aerospace Engineering, Delft Univ. of Technology, Rept. LR-781, Delft, The Netherlands*, 1996.

[21] Si, J., Barto, A. G., Powell, W. B., and Wunsch, D., *Handbook of learning and approximate dynamic programming*, Vol. 2, John Wiley & Sons, 2004.

[22] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, No. 7540, 2015, p. 529.

[23] 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 preprint arXiv:1509.02971*, 2015.

[24] Zhou, Y., van Kampen, E., and Chu, Q., "Incremental model based heuristic dynamic programming for nonlinear adaptive flight control," *Proceedings of the International Micro Air Vehicles Conference and Competition 2016, Beijing, China*, 2016.

[25] Klein, V., and Morelli, E. A., *Aircraft system identification: theory and practice*, American Institute of Aeronautics and Astronautics Reston, VA, 2006.

[26] Hunter, J., *Jane's All the World's Aircraft: In Service 2017/2018: Yearbook*, IHS, 2017.

[27] Ng, A. Y., "Feature selection, L 1 vs. L 2 regularization, and rotational invariance," *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004, p. 78.

II

Report

2

Literature Review

2.1. History of reinforcement learning

Reinforcement learning finds its origins from three major branches that combined over the years [46]. The first branch is formed by the psychology of animal learning, focusing on the learning behavior of animals by means of trial and error. Some famous and fundamental work in this field includes the Law of Effect theory by Edward Thorndike in 1898 and the work of B.F. Skinner in 1990 on operant conditioning [42, 47]. Briefly explained, the Law of Effect states that actions which cause a satisfactory effect become more likely to occur while those actions which cause a discomforting effect become less likely to occur with respect to a certain state [47]. The work of B.F. Skinner on operant conditioning resulted in the Skinner Box, a controlled environment which allowed to observe the learning behavior in combination with a measurable action of the subject. The measured action, often a lever or button, influenced the distribution of food or reward [42].

The second branch is the branch of optimal control, which initially did not consider learning. The problem at hand in optimal control is to design a controller such that it optimizes a defined measure of performance on the behavior of a dynamical system. In the 1950s Richard Bellman with others combined the ideas of a system state and a value function to form an equation now well known as the Bellman Equation [46]. In turn this lead to the introduction of dynamic programming in 1957, which tried to solve the optimal control problem by solving the associated Bellman Equation [9]. Dynamic programming is considered as one of the only practical ways to solve general stochastic optimal control problems, but suffers from what is known as the 'curse of dimensionality' [46].

Temporal difference learning, unique to reinforcement learning, forms the third and final branch. The method is distinctive due to the temporal successive estimates of the same quantity [46]. The origins of this method lie in the psychology of animal learning and can be traced back to the work of Minsky in 1954 and Samuel in 1959. Minsky was the first to realize temporal difference ideas could be used for artificial learning [28], while Samuel was the first to propose and implement temporal difference ideas [34]. The first temporal difference update rule, now known as tabular TD(0), was proposed by Ian Witten in 1976 and was the first to combine the optimal control branch with the temporal difference branch [46, 56]. All threads were fully combined by the development of Q-learning by Chris Watkins in 1989, as it advanced and integrated prior work in all branches of reinforcement learning [46].

2.2. Reinforcement Learning essentials

All reinforcement learning frameworks rely on the same basic interaction of an agent with its environment. A schematic overview of this interaction is shown in figure 2.1. The environment represents the controlled system or plant from which the state and reward are observed. It has as input an action $\mathbf{a}_t \in \mathcal{A}(s)$ which influences the environment and generates a new state and reward. The agent, or controller, is an entity which is responsible for collecting the state $\mathbf{s}_t \in \mathcal{S}$ and reward $r_t \in \mathcal{R}$ and use it to update its policy and to generate a new action based on this policy.

The finite Markov Decision Process (MDP), introduced by Richard Bellman, forms a fundamental basis for most reinforcement learning frameworks as it is an idealized form of the reinforcement learning problem [8, 46]. The MDP introduces several key elements used throughout the field of reinforcement

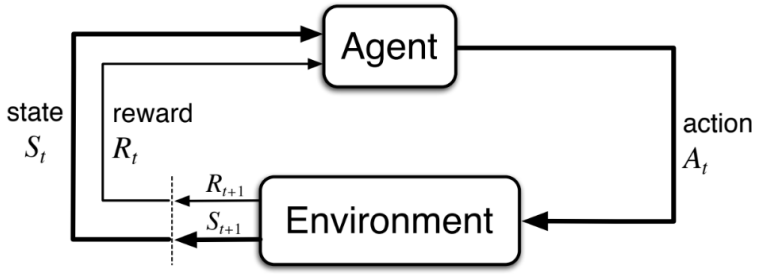


Figure 2.1: The interaction between agent and environment in reinforcement learning [46].

learning, most notably state transitions, rewards and value functions. The finite number of elements in the sets of states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}(s)$ and rewards $r \in \mathcal{R}$ allows for well-defined discrete probability distributions.

2.2.1. The Markov property

As the name implies, the Markov Decision Process possesses the Markov property. Formally this property is defined by eq. (2.1) for discrete processes, where $\mathbf{S}_t$ is a stochastic process with $t \geq 0$ and $\mathbf{s}_t, \mathbf{s}_{t-1}, \dots, \mathbf{s}_1, \mathbf{s}_0 \in \mathcal{S}$ [14]. This property allows to write the state transition probability as a function of only the preceding system state, instead of the complete history of states. In other words, the future is only depended on the immediate preceding state, instead of all states and actions leading up to this state [46].

$$\mathbb{P}\left(\mathbf{S}_t = \mathbf{s}_t \mid \mathbf{S}_{t-1} = \mathbf{s}_{t-1}, \dots, \mathbf{S}_0 = \mathbf{s}_0\right) = \mathbb{P}\left(\mathbf{S}_t = \mathbf{s}_t \mid \mathbf{S}_{t-1} = \mathbf{s}_{t-1}\right) \tag{2.1}$$

The Markov property simplifies the reinforcement learning problem by only having to consider the current system state. Unfortunately not all problems are Markov as system states are often only partial observable. This class of reinforcement learning problems are called Partial Observable Markov Decision Processes (POMDP) and require more advanced techniques to solve reliably, such as trajectory information or believe states [20, 21]. For this introduction, the process is considered to be Markov.

2.2.2. State transition and reward function

A Markov Decision Process can be fully defined by the state transition probability function and the expected reward $r \in \mathcal{R}$ following from the state transitions. The general form of the state transition probability equation is shown by eq.(2.2), with $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ describing the system dynamics of the MDP [46].

$$p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) = \mathbb{P}(\mathbf{S}_{t+1} = \mathbf{s}' \mid \mathbf{S}_t = \mathbf{s},\ \mathbf{A}_t = \mathbf{a}) \tag{2.2}$$

The reward function is a numerical way of representing what is considered positive or negative behavior within the environment. The general definition for the function $r(\mathbf{s}, \mathbf{a})$ is shown by eq. (2.3) [46], taking the expectancy of the reward given a system state and action. Often the reward function is designed to fit the main goal of the problem, but depending on the problem can also be defined by the environment itself.

$$r(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[R_t \mid \mathbf{S}_{t-1} = \mathbf{s},\ \mathbf{A}_{t-1} = \mathbf{a}\right] \tag{2.3}$$

2.2.3. Value functions and policies

The objective of Markov Decision Processes and reinforcement learning in general is to find an optimal policy which maximizes the expected current and future rewards. The expected current and future rewards is often called the return G_t. An important tool for finding the optimal policy is to express the current state and policy as a value function, which is the expectancy of the summation of current and discounted future rewards. The discount factor $\gamma \in [0, 1]$ is used to make the summation finite when episodes last infinitely long and allows to indicate the importance of rewards far in the future compared to immediate rewards [46]. In eq. (2.4) the state-value function is defined which starts in state $\mathbf{s} \in \mathcal{S}$

and thereafter follows policy π [46].

$$v^{\pi}(\mathbf{s}) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,\middle|\, \mathbf{S}_t = \mathbf{s}\right] \tag{2.4}$$

An often used alternative value function representation is the action-value function $q_{\pi}(\mathbf{s}, \mathbf{a})$, which represents the value of starting in state $\mathbf{s} \in \mathcal{S}$ and taking action $\mathbf{a} \in \mathcal{A}(s)$ and then following policy π onward [46].

$$q^{\pi}(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1} \,\middle|\, \mathbf{S}_t = \mathbf{s},\, \mathbf{A}_t = \mathbf{a}\right] \tag{2.5}$$

The use of value functions reduces the objective of the Markov Decision Process to an optimization problem, which involves finding the optimum state-value function or action-value function as defined by eq. (2.6) and (2.7).

$$v^{*}(\mathbf{s}) \quad = \max 2_{\pi}\, v^{\pi}(\mathbf{s}) \tag{2.6}$$

$$q^{*}(\mathbf{s}, \mathbf{a}) = \max 2_{\pi}\, q^{\pi}(\mathbf{s}, \mathbf{a}) \tag{2.7}$$

The optimal policy is then defined by (2.8) or (2.9) depending on the use of the state-value or action-value function [46]. This also shows why the action-value function is often used, as the policy definition is simple and allows for manually searching over the finite number of actions $\mathbf{a} \in \mathcal{A}(s)$.

$$\pi^{*}(\mathbf{s}) = \arg\max_{\mathbf{a}}\; r(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) v^{*}(\mathbf{s}') \tag{2.8}$$

$$\pi^{*}(\mathbf{s}) = \arg\max_{\mathbf{a}}\; q^{*}(\mathbf{s}, \mathbf{a}) \tag{2.9}$$

2.3. Solving the Reinforcement Learning Problem

To solve a MDP two simple methods can be used, value iteration and policy iteration. Value iteration uses an iterative process to estimate the optimal state value, after which the policy is extracted. The pseudo-code for this iterative process is given by algorithm 2.1. The second method is policy iteration, which cycles between a policy evaluation and policy update step and is given by algorithm 2.2.

Algorithm 2.1 Value Iteration [46]

1: Initialize $V(\mathbf{s})$ arbitrarily
2: **for** $s \in \mathcal{S}$ **do** until $V(\mathbf{s})$ converges.
3: $\quad V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} \sum_{\mathbf{s}', r} p(\mathbf{s}', r \mid \mathbf{s}, \mathbf{a})\,[r + \gamma\, V(\mathbf{s}')]$
4: **end for**
5: $\pi(\mathbf{s}) \leftarrow \arg\max_{\mathbf{a}} \sum_{\mathbf{s}', r} p(\mathbf{s}', r \mid \mathbf{s}, \mathbf{a})\,[r + \gamma\, V(\mathbf{s}')]$

Algorithm 2.2 Policy Iteration [46]

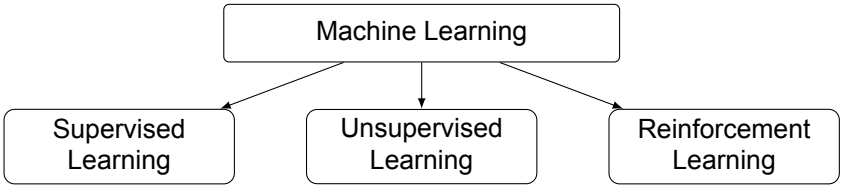
1: Initialize $V(\mathbf{s})$ and $\pi(\mathbf{s})$ arbitrarily
2: # 1. Policy Evaluation
3: **for** $\mathbf{s} \in \mathcal{S}$ **do** until $V(\mathbf{s})$ converges.
4: $\quad V(\mathbf{s}) \leftarrow \sum_{\mathbf{s}', r} p(\mathbf{s}', r \mid \mathbf{s}, \pi(\mathbf{s}))\,[r + \gamma\, V(\mathbf{s}')]$
5: **end for**
6:
7: # 2. Policy Improvement
8: **for** $\mathbf{s} \in \mathcal{S}$ **do**
9: $\quad \pi(\mathbf{s}) \leftarrow \arg\max_{\mathbf{a}} \sum_{\mathbf{s}', r} p(\mathbf{s}', r \mid \mathbf{s}, \mathbf{a})\,[r + \gamma\, V(\mathbf{s}')]$
10: **end for**
11: Return to step #1 (line 2).

Another approach used throughout the rest of this report is to use function approximators with parameters $\mathbf{w}$ to estimate these functions. The notation for these function approximators use a capital and state the parameter vector used. This results in the state value function $V(\mathbf{s} \mid \mathbf{w}_c)$, state action value function $Q(\mathbf{s}, \mathbf{a} \mid \mathbf{w}_c)$ and policy $\pi(\mathbf{s} \mid \mathbf{w}_a)$.

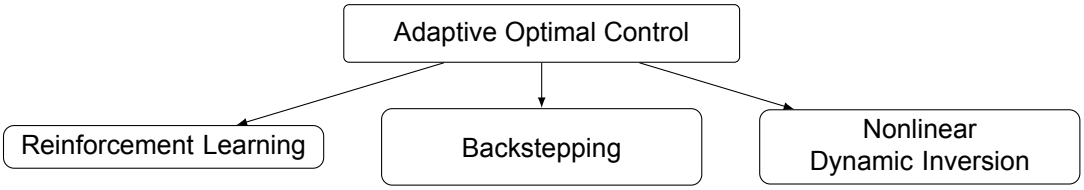
2.4. Overview of reinforcement learning for control tasks

Reinforcement learning finds it origin in several branches of research as discussed in section 2.1. Even the current state-of-the-art on reinforcement learning is researched in two distinct fields, namely Adaptive Optimal Control and Machine Learning. This provides an interesting difference in perspective. Reinforcement Learning within the field of Machine Learning focuses on the framework, data handling and function approximator structure to interpret increasingly complex forms of input. This is different from the field of (Adaptive) Optimal Control, which puts more emphasis on the controlled system and reward signal and uses the function approximator as a tool. It often uses rich continuous reward signals and processed state information as input.

The field of machine learning consists of three main approaches as shown in the tree below. These fields are supervised, unsupervised and reinforcement learning. The difference between the fields is based on the dataset, task and training method. If the training dataset contains what is considered the correct answer, also called the label, it is a supervised learning problem. An example of a supervised task is classification. In contrast, unsupervised learning is when no measurable ground-truth is available, examples of such tasks are clustering or association. The final branch is reinforcement learning, where the task is to optimize the received reward signal. This reward based learning is what separates it from supervised and unsupervised approaches.



Reinforcement Learning as applied in the branch of Machine learning uses two approaches. The most actively researched and most relevant approach uses Deep Learning, where deep neural networks are used as non-linear universal function approximators. This field is called Deep Reinforcement Learning and is treated in depth in section 2.4.2. The seconds approach is a collection of linear methods, using radial basis functions, polynomials, Fourier series or other linear combinations. This second approach is not further detailed in this survey, as artificial neural networks can model these linear structures and DeepRL is more actively researched.



The field of adaptive optimal control largely consists of Reinforcement Learning. Other methods include (Incremental) Backstepping and (Incremental) Non-linear Dynamic Inversion. The field of Reinforcement Learning can again be broken down in other sub-fields, of which two large ones are Inverse Reinforcement Learning and Approximate Dynamic Programming, treated in section 2.4.1 and 2.4.3 respectively. Furthermore, more high level concepts exist not bound to specific sub-fields. Section 2.4.4 elaborates on these high level concepts.

2.4.1. Inverse Reinforcement Learning

Inverse Reinforcement Learning encompasses methods which use observed optimal behaviour as input [31]. Instead of optimizing the policy, the problem is inverted to finding the cost function representing the goal of the observed behaviour. Unfortunately, this branch of RL does not fit the purpose of this

thesis research. The objective is to develop an automatic controller, and as a result can not rely solely on expert data to learn from.

2.4.2. Deep Reinforcement Learning (DeepRL)

DeepRL is an extension of Deep Learning, a Machine Learning approach where deep neural networks are used as universal function approximators. These deep neural networks consist of several types of layers, such as convolution, recurrent and dense layers. The deep architecture of the networks and the different type of layers allow the use of more complex inputs, such as images/frames, sound or even language. Using reinforcement learning techniques this is combined into DeepRL.

This branch of RL is characterized by this end-to-end approach, where raw unprocessed data is used as input to large neural networks. Internally the network has to learn a meaningful state representation in addition to the value function or policy it has to predict. As a result, DeepRL often needs large amounts of data and long training times. The structure of DeepRL agents is based on the Actor-Critic, where separate networks are used to represent the policy and value function. Furthermore, most DeepRL frameworks do not actively consider adaptiveness or continual learning and stop training once the performance does not improve or meets a threshold.

Over the years, many research has been conducted on how to further advance DeepRL. This resulted in an abundance of new algorithms or enhancements to the training process. Unfortunately, listing all this research is not within the scope of this Literature Survey. As a result, the most influential, important and relevant research is discussed in more detail only.

Deep Q-Network (DQN)

The first DeepRL framework is the Deep Q-Network, which showcased superhuman performance on a variety of Atari 2600 computer games using frames as input and score as reward [29]. Up until this point it was not possible to control environments using image data or other forms of vision as input. As the name implies, DQN is based on tabular Q-Learning where a deep neural network approximates the Q-table. Just as Q-learning, a critic only method, the action space is a finite set of actions. The success of this framework is based on two new methods the researchers applied, namely target networks and experience replay.

Experience replay stores previous obtained experiences in a buffer. In eq. (2.10) an experience vector $\mathbf{e}_t$ is shown. At random multiple samples are drawn from this buffer to update the DQN. Experience replay has two benefits: the first is it avoids learning temporal correlations and local approximations as the data is randomly selected and shuffled; the second is retaining memory and more efficiently using past experiences to update the networks. More research is conducted after the introduction of DQN to improve the selection of past experiences from the buffer, which resulted in Prioritized Experience Replay [35] and Hindsight Experience Replay [4].

$$\mathbf{e}_t = \begin{bmatrix} \mathbf{s}_t & \mathbf{a}_t & r_{t+1} & \mathbf{s}_{t+1} \end{bmatrix} \tag{2.10}$$

The second method applied is the use of a target network, a copy of the main network which is updated at a slower rate. The DQN network update uses gradient decent with the mean squared temporal difference error as the loss function. The additional target network is used to calculate the value target, part of the temporal difference error as shown by eq. (2.11). The benefit of using this method is that it prevent numerical instability caused by the value prediction and value target changing too rapidly due to the network update.

$$\text{TD Error} = \underbrace{Q(\mathbf{s}_t, \mathbf{a}_t \mid \mathbf{w}_c)}_{\text{Value Prediction}} - \underbrace{\left[r_{t+1} + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a} \mid \mathbf{w}_c^{target}) \right]}_{\text{Value Target}} \tag{2.11}$$

Deep Deterministic Policy Gradient (DDPG)

The next milestone in DeepRL is the transfer toward continues action spaces. This milestone is achieved by Deep Deterministic Policy Gradient [26]. The DDPG network is based on the same principles as DQN, but adds an actor network to predict the action of the agent. This added actor results in an often used architecture in reinforcement learning, namely the Actor-Critic Designs. The additional actor network is updated by using a policy gradient update, in this case the gradient which maximizes

the state action value. The gradient used to update the actor is shown by eq. (2.12) for a batch size of N.

$$\nabla_{\mathbf{w}_a} Q \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_a Q(\mathbf{s}_i, \mathbf{a}_i \mid \mathbf{w}_c) \mid_{\mathbf{a}_i = \pi(\mathbf{s}_i \mid \mathbf{w}_a)} \nabla_{\mathbf{w}_a} \pi(\mathbf{s}_i \mid \mathbf{w}_a) \tag{2.12}$$

The DDPG actor is a deterministic mapping between state and action. To ensure the agent explores sufficiently, action noise $\mathcal{N}$ is added generated by a Ornstein-Uhlenbeck process [49]. This leads to the policy $\pi'(s)$ as shown in eq. (2.13). Other forms of action noise are possible if they suit the environment.

$$\pi'(\mathbf{s}) = \pi(\mathbf{s} \mid \mathbf{w}_a) + \mathcal{N} \tag{2.13}$$

Asynchronous Advantage Actor Critic (A3C)
The Asynchronous Advantage Actor Critic paper proposes two new approaches, asynchronous updates and fixed length sequences of experiences to calculate the advantage [30]. The asynchronous updates uses multiple workers to asynchronous gather experiences and update if enough are gathered. Since then it has been suggested by OpenAI that the synchronous version called Advantage Actor Critic (A2C), where the agent waits to update until the workers are done, performs equally or better than A3C [32]. The other proposed method is to calculate the advantage and use this to update the actor using the policy gradient method described by eq. (2.14).

$$\nabla_{\mathbf{w}_a} V(\mathbf{s}) = \nabla_{\mathbf{w}_a} \log \pi(\mathbf{s} \mid \mathbf{w}_a) A(\mathbf{s}, \mathbf{a} \mid \mathbf{w}_a, \mathbf{w}_c) \tag{2.14}$$

The problem is then reduced to calculating an estimate for the advantage which is formally described by eq. (2.15).

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s}) \tag{2.15}$$

Using the sequences of experiences, this advantage can be estimated using the generalized advantage estimate (GAE) given by eq. (2.16) for sequence length k [37].

$$A(\mathbf{s}_t) = -V(\mathbf{S_t} \mid \mathbf{w}_c) + \sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(\mathbf{s}_{t+k} \mid \mathbf{w}_c) \tag{2.16}$$

The idea behind using the advantage is it more informative that the value, as advantages larger than zero indicate a better than average action, and lower than zero a worse than average action. Calculating the value for both would results in similar values, with only small differences.

Trust Region Policy Optimization (TRPO)
One of the problems within DeepRL has been the stability of the policy updates, which is partially solved using adaptive learn rates. Nevertheless, the step size is till overestimated on occasion which can deteriorate the policy instead of improving performance. The TRPO paper suggests using a trust region, to place an upper bound on updates to prevent too large changes to the policy [36]. This is done using a threshold on the KL Divergent, the dissimilarity between probability distributions. The resulting policy gradient and the equation upper bound the update is subjected to are given by eq. (2.17) and (2.18).

$$\nabla_{\mathbf{w}_a} V(\mathbf{s}) = \nabla_{\mathbf{w}_a} \log \left[\frac{\pi(\mathbf{s} \mid \mathbf{w}_a)}{\pi(\mathbf{s} \mid \mathbf{w}_{a_{old}})} A(\mathbf{s}) \right] \tag{2.17}$$

$$\mathbb{E}\left[KL\left[\pi(\mathbf{s} \mid \mathbf{w}_a), \pi(\mathbf{s} \mid \mathbf{w}_{a_{old}}) \right] \right] \leq \delta \tag{2.18}$$

As imposing an upper limit is computational difficult, another approach is to penalize the KL divergence direly in the policy gradient. Nevertheless, the KL divergence in itself is already a complex calculation. The Proximal Policy Optimization (PPO) paper, based on TRPO, suggest another alternative [38]. Instead of penalizing the KL Divergence, a simple clipping to the probability ratio is applied as shown in eq. (2.19) where ε is a hyper-parameter for which empirical results suggest a value of $\varepsilon = 0.2$.

$$\nabla_{\mathbf{w}_a} V(\mathbf{s}) = \nabla_{\mathbf{w}_a} \log \left[min \left(\frac{\pi(\mathbf{s} \mid \mathbf{w}_a)}{\pi(\mathbf{s} \mid \mathbf{w}_{a_{old}})} A(\mathbf{s}), clip \left(\frac{\pi(\mathbf{s} \mid \mathbf{w}_a)}{\pi(\mathbf{s} \mid \mathbf{w}_{a_{old}})}, 1+\varepsilon, 1-\varepsilon \right) A(\mathbf{s}) \right) \right] \tag{2.19}$$

Both TRPO and PPO currently show the best performance in learning speed and obtained rewards for a variety of Atari 2600 games and MuJoCo [48] environments within the field of DeepRL. PPO slightly outperforms TRPO, with the added benefit of a simpler algorithm and a reduced computational complexity.

2.4.3. Approximate Dynamic Programming

Approximate Dynamic Programming (ADP) uses function approximators to combat the curse of dimensionality inherent to Dynamic Programming. Within this field, two main approaches are identifiable, namely Linear Quadratic Regulator (LQR) Problems and Actor-Critic Designs. The LQR problems rely on linear system dynamics and therefore is not considered for this research. The Actor-Critic Designs opt to use the Actor-Critic architecture, which is a general approach to solve the ADP problems.

The Actor-Critic Designs within the field of Adaptive Optimal Control can be divided in three main categories, Heuristic Dynamic Programming, Dual Heuristic Dynamic Programming and Globalized Dual Heuristic Dynamic Programming [39]. The difference between the categories is based on the function of the critic. The Actor-Critic Designs can in principle use a variety of different function approximators where needed. In practice this however is mostly done using artificial neural networks. As a result, the Actor Critic Designs of ADP have multiple parallels with DeepRL.

Heuristic Dynamic Programming (HDP)

The simplest form of the ACD is Heuristic Dynamic Programming [39], using two function approximators for the actor and critic. The critic estimates the value function, which can be either the state value or state action value function as represented by eq. (2.20). The version using the state action value function is called Action Dependent HDP (ADHDP).

$$\text{Critic} \rightarrow V(\mathbf{s} \mid \mathbf{w}_a) \ \vee \ Q(\mathbf{s}, \mathbf{a} \mid \mathbf{w}_a) \tag{2.20}$$

The critic can be updated by minimizing the temporal difference error. The actor is in turn updated by use of policy gradient, using the gradient obtained from the critic. For a more in dept explanation of the update rules used, see the Preliminary Experiment in chapter 3 which is based on the HDP framework or [39].

Dual Heuristic Dynamic Programming (DHP)

The main difference of Dual Heuristic Dynamic Programming with HDP is the critic output. Whereas the HDP critic approximates the value, the DHP critic predicts the derivative of the value with respect to the state. The critic output of DHP is illustrated by eq. (2.21). The critic again has two versions based on the usage of the state value or state action value. The state action value version is named Action Dependent DHP (ADDHP).

$$\text{Critic} \rightarrow \frac{\partial V(\mathbf{s})}{\partial \mathbf{s}} \ \vee \ \frac{\partial Q(\mathbf{s}, \mathbf{a})}{\partial \mathbf{s}} \tag{2.21}$$

The benefit of DHP is an increase in accuracy of the estimated value derivative and therefore a more accurate policy update[39]. Furthermore, the actor update can directly use the derivative instead of calculating it first. As a result, DHP performs better than HDP for just a slight increase in complexity.

Globalized Dual Heuristic Dynamic Programming (GDHP)

The final category is Globalized Dual Heuristic Dynamic Programming, which combines HDP and DHP. The GDHP critic approximates both the value and the derivative of the value with respect to the state as shown in eq. (2.22).

$$\text{Critic} \rightarrow \left[V(\mathbf{s}) \quad \frac{\partial V(\mathbf{s})}{\partial \mathbf{s}} \right] \ \vee \ \left[Q(\mathbf{s}, \mathbf{a}) \quad \frac{\partial Q(\mathbf{s}, \mathbf{a})}{\partial \mathbf{s}} \right] \tag{2.22}$$

The globalized form tries to combine the best of HDP and DHP and as a result performs slightly better than DHP, which in turn performs better than HDP. Nevertheless, the increase in algorithm complexity is often far larger than the gain in performance. As a result, DHP is often the preferred method.

2.4.4. High Level Reinforcement Learning Concepts

Not all RL frameworks fall into a single well defined category. Some frameworks focus more on high level concepts which are indifferent from the exact update rule used. Two such fields are Model-Dependent Reinforcement Learning and Hierarchical Reinforcement Learning, which are further discussed in sections 2.4.4 and 2.4.4 respectively.

Model-Dependent Reinforcement Learning

In reinforcement learning, the names model-free, model-based and model-depend are used to characterize different methods and their reliance on models. Even in literature the definition of these categories differ. In this report model-free is considered every method which does not use any kind of model, both internal as external. Model-based does make use of models or approximations thereof, but are not dependent on a predefined, accurate and fixed model. Often the model representation is learned or locally approximated during the training of the agent or learned online. The final category is model-dependent. This category relies on accurate predefined model, and does not consider changes to this model.

The reliance on a fixed model results in a lack of adaptability to changing dynamics or model errors. Furthermore, the objective aims to develop a controller which does not rely on a pre-defined model. As a result, this thesis research will consider model-free or model-based approaches only, where model-based with internal models are preferred.

Hierarchical Reinforcement Learning

The idea of Hierarchical RL (HRL) draws inspiration from how humans tend to learn complex tasks. Instead of learning a complex task directly, first the agent is tasked to learn a small subset of the needed behaviour [11]. An example would be if an bipedal robot needs to learn to navigate through a space. It first needs to learn how it can walk and turn. I this case, it could be beneficial to first learn the robot to walk and turn and only afterwards learn to navigate. Using such Hierarchical approaches has proven to possibly accelerate learning [7], or allow for increased safety while exploring [53].

For the purpose of this research HRL is not yet considered. As no reference reinforcement learning controller is available on the PH-LAB model, it is too early to judge the possible worth of HRL for this research project. Instead the project will consider hierarchical approaches as possible enhancements of the baseline framework or for future research.

2.5. Reinforcement learning for flight control

After the overview of the most relevant categories, this section will elaborate on flight control applications of reinforcement learning. The history of reinforcement learning flight control dates back to 1996, where the first longitudinal agent was developed. Over the years, several publications either focussed on the flight control aspect or just used an aerospace vehicle as environment. This section will discuss the most relevant publications that formed reinforcement learning flight control and the conclusions drawn from the research.

2.5.1. Adaptive-Critic-Based Neural Networks for Aircraft Optimal Control (1996)

One of the first publications on a reinforcement learning based flight controller is presented in 1996 [5]. The paper describes using a Adaptive-Critic-Based Neural Networks agent and a Linear Time Invariant (LTI) state space system of a further unspecified longitudinal aircraft model. The agent architecture described is also known as DHP, which is treated in section 2.4.3. The control task of the agent is to maintain the trimmed flight state using a random initialization of the neural networks.

The results of the paper show that the agent is able to converge within several time steps to a stable controller which steers the agent toward the trimmed state. The resulting policy is compared to the initial policy and the optimal policy, from which is concluded that the agent quickly reaches a near-optimal policy.

2.5.2. Helicopter Flight Control using Direct Neural Dynamic Programming (2004)

The next step in flight control using reinforcement learning is presented by Enns & Si (2004) in [39]. The book chapter discusses a Direct Neural Dynamic Programming agent applied to FLYRT, an industrial validated Apache Helicopter model. The chapter discusses three different tasks: stabilization,

reference tracking and flight control reconfiguration, which is based on earlier publications of the same authors [15–17]. The Direct Neural Dynamic Programming agent is architectural the same as ADHDP as explained in section 2.4.3.

The results show ADHDP is capable of controlling complex and high dimensional systems, such as the Apache Helicopter, for stabilization and reference tracking tasks under varying atmospheric wind conditions. Two features were seen as paramount to the success of this research, the pre-trained trim neural network and the cascaded structure of the actor. The results of the reconfiguration task show the ADHDP based agent is able to retain control for a simple failure case, whereas a PID controller is shown to become unstable. It is noted these results do not hold for every failure case and more research is required on this subject. Finally, the authors conclude that this ADHDP approach is a viable candidate for complex MIMO systems, especially suited for online learning and coupled control.

2.5.3. Online Adaptive Critic Flight Control (2004)

The first full 6-degree of freedom aircraft flight control using reinforcement learning is developed in 2004 by Ferrari & Stengel [18]. The environment uses a generic business jet aircraft model [45] and a DHP based agent to control airspeed and attitude angles. The paper opts to use an offline and online learning phase. In the offline phase the agent is trained to incorporate the knowledge from proven gain scheduled linear controllers, while the online phase refines the policy for non-linearities, coupling effects and extreme flight conditions encountered outside the gain scheduled flight regions.

The paper shows results for four test cases, for nominal and large angle manoeuvres, actuator failures, and model parameter variations. The results of the adaptive controller are compared against a fixed agent originating from the offline training phase, which effectively forms a continues linear policy based on the gain scheduling knowledge. The adaptive agent shows better performance for the nominal manoeuvre, actuator failure and model parameter variation. Furthermore, the adaptive agent is able to retain stable control in the large angle manoeuvre, where the fixed agent enters stall and is unable to recover. It is noted that the stall region is not accurately modelled. The results lead to the conclusion that the adaptive DHP based agent is able to form a successful flight control system able to improve performance with respect to the offline controller. The agent learns new information, while retaining baseline performance and is able to take into account unexpected conditions, modelled dynamics, failures and variation in model parameters.

2.5.4. Online adaptive critic flight control using approximated plant dynamics (2006)

The Actor-Critic Designs have two forms frequently used, namely the action independent and action dependent version. In 2006 the practical differences between both versions is researched for the HDP framework [50]. The HDP and ADHDP agents are applied to the longitudinal General Dynamic F-16 aircraft model and designed to control the airspeed and pitch angle.

The results of the comparison show that HDP outperforms ADHDP on a number of aspects. The chance of reaching a successful control policy, adapting to changing plant dynamics and to varying flight conditions are better performed by the HDP framework for this flight control problem. The ADHDP network does however show higher performance when subjected to Gaussian measurement noise, as the HDP framework interprets this as a change in dynamics.

2.5.5. Online Reinforcement Learning Control for Aerospace Systems (2018)

In recent years, incremental forms of control have grown in popularity due to their robustness to changing dynamics and ability to control non-linear systems. In 2018 several incremental version of approximate dynamic programming frameworks have been proposed [57].

Incremental Approximate Dynamic Programming (iADP)

The first method, named Incremental Approximate Dynamic Programming and based on [59], uses an approximated quadratic cost function and local linearised incremental model to directly calculate the control increment. The method is applied to a non-linear aircraft model for both a direct and measured state feedback. The control task of the agent is to reject disturbances.

The results of the method show disturbances are actively rejected by the trained policies. To train the policy, only few iterations are needed as the only trained elements are the cost function kernel and

the incremental dynamics. From the results it is concluded that this method shows promise due to is simplicity and ability to control non-linear systems.

Incremental Actor-Critic Designs (IHDP & IDHP)
The second and third method developed are the incremental versions of Actor-Critic Designs, resulting in Incremental HDP (IHDP) and Incremental DHP (IDHP) based on the papers [58] and [60]. Instead of using a global system model, the methods approximate a local incremental model using recursive least squares. Both methods are implemented on a non-linear aircraft model for the purpose of tracking control.

The results of IHDP show accelerated online learning and higher tracking precision compared to the HDP framework. The same results of accelerated learning and higher tracking performance are observed for IDHP compared to DHP. In addition IDHP is shown to cope with a larger range of initial conditions and is robust to sudden changes in the system model, both at which DHP is unable to converge to the reference signal. Furthermore, the results revealed IDHP is also able to control a new unstable system before the state diverges. The conclusion is drawn that the incremental versions have great potential for general adaptive control tasks where the system dynamics are unknown.

2.6. Conclusion

The goal of this literature survey is to get acquainted with and present the state-of-the art of reinforcement learning relevant for flight control. First the field of reinforcement learning is introduced by means of the history and origin of the field, after which the important basics such as value functions and policies are presented through the Markov Decision Process. A broad overview of the relevant branches of reinforcement learning is discussed to find the state-of-the-art and identify promising methods or concepts applicable to flight control problems.

Deep Reinforcement Learning is an actively researched field, were new methods such as advantage based optimization show promise compared to traditional temporal difference and policy gradient methods. Furthermore, as the field uses complex end-to-end solutions, effort is put into retaining memory and enhancing the learning process in terms of speed and robustness. Examples include experience replay methods and trust region bounded network updates. Approximate Dynamic Programming is and adaptive optimal control based approach. The branch puts emphasis on the control task and how to alleviate the learning process by making smart design choices. The branch provides a variety of basic frameworks which form the foundation of Actor-Critic Designs, also used in Deep Reinforcement Learning. Finally, the survey presents the methods currently implemented to flight control problems. Most of these methods use Actor-Critic Designs and are based on Approximate Dynamic Programming approaches such as HDP and DHP. Recent advances use incremental methods to enhance the performance of the more traditional Actor-Critic Designs, up to the point where the controller can be applied online, without having prior knowledge of the dynamics. Furthermore, these incremental methods show promise for fault-tolerant control and starting with unstable dynamic systems.

For the purpose of this research the Approximate Dynamic Programming solutions such as (Incremental) Dual Heuristic Programming as in [18, 60] can be integrated in a single framework to provide the required performance for an online and and adaptive flight control application due to the low sample complexity. This field can be combined with the novel algorithms originating from DeepRL improve performance or stability of the learning process.

3

Preliminary Analysis

From the literature survey two promising branches were identified for the purpose of this research, namely Deep Reinforcement Learning and Approximate Dynamic Programming. Ultimately the decision to focus on Approximate Dynamic Programming is made as as it closely resembles Deep Reinforcement Learning, but applied from a control theory perspective. In this chapter a preliminary analysis is performed by applying a Heuristic Dynamic Programming framework to a linear short period aircraft model for a simple fight control task. The goal of the preliminary analysis is to test the feasibility of Approximate Dynamic Programming for flight control, gain practical experience and highlight some of the strengths and weaknesses of the method and Approximate Dynamic Programming as a whole.

The chapter starts with an introduction on the used environment in section 3.1, which gives insight in the dynamics, set-up and goal of the environment. Section 3.2 elaborates on the framework used for this experiment, namely Heuristic Dynamic Programming. It details the used state information, actor-critic structure, update rules and finally the pseudo-code algorithm used to perform the experiment. The results of the experiment are shown in section 3.3, describing the performance and stability of the algorithm. Finally, in section 3.4 the conclusion and recommendations are given based on the goals of the preliminary experiment and the influence on the thesis research to be conducted.

3.1. Environment

The controlled environment is based on the OpenAI Gym API [12] and features a custom linear short period model. The short period model consists of two states, the angle of attack α and pitch rate $q \in [-2.0, 2.0]$. The model input is the elevator deflection $\delta_e \in [-0.6, 0.6]$. The state space model is shown in eq. (3.1), with all states and inputs in radians. The initial state of the environment is initialized to $\mathbf{0}$.

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \end{bmatrix} = \begin{pmatrix} -0.7441 & 1.0000 \\ -1.4796 & -1.5773 \end{pmatrix} \begin{bmatrix} \alpha \\ q \end{bmatrix} + \begin{pmatrix} -0.08996 \\ -6.76789 \end{pmatrix} \delta_e \tag{3.1}$$

The goal of the environment is to follow an arbitrary pitch rate reference signal and the performance of the agent is measured over a single episode lasting 120 seconds. For this preliminary experiment the reference signal is based on a sinusoid of fixed frequency and amplitude and is described by eq. (3.2). The phase $\phi \in [0, 2\pi]$ can be used to randomly shift the reference signal between episodes. Furthermore, the reference is defined such that the sampled value is the target value to be attained by the next time step.

$$q_{ref} = 0.175 \sin(0.1\pi\ t + \phi) \tag{3.2}$$

The reward or cost function of the environment is defined as the squared tracking error divided by two as shown by equation (3.3). It is calculated by the environment after advancing the state to the next time step using the state space system in eq. (3.1) and comparing the updated pitch rate with the provided reference.

$$r(s, a) = \tfrac{1}{2}(q - q_{ref})^2 \tag{3.3}$$

3.2. Model-Dependent Heuristic Dynamic Programming

For the agent a Heuristic Dynamic Programming [39] framework is used as a basis. This framework is chosen due to the low complexity and ease of implementation and is well documented in various books and other sources of literature [23, 39, 55]. To further reduce the complexity an action independent and model based solution is used. In practice this means the critic is only a function of the state and the model network is replaced with the model used by the environment.

3.2.1. State augmentation

The state received from the environment is augmented before passing it to the actor and critic function approximators.The augmentation adds the tracking error $\Delta q = q - q_{ref}$ and removes the angle of attack and pitch rate. The addition of the tracking error allows the actor and critic to directly detect the correlation with the cost function. Empirically it is observed that the tracking error alone is sufficient for the framework to function for this particular linear flight control problem. The resulting augmented state is shown by eq. (3.4).

$$\mathbf{s}^{aug} = \begin{bmatrix} \Delta q \end{bmatrix} \tag{3.4}$$

3.2.2. Network structure and update rules

The actor and critic are modelled using artificial neural networks. The Heuristic Dynamic Programming framework is implemented in Python 3 together with the TensorFlow 1.10 [1] library. For this implementation the actor and critic network use the same layout, as shown in figure 3.1. The input layers consists of a single input neuron the match the size of the augmented state vector. A single hidden layer with six neurons and a hyperbolic tangent activation connects the input and output layers. The output layer of the critic consists of a single neuron as the state value is a scalar. The environment uses a single actuator resulting in the actor also having a single output neuron with no activation.

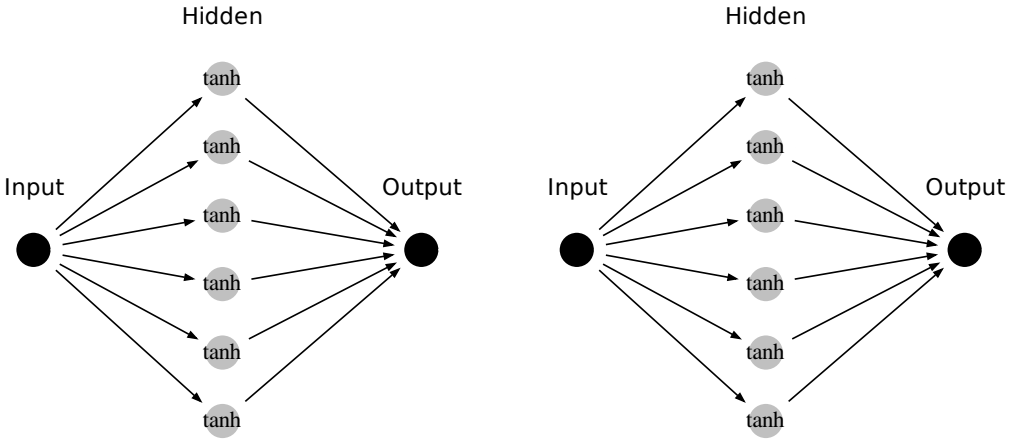


Figure 3.1: Neural network layout of the actor (left) and critic (right).

The actor and critic are updated using gradient decent. Eq. (3.5) shows the general formula for the gradient decent algorithm to update the parameters $\mathbf{w}$ of a function approximator using learn rate α and the gradients of the loss with respect to the parameters.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \frac{\partial E}{\partial \mathbf{w}_t} \tag{3.5}$$

To update the actor and critic network a loss function is defined after which the gradient of this loss function with respect of the parameters is calculated. This preliminary experiment opted to use the Adam optimizer [24] to perform gradient decent step. Adam uses an adaptive and individual learn rate for each parameter to calculate the parameter update for faster learning.

Critic

The critic loss is defined as the mean squared temporal difference error as shown in eq. (3.6). This expression follows the format for standard classification tasks, where $r_{t+1} + \gamma V(\mathbf{s}_{t+1} \mid \mathbf{w}_c)$ serves

as a static label or target value and $V(\mathbf{s}_t \mid \mathbf{w}_c)$ as the network prediction. TensorFlow can directly calculate the gradients and updates as the loss is a function of the network output. For a more elaborate description on the gradient calculation through neural networks see [33].

$$E_c = \frac{1}{2}\left(r_{t+1} + \gamma\, V(\mathbf{s}_{t+1} \mid \mathbf{w}_c) - V(\mathbf{s}_t \mid \mathbf{w}_c)\right)^2 \tag{3.6}$$

Actor
The actor loss is defined by the mean squared error between the current state value and the minimum state value as shown in eq. (3.7). The minimum state value is obtained by following a perfect policy resulting in zero cost and therefore a state value of zero ($V_{min} = 0$) independent of the state.

$$E_a = \frac{1}{2}(V(\mathbf{s} \mid \mathbf{w}_c) - V_{min})^2 \tag{3.7}$$

As the actor loss is not a direct function of the actor output additional steps are required to calculate the gradients. In eq. (3.8) the chain rule is used to expand the gradients as a chain of partial derivatives.

$$\frac{\partial E_a}{\partial \mathbf{w}_a} = \frac{\partial E_a}{\partial V}\frac{\partial V}{\partial \mathbf{s}}\frac{\partial \mathbf{s}}{da}\frac{\partial a}{\partial \mathbf{w}_a} \tag{3.8}$$

The first partial derivative $\frac{\partial E_a}{\partial V}$ is obtained by differentiation of the loss function of eq. (3.7) with respect to the value function. Conveniently this results in the state value prediction of the critic. The partial derivative $\frac{\partial \mathbf{s}}{\partial a}$ is derived by differentiating the state space system of the environment with respect to the action taking into account the augmented state $\mathbf{s}^{aug}$ and time step Δt. The final two partial derivatives $\frac{\partial V}{\partial \mathbf{s}}$ and $\frac{\partial a}{\partial \mathbf{w}_a}$ are obtained by differentiating the critic with respect to the input and the actor network with respect to the weights. This operation is performed by the TensorFlow library using standard back-propagation. The resulting chain of partial derivatives is given by eq. (3.9)

$$\frac{\partial E_a}{\partial \mathbf{w}_a} = V(\mathbf{s} \mid \mathbf{w}_c)\nabla_{\mathbf{s}} V(\mathbf{s} \mid \mathbf{w}_c)\left[-6.77\Delta t\right]\nabla_{\mathbf{w}_a}\pi(\mathbf{s} \mid \mathbf{w}_a) \tag{3.9}$$

3.2.3. Algorithm

With the network structure and update rules in place the framework is implemented using algorithm 3.1. All hyper-parameters needed for the creation of the neural networks are shown in table 3.1. The values presented are empirically obtained, based on often used values from literature. Further performance gains could be achieved by a hyper-parameter search.

Table 3.1: Hyperparameters used in the Heuristic Dynamic Programming implementation.

Hyperparameter	Symbol	Value
Discount factor	γ	0.8
Critic learn rate	α_c	0.01
Actor learn rate	α_a	0.001
Number of update cycles	N_{cycles}	5

Algorithm 3.1 starts by the set-up of the neural networks and environment. The training loop is then started, which first calculates the reference, augmented state and advances the environment to the next time step. The actor and critic are then repeatedly updated each time step using the update rules presented in section 3.2.2. The critic update uses a backward temporal difference scheme, as it is not possible to sample or predict the reference signal in the future which is in turn needed to create the augmented state. The value target is calculated only once per update cycle to prevent numerical instability in the critic network. The chosen update sequence alternates between a critic and actor update. Alternatively it is possible to first perform all critic updates followed by all actor updates within the update cycle. The complete implementation and all source code is hosted on GitHub[1]

[1] GitHub Repository: `http://github.com/dave992/msc-thesis`

3.3. Results and Discussion

The implemented Heuristic Dynamic Programming agent is used to control the short period environment for a total of 100 training runs. To make a fair comparison the reference signal defined by eq. (3.2) uses zero phase shift for all runs. This section is subdivided into two parts. Subsection 3.3.1 focuses on the performance and of a well performing run and compares it with a conventional PID controller. Afterwards, in subsection 3.3.2 the HDP framework stability is investigated.

3.3.1. Performance

The performance of the algorithm can be characterized by several metrics and parameters, such as loss or convergence time. For the purpose of this experiment, the performance is measured by the tracking performance and cost per time step during an episode. The tracking performance allows to inspect the behaviour of the agent, while the cost per time step gives insight in the variable minimized by the algorithm. Both metrics are shown for the best performing agent and a conventional PID controller to compare the results of this method. The best performing agent is selected based on the minimum cumulative cost obtained during the 100 training runs.

Tracking Performance

The tracking performance and commanded deflection of the HDP agent run with lowest cumulative cost is shown in figures 3.2. The HDP agent is able to follow the provided reference signal relatively closely. It is interesting to note that several unstable regions are present, where the agent commands sudden large deflections before quickly recovering. These regions occur frequently after the maxima and minima of the reference pitch rate signal. Furthermore, as the episode progresses the commanded deflection and response become smoother.

The behaviour of this selected run is shared among the other successful runs, which all showcase different degrees of unstable regions after the maxima and minima of the reference signal. The increase in smoothness of the commanded deflection and response is also shown by the other runs. The differentiating factor between this best and other successful runs is the magnitude and length of the unstable responses.

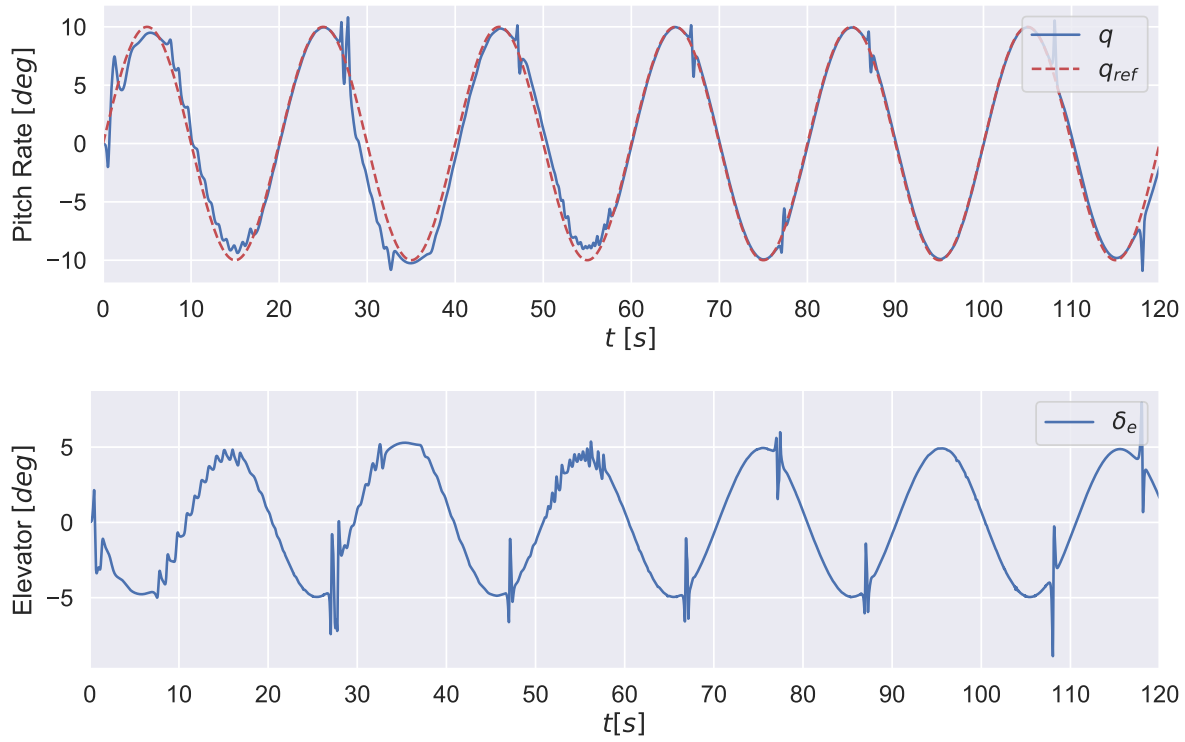


Figure 3.2: Pitch rate tracking performance and commanded deflection of the lowest cumulative cost HPD agent.

In figure 3.3 the tracking performance of a conventional PID controller is shown. The gains of the PID controller are arbitrarily chosen with values of $K_P = -12.0$, $K_I = -1.0$, $K_D = 0.0$. The tracing performance of the PID controller is noticeable better than the HDP agent. As expected it does not display any local unstable regions and the commanded deflection and response is consistent over the complete episode.

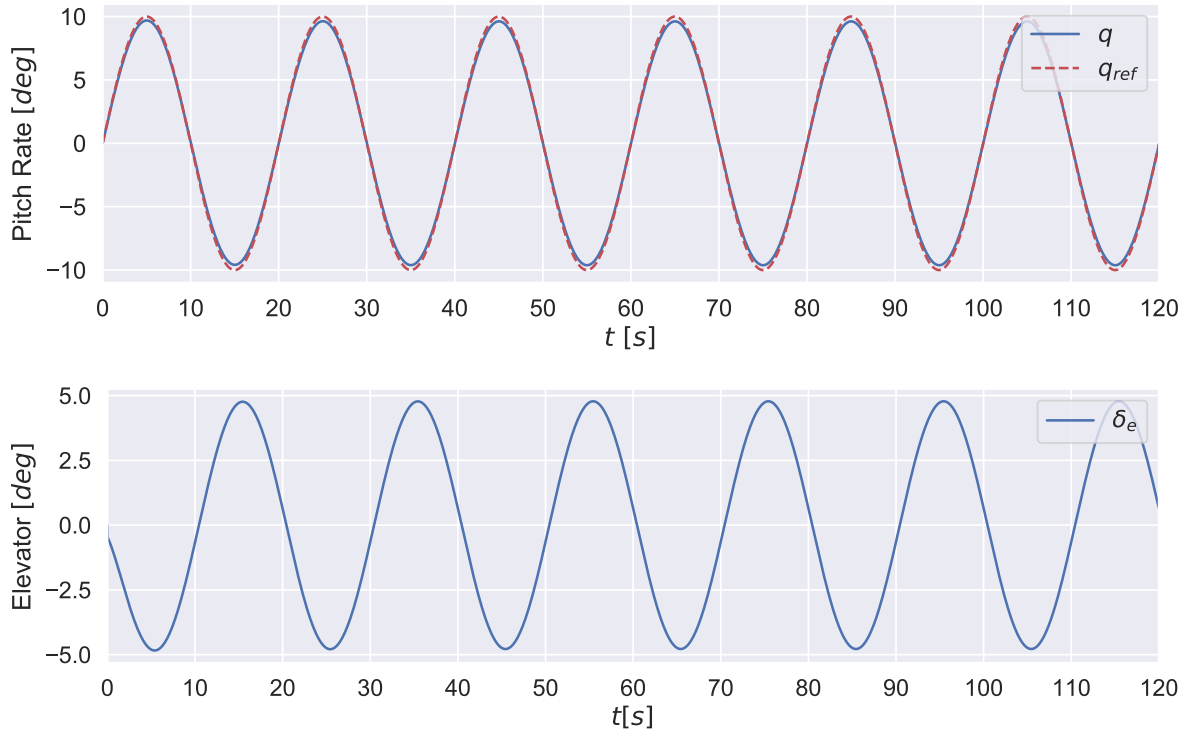


Figure 3.3: Pitch rate tracking performance and commanded deflection of a PID controller.

The comparison between PID and the HDP controller is made to display the tracking performance of what is expected from a pitch rate controller. The comparison is not completely fair, as the HDP controller has to learn and adjust online, whereas the PID controller is arbitrarily tuned beforehand.

Cost Performance

The second performance metric of interest for this preliminary experiment is the cost per time step, the variable which the HDP algorithm minimizes online. In figure 3.4 the cost per time step of the best performing HDP agent is shown, the same run from which the tracking performance is visualized.

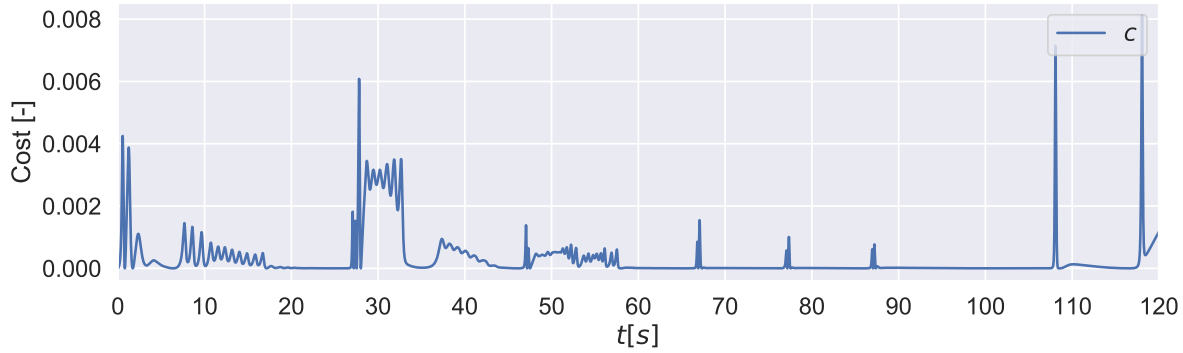


Figure 3.4: Cost per time step of the HDP agent.

At the start of the episode the cost oscillates as the agent learns its control policy. The agent quickly

learns a policy where the cost gets close to the ideal of zero. The unstable regions are clearly visible in the cost, seen as large spikes which sometimes cause a period of higher cost. The increase in smoothness discussed in the Tracking Performance section is also visible from the cost plot, as the spikes up until 100 seconds get less in magnitude and duration.

For comparison the PID controller also produces a cost per time step using the same calculation as for the HDP agent, shown in figure 3.5. The cost figure displays a small transient not visible in the tracking performance plots as the integrator start accumulating. As expected, the error is several magnitudes smaller than the cost of the HDP agent.

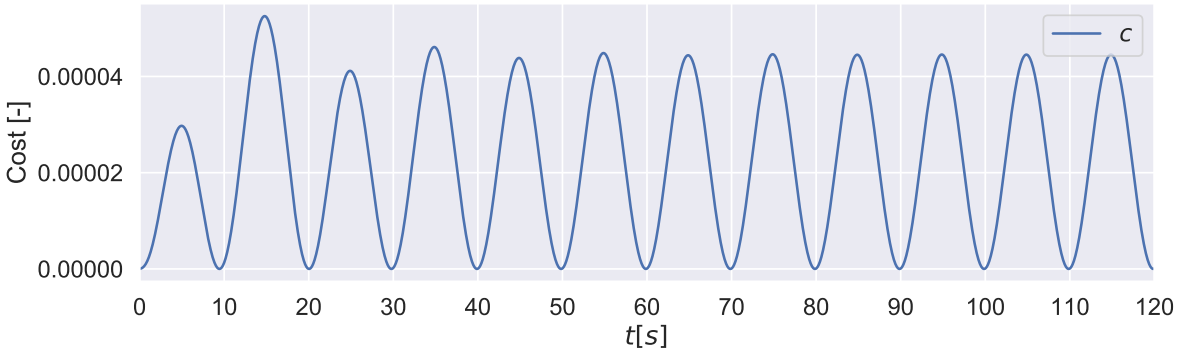


Figure 3.5: Cost per time step of the PID controller.

3.3.2. Algorithm stability

The learning stability or convergence rate is an important aspect of an online learning framework. It is a measure of how often the controller converges to a stable solution starting from a random initialization. Ideally this should be 100% for self learning controllers. The 100 HDP agent runs performed for this preliminary experiment are used to judge the stability of the algorithm. A run is labelled as failed if the cumulative cost of the complete episode is larger than 200. This corresponds to the cumulative cost of a run with a constant pitch rate error of ≈ 7.4 degrees. It should be noted that cost per time step is exponential with respect to the error, resulting in a higher weighting of large errors compared to small errors. To put the failure threshold in perspective, the best run (used for the figures in section 3.3.1) has a cumulative error of 3.604.

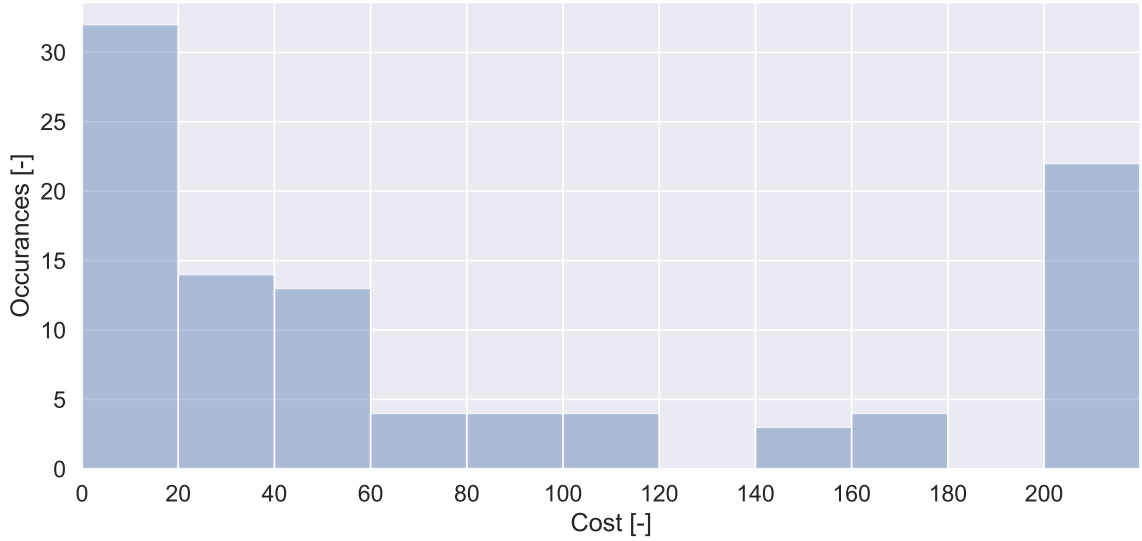


Figure 3.6: Histogram of cumulative cost over 100 runs, using 11 bins in the range $[0, 220]$.

The failure threshold of a cumulative cost higher of equal to 200 resulted in 22 out of the 100 runs to be labelled as failed. To get a better overview of the likelihood of a functioning controller, the cost per run is used to create a histogram. The histogram consists of 11 bins in the range of $[0, 220]$, where all failed runs are clipped to a cumulative cost of 200. The histogram is shown in figure 3.6. The distribution the histogram visualizes confirms the algorithm is likely to converge, with the first bin showing most occurrences. As expected the occurrences of higher cumulative cost quickly decrease, before showing a spike at the final bin. The final spike is due to the clipping of all episodic costs over 200.

3.3.3. Performance of a failed run

The high level stability analysis of the HDP algorithm showed that 22 out of the 100 runs were labelled as failed based on their cumulative cost. This section will show the performance of such a failed run to visualize and explain what happens when an agent run is labelled as failed.

Tracking Performance

The tracking performance of a random failed run, with a cumulative cost of 225.32, is shown in figure 3.7. This run shows typical behaviour observed in the set of failed runs. An initial large unstable response causes the system to diverge from the reference. Interesting is the agent still follows the sine signal at an decreasing offset and eventually at the end of the episode catches the reference signal. The smaller oscillations are typical for failed runs, but do also occur in successful runs within the unstable regions.

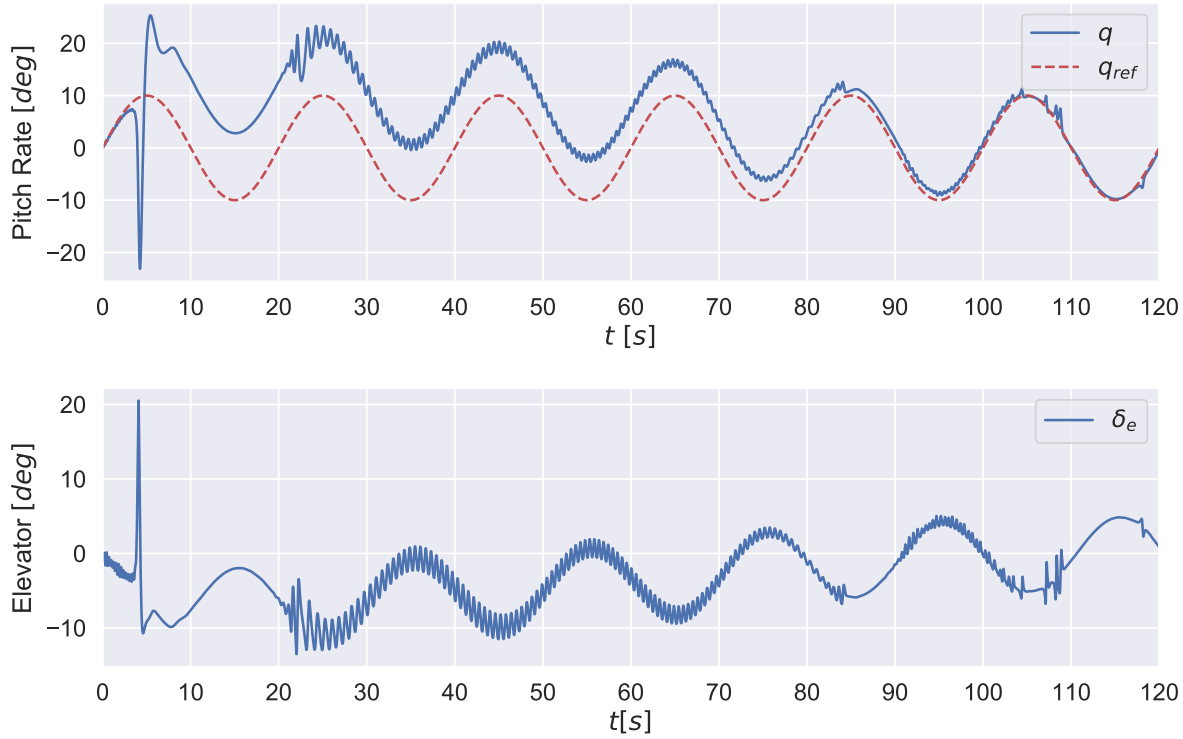


Figure 3.7: Pitch rate tracking performance and commanded deflection of an at random selected failed HDP agent.

Cost

In figure 3.8 the cost per time step of the at random selected failed run is shown. The figure displays a large spike after a short period of following the reference. This large initial spike is a characteristic seen in multiple failed runs. After the cost stabilizes a steady downward trend is visible. Most runs labelled as failed also show a clear downward trend in the cost per time step. Another interesting feature of these failed runs is that often they do follow the shape of the reference signal, but at an offset.

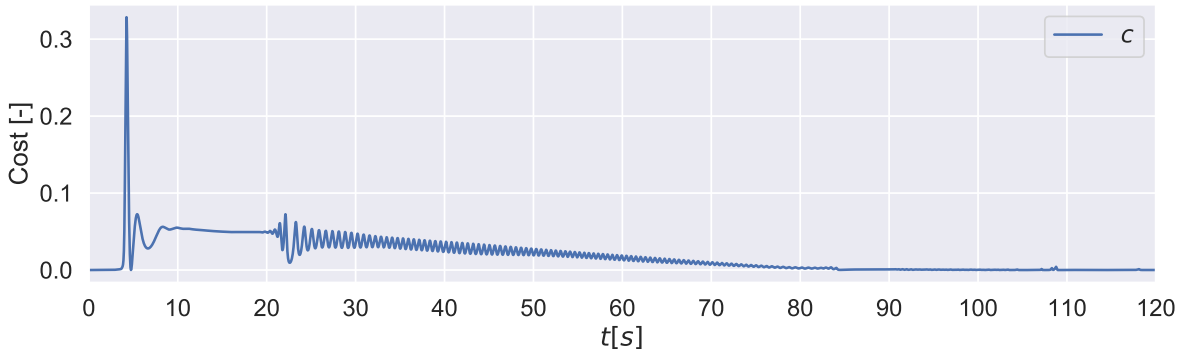


Figure 3.8: Cost per time step of an at random selected failed HDP agent.

3.4. Conclusion

The online reinforcement learning controller based on the HDP algorithm is able to follow a pitch rate reference signal for a two state short period aircraft model starting from a random controller initialization. The tracking performance and cost per time step of the HDP agent is worse than an arbitrary PID controller, which serves as a target for nominal performance. Nevertheless, the experiment fulfils the purpose of a feasibility check using a simple algorithm and aircraft model. A high level stability analysis of the framework showed that 22 of the 100 performed runs were labelled failed based on their cumulative cost not meeting the maximum threshold value of 200. The distribution of cumulative costs over all runs shows a higher probability toward a successful run, with occurrences of higher cost runs quickly lowering past the first bin. The runs labelled as failed still follow the overall shape and magnitude of the reference signal at an offset. The offset is often initiated by a large spike in the commanded elevator deflection from which the agent is not able to recover instantly. Over time the cost does show a clear downward trend, where the cost per time step decreases. Some of the failed runs even manage to intercept the reference signal before the episode terminates.

Based on these results and the literature survey, Dual Heuristic Dynamic Programming is selected as the baseline framework for the online and adaptive flight control of the Cessna Citation 550. Compared to other ADP frameworks such as Heuristic Dynamic Programming and Globalized Dual Heuristic Programming, DHP provides a good balance between algorithm complexity and control performance. This framework is able to control partial-observable systems with continuous state and action spaces as encountered in the Cessna Citation 550 aircraft and the simulation model. Based on the results of the preliminary analysis, the frameworks will provide body rate tracking control, to reduce the learning complexity of the problem and thereby increase the learning speed. The rate controller can in turn be used in a conventional feedback controller to provide a more high level form of control.

Algorithm 3.1 Heuristic Dynamic Programming implementation

Require: γ, $V(\mathbf{s} \mid \mathbf{w}_c)$, $\pi(\mathbf{s} \mid \mathbf{w}_a)$, $\frac{d\mathbf{s}}{da} = [6.77\Delta t]$, $N_{cycles} = 2$

1: Initialize $\mathbf{w}_c \leftarrow \mathbf{w}_{c_{0,0}}$ and $\mathbf{w}_a \leftarrow \mathbf{w}_{a_{0,0}}$
2: $Env \leftarrow$ create_environment()
3: $\mathbf{s}_0 \leftarrow$ reset(Env)
4:
5: # Training for-loop, 1st iteration: $t = 0$
6: $q_{ref,0} \leftarrow$ sample_reference()
7: $\mathbf{s}_0^{aug} \leftarrow$ augment($\mathbf{S}_0, q_{ref,0}$)
8: $\mathbf{a}_0 \leftarrow \pi(\mathbf{s}_0^{aug} \mid \mathbf{W}_{a_{0,0}})$
9: $\mathbf{s}_1, r_1 \leftarrow Env$.step($A_0, q_{ref,0}$)
10:
11: # Missing r_0: Skip parameter update
12:
13: # Update network parameter indexes
14: $\mathbf{w}_{a_{1,0}} \leftarrow \mathbf{w}_{a_{0,0}}$
15: $\mathbf{w}_{c_{1,0}} \leftarrow \mathbf{w}_{c_{0,0}}$
16:
17: # Training for-loop, 2nd iteration: $t = 1$
18: $q_{ref,1} \leftarrow$ sample_reference()
19: $\mathbf{w}_1^{aug} \leftarrow$ augment($\mathbf{w}_1, q_{ref,1}$)
20: $\mathbf{a}_1 \leftarrow \pi(\mathbf{s}_1^{aug} \mid \mathbf{w}_{a_{1,0}})$
21: $\mathbf{s}_2, r_2 \leftarrow Env$.step($\mathbf{a}_1, q_{ref,1}$)
22:
23: $V_{target} \leftarrow r_1 + \gamma\, V(\mathbf{s}_1^{aug} \mid \mathbf{w}_{c_1})$
24: # Update cycle for-loop, 1st iteration of N_{cycles}: $i = 1$
25: $\mathbf{w}_{c_{1,1}} \leftarrow$ update_critic($\mathbf{s}_0^{aug}$, $V_{target} \mid \mathbf{w}_{c_{1,0}}$)
26: $\frac{dE_a}{da} \leftarrow V(\mathbf{s}_1^{aug} \mid \mathbf{w}_{c_{1,1}}) \nabla_{\mathbf{s}} V(\mathbf{s}_1^{aug} \mid \mathbf{w}_{c_{1,1}}) \left[6.77\Delta t\right]$
27: $\mathbf{w}_{a_{1,1}} \leftarrow$ update_actor($\mathbf{s}_1^{aug}$, $\frac{dE}{da} \mid \mathbf{w}_{a_{1,0}}$)
28:
29: # Update cycle for-loop, 2nd iteration of N_{cycles}: $i = 2$
30: $\mathbf{w}_{c_{1,2}} \leftarrow$ update_critic($\mathbf{s}_0^{aug}$, $V_{target} \mid \mathbf{w}_{c_{1,1}}$)
31: $\frac{dE_a}{da} \leftarrow V(\mathbf{s}_1^{aug} \mid \mathbf{w}_{c_{1,2}}) \nabla_{\mathbf{s}} V(\mathbf{w}_1^{aug} \mid \mathbf{w}_{c_{1,2}}) \left[6.77\Delta t\right]$
32: $\mathbf{w}_{a_{1,2}} \leftarrow$ update_actor($\mathbf{S}_1^{aug}$, $\frac{dE}{da} \mid \mathbf{w}_{a_{1,1}}$)
33:
34: # Update network parameter indexes
35: $\mathbf{w}_{a_{2,0}} \leftarrow \mathbf{w}_{a_{1,2}}$
36: $\mathbf{w}_{c_{2,0}} \leftarrow \mathbf{w}_{c_{1,2}}$
37:
38: for t in $range(2, T)$:
39: Repeat logic of code lines 15-34, update indexes accordingly.

4

Conclusion

Rapid advancements of automatic and autonomous control resulted in new applications and with them new challenges related to the uncertain environment they operate in and unforeseen events such as failures or faults. Adaptive and fault-tolerant control techniques can severely reduce the impact of these challenges. This thesis research contributes to the development of on online, adaptive and non-linear continuous reinforcement learning based flight controller for fixed wing aircraft. The formal research objective, first presented in chapter 1, is repeated below. In this chapter the final conclusion is given on the results obtained throughout this research and how they answer the research question and fulfill the research objective in turn.

Research Objective: *The objective of this research is to develop an online, adaptive and non-linear continuous Reinforcement Learning based flight controller for fixed wing aircraft by investigating and implementing a novel Reinforcement Learning framework on the Cessna Citation 550 model and by proposing and investigating possible solutions to improve performance and robustness to random initializations of the agent.*

The research objective resulted in a main and several sub questions. To achieve this scientific contribution the main research question and sub-questions were answered throughout this report and the scientific paper. The conclusion based on these sub-questions is detailed below, before reviewing the main research question and providing a more general conclusion based on the presented work.

Research Question 1: *What is the state-of-the-art of Reinforcement Learning for flight control of fixed-wing aircraft?*
In the literature survey in chapter 2, the current state-of-of-the-art is researched, to determine the current landscape of Reinforcement Learning for flight control and answer the first sub question. The Adaptive Critic Designs, using function approximators for the policy and value estimation, is widely adopted for continuous control problems. This field consists of two large sub-fields, namely Approximate Dynamic Programming (ADP) and Deep Reinforcement Learning (DeepRL). Approximate Dynamic Programming is used for more traditional control tasks, while DeepRL is mostly applied to gaming problems. DeepRL is characterized by the high sample complexity, often resulting from the high level goals such as winning or reaching a specific way-point. This makes the approaches less suited for online purposes, where execution time is important. DeepRL features a more diverse landscape of algorithms utilizing multiple competing agents, stochastic policies and trust region bounded agent updates. Approximate Dynamic Programming often relies on a form of model information to update the agent, but as a result has a reduced sample complexity and is fit for use in online applications. The algorithms are based on the same fundamental basis, namely Heuristic Dynamic Programming (HDP) and Dual Heuristic Dynamic Programming (DHP).

For the purpose of this research the Approximate Dynamic Programming solutions such as (Incremental) Dual Heuristic Programming as in [18, 60] can be integrated in a single framework to provide the required performance for an online and and adaptive flight control application due to the low sample complexity. This field can be combined with the novel algorithms originating from DeepRL improve performance or stability of the learning process.

Research Question 2: *What is the proposed baseline framework for adaptive and online Reinforcement Learning flight control on the PH-LAB research aircraft model?*

(a) *What are the characteristics of the PH-LAB model and is this environment fully observable?*

(b) *What type of control should the framework provide?*

(c) *What Reinforcement Learning framework is best suited for the purpose of this research and how does this compare against other promising methods?*

Dual Heuristic Dynamic Programming is selected as the baseline framework for the online and adaptive flight control of the Cessna Citation 550 based on the results of chapter 2 and 3. Compared to other ADP frameworks such as Heuristic Dynamic Programming and Globalized Dual Heuristic Programming, DHP provides a good balance between algorithm complexity and control performance. This framework is able to control partial-observable systems with continuous state and action spaces as encountered in the Cessna Citation 550 aircraft and the simulation model. Based on the results of the preliminary analysis, the frameworks will provide body rate tracking control, to reduce the learning complexity of the problem and thereby increase the learning speed. The rate controller can in turn be used in a conventional feedback controller to provide a more high level form of control.

Research Question 3: *How can the robustness to random agent initialization and tracking performance be improved for this framework?*
One of the failure causes in reinforcement learning is numerical instability resulting from the temporal difference calculations. In DeepRL slow moving target networks are seen to reduce instability by replacing the target value. For the agent framework a target critic is implemented. An experiment with and without the target network is performed to capture the influence of the target on the robustness to random initializations of the agent. The effect of the initialization of the agent itself on the robustness is researched by adjusting the standard deviation used to initialize the artificial neural network parameters. Limited system knowledge is expected to be an effective method to increase the tracking performance of the framework. The local system gradients govern the direction of the agent updates and during the experiment will be replaced with a fixed set of gradients obtained from linearizing the Cessna Citation 550 around flight condition FC3. The gradients are kept fixed throughout the episode to investigate the effect of errors on the performance.

Research Question 4: *How do the proposed methods compare to the baseline framework with respect to adaptability, online learning and tracking performance?*
The results presented in the scientific paper in part I show two methods proposed to improve the robustness and performance of the agent. The introduction of a target critic network is shown to significantly improve the robustness of the agents to failure by preventing numerical instabilities in the temporal difference calculation. The added critic target network did not influence the tracking performance of the agent significantly in the experiment with comparable reward distributions for both cases. Furthermore, the online learning and adaptability also did change with the addition of the critic target network. The second method proposed is including limited model knowledge in the form of fixed model gradients obtained from linearizing the Cessna Citation 550 around flight condition FC3. The policy is still randomly initialized. The model information increased the robustness of the controller with only a single failed episode in the complete experiment case. The performance drastically increased with significant improved rewards and less variance. The model information improves the adaptability and online learning of the agent, by providing a more stable and correct optimization direction for the actor network update. The initialization of the agent parameters greatly influences the robustness of the framework. Increasing the standard deviation of the truncated normal distribution results in all episodes failing within seconds of the episode starting. The networks become oversensitive to updates, which immediately saturate the control output of the agent. Decreasing the standard deviation results in agents being unable to learn for prolonged periods of time.

Main Research Question: *How can a novel online and continuous Reinforcement Learning framework be implemented on the Cessna Citation 550 model as a proof of concept for the purpose of adaptive fight control and be further improved to enhance performance and increase robustness to random agent initializations compared to the baseline framework?*

The scientific paper presents the answer to the main research question and uses the information gathered throughout this research and as a result completes the research objective introduced in the introduction in chapter 1. This thesis contributed to the development of an online adaptive and non-linear continuous reinforcement learning based flight controller for fixed wing aircraft. A Dual Heuristic Dynamic Programming framework is implemented and analyzed on the simulated Cessna Citation 550 aircraft. The framework is demonstrated to learn a near optimal control policy starting from a random initialized policy at different flight conditions. The performance is greatly improved by adding limited model information in the form of local system derivatives. Utilizing Deep Reinforcement Learning techniques in the form of target networks increased the learning robustness and numerical stability of the proposed framework. Additional research is needed to advance the proposed framework from the simulated Cessna Citation 550 to actual flight test on the PH-LAB research aircraft of Delft University of Technology. However, this research has demonstrated the feasibility and capabilities of online and adaptive reinforcement learning with no prior model knowledge for body rate control and sideslip regulation on a fixed-wing CS-25 aircraft.

Reinforcement learning is shown to be a feasible candidate for online adaptive flight control of a CS-25 aircraft and should be further researched by means of flight tests on the PH-LAB aircraft. Additional research is needed to improve the robustness of the framework and reduce the reliance on the online identification of the system model. In the future, reinforcement learning will be able to learn and adapt online to changing environments and provide an effective alternative for fault tolerant flight controllers, reducing the impact of failures and increasing the overall safety of automatic and autonomous controlled aerospace vehicles.

5

Recommendations for future work

As a new application of reinforcement learning, additional research is required to advance RL based flight controllers to industry applications. The following areas of future work are identified as instrumental in advancing the field of reinforcement learning for flight control based on the conclusions and insights obtained during this research:

- In this research a first step toward a model-independent reinforcement learning flight controller is made. The focus was on demonstrating the viability of the approach, resulting in three important simplifying assumptions which are not true in flight tests. First, the aircraft system state is assumed to be measurable and observable. The aerodynamic angles can violate this assumption, based on the available sensors and configuration of the PH-LAB research aircraft. Second, the sensor dynamics and signal delays are neglected. The delay may cause system identification problems if not synchronized properly. The effect of sensor dynamics is expected to be small as effective methods such as Kalman-filters are available to improve the measurements. Furthermore, the RLS model estimation acts as an additional filter. This assumption will not always hold. Finally, the aircraft is assumed to fly in undisturbed air, using the turbulence model part of the simulation model can produce different results. Further research should be completed to investigate the effect of these assumptions on the proposed framework.

- The implemented framework uses a build-in airspeed controller, which is not present in the PH-LAB. In principle this functionality can be provided by the agent too. The slower engine dynamics can be a challenge for the agent to correctly capture. Additional research is needed to implement speed control through the agent and evaluate the effects.

- To produce the results of this research a fixed length episode of five minutes is used. In reality, the episode length will vary based on the duration of the flight. Continued learning can cause instability in the neural networks as the parameters can keep growing over time as it tries to minimize tracking errors. Possible solutions can be L2-regulation, parameter clipping as in [60] or penalizing large commanded actuator rates. Furthermore, the continued learning can negatively impact the RLS system model if an insufficient amount of excitation is present in the flight profile. This can result in covariance wind-up. Additional research is needed to find the effect and solutions of continual learning on the proposed framework.

- In this research the actor internally decouples the longitudinal and lateral controls. As the agent starts from a random initialization, this aids the controller by preventing interference of decoupled states on the commanded actions. Additional research is needed to compare the effects of the decoupling against a fully connected actor and investigate other solutions which mitigate the unwanted interference during forming of the initial policy.

- The reward signal only communicates the primary objective in the form of reference tracking to the agent. As a result the agent is unaware of secondary objectives and limitations of both the controller and the controlled system. For example, the agent is unaware of flight envelope limits or

passenger comfort. Future research can implement more complex reward functions, which take these limitations and secondary objectives into account and result in a more global form of optimal control. Another advantage of more advanced reward functions is an increase in robustness of agents, as the policy in no longer steered toward unobtainable objectives resulting in large errors and blow-up of the actor-critic.

- Additional research is needed to evaluate the used RLS system model and alternative online system identification methods. The results indicate the linear model gradients do not have to be perfect to still have near optimal control. Instead the sign and relative magnitude are important. Other system identification methods or combinations thereof may be more suited for this approach or can be developed to tailor to the specific needs of this implementation.
- The results showed the framework is not fully robust to the random initializations of the actor-critic, with an increase in failed episodes as the aerodynamic damping decreased. One of the causes was identified to be the outer loop PID controller, which did not provide any damping on the body rate references. The agent can also directly control these higher levels dynamics at the cost of learning speed. Additional research should be conducted to mitigate the learn speed decrease by using higher levels of control. A possible solution is to use a cascaded actor structure as in [18].
- Most machine learning applications benefit from adaptive learn rates to obtain better results and faster convergence. Additional research should be conducted to develop adaptive learn rate strategies. A possibility is to implement trust region based updates such as [36, 38] and adapt this for deterministic policies.

A

Additional Results

The results summarized in the scientific paper in part I show the most relevant results only. In this appendix a closer look at the results are presented to get a more complete overview of all results obtained during this thesis research. In section A.1 additional episodes of the agent are shown for all flight conditions. Furthermore, the return contribution per tracked state is given to show the influence of each state on the return. Finally, in section A.2 the failed episodes are explained to get a better understanding of how and why a subset of the episodes fail.

A.1. Experiment Cases

In part I the experiment details four cases, which are repeated in table A.1. The performance results of all cases are visualized for the respective cases in section A.1.1, A.1.2 and A.1.3. Case D is not shown, as it did not result in any successful episodes.

Table A.1: The experiment cases as used in part I.

Experiment Case	Target Critic	Fixed Model Gradients	Network Initialization[1]
A	Yes	No	$\sigma = 10^{-2}$
B	No	No	$\sigma = 10^{-2}$
C	Yes	Yes	$\sigma = 10^{-2}$
D	Yes	No	$\sigma = 10^{-1}$
E	Yes	No	$\sigma = 10^{-3}$

[1] Truncated normal distribution $N_{trunc}\left(\mu, \sigma^2\right)$, with $\mu = 0.0$

A.1.1. Case A: proposed adaptive critic design

Experiment case A uses the proposed adaptive critic design for online learning without any prior model knowledge. For each of the four initial flight condition a random episode is selected and illustrated in figures A.2, A.3, A.4 and A.5 for the maneuver phase.

The initial oscillations seen in all flight conditions are the result of the recovering of the forced excitation phase and the agent still learning to improve its policy. For flight condition FC0 and FC1 the oscillation in pitch and pitch rate is insufficiently damped and as a result even visible in the altitude profile. For FC1 the insufficient damping is also visible in the bank angle. In FC1, FC2 and FC3 the dynamic response of the pitch rate and roll rate is shown to not fully reach the desired reference signal. This is a result of errors in the prediction of the next state by the Recursive Least Square system model. The update gradients are small as the predicted state is closer to the desired reference than the actual obtained next state, highlighting the importance of the online system identification model. Finally, in the results of FC0 and FC2 the effect of the aggressive learn rates is visible in the roll rate plots. High frequency oscillations around the desired roll rate are the result of the actor network updates overshooting the intended optimum. The high frequency oscillations are less visible in the dynamic response of the

environment due to the actuator dynamics filtering out the high frequency content of the commanded actuator deflection. This behavior is critical in the observed behavior in failed episodes and is further detailed in section A.2.

As in part I, the sum of rewards over the episode is taken as a measure of performance as it is directly proportional to the tracking errors. This quantity is also known as the return. To complete the additional results on experiment case A, the performance of all episodes is visualized in figure A.1 per tracked state.

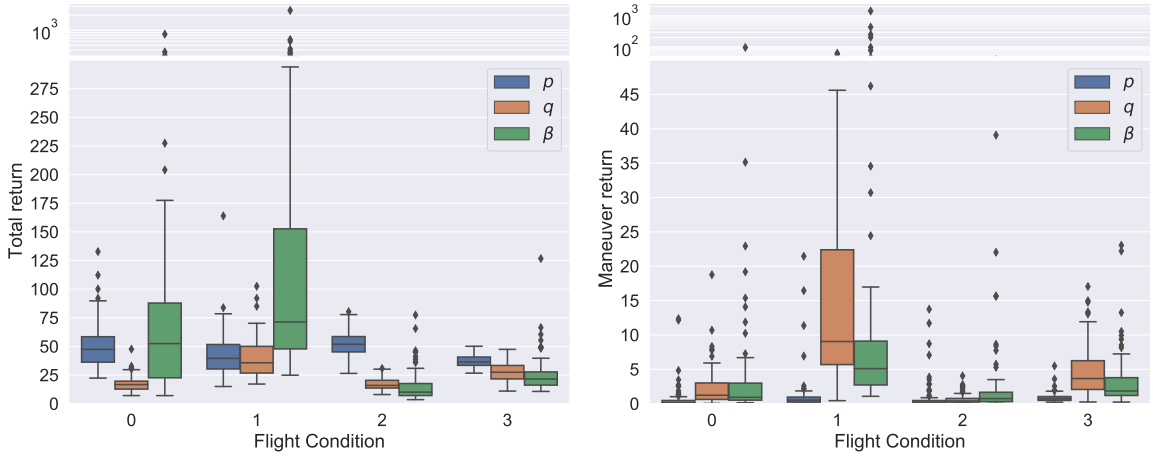


Figure A.1: The distribution of the returns per tracked state for experiment case A. The return is visualized for all flight conditions and for both the complete episode and the maneuver phase only.

The total return and maneuver return show a clear difference in the contribution per tracked state. From this figure a general pattern is observed related to the largest contributing state to the total return. The relative largest contributor in the total return is no longer the largest contributor in the maneuver return. The agent learns by experience and as a result of errors made in the past. The most contributing state is therefore most likely to be reduced, which is visible in the distribution of returns.

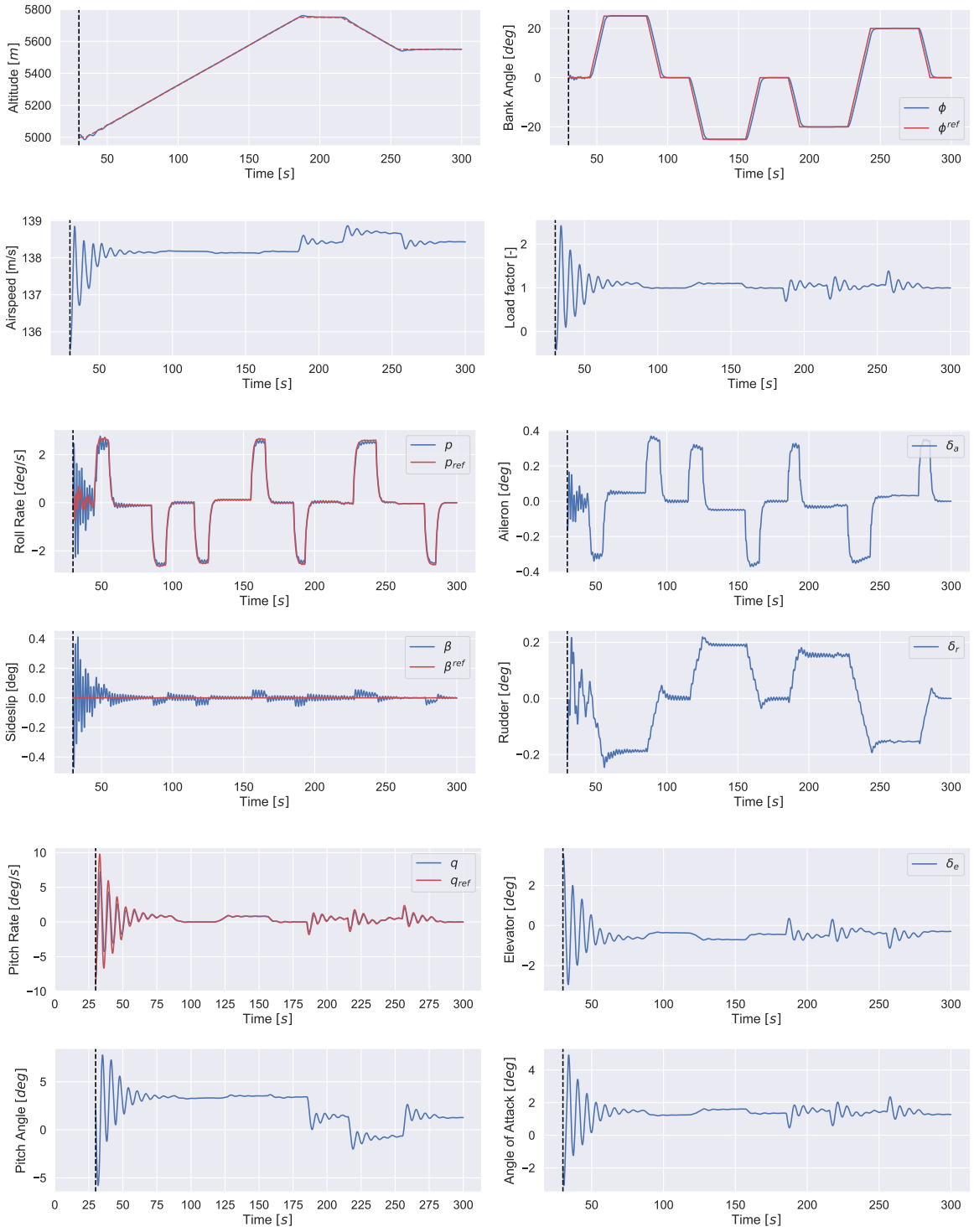


Figure A.2: Case A with flight condition FC0 at $h_e = 5000\ m$ and $V_{TAS} = 140\ m/s$.

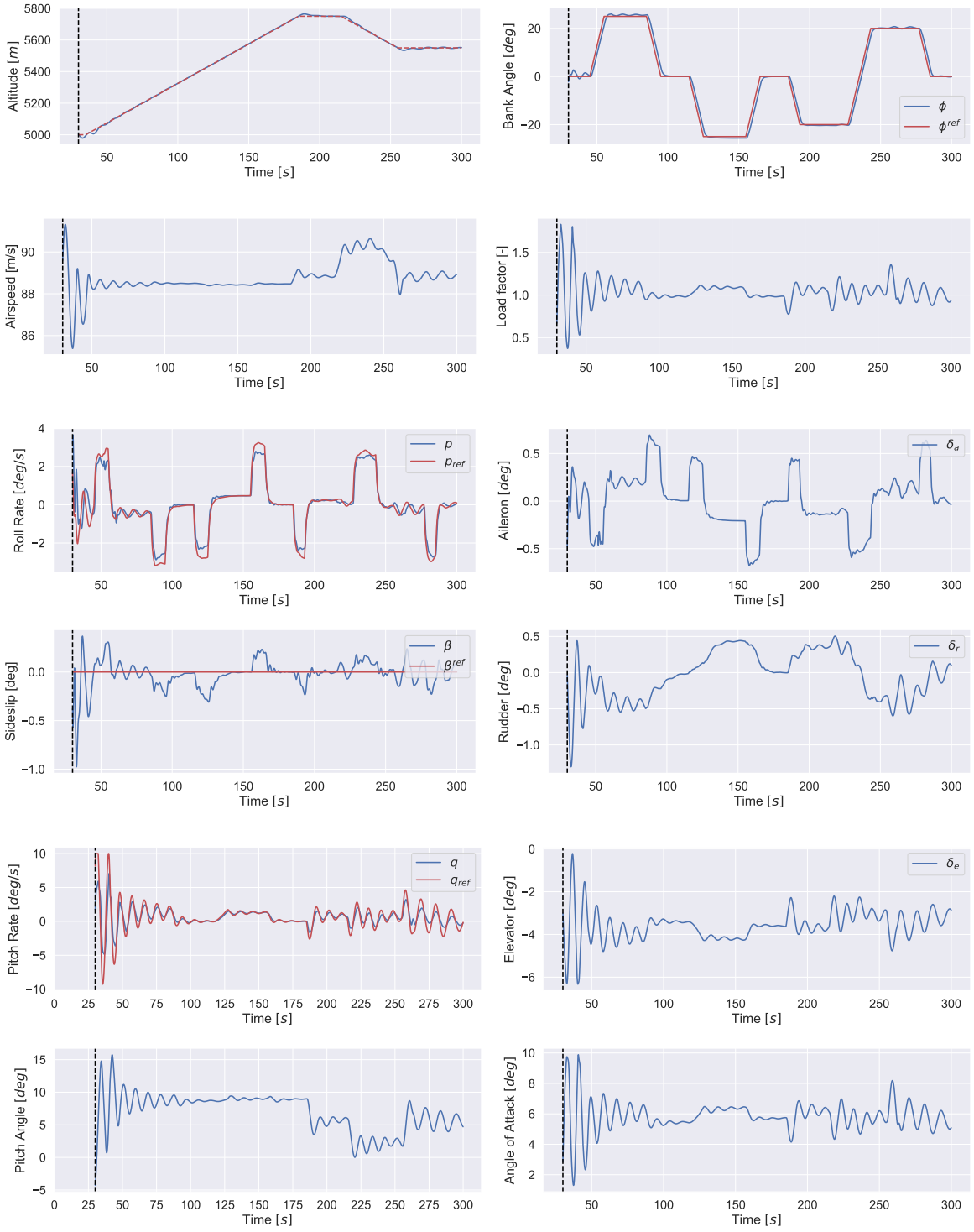


Figure A.3: Case A with flight condition FC1 at $h_e = 5000\ m$ and $V_{TAS} = 90\ m/s$.

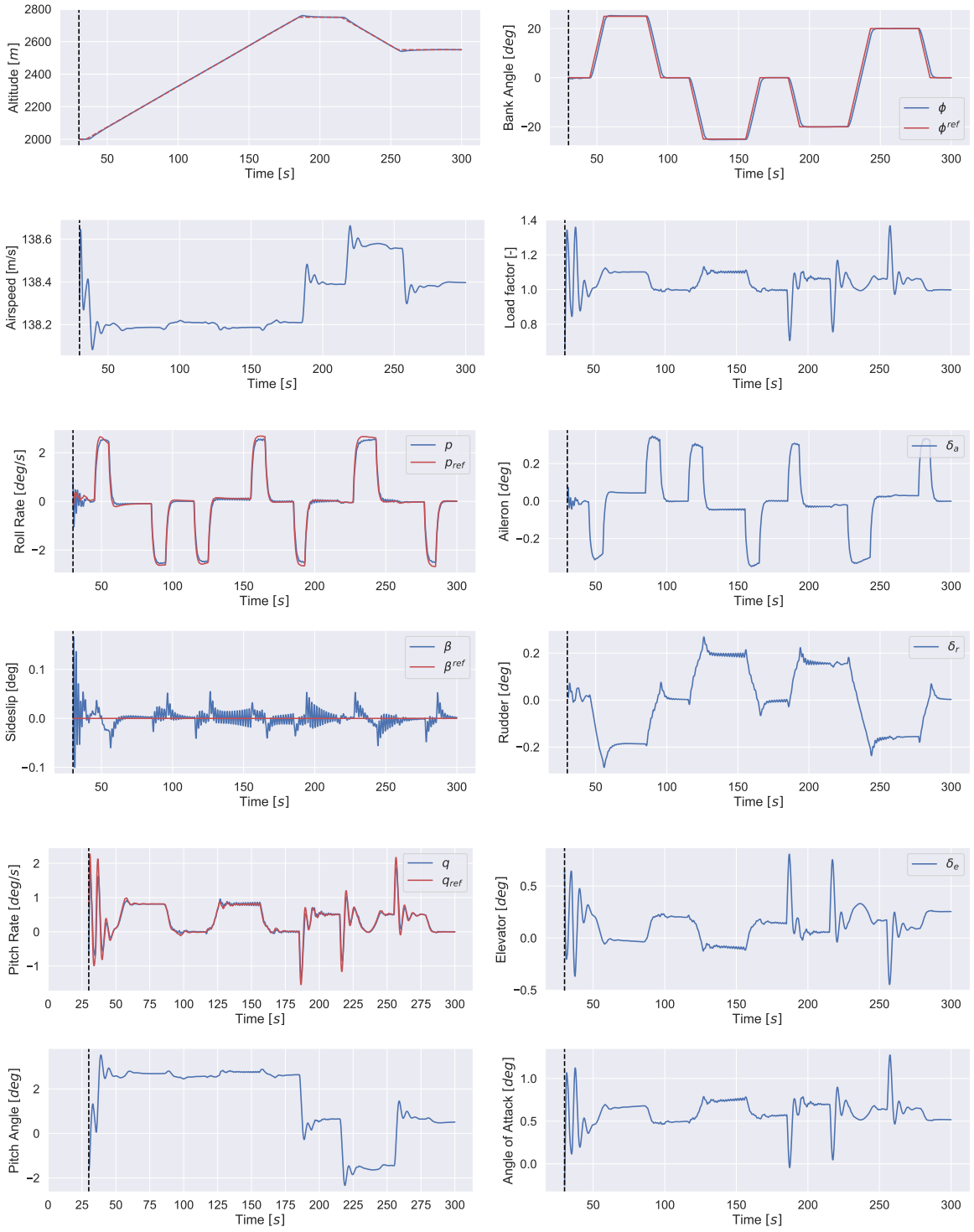


Figure A.4: Case A with flight condition FC2 at $h_e = 2000\ m$ and $V_{TAS} = 140\ m/s$.

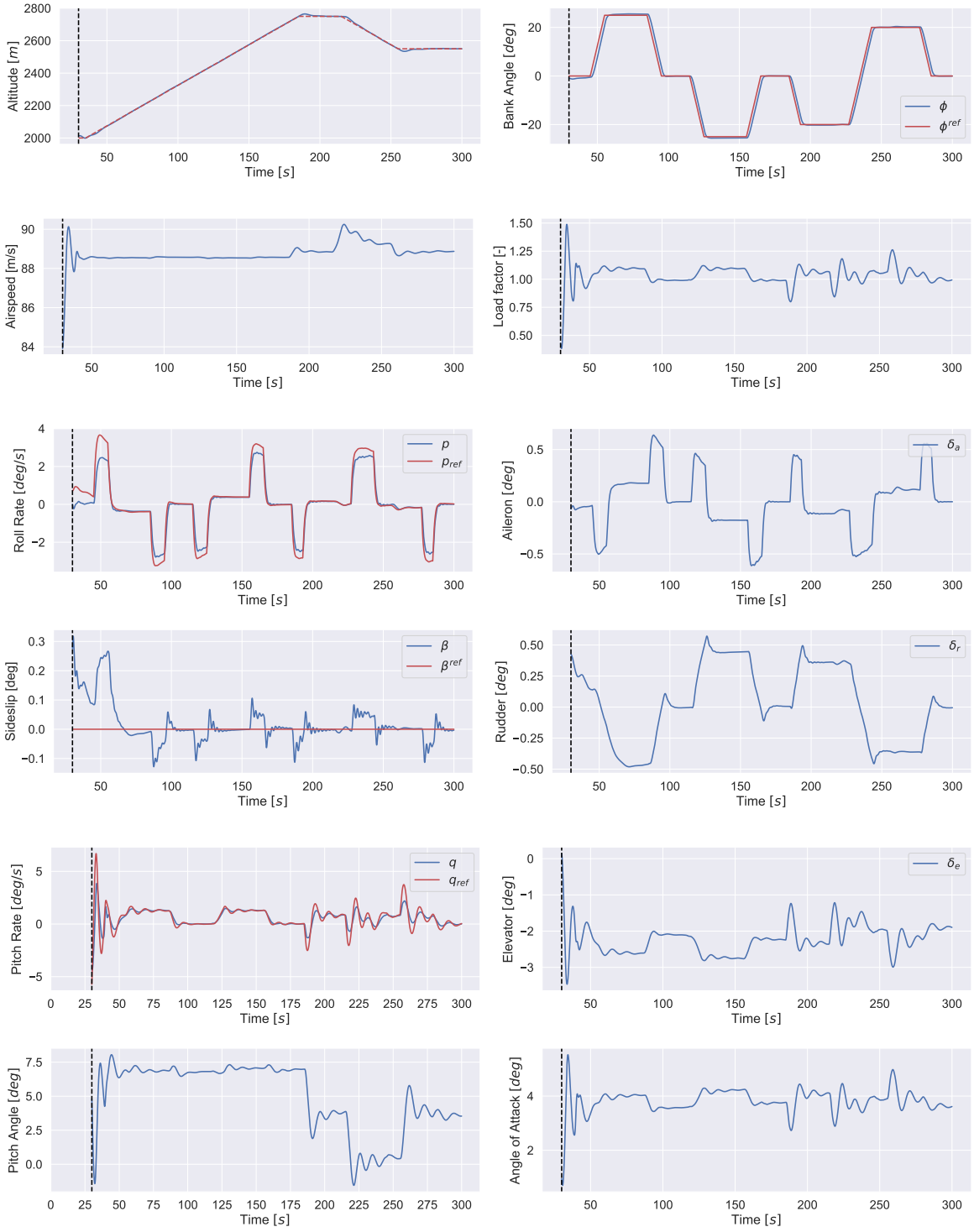


Figure A.5: Case A with flight condition FC3 at $h_e = 2000\ m$ and $V_{TAS} = 90\ m/s$.

A.1.2. Case B: without target critic

The agent without the target network performs comparable to the agents of experiment case A. For each flight condition the maneuver phase is again visualized in figure A.7, A.8, A.9 and A.10. The most visible and significant difference between case A and B is the increased amplitude, duration and occurrences of the high frequency oscillations in the roll rate and sideslip angle, which is also observed in the failed episodes detailed in section A.2. The target critic network lessens the impact and occurrences of the numerical instabilities causing the osculations, in turn increasing the robustness of the controller to random initializations.

In figure A.6 the distribution of returns for experiment case B is shown. The return distributions are not significantly different compared to the results of experiment case A.

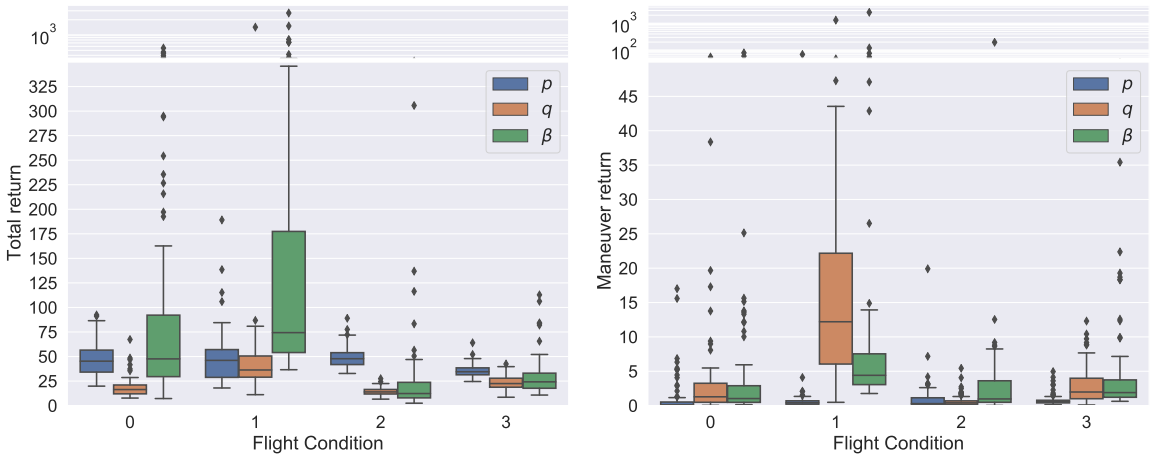


Figure A.6: The distribution of the returns per tracked state for experiment case B. The return is visualized for all flight conditions and for both the complete episode and the maneuver phase only.

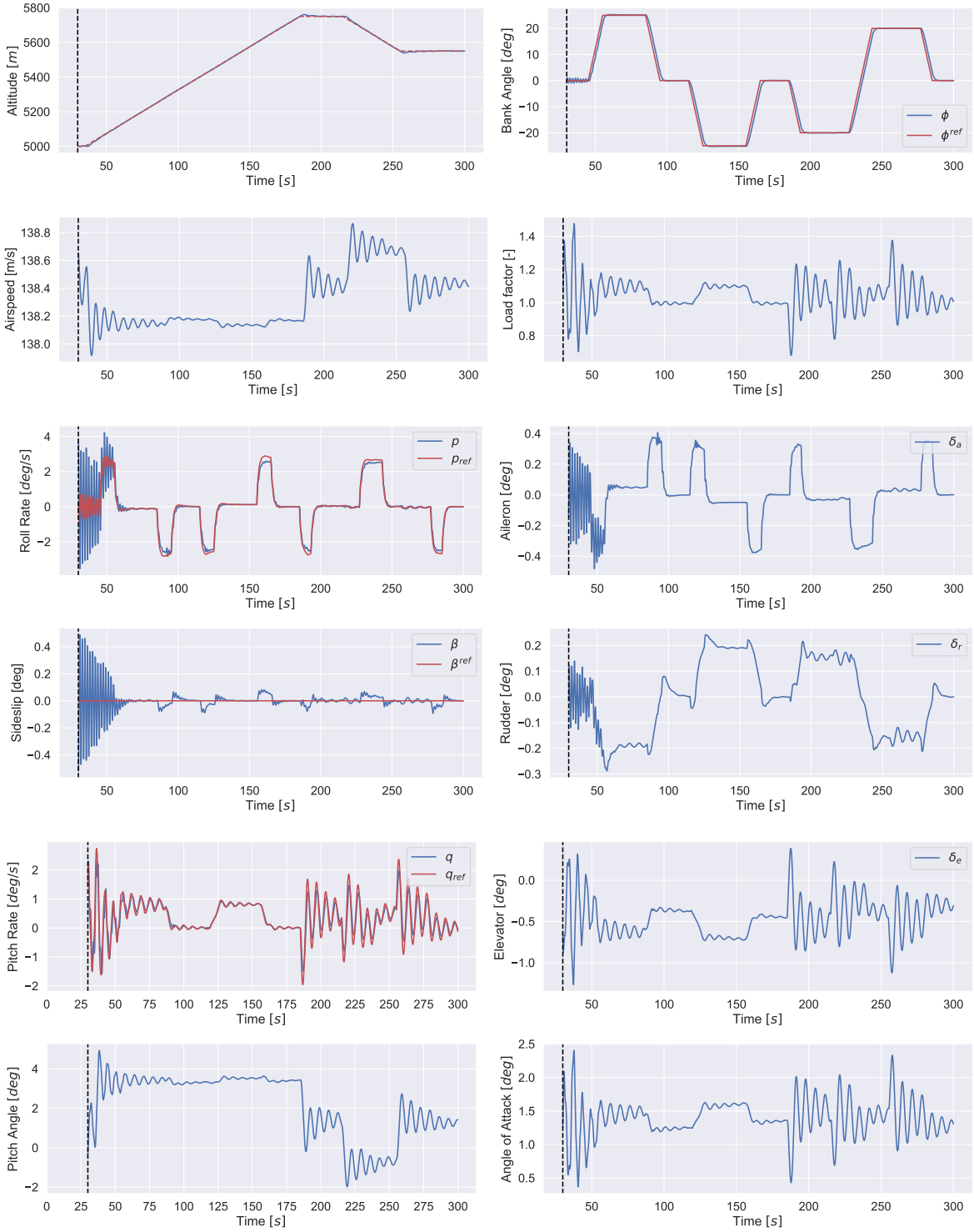


Figure A.7: Case B with flight condition 0 at $h_e = 5000\ m$ and $V_{TAS} = 140\ m/s$.

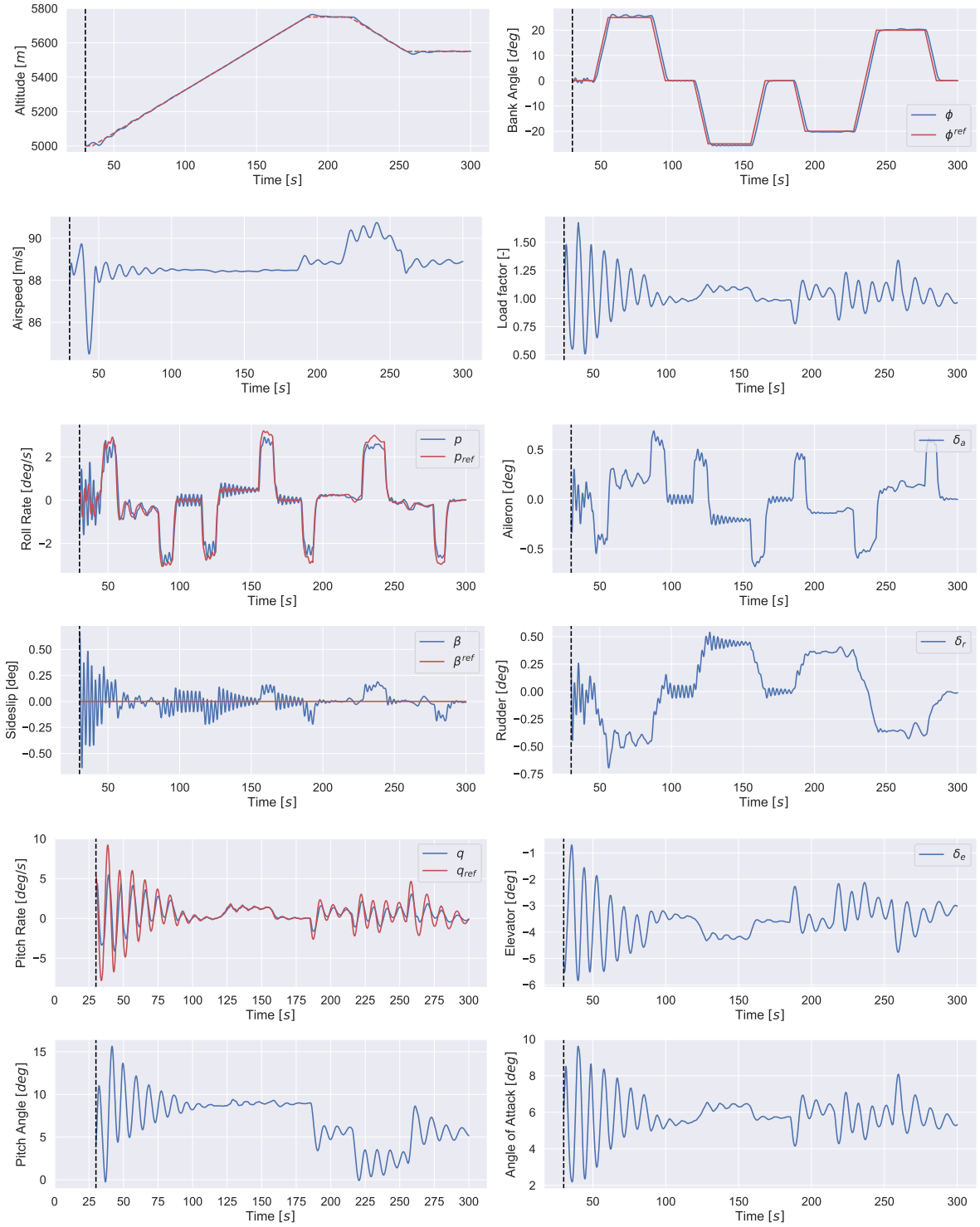


Figure A.8: Case B with flight condition 1 at $h_e = 5000\ m$ and $V_{TAS} = 90\ m/s$.

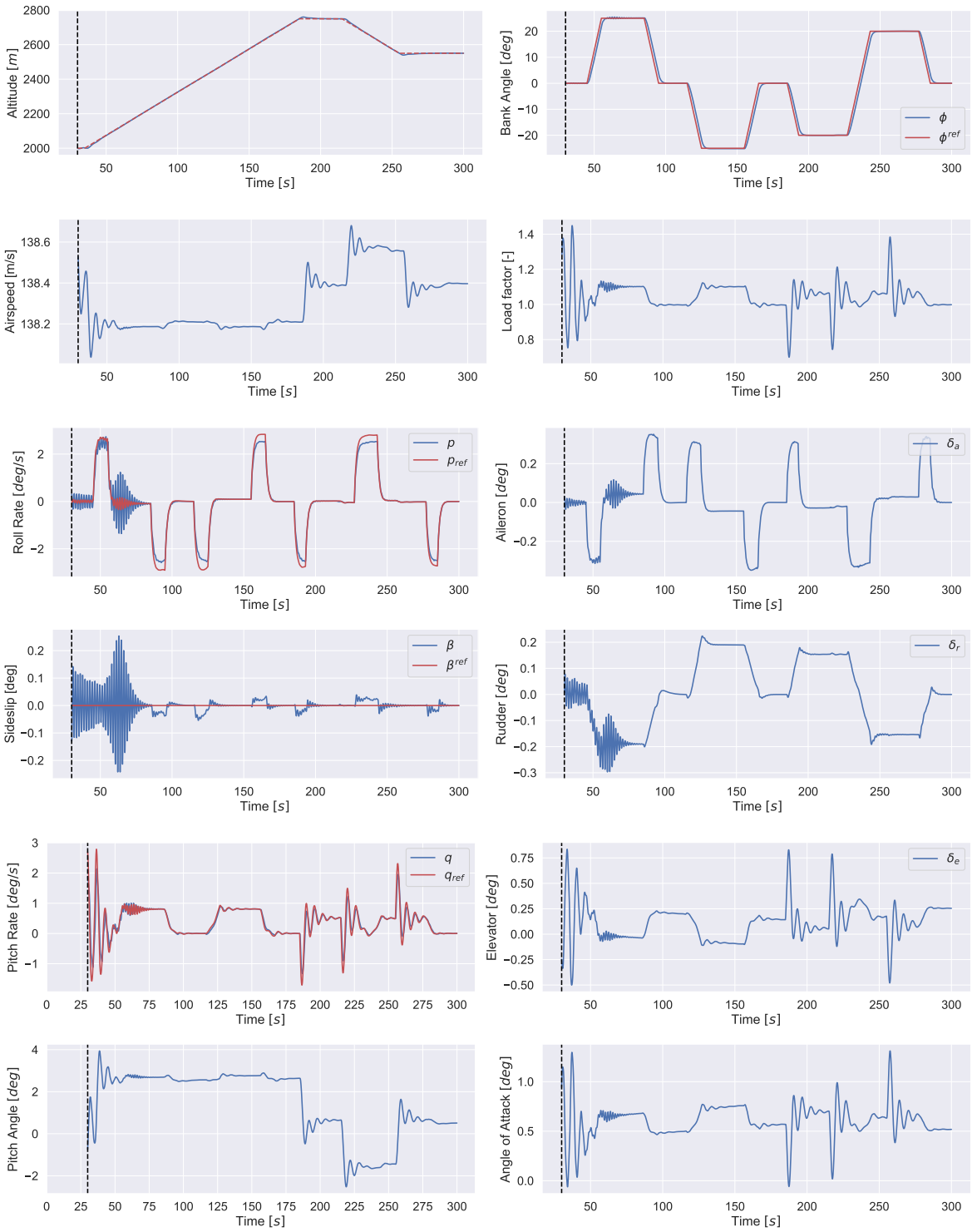


Figure A.9: Case B with flight condition 2 at $h_e = 2000\ m$ and $V_{TAS} = 140\ m/s$.

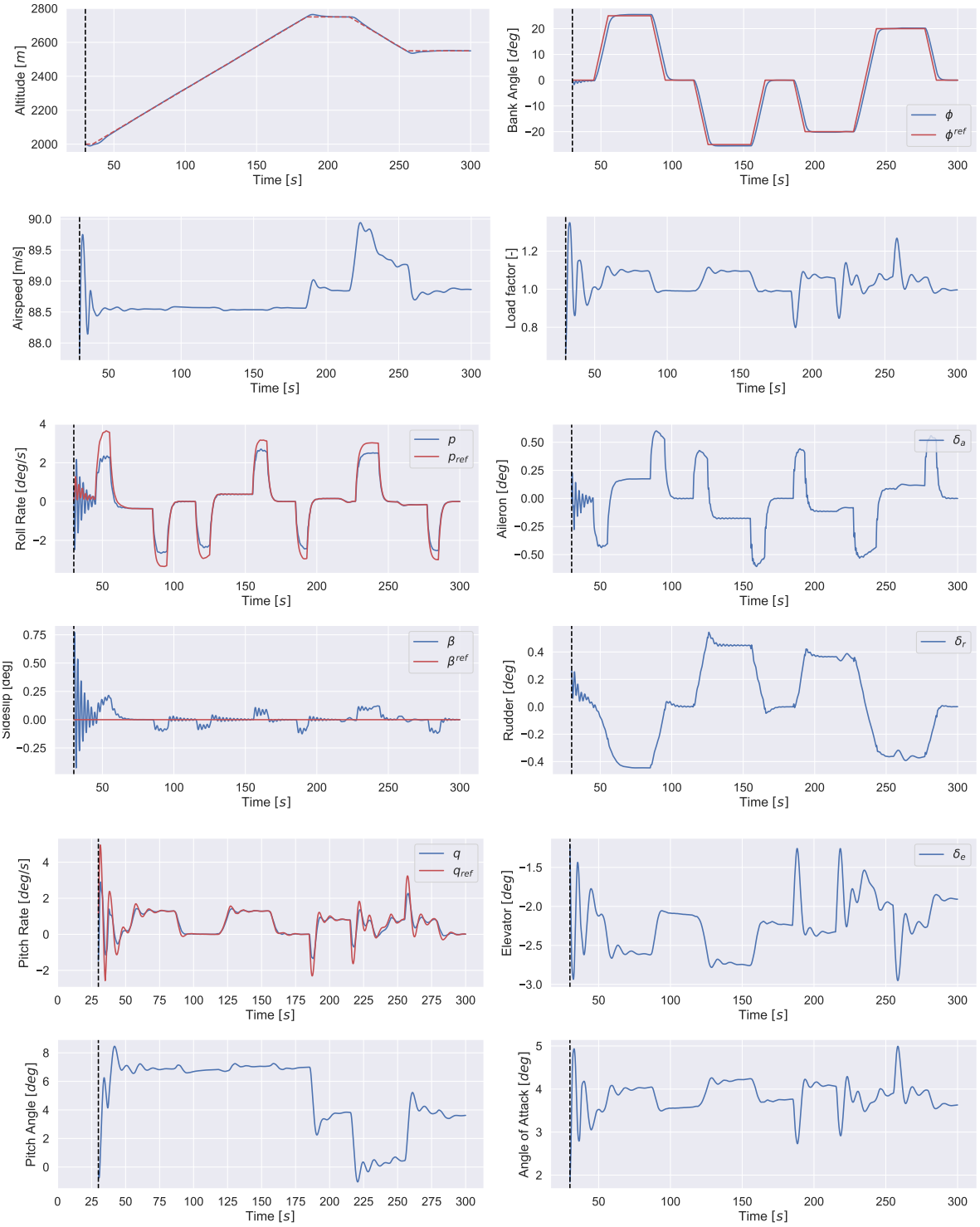


Figure A.10: Case B with flight condition 3 at $h_e = 2000\ m$ and $V_{TAS} = 90\ m/s$.

A.1.3. Case C: with fixed model gradients

In experiment case C the system model gradients $\mathbf{F}$ and $\mathbf{G}$ are replaced with a fixed set of gradients obtained by linearizing the Cessna Citation around flight condition FC3. The fixed model gradients replaces part of the function of the RLS system model and is done to show the effect of prior system knowledge. Even for flight condition FC3, the aircraft does not maintain the linearization point during the flight profile. As a result the fixed model gradients contain an error with respect to the current gradients. This error is larger for flight conditions further away from the linearization point, most notably FC0.

In figures A.12, A.13, A.14 and A.15 random episodes for all four flight conditions of experiment case C are shown. As can be seen from the figures, the tracking performance increases by incorporating prior system information. In figure A.11 the return per tracked state confirms this observation, with significant lower total returns for both the total and maneuver return.

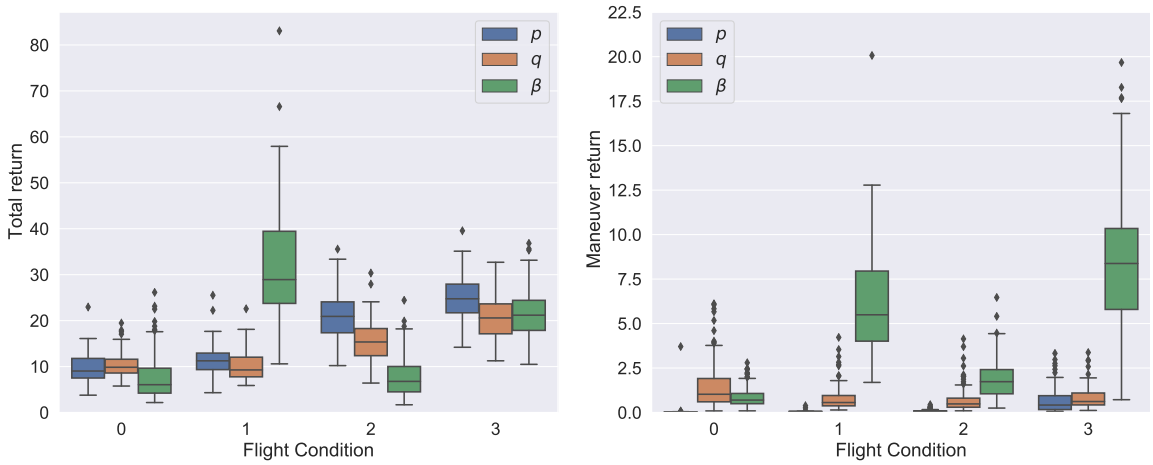


Figure A.11: The distribution of the returns per tracked state for experiment case C. The return is visualized for all flight conditions and for both the complete episode and the maneuver phase only.

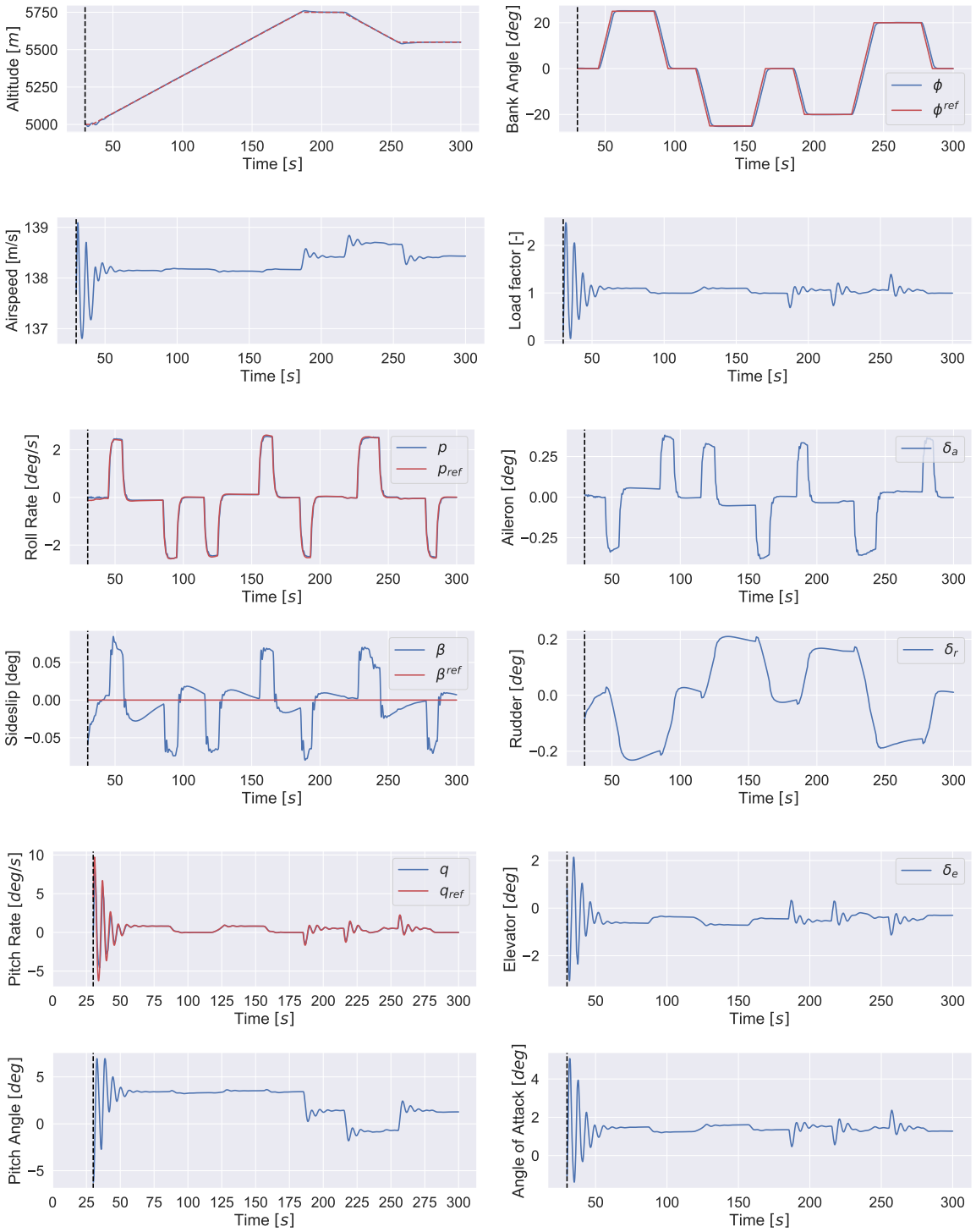


Figure A.12: Case C with flight condition 0 at $h_e = 5000\ m$ and $V_{TAS} = 140\ m/s$.

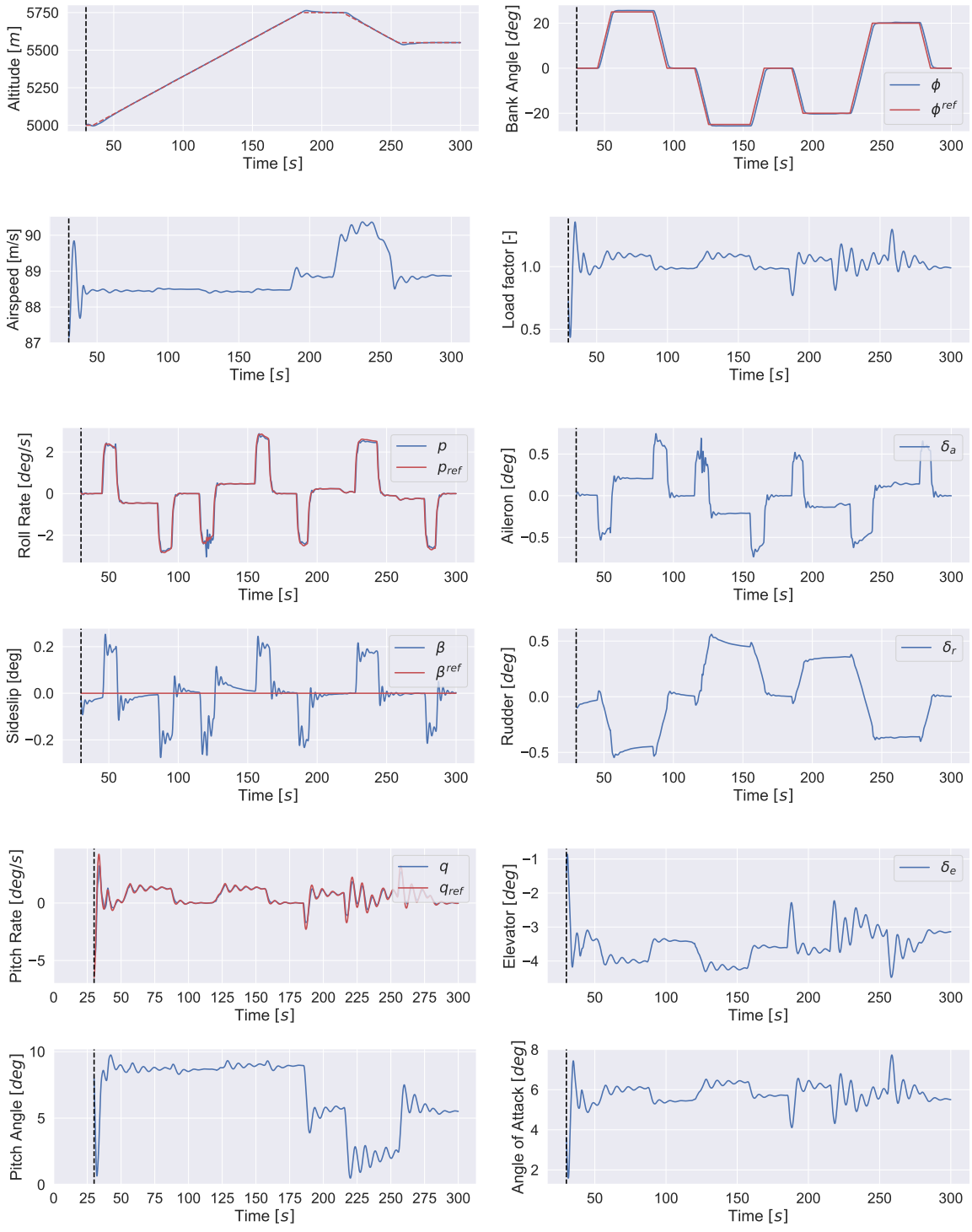


Figure A.13: Case C with flight condition 1 at $h_e = 5000\ m$ and $V_{TAS} = 90\ m/s$.

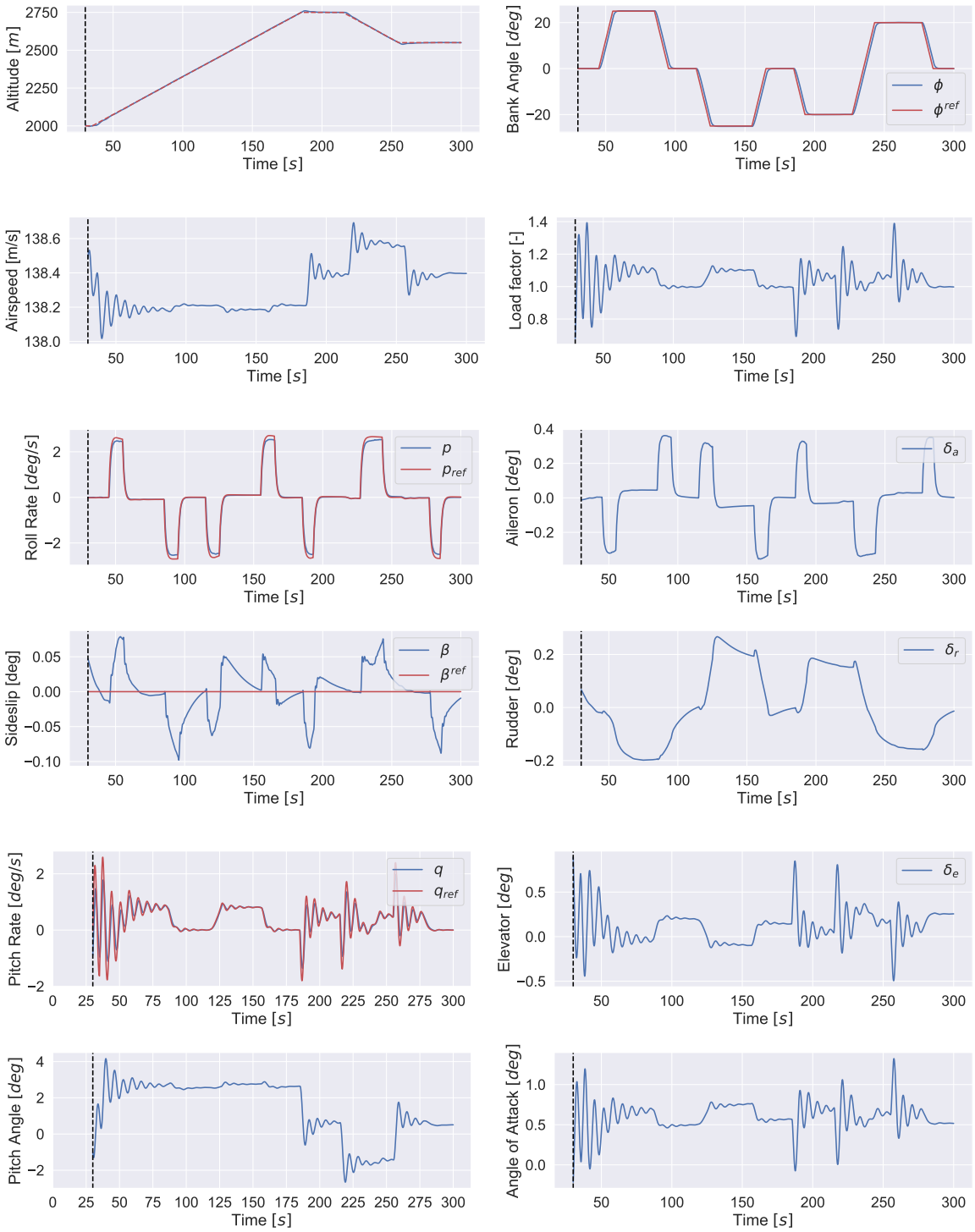


Figure A.14: Case C with flight condition 2 at $h_e = 2000\ m$ and $V_{TAS} = 140\ m/s$.

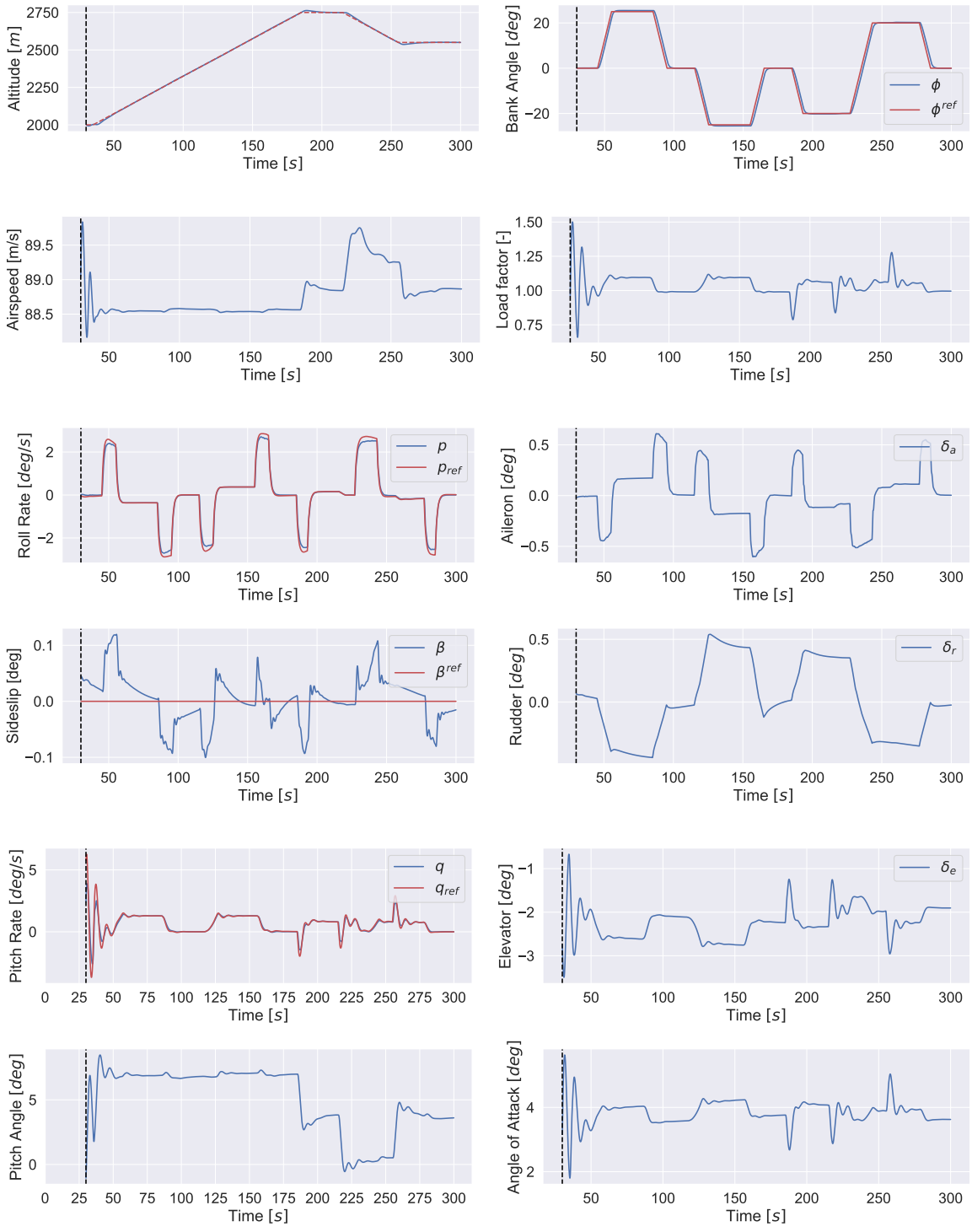


Figure A.15: Case C with flight condition 3 at $h_e = 2000\ m$ and $V_{TAS} = 90\ m/s$.

A.2. Failure Analysis

From the results discussed in the scientific paper, the only failed episodes occurred at flight conditions FC0 and FC1, with FC1 having a significant increased number of failed runs compared to FC0. The aerodynamic damping is less in these cases and is together with the outer loop controller seen as the main reason of the failure.

From the failed runs three types of behavior is observed which results in a failed run, namely upset flight, stall and prolonged error propagation or an combination thereof. It should be noted that the aircraft model is not validated for the flight regimes encountered in the failed episodes as these runs encounter stall or are outside the flight envelope. The focus therefore is not on the encountered dynamics, but on the agent behavior leading toward this failure.

A.2.1. Upset flight

The first failure cause is upset flight. This category is categorized by an unstable policy on the lateral control surfaces. For this class of failures, the learn rate or gradients are to aggressive/large in magnitude and cause the commanded action to alternate between the limits of the actuator. This effect is further enhanced by the critic, which becomes numerically unstable if both the actor and critic updates result in large changes. At some point the actions result in the aircraft exceeding an bank angle of 90 degrees. The altitude control feedback loop as illustrated in part I is no longer valid for bank angles over 90 degrees, further destabilizing the aircraft. The large errors shortly after result in the episode failing as the parameters overflow. This failure mode is often combined with stall.

A.2.2. Stall

The second cause of failure is stall and can result from an overaggressive policy or from poor altitude tracking during the forced excitation phase. In the stall region the system model is highly nonlinear and rapidly changes. The RLS model identification is unable to capture the change in dynamics fast enough resulting in unstable estimated parameters. This is enhanced by covariance wind-up resulting from insufficient excitation. Furthermore, in stall the body rates are no longer controllable resulting in an rate error which cannot be reduced without first recovering from stall. The continuous error causes the agent parameters to overflow.

A.2.3. Error accumulation

As the artificial neural networks do not contain a form of regulation, continuous small errors can cause parameters to grow without bounds. The actor and critic output become increasingly sensitive to the input as a result. At some point a small change in input will cause an instant saturation of the commanded control output. Depending on which control output this is, this results in upset flight, stall or a combination of both. The sudden large tracking errors quickly result in parameter overflow. An example of this failure case is observed in experiment case C, with use of the fixed model gradients. In figure A.16 this episode is shown.

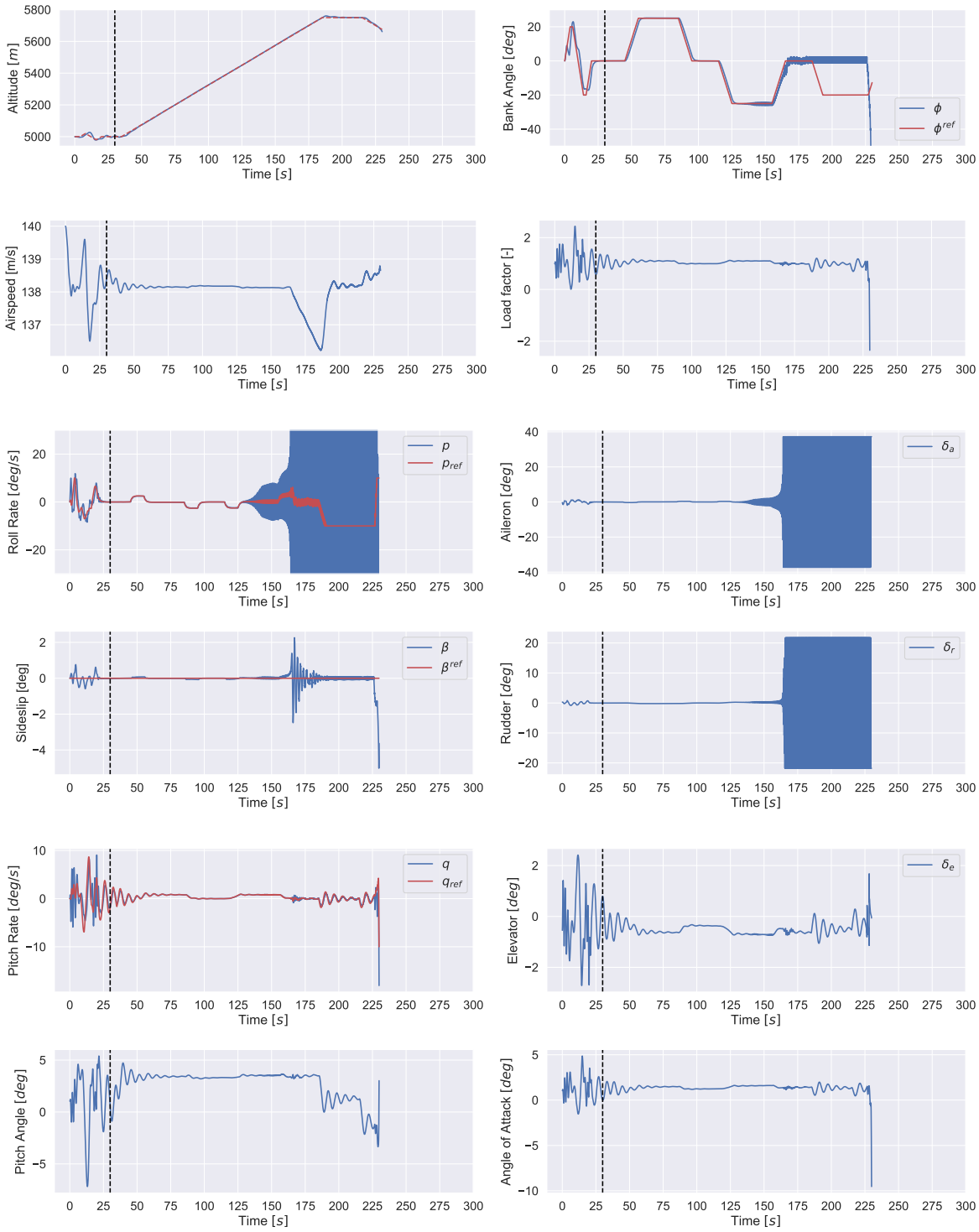


Figure A.16: Failed episode due to error accumulation from case C. Flight condition 0 at $h_e = 5000\ m$ and $V_{TAS} = 140\ m/s$. The initial cause is due to the error accumulation, but also upset flight and a negative stall angle are observed just before failure.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from `https://www.tensorflow.org/`.

[2] Paul Acquatella, Wouter Falkena, Erik-Jan van Kampen, and Q Ping Chu. Robust nonlinear spacecraft attitude control using incremental nonlinear dynamic inversion. In *AIAA Guidance, Navigation, and Control Conference*, page 4623, 2012.

[3] Paul Acquatella, E van Kampen, and Qi Ping Chu. Incremental backstepping for robust nonlinear flight control. In *Proceedings of the EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation and Control*, pages 1444–1463, 2013.

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[5] SN Balakrishnan and Victor Biega. Adaptive-critic-based neural networks for aircraft optimal control. *Journal of Guidance, Control, and Dynamics*, 19(4):893–898, 1996.

[6] Gary J Balas. Flight control law design: An industry perspective. *European Journal of Control*, 9 (2-3):207–226, 2003.

[7] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

[8] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[9] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[10] Boeing. Boeing autonomous passenger air vehicle completes first flight, 2019. `http://www.boeing.com/features/2019/01/pav-first-flight-01-19.page` [Accessed: 02-03-2019].

[11] Matthew Michael Botvinick. Hierarchical reinforcement learning and decision making. *Current opinion in neurobiology*, 22(6):956–962, 2012.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[13] Daimler AG. On the road in self-driving vehicles, 2018. `https://www.daimler.com/innovation/autonomous-driving/special/changes.html` [Accessed: 20-08-2018].

[14] Rick Durrett. *Probability: theory and examples*. Cambridge university press, 2010.

[15] Russell Enns and Jennie Si. Apache helicopter stabilization using neural dynamic programming. *Journal of guidance, control, and dynamics*, 25(1):19–25, 2002.

[16] Russell Enns and Jennie Si. Helicopter flight-control reconfiguration for main rotor actuator failures. *Journal of guidance, control, and dynamics*, 26(4):572–584, 2003.

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

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

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

[20] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015.

[21] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

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

[23] Said G Khan, Guido Herrmann, Frank L Lewis, Tony Pipe, and Chris Melhuish. Reinforcement learning and optimal adaptive control: An overview and implementation examples. *Annual Reviews in Control*, 36(1):42–59, 2012.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[25] S.H. Lane and R.F. Stengel. Flight control design using non-linear inverse dynamics. *Automatica*, 24(4):471–483, 1988.

[26] 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 preprint arXiv:1509.02971*, 2015.

[27] Peng Lu, Erik-Jan van Kampen, Cornelis de Visser, and Qiping Chu. Aircraft fault-tolerant trajectory control using incremental nonlinear dynamic inversion. *Control Engineering Practice*, 57: 126–141, 2016.

[28] Marvin Lee Minsky. *Theory of neural-analog reinforcement systems and its application to the brain model problem*. Princeton University., 1954.

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

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

[31] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[32] OpenAI. Openai baselines: Acktr and a2c, 2018. `https://blog.openai.com/baselines-acktr-a2c/` [Accessed: 16-09-2018].

[33] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[34] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[35] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[36] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[37] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

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

[40] S Sieberling, QP Chu, and JA Mulder. Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction. *Journal of guidance, control, and dynamics*, 33(6): 1732–1742, 2010.

[41] P Simplício, MD Pavel, E Van Kampen, and QP Chu. An acceleration measurements-based approach for helicopter nonlinear flight control using incremental nonlinear dynamic inversion. *Control Engineering Practice*, 21(8):1065–1077, 2013.

[42] Burrhus Frederic Skinner. *The behavior of organisms: An experimental analysis*. BF Skinner Foundation, 1990.

[43] L. Sonneveldt, E.R. Van Oort, Q.P. Chu, C.C. De Visser, J.A. Mulder, and J.H. Breeman. Lyapunov-based fault tolerant flight control designs for a modern fighter aircraft model. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2009.

[44] Lars Sonneveldt, QP Chu, and JA Mulder. Nonlinear flight control design using constrained adaptive backstepping. *Journal of Guidance, Control, and Dynamics*, 30(2):322–336, 2007.

[45] Robert F Stengel. *Flight dynamics*. Princeton University Press, 2015.

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press Cambridge, 2 edition, 2018. ISBN 9780262039246.

[47] Edward L Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, 2(4):i, 1898.

[48] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[49] George E. Uhlenbeck and Leonard S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[50] E Van Kampen, QP Chu, and JA Mulder. Online adaptive critic flight control using approximated plant dynamics. In *2006 International Conference on Machine Learning and Cybernetics*, volume 1, 2006.

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

[52] M.B. Vankadari, K. Das, C. Shinde, and S. Kumar. A reinforcement learning approach for autonomous control and landing of a quadrotor. pages 676–683, 2018.

[53] Stephen Verbist, Tommaso Mannucci, and Erik-Jan Van Kampen. The actor-judge method: safe state exploration for hierarchical reinforcement learning controllers. In *2018 AIAA Information Systems-AIAA Infotech@ Aerospace*, page 1634. 2018.

[54] Waymo. On the road, 2018. `https://waymo.com/ontheroad/` [Accessed: 20-08-2018].

[55] Paul Werbos. Approximate dynamic programming for realtime control and neural modelling. *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, pages 493–525, 1992.

[56] Ian H Witten. An adaptive optimal controller for discrete-time markov environments. *Information and control*, 34(4):286–295, 1977.

[57] Y Zhou. *Online reinforcement learning control for aerospace systems*. PhD thesis, Delft University of Technology, 2018.

[58] Y Zhou, E van Kampen, and QP Chu. Incremental model based heuristic dynamic programming for nonlinear adaptive flight control. In *Proceedings of the International Micro Air Vehicles Conference and Competition 2016, Beijing, China*, 2016.

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

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