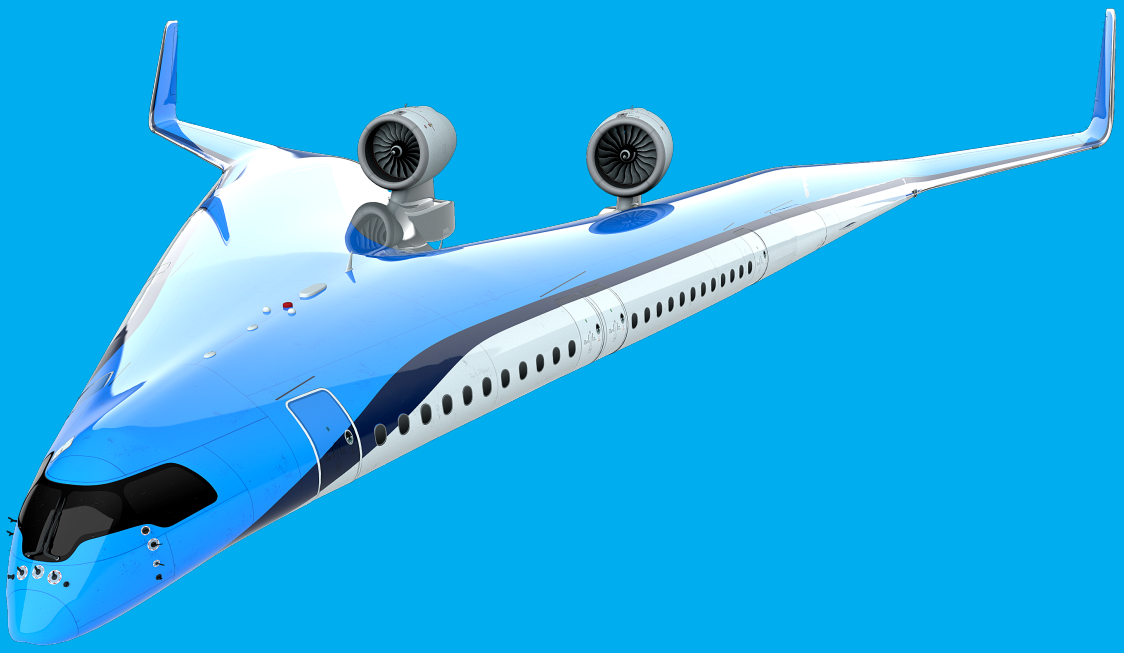
Reinforcement Learning for Flight Control of the Flying V

MSc Thesis Report

W.J.E. Völker



TU Delft

Reinforcement Learning for Flight Control of the Flying V

MSc Thesis Report

by

W.J.E. Völker

to obtain the degree of Master of Science
at Delft University of Technology,
to be defended publicly on Monday July 4, 2022 at 10:00 AM.

Student number:	4303067	
Project duration:	September 2021 - June 2022 (incl. lit. study)	
Thesis committee:	Dr. ir. C. de Visser,	TU Delft, Aerospace Control and Simulation, chair
	Dr. ir. E. van Kampen,	TU Delft, Aerospace Control and Simulation, supervisor
	Dr. ir. T. Sinnige,	TU Delft, Flight Performance and Propulsion, external
	Ir. Y. Li,	TU Delft, Aerospace Control and Simulation, additional

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

TU Delft

Preface

When I first learned about reinforcement learning I was intrigued by this idea of enabling machines to learn in the way that humans and animals learn. I was overwhelmed by the seemingly infinite number of possibilities that reinforcement learning would offer the world. After all, the fact that in theory you only need to tell a machine what its goal is, after which it will autonomously learn how to reach that goal, means that you can practically teach a machine anything, without even requiring much engineering knowledge.

Writing this thesis, however, made me realise that making a complex controller based on reinforcement learning does indeed require extensive knowledge of the dynamics of the controlled system - not only to perform sound validation, but also to get the controller to work properly in the first place. Furthermore, it requires a lot of trial and error, which then again often needs very little engineering knowledge.

However, being able to use knowledge of flight dynamics to understand the behaviour of a reinforcement learning agent that tries to control the Flying V was also what made this thesis so enjoyable. The Flying V, with its unconventional configuration, was the perfect platform for these thought experiments. Moreover, I believe that showing what the Flying V can do in terms of controlled flight is important, as it helps to generate support for this aircraft that plays a fundamental role in the transition to more sustainable flight.

I would like to thank my supervisor Erik-Jan for stimulating me to continue searching for possible causes when the controllers I developed for this research would not work, and for providing many useful insights. I would also like to sincerely thank him for always keeping up to date on my thesis, for always making time to give advice when I asked for it, and for being flexible. Furthermore, I would like to thank my co-supervisor Yifei for offering suggestions for improving my report. Lastly, thanks to Juul and my parents for supporting me during the busy times of performing this thesis research.

The field of reinforcement learning is continuously evolving, with more sample-efficient, stabler, and more explainable algorithms being published frequently. I hope you enjoy reading this thesis and encourage you to use it as inspiration to investigate the usefulness of other, perhaps better reinforcement-learning algorithms for flight control, or on any type of flight control for the Flying V.

W.J.E. Völker
Delft, June 2022

Contents

Nomenclature

Abbreviation	Definition
AFCS	automatic flight control system
ACD	adaptive critic design
ADP	approximate dynamic programming
ANN	artificial neural network
CS	control surface
DDPG	deep deterministic policy gradient
DHP	dual heuristic programming
DP	dynamic programming
DRL	deep reinforcement learning
DQN	deep Q-networks
FCS	flight control system
GDHP	globalised dual heuristic programming
HDP	heuristic dynamic programming
iADP-OP	incremental ADP with output feedback
IDHP	incremental model based DHP
LSTM	long short term memory layer
MC	Monte Carlo
MDP	Markov decision process
PID	proportional integral derivative
PPO	proximal policy optimisation
RL	reinforcement learning
SAC	soft actor critic
SAS	stability augmentation system
SCAS	stability and control augmentation system
TD	temporal difference
TD3	twin-delayed deep deterministic policy gradient
UAV	unmanned aerial vehicle
VLM	vortex lattice method

Symbols

Symbol	Definition
a	action
b	behaviour policy
G	return
q	action value
r	reward
s	state
v	state value
w	approximate value function weight
α	step size
γ	discount rate
ϵ	probability of random action
π	policy
ρ	importance sampling ratio
θ	approximate policy weight

1 Introduction

Why is researching flight control of the the Flying V relevant and why is it a good idea to use reinforcement learning for it? This introduction will try to give an answer to these questions in section 1.1 by providing background knowledge on the Flying V and reinforcement learning and explaining the problems faced in the development of the Flying V. Thereafter section 1.2 will motivate the choice for offline learning and thereby narrow down the scope of this research. Section 1.3 lays out the aim and the research questions that served as a guide for the literature review and experimental research described in this report. Finally section 1.4 gives the outline of the chapters that follow this introduction.

1.1. Motivation

Aviation is responsible for 3.5% of the human impact on climate change - measured in amount of radiative forcing - or even 5% if the effect of cirrus cloud enhancement is taken into account [53]. As global flight traffic is projected to increase with 4.3% annually [1], while average fuel burn of new commercial jet aircraft decreases by only 1% annually [101], novel solutions are needed.

Aircraft's fuel burn and thereby its climate impact can be reduced by improving operations, propulsion systems and airframe designs. In terms of improving propulsion systems electrification of short flights seems feasible. The low specific energy of batteries compared to fuel makes electrification of longer-range aircraft - which cause roughly two-thirds of CO2 emissions [40] - unrealistic in the medium term however [46]. Liquid hydrogen has a higher specific energy than batteries and is therefore a better energy carrier for longer-range aircraft, but it suffers from a lower volumetric energy density than both batteries and jet fuel, thereby complicating on-board storage [29].

Although novel propulsion systems are part of the long-term solution to aircraft's climate impact, a more realistic solution for longer-range aircraft may be to improve their airframe design, by increasing aerodynamic efficiency and decreasing structural weight. Aerodynamic efficiency gains (measured by the increase in lift-to-drag ratio) seem to have reached an asymptote though [58], which may mean that the the traditional tube-and-wing airframe is approaching its limits [59]. Therefore, a rethink of the tube-and-wing airframe is needed.

The Flying V is a novel aircraft concept that may enable a step change in aerodynamic efficiency. The V-shaped aircraft concept proposed by Benad [9] of the Berlin University of Technology and Airbus in 2015 promises a lift-to-drag ratio 25% higher than conventional wide-body commercial passenger aircraft [32]. Like other flying wing designs the Flying V realises aerodynamic efficiency gains mainly by having a smaller wetted area for a given payload than tube-and-wing aircraft, as for flying wings providing lift, trimming the aircraft and accommodating the payload are all performed by the same integrated component. Moreover, the the Flying V promises to have a lower structural weight than comparable tube-and-wing aircraft [9].

Despite several promising studies on flying wings over the past century, none have made it to market as commercial passenger aircraft. Doubts on the stability and control of flying wings are an important factor hindering acceptance [100]. To ensure stable, controlled and thereby safe flight an automatic flight control system (AFCS) may be a solution. However, the design of an AFCS is difficult for novel aircraft, as no off-the-shelf modelling and design tools are available and thereby accurate simulation models are hard to obtain and control design cannot rely on standard methods and software tools. Furthermore, the flight dynamics

of novel aircraft often involve non-linearities, which are particularly hard to model and control. This means that conventional control design - which is usually based on linear control theory - is not likely to provide an adequate solution, as it requires an accurate simulation model for initial gain tuning and its ability to control a plant with non-linear dynamics is limited.

This research proposes the application of reinforcement learning (RL) - a bio-inspired method based on the principle of learning through interaction with a potentially unknown environment - to automatic flight control of the Flying V. RL methods have several attractive properties that make them promising for AFCS design, especially for novel aircraft. Firstly and most importantly, RL methods are well suited to systems for which no (accurate) model is available. An RL agent may learn without any prior knowledge of the plant dynamics, thereby allowing online learning, on the real system. Alternatively an RL agent may learn offline with a simulation model and subsequently generalise the learned behaviour to the real world. If learning takes place in simulation a function approximator such as a deep artificial neural network that is robust to model uncertainties may be integrated in the RL method. And secondly, an RL agent with function approximation may find a solution to a problem that involves non-linear and complex plant dynamics, even if the designer does not know the general shape of a solution a priori.

1.2. Choice for offline learning

An AFCS based on RL for the Flying V may learn offline, online or both. The preferable option is not obvious and the choice of an RL method depends on whether learning happens online or offline. Online learning has as its main advantages that it does not require a model of the plant dynamics and that it allows adaptation to environmental and dynamic changes. Online RL methods for flight control are usually some form of adaptive critic design, or ACD (also known as adaptive dynamic programming or ADP), as are all online RL methods treated in this introduction (see [68] for an overview of the main families of ACDs).

Successes in the application of RL to low-level flight control go back to the beginning of this millennium. These first successes were applications of ADP methods to simulation models, specifically *direct neural dynamic programming* to an Apache helicopter model [31], *dual heuristic programming* to a business jet model [34] and (action-dependent) dual heuristic programming to an F-16 model [90]. For the development of the AFCSs presented in these studies an initial offline learning phase was required and therefore a model of the plant dynamics.

Later Zhou et al. [103, 104] showed that simple reference tracking tasks can be learned fully online by developing versions of ADP algorithms that increase sample efficiency by learning an incremental model of the plant online (IDHP and iADP-OP). Fully online learning of more complex control tasks is a subject of ongoing research and recent experiments such as the work by Lee and Kampen [54] show promising results. However, the failure rates reported in state-of-the-art literature on online RL such as [54] indicate that online RL methods are currently not mature enough to meet regulatory requirements, if they were to be used to develop a baseline controller for commercial passenger aircraft. Moreover, validation of a flight controller that learns online is hard. Predicting the way the controller will adapt to unforeseen circumstances is after all inherently difficult.

Occasional failures and unclear validation methods of an AFCS for the Flying V would not contribute to the acceptance of flying wings, which is one of the aims of this research. Therefore, this research will use an offline-learning approach. Offline learning has as its main advantages that it can handle more complex control tasks and involves no danger of trying unsafe control signals during the learning phase. As offline learning requires no restrictions on the safety of control signals *tried* during learning and as offline learning allows for a larger amount of samples to be collected than online learning, a wider variety of RL methods is available. A particularly promising family of RL methods that may not be sample-efficient enough for online learning for flight control, but do show state-of-the-art performance on a wide variety of complex problems (such as the game of Go [76]) is the family of deep reinforcement learning (DRL) algorithms. A comprehensive introduction to DRL can be found in [35], but the essentials are also introduced in the present report.

The amount of literature on DRL for fixed-wing flight control is limited, but the available literature shows promising results. An example of a state-of-the-art DRL algorithm applied to flight control is [14], in which a controller based on proximal policy optimisation (PPO) outperforms a PID-controller. The flight controller is applied to a small, unmanned tailless aircraft. More recently [24] showed that with soft actor-critic (SAC) a successful AFCS can be designed for coupled manoeuvres, such as a 40-degree-bank climbing turn.

An important downside of offline learning is that the adaptive nature of RL is not exploited. Whereas an RL controller that continually learns online (or switches on when a failure is detected) can adapt its behaviour

when a control failure occurs or a discrepancy between the simulation model and reality exists, an offline RL controller has to rely purely on behaviour learned in simulation. However, Dally [24] showed that with SAC a robust controller can be designed, which can generalise behaviour learned in simulation to, for example, failure cases and biased, noisy sensors that it has not experienced in simulation. A more in-depth exploration of the opportunities and challenges of DRL for flight control through a review of published literature is part of the present report.

1.3. Aim and research questions

The Flying V serves as an excellent platform for stability-and-control research of flying wings as passenger carriers, as its performance is promising and previous modelling efforts at Delft University of Technology allow the developing and testing of an AFCS in simulation. This research sets out to contribute to the acceptance of flying-wing designs by investigating a solution for automatic flight control of the Flying V based on the latest developments in the field of RL. Specifically, this research will propose an automatic altitude controller based on DRL algorithm TD3 and test it on a 6-degree-of-freedom simulation model of the Flying V. The following research questions will guide both this research's literature study and the experimental phase.

The main research question of this thesis is:

> **"How can an automatic altitude control system that is robust to aerodynamic-model uncertainties be designed for the Flying V, by using a deep reinforcement learning algorithm to autonomously learn altitude control?"**

The sub-questions that will together provide the answers to the main research question are:

1. What reinforcement-learning method is most promising for flight control of the Flying V?
   (a) What are the problem characteristics that dictate the choice of method?
   (b) Based on the identified problem characteristics, is learning online or offline more suitable?
   (c) What is the preliminary knowledge of reinforcement learning required to make an informed choice among existing reinforcement-learning algorithms?
   (d) What are the most promising state-of-the-art algorithms for the learning setting (online or offline) determined from question 1(c)?
   (e) How can these algorithms solve flight-control problems that involve coupled, non-linear dynamics?
   (f) How robust are these algorithms to uncertainties?
2. How can the reinforcement-learning method determined from research question 1 be implemented on a simulation model of the Flying V to demonstrate RL flight control for the Flying V?
   (a) What existing simulation model of the Flying V is most suited for this research?
   (b) How can the states and control signals defined in the simulation model of the Flying V be used to formulate the states and actions in a reinforcement-learning problem?
   (c) How can the reward function be defined?
   (d) How can the controller be structured, i.e. cascaded or not, to exploit the ability of RL to learn a complex task with minimal input from the human expert?
   (e) How can a training strategy be devised that increases sample efficiency and ease of implementation?
   (f) How do the hyperparameter settings affect learning efficiency and stability?
   (g) Are the available computational resources sufficient?
3. How well does the implemented flight control system perform on a nominal simulation model of the Flying V?
   (a) What metrics are most suited to characterise performance?

   (b) How does the flight control system compare to a flight control system that is based on a different method?

4. How robust is the implemented flight control system to changes in the simulation model of the Flying V?

   (a) How well does the nominal flight control system perform on the "real system", i.e., an altered version of the simulation model of the Flying V that is used to simulate a sim-to-real transfer?

   (b) How well does the nominal flight control system perform under measurement noise or bias?

   (c) How well does the nominal flight control system perform in response to various shapes of altitude-reference signals?

   (d) How well does the nominal flight control system perform if initial flight conditions deviate from the nominal (trimmed) initial flight conditions?

1.4. Report outline

The body of this report starts with part I, which consists of a scientific paper that presents the research methodologies, results and discussion of the main research phase of this thesis. Thereby the scientific paper presents a single-loop automatic altitude controller for the Flying V, based on TD3. The scientific paper is followed by part II, which includes a literature study and a preliminary analysis. Readers who are less familiar with reinforcement learning are advised to read 2 and 3 first, before reading the scientific paper. Section 2 covers the fundamental knowledge of reinforcement learning, as well as several state-of-the-art model-free algorithms, from which the most promising algorithms are identified. Chapter 3 discusses how state-of-the-art model-free algorithms can be applied to flight control, treating several challenges and opportunities of implementing these algorithms. Readers who are less familiar with the Flying V are advised to read chapter 4 before reading the scientific paper, which gives an overview of past and present research on simulation and control of the Flying V and thereby analyses the tools available and the knowledge gaps. Part II ends with a preliminary analysis of a TD3 agent in a simple environment in chapter 5, to empirically discover the function of TD3's hyperparameters and gain practical experience with implementing TD3. Then, part III follows with additional results from the experimental research phase, which were not included in the scientific paper. Finally, part IV, chapter 8, ends this report by summarising the answers that this report provides to several of the research questions presented in section 1.3 and providing recommendations for further research.

I

Scientific paper*

***This paper has been submitted to AIAA SciTech 2023 and is currently under review.**

Twin-Delayed Deep Deterministic Policy Gradient for altitude control of a flying-wing aircraft with an uncertain aerodynamic model

Willem Völker*
Faculty of Aerospace Engineering, Delft University of Technology, Delft, 2629HS, The Netherlands

Recent research on the Flying V - a flying-wing long-range passenger aircraft - shows that its airframe design is 25% more aerodynamically efficient than a conventional tube-and-wing airframe. The Flying V is therefore a promising contribution towards reduction in climate impact of long-haul flights. However, some design aspects of the Flying V still remain to be investigated, one of which is automatic flight control. Due to the unconventional airframe shape of the Flying V, aerodynamic modelling cannot rely on validated aerodynamic-modelling tools and the accuracy of the aerodynamic model is uncertain. Therefore, this contribution investigates how an automatic flight controller that is robust to aerodynamic-model uncertainty can be developed, by utilising Twin-Delayed Deep Deterministic Policy Gradient (TD3) - a recent deep-reinforcement-learning algorithm. The results show that an offline-trained single-loop altitude controller that is fully based on TD3 can track a given altitude-reference signal and is robust to aerodynamic-model uncertainty of more than 25%.

I. Introduction

Aviation is responsible for 3.5% of the human impact on climate change - measured in amount of radiative forcing - or even 5% if the effect of cirrus cloud enhancement is taken into account [1]. As global flight traffic is projected to increase with 4.3% annually [2], while average fuel burn of new commercial jet aircraft decreases by only 1% annually [3], novel solutions are needed.

Airframe designers conventionally decrease fuel burn of next-generation aircraft by increasing aerodynamic efficiency and decreasing structural weight through optimisation of current-generation airframe designs. Aerodynamic-efficiency gains (measured by the increase in lift-to-drag ratio) seem to have reached an asymptote though [4], which may mean that the traditional tube-and-wing airframe is approaching its limits [5]. Therefore, a rethink of the tube-and-wing airframe is needed.

The Flying V is a novel aircraft concept that may enable a step change in aerodynamic efficiency. The V-shaped aircraft concept proposed by Benad [6] of the Berlin University of Technology and Airbus in 2015 promises a lift-to-drag ratio 25% higher than conventional wide-body commercial passenger aircraft [7]. Like other flying wing designs the Flying V realises aerodynamic efficiency gains mainly by having a smaller wetted area for a given payload than tube-and-wing aircraft, as for flying wings providing lift, trimming the aircraft and accommodating the payload are all performed by the same integrated component. Moreover, the Flying V promises to have a lower structural weight than comparable tube-and-wing aircraft [6].

Despite several promising studies on flying wings over the past century, none have made it to market as commercial passenger aircraft. Doubts on the stability and control of flying wings are an important factor hindering acceptance [8]. To ensure stable, controlled and thereby safe flight an automatic flight control system (AFCS) may be a solution. However, the design of an AFCS is difficult for novel aircraft, as no off-the-shelf modelling and design tools are available and thereby accurate simulation models are hard to obtain and control design cannot rely on standard methods and software tools. Furthermore, the flight dynamics of novel aircraft often involve non-linearities, which are particularly hard to model and control. This means that conventional control design - which is usually based on linear control theory - may not provide an adequate solution, as it requires an accurate simulation model for initial gain tuning and its ability to control a plant with non-linear dynamics is limited.

This research proposes the application of reinforcement learning (RL) - a bio-inspired method based on the principle of learning through interaction with a potentially unknown environment - to automatic flight control of the Flying V. RL methods have several attractive properties that make them promising for AFCS design, especially for novel aircraft.

*MSc Student, Control and Simulation Research Group, Delft University of Technology, willem.volker@gmail.com

Firstly and most importantly, RL methods are well suited to systems for which no (accurate) model is available. An RL *agent* may learn without any prior knowledge of the plant dynamics, thereby allowing online learning, on the real system. Alternatively an RL agent may learn offline with a simulation model and subsequently generalise the learned behaviour to the real world. If learning takes place in simulation a function approximator such as a deep artificial neural network that is robust to model uncertainties may be integrated in the RL method. And secondly, an RL agent with function approximation may find a solution to a problem that involves non-linear and complex plant dynamics, even if the designer does not know the general shape of a solution a priori.

Fully online learning of more complex control tasks is a subject of ongoing research and recent experiments such as the work by Lee and Kampen [9] show promising results. However, the failure rates reported in state-of-the-art literature on online RL such as [9] indicate that online RL methods are currently not mature enough to meet regulatory requirements, if they were to be used to develop a baseline controller for commercial passenger aircraft. Moreover, validation of a flight controller that learns online is hard. Predicting the way the controller will adapt to unforeseen circumstances is after all inherently difficult.

Occasional failures and unclear validation methods of an AFCS for the Flying V would not contribute to the acceptance of flying wings, which is one of the aims of this research. Therefore, this research will use an offline-learning approach. Offline learning has as its main advantages over online learning that more complex control tasks can be learned and the danger of trying unsafe control signals during the learning phase is not present. As offline learning requires no restrictions on the safety of control signals *tried* during learning and as offline learning allows for a larger amount of samples to be collected than online learning, a wider variety of RL methods is available. A particularly promising family of RL methods that may not be sample-efficient enough for online learning for flight control, but do show state-of-the-art performance on a wide variety of complex problems (such as the game of Go [10]) is the family of deep reinforcement learning (DRL) algorithms.

The amount of literature on DRL for fixed-wing flight control is limited, but the available literature shows promising results. An example of a state-of-the-art DRL algorithm applied to flight control is [11], in which a controller based on proximal policy optimisation (PPO) outperforms a PID-controller. The flight controller is applied to a small, unmanned tailless aircraft. More recently [12] showed that with soft actor-critic (SAC) a successful AFCS can be designed for coupled manoeuvres, such as a 40-degree-bank climbing turn.

An important downside of offline learning is that the adaptive nature of RL is not exploited. Whereas an RL controller that continually learns online (or switches on when a failure is detected) can adapt its behaviour when a control failure occurs or a discrepancy between the simulation model and reality exists, an offline RL controller has to rely purely on behaviour learned in simulation. However, Dally and van Kampen [12] showed that with SAC a robust controller can be designed, which can generalise behaviour learned in simulation to, for example, failure cases and biased, noisy sensors that it has not experienced in simulation.

The contributions of this paper are the following. Firstly, this paper presents an assessment of the robustness of a flight controller based on TD3 to uncertainties in the aerodynamic model of the Flying V. Secondly, this paper provides an indication of the usefulness of TD3 for flight control of an aircraft for which the aerodynamic model is uncertain. Finally, this paper presents the first application of RL to the Flying V. Therefore, the methods presented in this paper may serve as a starting point for further research into RL for flight control of the Flying V.

To investigate whether an AFCS based on TD3 is indeed robust to aerodynamic model uncertainties an altitude controller was developed. Hereby control was fully based on TD3, without inner-loop PID control. Moreover, the controller was structured as a single loop, in order to investigate the ability of TD3 to learn the nonlinear and coupled dynamics associated with altitude control. An additional reason for using a single control loop was to exploit the ability of an RL agent to autonomously learn a task, with minimal input based on domain knowledge from the human control engineer. To the best knowledge of this author this research represents the first application of a model-free DRL algorithm for single-loop altitude control of a passenger aircraft. The offline-trained altitude controller was tested on a flight-simulation model of the Flying V that simulated varying levels of aerodynamic-model uncertainty. The effect of model uncertainty on the altitude-tracking error was then used to evaluate the robustness of an AFCS based on TD3 to aerodynamic-model uncertainty.

The paper is structured as follows. Section II describes the algorithm behind TD3, the modelling and simulation methodology used to simulate the Flying V, and the methodology used to train a TD3 agent offline for flight control of the Flying V. Section III shows the responses and errors corresponding to simulations of the trained TD3 agent for the nominal Flying-V model, as well as the model with simulated aerodynamic-model uncertainty, sensor noise and altered initial conditions. Finally section IV presents the conclusion of the present research.

II. Methodology

This section introduces the methodology used to produce the results in this paper. Firstly, subsection II.A introduces the reinforcement-learning algorithm, TD3, used to develop the altitude controller. Secondly, subsection II.B introduces the controlled system: the Flying V and its flight-simulation model. Thirdly, subsection II.C describes how altitude control for the Flying V was formulated as a reinforcement-learning problem. Fourthly, subsection II.D presents the settings of the hyperparameters and other parameters used to train a TD3 agent for altitude control. Lastly, subsection II.E describes how aerodynamic-model uncertainty, sensor noise, and altered initial conditions were modelled.

A. Twin-Delayed Deep Deterministic Policy Gradient

Twin-Delayed Deep Deterministic Policy Gradient or TD3 is a state-of-the-art model-free DRL algorithm for continuous action spaces introduced by Fujimoto et al. [13] in 2018. As TD3 is based on predecessor algorithm Deep Deterministic Policy Gradient (DDPG), published by Lillicrap et al. [14] in 2016, this section starts with an introduction to the fundamentals of DDPG.

While conventional policy-gradient methods use a stochastic policy to ensure sufficient exploration, Deep Deterministic Policy Gradient (DDPG) improves on these methods by using a deterministic policy $\mu(s, a, \boldsymbol{\theta})$, with s the state, a the action, and θ the parameter vector. DDPG is similar to earlier reinforcement-learning algorithms Q-learning and DQN in the way it approximates the optimal action-value function, through an implementation of the Bellman optimality equation. The parameters that define the action-value function are approximated by networks known as Q-Networks.

Also similarly to DQN, DDPG uses experience replay. A set of previous experiences at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$, with r the reward, is stored in a *replay buffer* D. DDPG applies Q-learning updates to samples of experience (s, a, r, s') which are randomly chosen from the replay buffer.

Furthermore, DDPG borrows the idea of using target networks from DQN. As the target y_{DQN}, given by Eq. (1) (with γ the discount factor), depends on the same parameters $\mathbf{w}$ which are being updated, learning can become unstable. With the use of a separate parameter vector $\mathbf{w}^-$ (a time-delayed version of $\mathbf{w}$) to construct a target network $\hat{Q}$, stability is improved. $\hat{Q}$ is constructed by simply cloning Q every C steps. DDPG updates and averages the target networks once every main network update, instead of every C steps as DQN does.

$$y_{DQN} = r + \gamma \max_{a'} Q(s', a', \mathbf{w}) \tag{1}$$

DDPG adds an additional technique to DQN to enable maximisation over continuous action spaces, which would be too expensive with a normal optimisation algorithm. The optimal action-value function is assumed to be differentiable with respect to its action arguments, such that its gradient can be computed and maximisation can be approximated. For this DDPG uses a *target policy network* μ_{target}, which is constructed in the same way as the target Q-network. The target policy network approximates the maximising action for the target action-value function. The resulting loss function is mimised by stochastic gradient descent.

DDPG can be characterised by its target formulation. The target of DDPG can be represented by

$$y_{DDPG} = r + \gamma Q_{target}\left(s', \mu_{target}\left(s', \boldsymbol{\theta}\right), \mathbf{w}\right)^2. \tag{2}$$

The policy learning step in DDPG is performed by applying gradient ascent with respect to the policy parameters to solve

$$\max_{\theta} \underset{s\sim D}{\mathrm{E}} \left[Q\left(s, \mu(s, \boldsymbol{\theta})\right)\right]. \tag{3}$$

TD3 aims to improve on DDPG and other actor-critic methods by addressing function approximation errors that cause overestimation of action values and sub-optimal policies. Fujimoto et al. [13] brought about this improvement by adapting DDPG with three techniques, the first of which is *double learning*, introduced by van Hasselt [15, 16]. Double learning works by first producing two independent estimates (i.e., from different samples) of a value and using one estimate to choose the maximising action, while using the other to estimate its value. The value estimate will then be unbiased. Secondly the process is repeated with the role of the two estimates reversed, resulting in a second unbiased estimate. TD3 applies double learning by constructing the target in its loss functions with the smallest of the two action values it learns and names this *clipped double Q-learning*. Thereby underestimation is favoured, which is unlikely to persist during learning, as the policy will not favour actions with low values.

The term *delayed* refers to the second technique, namely that of *delaying policy improvement* until policy evaluation converges. TD3 improves its policy once for every two policy evaluation steps. Delaying the policy together with the use of target networks should reduce variance, a possible cause of overestimation bias.

The third technique is *target policy smoothing*. The smoothing is performed to avoid that incorrectly highly valued estimates are exploited by the agent. By adding noise to the target policy and averaging over mini-batches, variance in the target (caused by function approximation errors) can be smoothed. The reasoning behind this technique is that similar actions should have similar values and outliers are therefore probably incorrect.

Target policy smoothing is applied by injecting clipped noise to the target policy and then clipping the action that results from the target policy with added noise. The target action is then:

$$a'(s') = \text{clip}\left(\mu_{\text{target}}(s', \boldsymbol{\theta}) + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma), \tag{4}$$

in which c is the maximum noise value, and ϵ the probability of taking a random action. The action is subsequently clipped to ensure it lies within the valid action range $a_{Low} \leq a \leq a_{High}$.

Clipped double Q-learning is performed by constructing the target y_{TD3} for both action-value functions as

$$y_{TD3} = r + \gamma \min_{i=1,2} Q_{i,\text{target}}(s', a'(s'), \mathbf{w}), \tag{5}$$

and then choosing the smallest target value of the two, to which both action-value functions are regressed.

Then, like in DDPG, Eq. (3) is used for policy improvement, although less frequently than in DDPG, as prescribed by the delay technique. For this DDPG simply uses the first of the two action-value function approximations it has learned. Also similar to DDPG, noise in the behaviour policy is added to ensure exploration.

With the improvements to DDPG, which is known to be unstable [17, 18] but already often outperformed competitor algorithms on benchmark tasks when it was introduced, Fujimoto et al. [13] produced a relatively stable state-of-the-art RL algorithm. Benchmarking research by Lazaridis et al. [19], Ball and Roberts [20] shows that comparison of RL algorithms is not straightforward, as performance is highly task-dependent, but it is clear that SAC [21] shows comparable performance to TD3 on benchmark tasks. However, research published by Dong et al. [22] suggests that the stochastic nature of SAC (randomness is maximised during training) might lead to learning more oscillatory behaviour, compared to TD3. Whilst the oscillations might not increase tracking error, they would likely increase actuator wear and decrease passenger comfort in flight. Therefore, to decrease the chance that the RL agent learns a policy that is more oscillatory than desirable for flight control, TD3 was chosen for the present research.

B. Flying-V Model and Simulation

The Flying V is a V-shaped flying wing with a pressurised cabin that extends through both wings, as illustrated in Fig. 1. The cabins are located in the wing's leading edge and the engines at the trailing edge, on top of the wing to reduce noise propagation to the ground. The Flying V design aims to outperform state-of-the-art commercial passenger aircraft designs similar to that of the Airbus A350-900, mainly in terms of lift-to-drag ratio and structural weight. The aircraft has a capacity of 314 passengers in a two-class configuration, a cruise speed of M=0.85, a range of 15,000 km and a wing span of 64.75m, all similar to those of the Airbus A350-900.

The flight control system of the Flying V consists of a pair of inboard elevons δ_{e_i}, a pair of outboard elevons δ_{e_o}, a pair of rudders δ_r and two engines. The present research will use a conventional approach to control allocation and thereby allocate the inboard elevons to pitch control (through symmetrical deflection), the outboard elevons to roll control (through asymmetrical deflection) and the rudders to yaw control (through deflection in the same direction around the aircraft body's vertical axis).

The flight simulation model used for this research was based on stability-and-control derivatives obtained from the vortex-lattice method applied to a numerical model of the Flying V [25]. From the stability-and-control derivatives the force and moment coefficients C_X, C_Y, C_Z, C_L, C_M, and C_N were computed according to Eq. (6), in which X is interchangeable with a force along, or moment around, a different axis. Hereby X, Y and Z are the forces along, and L, M and N are the moments around the body x-, y- and z-axes.

$$\begin{aligned} C_X = C_X(\alpha) + C_X(\alpha, \beta) + C_X(\alpha, p) + C_X(\alpha, q) + C_X(\alpha, r) + C_X(\alpha, \delta_{r_{left}}) + C_X(\alpha, \delta_{e_{o,left}}) + \\ C_X(\alpha, \delta_{e_{i,left}}) + C_X(\alpha, \delta_{r_{right}}) + C_X(\alpha, \delta_{e_{o,right}}) + C_X(\alpha, \delta_{e_{i,right}}) \end{aligned} \tag{6}$$

The computed force and moment coefficients, the current state and control input and the geometrical properties of the Flying V were subsequently used as inputs for the equations of motion, from which the 12 states in Table 1 could be obtained, describing the position, velocity, Euler angles and angular rates of the Flying V. The control input consisted of

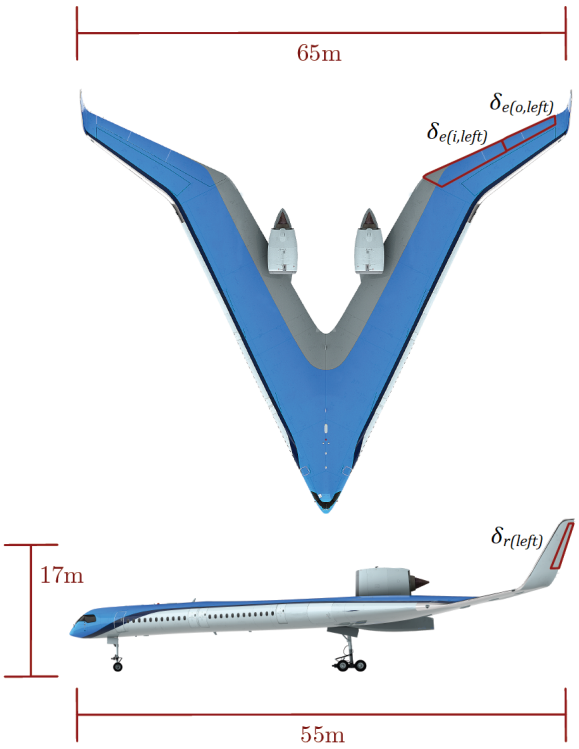


Fig. 1 Illustration of the control-surface layout and outer dimensions of the Flying V [23, 24].

the deflection of the three pairs of control surfaces and the thrust force, as summarised in Table 2, whereby rudder deflection is positive to the left, and elevon deflection is positive down.

Table 1 States in the flight-simulation model of the Flying V

State	Definition	Unit
X_e	position along earth x-axis	m
Y_e	position along earth y-axis	m
Z_e	position along earth z-axis	m
U_b	speed along body x-axis	m/s
V_b	speed along body y-axis	m/s
W_b	speed along body z-axis	m/s
p	rotational rate around body x-axis	rad/s
q	rotational rate around body y-axis	rad/s
r	rotational rate around body z-axis	rad/s
ϕ	rotational angle around body x-axis	rad
θ	rotational angle around body y-axis	rad
ψ	rotational angle around body z-axis	rad
α	angle of body x-axis w.r.t. airflow vector (around body y-axis)	rad
β	angle of body x-axis w.r.t. airflow vector (around body z-axis)	rad

For this research actuator dynamics were modelled as a first-order system with a time constant of 0.05 s, which approximates the dynamics of a fast elevator actuator. The actuator time constant and rate limit values were based on examples from [26]. Control surface deflections were limited at angles of ± 30 deg, as proposed by Cappuyns [25].

Table 2 Control inputs to the flight-simulation model of the Flying V

Control input	Definition	Unit
$\delta_{r_{left}}$	left rudder angle	deg
$\delta_{e_{o,left}}$	left outboard-elevon angle	deg
$\delta_{e_{i,left}}$	left inboard-elevon angle	deg
$\delta_{e_{i,right}}$	right inboard-elevon angle	deg
$\delta_{e_{o,right}}$	right outboard-elevon angle	deg
$\delta_{r_{right}}$	right rudder angle	deg
T_1	left-engine thrust force	N
T_2	right-engine thrust force	N

C. Altitude Control as a Reinforcement-Learning Problem

The task of the TD3 agent in this research was to track a given altitude reference signal h_{ref}. The reward was defined as Eq. (7), meaning that the agent measures performance through the absolute value of the altitude error $h_{error} = h - h_{ref}$, where h is the measured altitude and h_{ref} the given reference altitude. The reward space was limited by clamping the altitude error value if it was larger than 50 m.

$$r = -|h_{error}|, \quad h_{error} \leq 50 \text{ m} \tag{7}$$

To learn and subsequently perform an altitude reference tracking task the agent could manipulate the Flying V's pitch rate by deflecting the inboard elevons symmetrically. The agent effectively controlled the elevon-deflection *change* $\Delta\delta_{e_i}$, which was restricted to the interval $(-0.6, 0.6)$ deg. The restriction of $\Delta\delta_{e_i}$ effectively imposed a rate limit of 60 deg/s, as the sampling rate was set to 100 Hz. Control of $\Delta\delta_{e_i}$ by the agent was implemented through Eq. (9) (inspired by the work of Dally and van Kampen [12]) in which $\Delta\delta_{e_i min} = -0.6$ and $\Delta\delta_{e_i max} = 0.6$.

$$\delta_{e_i t} = \delta_{e_i t-1} + \Delta\delta_{e_i t}, \quad -0.6 \text{ deg} \leq \Delta\delta_{e_i} \leq 0.6 \text{ deg} \tag{8}$$

$$\Delta\delta_{e_i} = \Delta\delta_{e_i min} + (a+1)\frac{\Delta\delta_{e_i max} - \Delta\delta_{e_i min}}{2}, \quad a \in (0, 1) \tag{9}$$

To learn a relation between aircraft states, the agent's actions and performance with respect to the reward, the agent was given four observations. The first observation available to the agent was the altitude error h_{error}, necessary to distinguish between positive and negative altitude errors, as the reward was defined in terms of the absolute value of h_{error}. The second observation was the measured pitch angle θ, which directly influences change in altitude. The third observation was the pitch rate q, which in turn directly influences pitch angle. Pitch rate is itself directly influenced by the inboard elevon deflection angle δ_{e_i}. The fourth observation was the inboard elevon angle δ_{e_i}, which was added because action a was formulated in terms of a change with respect to an otherwise-unknown current deflection angle. All observations were normalised through before being given to the agent, with the ranges given in Table 3. The observation s is summarised in Eq. (10). The agent-environment interaction resulting from the reward, action and observation specified in this section is summarised by the block diagram shown in Fig. 2.

Table 3 Ranges for normalisation of the observation

Observation	Range	Unit
h_{error}	(-300,300)	m
θ	(-20,20)	deg
q	(-20,20)	deg
δ_{e_i}	(-30,30)	deg

$$s = [h_{error}, \theta, q, \delta_{e_i}]^T \tag{10}$$

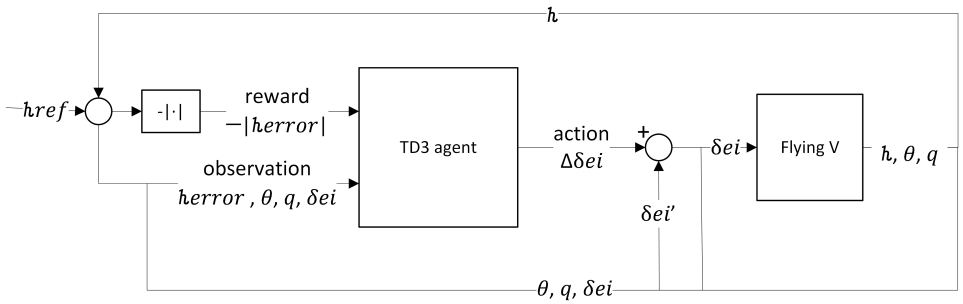


Fig. 2 Agent-environment interaction used to train the TD3 agent for altitude control.

D. Training

Training of the TD3 agent was performed in episodes of 20 seconds each, with a sampling rate of 100 Hz. The reference signal during training was given by Eq. (11), where h_0 is the initial altitude and g_{climb} is the climb gradient. g_{climb} was randomly initialised at the start of each episode and was drawn from a uniform distribution in the interval $[-3000, 3000]$ ft/min. Thereby the climb gradients during training approximately corresponded to the climb gradients prescribed for an Airbus A350-900 [27], the reference aircraft for the Flying V.

$$h_{ref} = \begin{cases} h_0, & \text{if } t < 2 \\ h_0 + g_{climb}(t-2), & \text{otherwise} \end{cases} \tag{11}$$

For this research the Flying V was simulated in the cruise condition, with a neutral centre of gravity location and a weight equal to the maximum takeoff weight. The Flying V was initialised with initial conditions at or near the trimmed state at 10 km at the start of each episode, resulting in the initial conditions summarised in Table 4, whereby random initialisation was based on a uniform distribution. The outboard elevons and rudders were kept constant at zero degrees. The thrust force was kept constant at the trim value for this flight condition, equal to 17479 N per engine. All states that are not included in Table 4 were initialised as zero.

Table 4 Initial conditions

State	Value or range	Unit
M	0.85	-
h_0	10	km
U_b	253.9 ± 1	m/s
W_b	18.2 ± 0.5	m/s
θ	4.1 ± 1	deg
q	0 ± 1	deg/s

Hyperparameters were tuned manually through trial and error. Most hyperparameter values were copied from Fujimoto et al. [13], but some hyperparameters required tuning. Especially the exploration noise required a relatively large increase compared to the value originally used by Fujimoto et al.. The same type of artificial neural network was used to represent the actor and the critic. The networks' first layers consisted of 4 input neurons (one for each observation), followed by two fully connected hidden layers with 128 neurons each. The hidden layers were followed by a ReLu layer, a fully connected output layer, and finally a tanh function. Hyperparameter settings are summarised in Table 5.

The episode-reward curve for training a TD3 agent for altitude control typically did not converge to a steady episode-reward value. Therefore, the learned policy was saved every time the average episode reward reached a value of at least -20,000. Hereby the episode reward was averaged over the last 5 episodes. After a successful training run all saved policies were compared, based on their mean-absolute error when following a predefined test-reference signal. Hereby the test-reference signal included climb and descent sections as well as horizontal sections. The climb and descent gradients were equal to ± 1500 ft/min, comparable to the standard gradients of the Airbus A350-900 at high

Table 5 Hyperparameter settings (adapted from [13])

Parameter	Value
hidden units per layer	128
hidden layers	2
exploration noise	0.4
smoothing noise	0.2
smoothing noise limits	± 0.5
learning rate	1e-3
discount rate	0.99
batch size	100
experience buffer length	1e6
target smoothing factor	0.005
policy update frequency	2
target update frequency	2
optimiser	Adam

altitude [27]). The policy that resulted in the lowest mean-absolute-reference-tracking error for the test-reference signal was used to produce the results presented hereafter.

E. Simulation of Altered Conditions for Robustness Testing

1. Aerodynamic-Model Uncertainty

To simulate the discrepancy that may exist between the aerodynamic model of the Flying V and the real-world Flying V, uncertainty factors were applied to the stability-and-control derivatives on which the flight simulation model for this research was based. Each of the 11 stability-and-control derivatives, as named in Eq. (6), was multiplied by an uncertainty factor ν_i, whereby the left and right surfaces of a control surface pair were given the same factor. This process resulted in an altered aerodynamic coefficient $\hat{C}_X(\dots,\dots)$, as summarised in Eq. (12), where X can be substituted for a different force or moment coefficient.

$$\hat{C}_X(\dots,\dots) = C_X(\dots,\dots)\cdot\nu_i, \quad \nu_i \sim \mathcal{N}(1,\sigma) \quad \text{and} \quad \sigma \in [0.0625, 0.125, 0.25] \tag{12}$$

Hereby the means of the stability-and-control derivatives of the Flying V were assumed to be at the values obtained from the vortex-lattice method applied to a numerical model of the Flying V. As reported by van Overeem and van Kampen [28] the maximum error in the aerodynamic model of the Flying V used for the present research is estimated to be 25%. To simulate an uncertainty with a maximum of 25% the uncertainty factors were drawn from a normal distribution with a standard deviation of 0.125, such that 95% of the uncertainty-factor samples would fall in the aerodynamic-error range $[-25\%, 25\%]$. After the main set of simulation runs with a standard deviation of 0.125, two other sets of simulations were run, of which one had a standard deviation of 0.0625, to simulate the scenario that the actual maximum aerodynamic-model uncertainty is less than 25%, and the other had a standard deviation of 0.25, to simulate the scenario that the actual maximum aerodynamic-model uncertainty is more than 25%. For each of the three standard deviations a set of 100 simulations of 100 s each was run. During each simulation of 100 s the altitude-reference signal remained horizontal for $0 \leq t < 10$ s, descended at a rate of 1500 ft/min for $10 \leq t < 30$ s, was again horizontal for $30 \leq t < 50$ s, climbed at a rate of 1500 ft/min for $50 \leq t < 70$ s and was horizontal for $70 \leq t < 100$ s. This reference signal will hereafter be used as the standard reference signal for testing the altitude controller developed for this research and will be referred to as reference signal *RS*.

2. Sensor Noise

As the Flying V is still under development it is hard to precisely estimate the noise that will be present in the real-world aircraft. However, by using measurements of the sensor noise in other aircraft as a reference, a first assessment

of the robustness of the altitude controller developed in this research can be made. To assess the robustness of the altitude controller, sensor noise was not simulated during training, but only added to the simulated Flying V during tests for robustness to sensor noise.

Research by Grondman et al. [29] into sensor noise in the Cessna Citation PH-LAB aircraft was used as a reference for simulating sensor noise in the present research, because of the well-documented noise bias and standard deviation values for several types of sensors. Noise was simulated for the present research by adding a noise sample at each time step, from normal distributions with the biases and standard deviations shown in Table 6.

Table 6 Sensor noise [29]

Sensor	Bias	Standard deviation	Unit
altitude	8.0e−3	6.7e−2	m
pitch angle	4.0e−3	3.2e−5	rad
pitch rate	3.0e−5	6.3e−4	rad/s

3. Altered Initial Conditions

Altered initial conditions were simulated by randomly initialising states at the start of each of 100 simulation runs of 100 s in duration each. Hereby the trimmed initial condition I_{trim} was altered through Eq. (13), resulting in altered initial condition I_{alt}. ξ is a scaling factor drawn from a uniform distribution.

$$I_{alt} = I_{trim} + \xi \cdot I_{max}, \quad \xi \sim \mathcal{U}(-1, 1) \tag{13}$$

III. Results and Discussion

This section presents results obtained from simulated tests with the altitude controller developed for this research, accompanied with a discussion of these results. Firstly, subsection III.A shows and discusses the altitude-tracking performance of the controller under nominal conditions. Secondly, subsection III.B shows and discusses the robustness of the controller to aerodynamic-model error. Thirdly, subsection III.C shows and discusses the robustness of the controller to sensor noise, in combination with alternative reference-signal shapes. Fourthly, subsection III.D shows and discusses the robustness of the controller to altered initial conditions. Lastly, subsection III.E shows and discusses the stability during training and the sampling efficiency of TD3, as observed in this research.

A. Altitude-Tracking Performance under Nominal Conditions

In [30] the Federal Aviation Authority (FAA) specifies that for aircraft to be authorised to fly in Reduced Vertical Separation Minimum airspace (between 8.8 and 12.5 km altitude), an automatic altitude control system should be

> "... capable of controlling aircraft height within a tolerance band of ±65 ft (±20 m) about the acquired altitude when the aircraft is operated in straight and level flight under nonturbulent, nongust conditions.".

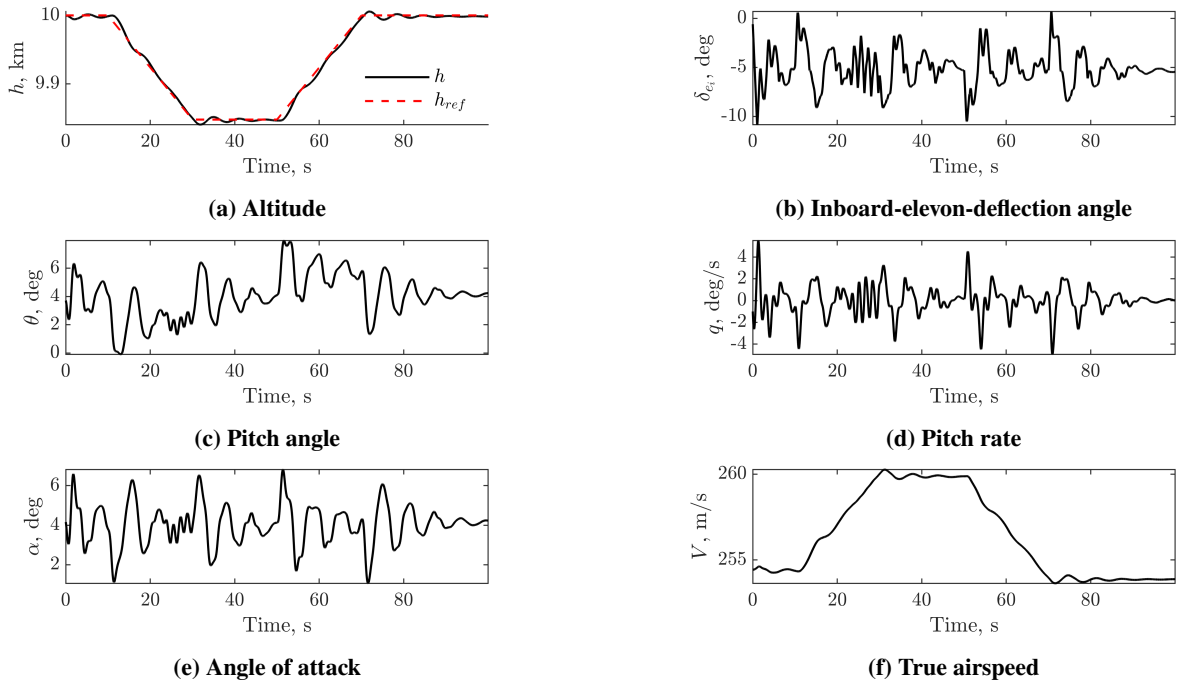
Therefore, a maximum-absolute-altitude-tracking error of 20 m, under nominal flight conditions, is used to determine whether the altitude controller in this research is successful. Figure 3 shows the simulated response of an offline-trained TD3 agent to a given altitude-reference signal with climb and descent gradients of 1500 ft/min, whereby the Flying V started from trimmed initial conditions and thrust was kept constant during the whole manoeuvre. Figure 3a shows that the agent is able to track the altitude-reference signal in both horizontal, descent, and climb phases. The mean-absolute-altitude-tracking error for the reference signal shown in Fig. 3a was 3.0 m, and the maximum-absolute-altitude-tracking error was 11.6 m. Therefore, even as the reference signal included climb and descent phases, rather than level flight as specified by the FAA [30], the test shown in Fig. 3 indicates that the altitude controller is successful under nominal conditions.

The measured altitude h in Fig. 3a oscillated around the reference altitude h_{ref} during the limited simulation time shown. However, the response for $t \geq 70$ s shows that the oscillations died out over time. Oscillations were also visible in the angle of attack (Fig. 3e). As angle of attack is related to load factor, the oscillations may cause passenger discomfort if the policy that was simulated to produce Fig. 3 were applied in the real-world aircraft. Moreover, as

Fig. 3b shows, oscillations were also present in the inboard-elevon-deflection angle during the whole manoeuvre, which may increase actuator wear. Passenger comfort and actuator wear were not part of the scope of the current research however. Therefore the reward function (Eq. (7)) did not include a penalty for oscillations.

One source of the oscillations is likely the short-period eigenmode of the Flying V. At the flight condition simulated in Fig. 3 the short-period frequency is approximately 0.34 Hz, as determined by simulating an impulse response of the Flying V at 10 km altitude, starting from a trimmed condition, and subsequently measuring the frequency of the short-period oscillation in the pitch rate[*]. The pitch-rate signal shown in Fig. 3d has a spike in the frequency content at a frequency of 0.34 Hz[†], which may be related to the short period. However, a larger spike in the frequency content of the pitch-rate signal shown in Fig. 3d, compared to the spike at 0.34 Hz, is at a frequency of 0.14 Hz, which is not close to the short-period frequency nor the phugoid frequency of 0.0084 Hz. Therefore, the oscillations with a frequency of 0.14 Hz are likely caused mainly by the oscillating inboard-elevon deflections resulting from the policy learned by the TD3 agent.

Oscillations in the policy learned by the agent were already greatly reduced by formulating the action as a change in elevon angle. Earlier attempts to allow the agent to manipulate the elevon-deflection angle directly consistently resulted in a policy with excessively high-gain control. Adding a penalty term for high elevon-deflection rates in the reward function was not found to offer a solution, as the added penalty term caused learning to be unsuccessful. Thereby it is hypothesised that this added penalty term makes the reinforcement-learning problem too complex for the TD3 agent.



(a) Altitude **(b) Inboard-elevon-deflection angle**

(c) Pitch angle **(d) Pitch rate**

(e) Angle of attack **(f) True airspeed**

Fig. 3 Nominal response of the TD3 altitude-control agent to an altitude-reference signal with climb and descent gradients of 1500 ft/min.

B. Robustness to Aerodynamic-Model Error

Figure 4 shows the mean-absolute-altitude-reference-tracking errors for three levels of aerodynamic uncertainty. The middle boxplot in Fig. 4 corresponds to simulations of the previously estimated aerodynamic-model-uncertainty range of the Flying-V model used for this research. The right boxplot shows the scenario that the actual uncertainty range is higher than the previously estimated range. As Fig. 4 shows, the mean, third quartile, and highest outlier increase with aerodynamic uncertainty. The outlier that can be seen in Fig. 4 at a mean-absolute error of 3.6 m, for a standard

[*]The eigenmode frequencies were obtained from the non-linear simulation model. Therefore, the mentioned eigenmode frequencies are an approximation of the values that would have been obtained by first linearising and reducing the simulation model.

[†]Spikes in the frequency content were found from the periodogram of the pitch-rate signal.

deviation of $\sigma = 0.25$, represents the worst-case scenario that came forth from the simulations that were run for this research.

As van Overeem and van Kampen [28] adopted a similar methodology for simulating aerodynamic-model uncertainty to the present research, with also a maximum standard deviation of 0.25, the results presented in this paper indicate that the TD3 controller presented here is robust to at least the same level of aerodynamic-model uncertainty as the INDI controller presented by [28]. However, differences between the research of van Overeem and van Kampen and the present research in the flight control task and the flight conditions simulated limits the possibilities for a direct comparison.

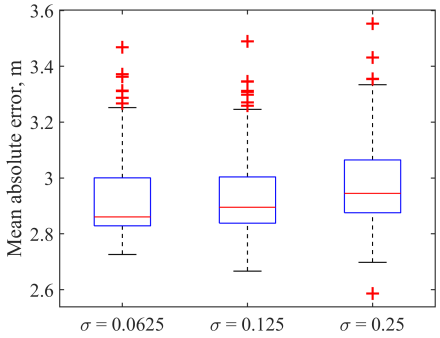


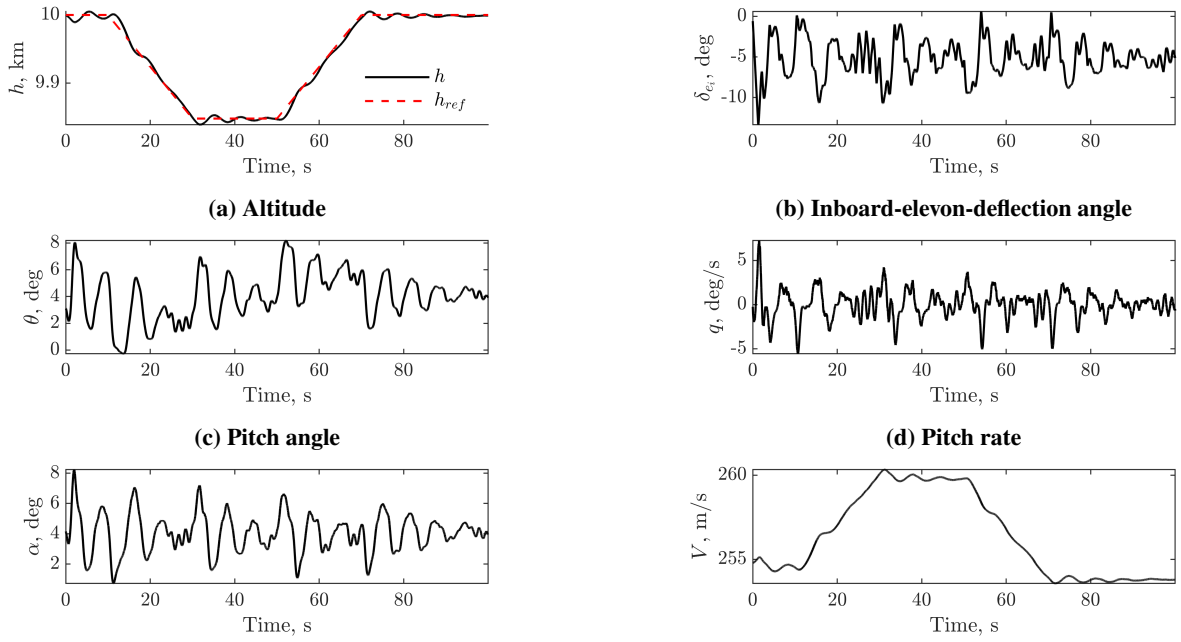
Fig. 4 Altitude-reference-signal-tracking error for three levels of aerodynamic-model uncertainty. The middle boxplot corresponds to a simulation of the uncertainty range estimated in previous research. The left and right boxplots show the scenarios that the uncertainty range is lower or higher than was estimated.

To assess whether flight control is safe in the worst-case scenario in terms of aerodynamic uncertainty, the response corresponding to the simulation with a mean-absolute error of 3.6 m was plotted, shown in Fig. 5. The signal shown in Fig. 5a shows that the measured altitude h remains close to the reference altitude h_{ref}. The maximum-absolute-altitude-tracking error was 14.4 m, which is smaller than 20 m and thereby complies with the requirement set in subsection III.A. The transient response dies out and the steady-state response has a small error, as can be most clearly observed at $t > 70$ s in Fig. 5a. Moreover, Fig. 5e shows that the angle of attack does not come near the stall angle of 18 degrees [25], nor the angle of attack at which the Flying V has a pitch-break tendency, equal to 20 degrees [31].

C. Robustness to Sensor Noise and Alternative Reference-Signal Shapes

As mentioned in subsection III.A the nominal mean-absolute-reference tracking error was 3.0 m. Figure 6 was simulated for the same reference signal and therefore shows that sensor noise (which the TD3 agent did not encounter during training) results in an increase of approximately 0.02 m in the mean-absolute reference tracking error. The figure also shows that the worst outlier for the 100 simulations that were run for this research was at an increase of 0.06 m in the mean-absolute-reference tracking error. Therefore, these simulations show that the altitude controller developed for the present research is robust to the simulated sensor noise.

To assess how the shape of the altitude-reference signal affects tracking error and safety of the control policy, the altitude controller developed for this research was simulated for two alternative reference-signal shapes, with simulated sensor noise. The same policy as was simulated to produce the results presented in previous sections of this paper was used, so the agent had not encountered these reference signals nor experienced the sensor noise during training. Figure 7 shows the response to a sinusoidal signal with a frequency of 0.01 Hz and an amplitude of 500 m, thereby simulating average climb and descent gradients of approximately 4000 ft/min. Figures 7b-7e show that oscillations in the control signal and the Flying V's attitude subside faster than for the reference signal shown in Fig. 5, which may be explained by the more gradual change in flight path angle corresponding to the sinusoidal altitude-reference signal. The oscillations that start after around 80 s may be explained by the relatively large decrease in airspeed (larger than for the nominal signal, see Fig. 3f), which causes aerodynamic damping and control effectiveness to decrease. The mean-absolute-reference-tracking error for this sinusoidal signal was 13.3 m. The increase in error with respect to standard reference signal *RS* is reasonable because of the higher gradients of the sinusoidal signal. The maximum-absolute-reference-tracking for the sinusoidal signal was 50 m, but this maximum error occurred at the start



(a) Altitude **(b) Inboard-elevon-deflection angle**

(c) Pitch angle **(d) Pitch rate**

(e) Angle of attack **(f) True airspeed**

Fig. 5 Worst response out of 100 runs in terms of altitude-tracking error for the TD3 altitude-control agent on a model of the Flying V with a simulated maximum aerodynamic-model error of 50%.

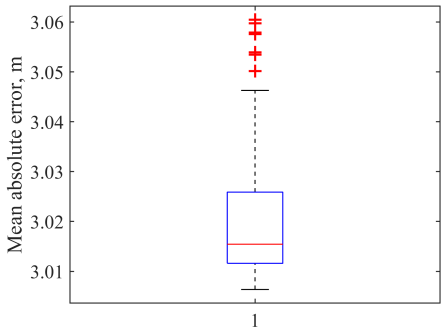
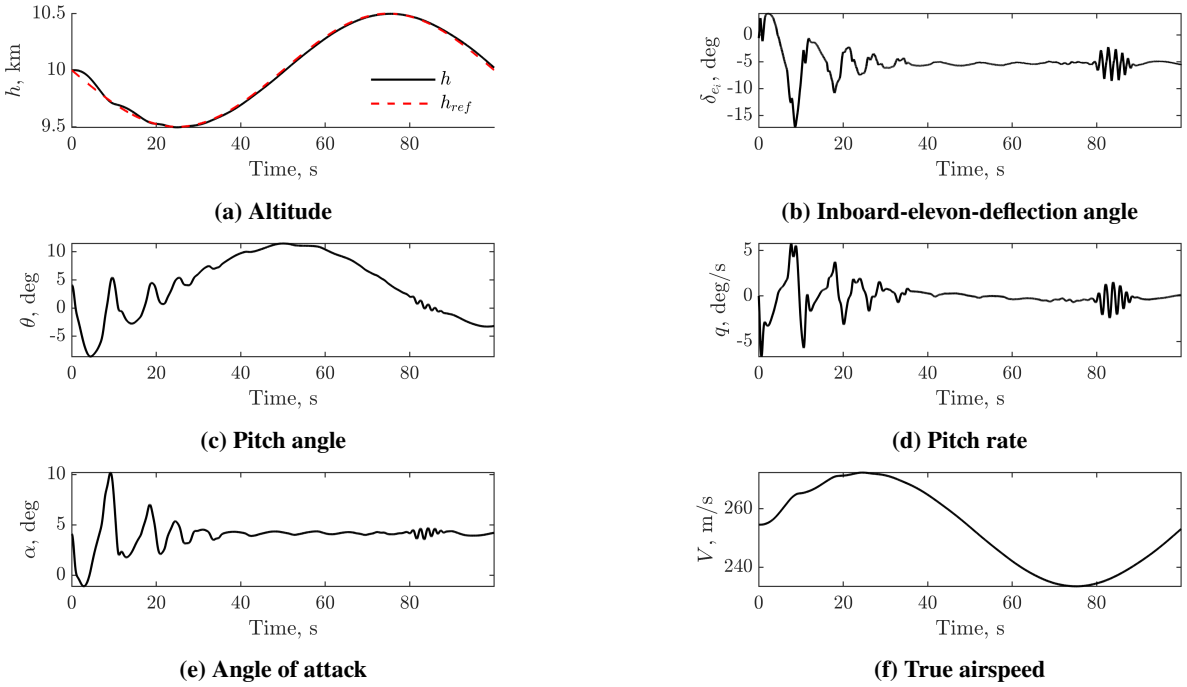


Fig. 6 Robustness to sensor noise and bias measured in terms of the altitude-reference-signal-tracking error. The error for the same reference signal with ideal sensors was 3.0 m.

of the simulation (after 3.6 s), when a sudden change in climb gradient of -4000 ft/min was commanded. After allowing a settling time of 15 s to adapt to the initial sudden change in climb gradient, the maximum-absolute-reference-tracking was 25 m.

Figure 8 shows the response to a triangular altitude-reference signal with a gradient of 1500 ft/min and simulated sensor noise, for which the mean-absolute-reference-tracking error was 5.9 m and the maximum-absolute-reference-tracking error was 22.2 m. The triangular signal has more extreme changes in climb gradient than a realistic reference signal that would be encountered in the real world, but it provides insight into the safety of the controller in case sudden changes in climb gradient are commanded by the pilot. Although oscillations are visible in all signals shown in Fig. 8, the angle of attack shown in Fig. 8e does not come near the stall or pitch-break angle and the airspeed shown in Fig. 8f remains near the nominal value of 254 m/s, so the controller keeps the aircraft in a safe state. Furthermore, as the maximum-absolute-reference-tracking error for a sinusoidal signal exceeds the requirements stated in subsection III.A for level flight (i.e., a horizontal reference signal) by only 2.5%, and the error for a triangular signal exceeds the requirements by only 1%, the controller is considered robust to alternative reference signals with sensor noise.



(a) Altitude

(b) Inboard-elevon-deflection angle

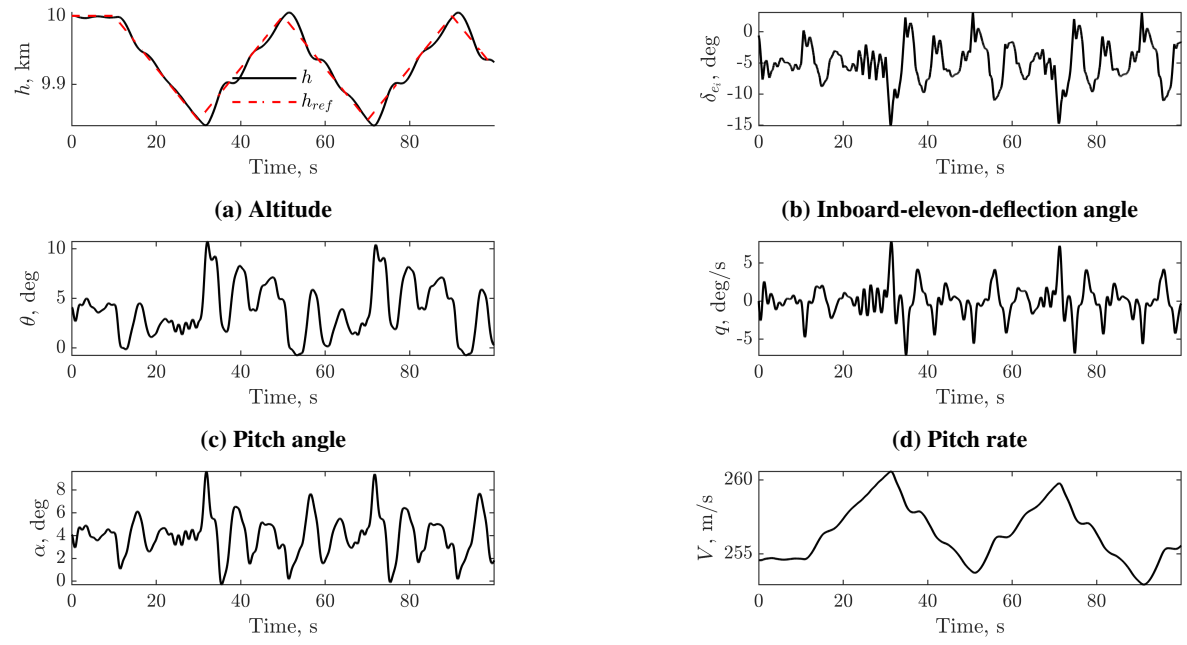
(c) Pitch angle

(d) Pitch rate

(e) Angle of attack

(f) True airspeed

Fig. 7 Robustness to sensor noise and reference-signal shape, shown through the response to a sinusoidal reference signal.



(a) Altitude

(b) Inboard-elevon-deflection angle

(c) Pitch angle

(d) Pitch rate

(e) Angle of attack

(f) True airspeed

Fig. 8 Robustness to sensor noise and reference-signal shape, shown through the response to a triangular reference signal with climb gradients of 1500 ft/min.

Random initialisation of the flight condition during training, as specified in subsection II.D, was found to have a large positive effect on robustness to alternative reference-signal shapes and altered initial conditions (see subsection III.D) during testing. However, randomly initialising the flight condition was found to only work for small ranges near the trimmed condition, as the problem became too hard for the agent to find a successful policy if the initial flight condition differed too much from the trimmed condition. Therefore, it is recommended to experiment with the range in initial conditions implemented during training if a similar future research project is undertaken, to find a range that is not too large to hinder training, nor too small to diminish the effect on robustness of the resulting policy.

D. Robustness to Altered Initial Conditions

In the real world the Flying V may not always be in a trimmed state when the altitude controller is engaged, due to atmospheric disturbances or pilot inputs that temporarily force the Flying V to deviate from the trimmed state. Therefore the robustness of the altitude controller to initial conditions that deviate from the trimmed state was tested, hereafter referred to as *altered initial conditions*. Hereby the states that directly affect longitudinal motion - U_b, W_b, q and θ - and the altitude h were altered. A maximum deviation I_{max} (see Eq. (13)) was chosen for each altered state, as shown in Table 7. Figure 9 shows that for some combinations of altered initial conditions the mean-absolute-reference-tracking error is higher than the nominal error of 3.0 m for the same reference signal. However, the increase in error is limited to a maximum of 1.3 m. Moreover, Fig. 9 shows that, for reference signal *RS*, the mean-absolute-reference-tracking error for some combinations of initial conditions is lower than for the nominal initial conditions.

Table 7 Limits of altered initial conditions

Initial condition	I_{max}	Unit
U_b	10	m/s
W_b	1	m/s
θ	2	deg
q	2	deg/s
h	100	m

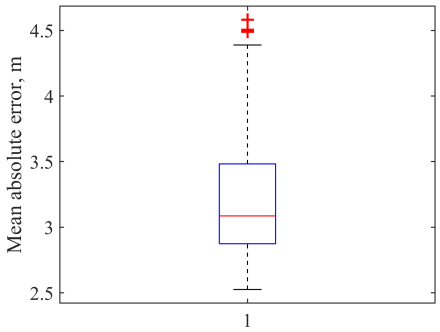
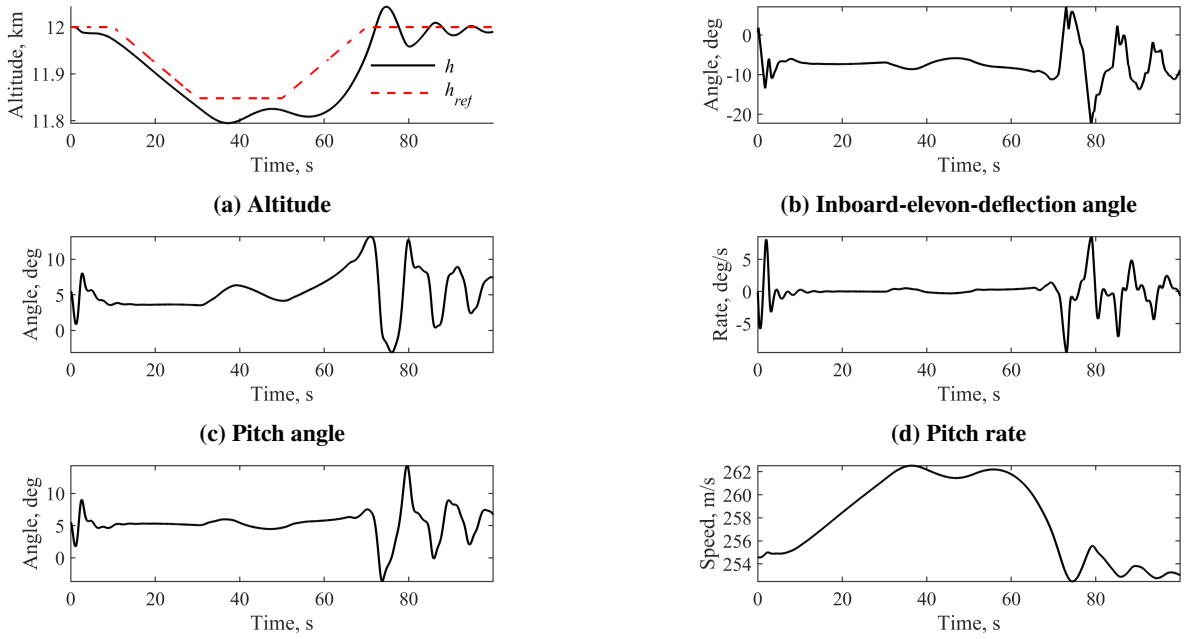


Fig. 9 Robustness to random combinations of initial horizontal and vertical airspeeds, pitch angles, pitch rates and altitudes that differ from the nominal values, measured in terms of the altitude-reference-tracking error in response to reference signal *RS*.

To assess how well the altitude controller can generalise behaviour to initial conditions that differ more extremely from the conditions that the TD3 agent was trained for (namely trimmed cruise flight at 10 km altitude), additional simulations were run for more extremely altered initial altitudes. Simulation starting at 8 km altitude for reference signal *RS* resulted in a mean-absolute-reference-tracking error of 2.8 m, which is 0.3 m smaller than the error at the nominal altitude of 10 km. Simulation at 12 km altitude, on the other hand, resulted in a mean-absolute-reference-tracking error of 27.7 m, which is 24.7 m larger than the error at the nominal altitude of 10 km. The smaller error at 8 km altitude may be explained by the higher air density at lower altitude, resulting in more control effectiveness and more damping of

eigenmodes, while the larger error at 12 km altitude may similarly be explained by less control effectiveness and less damping of the eigenmodes. As Fig. 10a shows, the agent is not able to accurately follow the commanded reference signal. However, as Fig. 10e shows, the controller does keep the Flying V in a safe flight regime, as the angle of attack does not come near the stall angle of 18 deg. Moreover, airspeed barely falls to values lower than the nominal airspeed, as Fig. 10f shows.



(a) Altitude

(b) Inboard-elevon-deflection angle

(c) Pitch angle

(d) Pitch rate

(e) Angle of attack

(f) True airspeed

Fig. 10 Response of the altitude controller when initialised at an altitude of 12 km, which is 2 km higher than the altitude it was trained for.

E. Training Stability and Sampling Efficiency of TD3

Figure 11 shows the moving average of the episode reward of the training run that produced the TD3 agent used to produce the results presented in this paper, averaged over 50 episodes of 20 s each. The figure shows that the TD3 agent improved its policy relatively consistently during the first approximately 150 episodes, but did not converge to that policy. Only after approximately 1300 episodes did the policy temporarily converge to a policy that more consistently resulted in relatively high episode rewards. Three aspects visible from the reward curve shown in Fig. 11 are typical for the reward curves of the several flight control problems tested for the research presented in this paper.

Firstly, fast initial improvement in the policy did not immediately lead to finding a successful policy, but was usually followed by divergence to unsuccessful policies. Secondly, once a successful policy was found, the agent eventually diverged from this policy. Thirdly, sample efficiency was low, leading to long training times. Finding the policy used to produce the results for this research (at 1466 episodes) took approximately 18 hours on an Intel Xeon CPU E5-1620 v3 @ 3.50GHz.

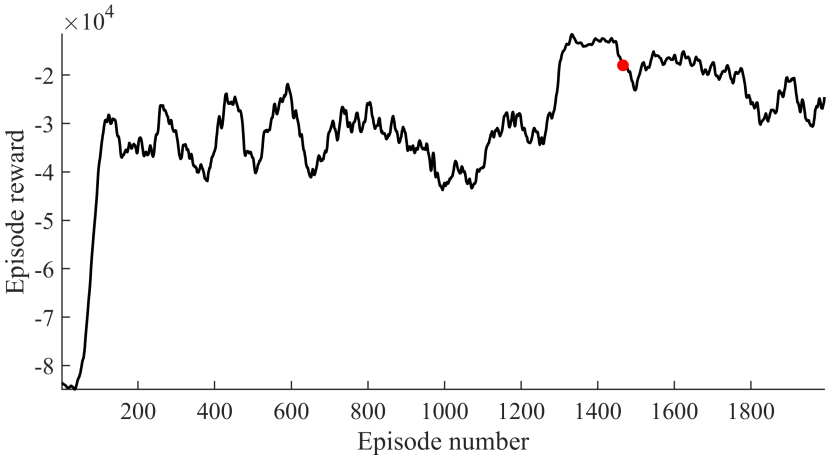


Fig. 11 Average reward during training of a TD3 agent for altitude control of the Flying V. The policy obtained at 1466 episodes (marked in red) was used to produce the results presented in this paper.

IV. Conclusion

The research presented in this paper shows that a single-loop controller based on TD3 can learn altitude control of a non-linear simulation model of the Flying V in an offline setting, and satisfies the set requirement of a maximum-absolute-altitude-tracking error of 20 m. Hereby the controller only observes the altitude-tracking error, the pitch angle, the pitch rate, and the elevon-deflection angle. The results also show that the controller is robust to aerodynamic-model error, sensor noise, various shapes of the altitude-reference signal, and unfavourable initial flight conditions. Therefore, the research presented in this paper suggests that deep-reinforcement learning and in particular TD3 has the potential to be used for creating robust flight controllers. However, several questions remain open to investigation.

To build on this research we recommend to investigate how robust a controller with the structure proposed in this research is to faults and atmospheric disturbances that were not simulated for this research. Furthermore, to increase the applicability of the controller to a wider variety of flight regimes we recommend to investigate the addition of airspeed and height in the observation and training at various altitudes and airspeeds. To prevent the problem from becoming too inconsistent during training and too complex for the TD3 agent to find a successful policy, we suggest the use of a learning curriculum. The learning curriculum may include progressively more difficult initial conditions, atmospheric disturbances and a varying aerodynamic model to increase robustness of the learned policy. Lastly, the use of TD3 or similar RL algorithms for lateral-directional control or a combination of lateral-directional and longitudinal control may be investigated.

By continuing research into reinforcement learning for flight control, with the use of more sample-efficient reinforcement-learning algorithms, the development of new methods to explain control policies, and the creation of standardised practices to develop reinforcement-learning-based flight controllers, researchers may bring reinforcement learning for flight control to a level at which it can be used in industry. In that way, safer, more autonomous flight of passenger aircraft with novel airframe designs such as the Flying V may one day become reality.

References

[1] Lee, D. S., Pitari, G., Grewe, V., Gierens, K., Penner, J. E., Petzold, A., Prather, M. J., Schumann, U., Bais, A., Berntsen, T., Iachetti, D., Lim, L. L., and Sausen, R., "Transport impacts on atmosphere and climate: Aviation," *Atmospheric Environment*, Vol. 44, No. 37, 2010, pp. 4678–4734. https://doi.org/10.1016/J.ATMOSENV.2009.06.005.

[2] Airbus, "Cities, airports & aircraft," Tech. rep., Airbus S.A.S., 2019. URL https://www.airbus.com/aircraft/market/global-market-forecast.html.

[3] Zheng, X. S., and Rutherford, D., "Fuel burn of new commercial jet aircraft: 1960 to 2019," Tech. rep., The International Council on Clean Transportation, 9 2020.

[4] Martinez-Val, R., Palacin, J. F., and Perez, E., "The evolution of jet airliners explained through the range equation," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 222, No. 6, 2008, pp. 915–919. https://doi.org/10.1243/09544100JAERO338.

[5] Martinez-Val, R., "Flying wings. An new paradigm for civil aviation?" *Acta Polytecnica*, Vol. 47, No. 1, 2007. URL http://ctn.cvut.cz/ap/.

[6] Benad, J., "The Flying V - A new Aircraft Configuration for Commercial Passenger Transport," Tech. rep., Airbus Operations GmbH and Berlin University of Technology, 2015. https://doi.org/10.25967/370094.

[7] Faggiano, F., Vos, R., Baan, M., and Van Dijk, R., "Aerodynamic design of a flying V aircraft," *17th AIAA Aviation Technology, Integration, and Operations Conference, 2017*, American Institute of Aeronautics and Astronautics Inc, AIAA, 2017. https://doi.org/10.2514/6.2017-3589.

[8] Wood, R. M., and Bauer, S. X., "Flying wings / flying fuselages," *39th Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics Inc., 2001. https://doi.org/10.2514/6.2001-311.

[9] Lee, J., and Kampen, E.-J., "Online reinforcement learning for fixed-wing aircraft longitudinal control," *AIAA Scitech 2021 Forum*, 2021, pp. 1–20.

[10] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Graepel, T., and Hassabis, D., "Mastering the game of Go without human knowledge," *Nature*, Vol. 550, No. 7676, 2017, pp. 354–359. https://doi.org/10.1038/nature24270.

[11] Bøhn, E., Coates, E. M., Moe, S., and Johansen, T. A., "Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization," *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 523–533. https://doi.org/10.1109/ICUAS.2019.8798254, URL http://arxiv.org/abs/1911.05478http://dx.doi.org/10.1109/ICUAS.2019.8798254.

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

[13] Fujimoto, S., van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," *35th International Conference on Machine Learning*, Stockholm, 2018. URL http://arxiv.org/abs/1802.09477.

[14] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[15] van Hasselt, H., "Double Q-learning," *Advances in neural information processing systems*, Vol. 23, 2010, pp. 2613–2621.

[16] van Hasselt, H., "van Hasselt, Hado Philip. Insights in reinforcement rearning: formal analysis and empirical evaluation of temporal-difference learning algorithms," Ph.D. thesis, Utrecht University, 2011.

[17] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P., "Benchmarking Deep Reinforcement Learning for Continuous Control," *33rd International Conference on Machine Learning*, New York, 2016. URL http://arxiv.org/abs/1604.06778.

[18] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D., "Deep reinforcement learning that matters," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, AAAI press, 2018, pp. 3207–3214.

[19] Lazaridis, A., Fachantidis, A., and Vlahavas, I., "Deep Reinforcement Learning: A State-of-the-Art Walkthrough," *Journal of Artificial Intelligence Research*, Vol. 69, 2020, pp. 1421–1471.

[20] Ball, P., and Roberts, S., "OffCon3: What is state-of-the-art anyway?" *arXiv*, 2021. URL http://arxiv.org/abs/2101.11331.

[21] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *arXiv*, 2018. URL http://arxiv.org/abs/1801.01290.

[22] Dong, Y., Shi, Z., Chen, K., and Yao, Z., "Self-learned suppression of roll oscillations based on model-free reinforcement learning," *Aerospace Science and Technology*, Vol. 116, 2021. https://doi.org/10.1016/j.ast.2021.106850.

[23] Delft University of Technology, "Flying-V," , 2019. URL https://www.tudelft.nl/lr/flying-v.

[24] Van Overeem, S., Wang, X., and van Kampen, E.-J., "Modelling and Handling Quality Assessment of the Flying-V Aircraft," *AIAA Scitech 2022 Forum*, Delft University of Technology, 2022.

[25] Cappuyns, T., "Handling qualities of a Flying V configuration," Tech. rep., TU Delft Aerospace Engineering, 2019.

[26] Stevens, B., Lewis, F., and Johnson, E., *Aircraft control and simulation: Dynamics, controls design, and autonomous systems: Third edition*, 2015. https://doi.org/10.1002/9781119174882.

[27] Skybrary, "AIRBUS A350-900 | SKYbrary Aviation Safety," , 2012. URL https://skybrary.aero/aircraft/a359.

[28] van Overeem, S., and van Kampen, E., "Modelling, Control, and Handling Quality Analysis of the Flying-V," Tech. rep., Delft University of Technology, Delft, 2022. URL http://resolver.tudelft.nl/uuid:7fd04eec-41d4-4967-b246-89fdfac2446e.

[29] Grondman, F., Looye, G. H., Kuchar, R. O., Chu, Q. P., and van Kampen, E. J., "Design and flight testing of incremental nonlinear dynamic inversion based control laws for a passenger aircraft," *AIAA Guidance, Navigation, and Control Conference, 2018*, Vol. 0, American Institute of Aeronautics and Astronautics Inc, AIAA, 2018. https://doi.org/10.2514/6.2018-0385.

[30] Carty, R. C., "Advisory Circular Subject: Authorization of Aircraft and Operators for Flight in Reduced Vertical Separation Minimum (RVSM) Airspace," Tech. rep., Federal Aviation Administration, 2019.

[31] Palermo, M., and Vos, R., "Experimental aerodynamic analysis of a 4.6%-scale flying-v subsonic transport," *AIAA Scitech 2020 Forum*, Vol. 1 PartF, American Institute of Aeronautics and Astronautics Inc, AIAA, 2020. https://doi.org/10.2514/6.2020-2228.

II

Literature study & preliminary analysis*

***The literature study & preliminary analysis have already been graded.**

2

Reinforcement learning: from fundamentals to state of the art

To make an informed choice among existing reinforcement learning (RL) methods a solid background knowledge is required (see also research question 1(c). This chapter sets out to introduce the fundamentals of RL to a reader with no or little prior knowledge of the field. The language used in this chapter to explain RL is based on the language used by artificial intelligence researchers and is heavily influenced by Sutton and Barto [82]. In the field of optimal control slightly different terms may be used for some RL concepts, but the fundamental ideas of RL described in this chapter are applicable to both fields. This text tries to indicate some of the differences where appropriate. After covering the fundamentals this chapter builds up to the latest developments in the field of RL, focussing on model-free deep RL methods. The text will try to argue that this class of methods is most promising within the feasible scope of the proposed research.

The outline of this chapter is as follows. Section 2.1 defines the basic principle of RL and demarcates it from other forms of machine learning. Section 2.2 defines the RL problem in a mathematical way with the *Markov decision process*. Section 2.3 then introduces *dynamic programming*, a basic way to solve the RL problem on which all later RL algorithms are based. Section 2.4 introduces *temporal-difference* learning, a method that allows the construction of model-free RL algorithms. Thereafter section 2.5 explains a way of constructing potentially more sample efficient RL algorithms through *off-policy* learning and introduces the first algorithm of the chapter: Q-learning. Section 2.6 sets out how RL can be combined with function approximators to extend RL methods to large state and action spaces and introduces the algorithm DQN. An introduction to *policy-based* methods follows thereafter in section 2.7, which also introduces the algorithm REINFORCE. The intersection of policy-based and value-based methods is discussed in section 2.8 with *actor-critic* methods, including the three algorithms DDPG, TD3 and SAC. Section 2.9 follows with a brief discussion of RL methods that combine learning with planning. Lastly section 2.10 gives a comparison of state-of-the-art model-free RL methods DDPG, TD3 and SAC and section 2.11 offers a perspective on the research field of model-free RL. Sections of text that specifically discuss an algorithm are demarcated by a black square ("■") at the end.

2.1. Basic principle and demarcation

RL is a bio-inspired machine learning method, based on the principle of trial-and-error learning. Just like a dog learns that if he sits he gets a sweet, or like a child learns that if he sticks his finger in a flame it hurts, an RL *agent* learns through interaction with its *environment* that certain *actions*, performed in a certain state, produce positive *rewards* and others negative ones. The terms *agent*, *environment* and *action* are analogous to the *controller*, *controlled system* and *control signal* in a control problem.

RL is different from other machine learning methods because it is oriented towards achieving a certain goal (defined by the designer) without being told which actions to take to achieve that goal. This way of formulating a machine learning problem allows for more creative solutions than defining it as a supervised learning problem (e.g., deep learning), the field most studied by machine learning researchers. In supervised learning the objective is to generalise actions provided by an external expert to unknown situations. This kind of learning works very well in situations in which labelled training data is accurate and plentiful, but

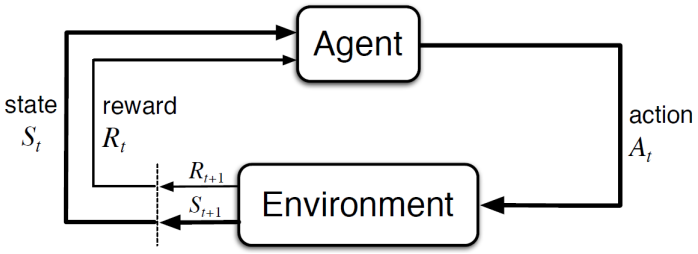


Figure 2.1: A schematic representation of the agent-environment interaction. After receiving reward r_t in state s_t the agent performs an action a_t. The environment responds with a next state s_{t+1} and a reward r_{t+1}, which can in turn be sensed by the agent. From [82].

it does not allow learning from interaction with an unknown environment. Some flight control problems cannot be adequately solved by a human pilot or a PID controller (the external expert), others are unsafe to demonstrate and yet others are unpredictable. For these situations it is infeasible to produce labelled data, posing a problem for supervised learning but not necessarily for RL.

To fully demarcate RL it must be noted that it does not fit the class of unsupervised learning. Unsupervised learning is concerned with learning a hidden structure from unlabelled data. While RL also does not rely on labelled data, its objective is to maximise a reward signal, not to uncover a hidden structure.

Another characteristic that sets RL apart from supervised and unsupervised learning is the dilemma of *exploration* and *exploitation*. To maximise the reward signal an agent might *exploit* the actions it knows to produce high rewards, but that might mean it misses other actions that produce higher rewards, but which it has not visited yet. To discover these actions the agent must *explore*. To find an optimal solution without doing too much damage on the way a balance must be found between exploration and exploitation.

This section ends with fully defining the RL system of agent-environment interaction through four subelements: the *policy*, the *reward signal*, the *value function* and optionally a *model* of the environment. The policy is a mapping from states to actions, which are allowed to be stochastic. The reward signal is a scalar signal that defines the goal of the agent and may also be stochastic. The sole objective of the agent is to maximise the total reward in the long run. The value function specifies what is good in the long run; the value of a state is the total reward an agent can expect in the future, starting from that state. Value estimation is a vital component of any RL algorithm. A value function may be interpreted as the long-term memory of an agent, while the reward signal is his short-term memory. Finally a model is an approximation or a prediction of the environment dynamics, such as for example a prediction of the state transition, given a state and action. It is used for *planning*. Reinforcement-learning methods that do or do not use models are known as *model-based* and *model-free* methods respectively. These four subelements will return in the discussion of almost all RL concepts hereafter. The next section will introduce a more formal definition of the RL problem: the Markov Decision Process.

2.2. Markov decision processes

This section introduces a mathematical formulation for the RL problem: the Markov decision process or MDP, first connected to RL by Andreae [4] and later integrated by Watkins [96] to the form most used today. Any method that aims to solve an MDP can be considered an RL method. MDPs involve key concepts integrated in any RL algorithm such as *returns*, *policies*, *value functions* and *Bellman equations*. Each will be addressed in this section.

Figure 2.1 displays the agent-environment interaction central to RL. For each state s_t at time step t the agent has a set of actions $a(s_t)$ to choose from. With an action $a_t \in a(s_t)$ it can influence his environment. The environment will respond with a next state s_{t+1} and a reward r_{t+1}, which are both sensed by the agent. The agent-environment interaction can be summarised by the following sequence, known as a trajectory.

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots \tag{2.1}$$

In most RL methods states are assumed to have the *Markov property*. Such states contain all information on past interactions with the environment that influences the future. In an MDP the probabilities of next state s' and reward r can therefore be formulated as $p(s', r \mid s, a)$, the probability that s' and r follow from choosing action a in state s at time t:

$$p(s', r \mid s, a) \doteq Pr\{s_t = s', r_t = r \mid s_{t-1} = s, a_{t-1} = a\}, \tag{2.2}$$

where the set of actions and states may be finite or infinite.

In trying to achieve a long-term goal it is not always beneficial to aim for the highest immediate reward. The long-term cumulative reward, called the *return* G_t, is usually a better target. RL methods generally aim to maximise the expected discounted return, for infinite-horizon problems defined as:

$$\mathbb{E}[G_t] \doteq \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right], \tag{2.3}$$

with $0 \leq \gamma \leq 1$ the *discount rate*. The higher the discount rate, the higher rewards far in the future are weighted. Returns have a recursive relationship that is used in many RL algorithms, given by:

$$G_t = r_{t+1} + \gamma G_{t+1} \tag{2.4}$$

Which actions the agent chooses at a certain time step in a certain state is determined by the *policy* π_t it is following, where $\pi_t(a \mid s)$ is the probability that $a_t = a$ if $s_t = s$. RL methods aim to find the policy that maximises rewards over the long run.

In determining which action to take it is useful to know or at least predict what return is to be expected after moving to a certain state. In an MDP this expected return for a state is given by the *state-value function* $v_\pi(s)$ of a state s under policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right]. \tag{2.5}$$

Alternatively the expected return after being in a certain state *and* choosing a certain action can be computed or approximated, known as the *action-value function* q_π:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right]. \tag{2.6}$$

Just like returns, value functions can be computed recursively. This recursive relation for discrete-time-and-state problems is known as the *Bellman equation*:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right]. \tag{2.7}$$

Equation 2.7, the Bellman equation for v_π, represents a sum over all values of the three variables a, s' and r. For each triple it computes the probability $\pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)$ and weights the quantity in square brackets with that probability. Then it sums over all possibilities to get an expected value. The expression between square brackets states that the value of a state equals the discounted value of the next state plus the expected reward.

The discrete-time Bellman equation serves well to help understand the fundamental workings of RL algorithms. The Bellman equation's continuous-time-and-state analogue is known as the Hamilton-Jacobi-Bellman equation, the treatment of which falls out of the scope of the present report (see, e.g., Wikipedia for an introduction).

Approximating the *optimal state-value function* v^* or the *optimal action-value function* q^* helps in solving the RL problem. With the former an *optimal policy* π^* can be constructed by maximising over the action values; the latter already has the information on the best action.

The Bellman equation written in terms of v^* is known as the *Bellman optimality equation* [8]. It states that the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$v^*(s) = \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_*\left(s'\right)\right]. \tag{2.8}$$

In terms of q^* the equation can be written as:

$$q^*(s,a) = \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} q_*\left(s', a'\right)\right]. \tag{2.9}$$

The next section introduces a class of methods that use the Bellman equation as an update rule: *dynamic programming*. These methods use many of the fundamental ideas that modern RL algorithms are based on.

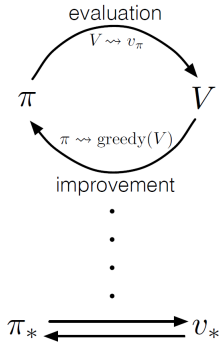


Figure 2.2: A visualisation of generalised policy iteration, the process in which policy evaluation and policy improvement alternate each other. In a policy-evaluation step the value function moves towards the optimal value function an arbitrary amount. Thereafter in the policy-improvement step the policy moves towards the optimal policy an arbitrary amount. The two steps alternate each other until convergence. From [82].

2.3. Dynamic programming

Dynamic programming (DP) is a class of methods that aim to find optimal policies for MDPs, introduced by Bellman [8]. DP algorithms require a complete model of transition probabilities and are therefore of limited practicality for flight control, but many of the concepts integrated in DP algorithms are used in *model-free* algorithms as well. This section therefore serves as an intuitive introduction to the concepts *policy evaluation*, *policy improvement*, *generalised policy iteration* and *bootstrapping* that originate from DP.

The first step in a general DP algorithm is to compute the state-value function v_π, known as *policy evaluation*, which can be done by using the Bellman equation (Equation 2.7) as an update rule for each state s, choosing v_0 arbitrarily:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_k\left(s'\right)\right]. \tag{2.10}$$

Because the Bellman equation tells us that this equation is an equality once $v_k = v_\pi$, v_k generally converges to v_π as $k \to \infty$.

The second step in a general DP algorithm is to find a better policy, known as *policy improvement*. It is based on the *policy improvement theorem* [8], which states that if

$$q_\pi\left(s, \pi'(s)\right) \geq v_\pi(s), \tag{2.11}$$

then the new policy π' is as good as, or better than π, i.e.,

$$v_{\pi'}(s) \geq v_\pi(s). \tag{2.12}$$

The procedure to make a better policy is therefore to make the new policy *greedy* (i.e., to choose the highest value) with respect to the value function of the original policy:

$$\pi'(s) = \underset{a}{\text{argmax}} \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi\left(s'\right)\right], \tag{2.13}$$

where a deterministic policy is considered. The same principle holds for stochastic policies however.

To construct a complete DP algorithm the policy evaluation and policy improvement steps must alternate each other, a process known as *generalised policy iteration*, shich is visualised in Figure 2.2. To find the optimal policy and value function it is not necessary to run one step until convergence before the next step is performed. An arbitrary number of iterations may be performed. This concept of basing estimates on estimates is known as *bootstrapping*.

The interaction of policy evaluation and policy improvement in some form is at the base of most RL algorithms. Stabilisation of both steps implies that the policy is greedy with respect to its own value function and the Bellman optimality equation holds. The policy and value function must then be optimal.

Although the development of classical DP methods formed an essential step towards practical application in areas such as flight control, these methods are still too computationally expensive and rely too much on a complete model of transition probabilities. A more useful approach to DP was introduced by Werbos [97]. He

proposed to approximate DP through *heuristic dynamic programming* (HDP). HDP uses gradient-descent for continuous-state problems and forms the basis of a group of RL algorithms known as *incremental dynamic programming*, connected to reinforcement learning by Watkins [96]. For a more extensive treatment of DP in the context of MDPs the reader is referred to [11, 12]. This chapter will continue by introducing a concept that allows the extension of RL to problems that are not perfectly modelled: *temporal difference learning*, the fundamentals of which are explained in the next section.

2.4. Temporal-difference learning

As the last section showed, DP offers a way of solving MPDs through *expected updates*, which require a complete and accurate model of transition probabilities. Such a model is not practical to obtain for a novel aircraft such as the Flying V. Therefore it may be useful to have RL methods that are based on *experience* and require no model at all. This section introduces these *model-free* methods, which *learn* value functions (rather than *compute* them as in DP), either from actual experience or simulated experience. The latter still requires a model, but this model has less strict requirements than the explicit model of transition probabilities required for DP.

To learn a value function an RL agent may sample the return observed after visiting a state, repeat this many times and then average the returns observed. This procedure may be followed for a problem that has a natural start and end (an *episodic* problem). If this is done for all states the average returns will converge to the expected values. Methods that learn values this way are known as *Monte Carlo* (MC) methods.

An important advantage of MC methods is that they do not fully rely on the Markov property being true, as they do not base value estimates on other value estimates, i.e., they do not bootstrap. Not bootstrapping comes at the expense of learning speed though, as each value update has to wait until the end of an episode. In applications where episodes are long or that are not episodic at all MC methods may fail. Additionally, MC methods may have to discard full episodes if they include exploratory actions that did not result in a higher return.

The obvious answer is then to bootstrap value estimates, .i.e., to not wait until the end of an episode to update value estimates. Methods that combine the ideas of MC and DP in this way are known as *temporal-difference* (TD) methods [96, 98]. They were inspired by animal learning psychology and artificial intelligence and developed through the work of Samuel [71] and Klopf [50].

To make the difference between methods that do and do not bootstrap clearer and to differentiate between RL methods in general it is insightful to compare the *update rules* they use for estimates of, e.g., the value of a state-action pair. The general form of an update rule is

$$estimate_{new} \leftarrow estimate_{old} + step\_size\left[target - estimate_{old}\right], \tag{2.14}$$

where $\left[target - estimate_{old}\right]$ is an *error* in the estimate that is reduced by moving toward the *target*. The step size is usually denoted by α.

Following this structure the update rule for MC value estimation is

$$v(s_t) \leftarrow v(s_t) + \alpha\left[G_t - v(s_t)\right], \tag{2.15}$$

with G_t the actual return after time step t. On the other hand a TD method that bootstraps 1 step ahead, known as TD(0) [81], might follow

$$v(s_t) \leftarrow v(s_t) + \alpha\left[r_{t+1} + \gamma v(s_{t+1}) - v(s_t)\right]. \tag{2.16}$$

MC and TD(0) methods are two extremes in terms of bootstrapping: MC does not bootstrap at all as it completes all steps until the end of an episode before it updates its value estimates, while TD(0) bootstraps by looking only one step ahead. Bootstrapping over only 1 time step might not be ideal, as in some applications a single time step might not be enough to observe a recognisable state change. Methods that look ahead n time steps, as proposed by [96] and further developed by Cichosz [20], van Seijen et al. [93], usually form the best compromise between TD(0) and MC. The update rule for such methods can be formulated as

$$v_{t+n}(s_t) \doteq v_{t+n-1}(s_t) + \alpha\left[G_{t:t+n} - v_{t+n-1}(s_t)\right], \quad 0 \leq t < T, \tag{2.17}$$

where T is the last time step of the episode and is equal to infinity for an infinite-horizon problem. The target in this update rule is the *n-step return* $G_{t:t+n}$. The value-estimation algorithm that can be created with Equation 2.17 is known as *n-step TD*.

As established above both MC and TD(0) have their advantages and disadvantages that must be considered in making a choice for an in-between algorithm with n steps. Three additional considerations are important, especially for online implementation: the amount of steps determines the *delay* that is incurred before updating, the amount of *computation* that is required at each time step and the *memory* required to record states, actions and rewards.

Once the number of steps has been chosen TD can be used to construct an algorithm for control by again using generalised policy iteration, as was introduced in the previous section. The policy improvement theorem (see section 2.3) then guarantees convergence to the optimal policy if all states are visited and an infinite amount of episodes is run. If no environment model is available there is no possibility to look ahead to the next states and choose the action based on the best reward and next state, so the estimated values are required to be action values.

In the real world it is often not possible to visit all states. Nor is it necessarily required though, as only part of the state set might be of interest. For aircraft for example usually a flight envelope is defined which specifies the extreme values that state variables may realistically reach. TD methods can be focused on the part of the state set that is of interest. The next section introduces another concept that helps implementing RL in the real world, by uncoupling behaviour from learning and thereby possibly increasing sample efficiency: off-policy learning.

2.5. On-policy and off-policy learning

Section 2.1 introduced a dilemma central to RL: exploration versus exploitation. Exploratory actions are essential to discover unknown territory, but a policy that occasionally explores will never be optimal. Therefore, if the value function learns on that policy, the value function will never be the optimal value function either. A possible solution is then to use two different policies: one that explores, called the *behaviour policy* b, and one that moves towards the optimal policy, called the *target policy* π. This approach is called *off-policy learning*, as the agent learns from data *off* the target policy. Methods that use one and the same policy for value estimation and policy iteration learn *on-policy*.

An implicit assumption of off-policy learning is that b *covers* π, i.e., $\pi(a \mid s) > 0$ implies $b(a \mid s) > 0$. Therefore the b must be stochastic in states in which it is not equal to π. A way to ensure this is to make the behaviour policy *soft*: it must select all actions in all states with nonzero probability.

To use the behavioural policy for estimating values under the target policy a scaling factor is needed, generally obtained through *importance sampling*. Importance sampling entails weighting returns according to the relative probabilities of their trajectories occurring under π and b, the importance sampling ratio:

$$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(a_k \mid s_k)}{b(a_k \mid s_k)} \tag{2.18}$$

A basic way of using off-policy learning in an RL algorithm is Q-learning, introduced by Watkins [96] in 1989. The algorithm is introduced below.

Q-learning: off-policy value-based model-free reinforcement learning

Q-learning is a model-free off-policy TD algorithm for discrete state-action pairs that converges if all state-action pairs are updated continuously. The algorithm is therefore of limited applicability in high-dimensional (continuous) state-action spaces, but it forms the basis for several state-of-the-art algorithms (e.g., DQN and DDPG) and studying it therefore helps understanding its successors.

The update rule (analogous to Equation 2.14) for Q-learning is given by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right], \tag{2.19}$$

so the Q-learning target y is:

$$y = r + \gamma \max_a Q(s', a) - Q(s, a). \tag{2.20}$$

With this update rule the action-value of the next state s_t - determined by the behaviour policy - is updated based on the action value obtained by following a *greedy* (i.e., maximising) policy $\max_a Q(s_{t+1}, a)$. This shows why Q-learning is an off-policy algorithm.

The algorithm is given below in pseudocode.

Algorithm 1 Q-learning, adapted from [82]

initialise $Q(s,a)$ arbitrarily
repeat(for each episode):
  initialise s
  repeat(for each step of episode):
    choose a from s using policy derived from Q (e.g., ϵ-greedy)
    take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_a Q\left(s',a\right) - Q\left(s,a\right)\right]$
    $s \leftarrow s'$
  until s is terminal

In algorithm 2.5 ϵ-greedy refers to a common mechanism for balancing exploration and exploitation. An ϵ-greedy policy chooses the greedy action with probability 1-ϵ and a (random) exploratory action with probability $0 \leq \epsilon < 1$ (usually small).

■

The Q-learning algorithm (algorithm 2.5) involves a maximisation operation (in this case to construct the target policy), which is common in RL algorithms. The maximisation serves as an estimate for the maximum value. This estimate has a positive bias though, a *maximisation bias*.

Van Hasselt [88, 89] introduced a method to avoid maximisation bias in RL algorithms, known as *double learning*. Double learning works by first producing two independent estimates (i.e., from different samples) of a value and using one estimate to choose the maximising action, while using the other to estimate its value. The value estimate will then be unbiased. Secondly the process is repeated with the role of the two estimates reversed, resulting in a second unbiased estimate. Double learning requires double memory, but does not introduce extra computation time, as only one of the two unbiased estimates is updated.

Although constructing an off-policy algorithm is slightly more tedious than constructing an on-policy algorithm, an important advantage of off-policy learning is that it allows an RL agent to learn from past experience. Learning from past experience (see for example algorithm 2.6) has the potential of decreasing the amount of samples required to learn an (approximately) optimal policy. However, off-policy algorithms inevitably suffer from a higher variance as samples are gathered from a different policy that the one that is updated. This variance has presented a major challenge for stability and convergence that long prohibited the effective use of off-policy algorithms in large continuous state and action spaces [13]. Nevertheless, recent breakthroughs seem to have largely solved these issues of off-policy learning, as will be discussed in section 2.8. First, the next section will introduce an important concept that allows both on-policy and off-policy RL to operate in large state and action spaces: function approximation.

2.6. Function approximation: extending to large state and action spaces

RL problems generally suffer from the *curse of dimensionality* [8], meaning that the number of states (and thereby the computational requirements) increases exponentially with the number of state variables. Extending RL methods to large or even continuous state spaces therefore poses a problem. To solve this problem RL has benefited greatly from interaction with the field of supervised learning.

With the help of a function approximator a value function or policy can be represented by a parameterised function with weight vector $\mathbf{w} \in \mathbb{R}^d$, where usually $d << |\ s\ |$. The approximate value function is then given by $\hat{v}(s,\mathbf{w})$ and the approximate action-value function by $\hat{q}(s,a,\mathbf{w})$. As the number of weights is smaller than the number of states the update of a single state or state-action pair might *generalise* to several other states or state-action pairs nearby. Apart from saving memory and computational requirements the generalisation also means that not all states may have to be visited to estimate a value for them, thereby increasing sample efficiency.

To approximate the value of a state s it is moved by a small amount toward its *update target*. Section 2.4 introduced the general update rule, Equation 2.14. To extend this idea to function approximation the update rule is allowed to take an arbitrary form and the update of a single state may influence many other states. All the function approximator does is learn to mimic output u if it is given input s. Usually this is done by minimising an error between the best estimate of the true value (or the target) and the old estimate.

By formulating the problem in this way in theory any function approximator can be used. A common

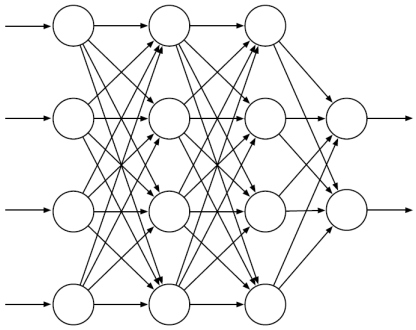


Figure 2.3: A feedforward ANN with two hidden layers.

and simple implementation of this strategy in the form of an update rule can be done through *semi-gradient descent*:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}\left(s_t, \mathbf{w}_t\right) \right] \nabla \hat{v}\left(s_t, \mathbf{w}_t\right), \tag{2.21}$$

where U_t is the update target at time t, which may for example be an n-step return.

Semi-gradient descent is a simple function approximation method that may work well for online RL. If a non-linear function approximator is more appropriate RL may be combined with artificial neural networks (ANNs). Other function approximator types that may be used for RL, with relevant literature, are linear models [3], SVMs [22], decision trees [38], Gaussian processes [69] and finally deep learning (see below). Note that in choosing a function approximator it is critical to consider how well the function approximator can handle non-stationary update targets and incrementally acquired data.

A class of function approximators that has had remarkable success with approximating complex and non-linear functions is that of deep ANNs (ANNs with multiple hidden layers of artificial neurons) used in the field of deep learning (DL), one of the most popular fields in machine learning.

An ANN is a network of interconnected units that are inspired by biological neurons. Figure Figure 2.3 shows a feedforward ANN that has two *hidden layers*, the layers between the input and output layers. Each connection in the network has a weight. Connections may also be looped, resulting in a *recurrent* ANN.

A unit in an ANN applies a non-linear *activation function* to the weighted sum of its input signals, resulting in an output or *activation*. The activation function is usually S-shaped (e.g., a logistic function, a rectifier non-linearity or a step function) and should be chosen based on the function to be approximated.

Input units have a set activation: the input values of the function being approximated by the ANN. Output units have as activation a function of the activation patterns over the input units (in a feedforward ANN), parameterised by the connection weights. A network with at least one hidden layer can approximate any continuous function with a finite number of non-linear activation functions [23]. The use of more than one hidden layer usually allows for more complex function approximation however, as has been shown in practice with deep ANNs [10].

The hidden layers in a deep ANN create a progressively more abstract representation of the ANN input. Each unit adds a feature to build a hierarchical representation of the function that is approximated. An important advantage of deep ANNs is therefore it is not necessary to select or design features manually.

An ANN's performance is measured through an objective function, which is usually minimised through a stochastic gradient method that adjusts the network's weights in the direction that minimises the error. The gradient therein consists of the partial derivatives of the objective function with respect to each weight value at that time. How the objective function is defined in an RL problem depends on the particular method.

A common problem of deep ANNs is *overfitting* on the training data set, resulting in poor generalisation to unknown environments. Common methods to reduce overfitting include *cross validation* (stopping training when performance on a validation data set decreases), *regularisation* (modifying the objective function) and *weight sharing* (reducing the number of degrees of freedom). In addition the dropout method (randomly removing units during training) has shown particularly good performance and thereby results in good generalisation [80].

For a more in-depth introduction to DL the reader is referred to the following literature. An extensive overview of DL literature is given in [72] and a more compact literature review can be found in [52]. For a comprehensive textbook introduction to DL the reader is referred to [39].

The intersection of DL and RL has led to the fruitful field of *deep reinforcement learning* (DRL), also reviewed in [72]. The use of ANNs as function approximators for RL in general goes back to 1954 [33]. The extension to value function and policy estimation and the use of multilayer ANNs followed in the 1980s through the work of Barto et al. [6], Werbos [98, 99], among others. A recent influential demonstration of the power of DRL is the work of [76, 78, 79], who used RL with deep convolutional ANNs to train algorithms that beat the world's best players at the Chinese game of Go, a previously unattainable benchmark for artificial intelligence researchers.

DL has an important advantage over previously used function approximators for RL that required delicately chosen features to accurately represent the function to be approximated. DL requires less prior knowledge because it can learn different levels of abstraction from data [35]. Therefore designers are spared the difficult task of choosing the right features and the approximation power is high. This makes DRL promising for real-world applications such as robots (including UAVs) [37, 55, 67], self-driving cars [15, 64] and passenger aircraft (the focus of the current research).

Several challenges in the design of DRL agents remain though, the most prevalent of which are efficient exploration of the environment and good generalisation of behaviour to different scenarios. Both challenges are expected to be faced in FCS design for the Flying V. Efficient exploration is for example important in an online learning setting and good generalisation is essential to transfer behaviour learned in a simulator to the real world. A more detailed discussion of these challenges can be found in section 3.1.

DQN: extending Q-learning to large state spaces

Mnih et al. [60] used the idea of using function approximators to extend RL to large state spaces and applied it to Q-learning (see algorithm 2.5). They used deep neural networks to approximate the optimal action-value function, resulting in the algorithm known as Deep Q-Networks (DQN), which was an important early success in the field of DRL. Extension to continuous state and action space was not possible with DQN though. However, DQN stands at the base of several more recently developed algorithms which do support continuous spaces.

To approximate the action-value function $q(s,a)$ it can be parameterised with parameter vector $\mathbf{w}$, as $\hat{q}(s,a,\mathbf{w})$. The parameters are approximated by networks known as Q-Networks, for which the target (see Equation 2.14) is given by

$$y = r + \gamma \max_{a'} Q(s', a', \mathbf{w}) \tag{2.22}$$

Apart from parameterisation of the policy DQN uses a few other techniques to improve on standard Q-learning. The first technique is known as *experience replay*. A set of previous experiences at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored in a *replay buffer* D. This replay buffer should not be too large, to avoid slow learning, but also not too small, to avoid overfitting on very recent experiences. Note that experience replay (which is used in several state-of-the-art algorithms) requires an off-policy learning algorithm, as the *old* experiences in the replay buffer may have been obtained with an outdated policy. DQN applies Q-learning updates to samples of experience (s,a,r,s') which are randomly chosen from the replay buffer.

The second technique is the use of *target networks*. As the target given by Equation 2.22 depends on the same parameters $\mathbf{w}$ which are being updated, learning can become unstable. With the use of a separate parameter vector $\mathbf{w}^-$ (a time-delayed version of $\mathbf{w}$) to construct a target network $\hat{Q}$, stability is improved. In DQN $\hat{Q}$ is constructed by simply cloning Q every C steps.

The target at step i can be defined as $y_i = r + \gamma \max_{a'} Q(s', a', \mathbf{w}_i)$, which can be used to define an error to perform a gradient descent step on. The error can be defined as the square of the *TD error*, $\left(y_i - Q(s,a,\mathbf{w}_i)\right)^2$ although other ways of defining the error are possible (see, e.g., [82] for a discussion of different types of error definitions). The loss function to be minimised is the expectancy of this squared TD error under the behaviour policy. In DQN the target is constructed with the target network.

Pseudocode for DQN is given below.

Algorithm 2 Deep Q-Networks, adapted from [60]

initialise replay memory D to capacity N
initialise Q with random weights $\mathbf{w}$
initialise target action-value function $\hat{Q}$ with weights $\mathbf{w}^- = \mathbf{w}$
repeat(for each episode):
  initialise s
  repeat(for each step of episode):
    choose a from s using policy derived from Q (e.g., ϵ-greedy)
    take action a, observe r, s' and store in D
    set $s' = s$
    sample random minibatch of transitions (s, a, r, s') from D
    if s is terminal **then**
      $y_i = r_i$
    else
      $y_i = r_i + \gamma \max_{a'} \hat{Q}(s', a', \mathbf{w}^-)$
    end if
    perform a gradient descent step on $\left(y_i - Q(s, a, \mathbf{w})\right)^2$ with respect to the parameters in $\mathbf{w}$
    every C steps reset $\hat{Q} = Q$
  until s is terminal

■

2.7. Policy-based methods

This section introduces a class of RL methods that do not use a value function to base action selection on, but learn a parameterised policy directly: *policy-based methods*. These methods have the important advantage over value-based methods that they can be applied to continuous action spaces. The most straightforward way of constructing a policy-based method is by using a stochastic policy, for which the parameters can be approximated through gradient ascent. Such methods are known as policy-gradient methods. A parameterised policy can be represented by

$$\pi(a \mid s, \boldsymbol{\theta}) = Pr\{a_t = a \mid s_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}, \tag{2.23}$$

with $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ the parameter vector.

Policy-gradient methods aim to maximise a scalar performance measure $J(\boldsymbol{\theta})$ by approximating gradient ascent in J:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}, \tag{2.24}$$

with $\widehat{\nabla J(\boldsymbol{\theta}_t)} \in \mathbb{R}^{d'}$ a stochastic estimate of the expected gradient of the performance measure with respect to $\boldsymbol{\theta}_t$. These estimates can be constructed through unbiased estimators based on sample trajectories [13].

A policy can be parameterised in any way, as long as its gradient exists for all possible states, actions and parameter values. By choosing a stochastic policy $\pi(a \mid s, \boldsymbol{\theta}) \in (0, 1)$ exploration can be ensured. The action preferences in a stochastic policy can be driven towards the optimal (deterministic or stochastic) policy.

Apart from offering a way of dealing with continuous action spaces, using a paramaterised policy rather than a parameterised action-value function may have several other advantages, depending on the problem. An important practical advantage of using a parameterised policy is that it offers a way of adding prior knowledge about its desired form. Furthermore, for some problems approximating the policy may be easier than approximating the action-value function. Finally, the *policy-gradient theorem* [84] offers a stronger theoretical convergence guarantee than the guarantees available for action-value methods.

REINFORCE: a base policy-gradient algorithm

To illustrate how policy-gradient methods can be implemented REINFORCE is shortly introduced here, the base algorithm for this class of methods. The update rule for REINFORCE is:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(a_t \mid s_t, \boldsymbol{\theta}_t)}{\pi(a_t \mid s_t, \boldsymbol{\theta}_t)} \tag{2.25}$$

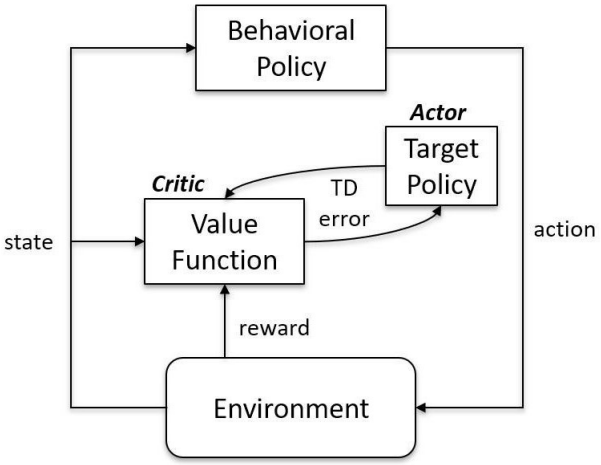


Figure 2.4: A generic actor-critic structure for off-policy learning. The behavioural policy selects an action. The environment responds with a next state and a reward, which are used by the value function (the critic) to update state- or action-values. The updated and the old values can be used to compute the TD error, which is in turn used to update the target policy (the actor) and the value function.

As the full return G_t is used for REINFORCE, it can be considered an MC method. The logic behind the update rule is that the parameter vector $\boldsymbol{\theta}$ is at each step moved in the direction of the gradient multiplied by $\frac{G_t}{\pi(a_t|s_t,\theta_t)}$. A higher return G_t increases the size of the step in the direction of this gradient, thus promoting actions that result in a high return. A higher probability $\pi(a_t \mid s_t,\theta_t)$ of choosing this action decreases the size of the step and thereby prevents that actions are favoured only because they are chosen more often. The fractional vector $\frac{\nabla\pi(a_t|s_t,\boldsymbol{\theta}_t)}{\pi(a_t|s_t,\boldsymbol{\theta}_t)}$ is often implemented in algorithms through the equivalent expression $\nabla \ln \pi(a_t \mid s_t,\boldsymbol{\theta}_t)$.

■

2.8. Actor-critic methods

At the intersection of value-based and policy-based methods is a class of methods that approximates both a value function and a policy: *actor-critic* methods. *Actor* refers to the learned policy and *critic* to the learned value function. The learning processes of the value function (e.g., a TD method) and policy (e.g., a policy-gradient method) are separable, but can be solved simultaneously. In DRL both the actor and the critic can be approximated by deep neural networks. The actor network then uses gradients to update the policy parameter vector $\boldsymbol{\theta}$ and the critic network the value-function parameter vector $\mathbf{w}$. Figure 2.4 displays a generic actor-critic structure for off-policy learning. Advances in off-policy actor-critic structures have allowed the design of many of the latest state-of-the-art DRL algorithms, some of shich will be discussed below.

Actor-critic methods have recently gained in popularity, although they have existed since the early days of RL [7, 83]. In the 1990s action-value methods were preferred due to their simplicity, but their limited convergence in combination with function approximation directed attention to policy-gradient methods. Policy-gradient methods suffer from high variance and thereby slow convergence however, for which actor-critic methods can provide a solution [13].

Although the fundamentals of RL are in this chapter primarily described in the language that is used by artificial intelligence researchers, the fields of optimal control and dynamic programming have developed methods similar to the actor-critic methods as they are described in this chapter. These methods are based on HDP, DHP and GDHP, which were briefly introduced in section 2.3. Prokhorov and Wunsch II [68] formulated a unified view of HDP, DHP and GDHP and their adaptive counterparts, naming these *adaptive critic designs*. In the literature these methods are also referred to as *actor-critic designs* and *action-critic designs*.

With the fundamentals of RL covered in this section and the previous sections, this chapter has arrived at a point where the latest developments in the field of model-free RL can be understood. The latest techniques that have made stable performance of off-policy model-free RL in continuous state and action spaces possible will be introduced through the introduction of three state-of-the-art algorithms: DDPG, TD3 and SAC.

DDPG: extending DQN to continuous action spaces, an actor-critic approach
While conventional policy gradient methods use a stochastic policy (Equation 2.23) to ensure sufficient exploration, Deep Deterministic Policy Gradient (DDPG) (published by Lillicrap et al. [56]) improves on these methods by using a deterministic policy $\mu(s,a,\boldsymbol{\theta})$. A deterministic policy gradient (DPG) method was first published by Silver et al. [77], who showed that DPG is in fact a special case of stochastic policy gradient. DPG requires integration over the state space only, while stochastic policy-gradient requires integration over both the state and action space, making DPG more sample efficient. DDPG showed state-of-the-art performance on high-dimensional tasks at a computational cost similar to that of its predecessors when it was introduced in 2014.

To still ensure sufficient exploration while using a deterministic (target) policy Silver et al. used an off-policy algorithm with a stochastic behaviour policy, which can be constructed by adding noise to the actions during training. DDPG uses the deterministic target policy to construct an actor-critic algorithm that uses a function approximator to estimate action values and then updates the policy parameters in the direction of the action-value gradient.

DDPG is similar to Q-learning and DQN (see section 2.5 and section 2.6) in the way it approximates the optimal action-value function Q^*, through an implementation of the Bellman optimality equation (Equation 2.9) for $Q(s,a,\mathbf{w})$ and the way it minimises a loss function. Furthermore, DDPG similarly uses experience replay and target networks. However, DDPG updates the target networks differently from DQN, as DDPG updates and averages them once every main network update, instead of every C steps.

DDPG adds an additional technique to DQN to enable maximisation over continuous action spaces, which would be too expensive with a normal optimisation algorithm. The optimal action-value function is assumed to be differentiable with respect to its action arguments, such that its gradient can be computed and maximisation can be approximated. For this DDPG uses a *target policy network* μ_{target}, which is constructed in the same way as the target Q-network. The target policy network approximates the maximising action for the target action-value function. The resulting loss function is mimised by stochastic gradient descent.

Like the other algorithms in this chapter DDPG can be characterised by its target formulation. The target of DDPG can be represented by

$$y = r + \gamma Q_{target}\left(s', \mu_{target}\left(s', \boldsymbol{\theta}\right), \mathbf{w}\right)^2. \tag{2.26}$$

The policy learning step in DDPG is performed by applying gradient ascent with respect to the policy parameters to solve

$$\max_{\theta} \underset{s\sim D}{\mathrm{E}}\left[Q\left(s, \mu(s,\boldsymbol{\theta})\right)\right] \tag{2.27}$$

■

TD3: reducing function approximation error in DDPG and other actor-critic methods
Twin Delayed DDPG or TD3 is a state-of-the-art model-free DRL algorithm for continuous action spaces introduced by Fujimoto et al. [36] in 2018. TD3 aims to improve on DDPG and other actor-critic methods by addressing function approximation errors that cause overestimation of action values and sub-optimal policies. The authors brought about this improvement by adapting DDPG with the following three techniques.

The term *twin* in the name of TD3 refers to the first of these techniques, namely double learning, which was introduced in section 2.5. TD3 constructs the target in its loss functions with the smallest of the two action values it learns and names this *clipped Double Q-learning*. Thereby underestimation is favoured, which is unlikely to persist during learning, as the policy will not favour actions with low values.

The term *delayed* refers to the second technique, namely that of *delaying policy improvement* until policy evaluation converges. TD3 improves its policy once for every two policy evaluation steps. Delaying the policy together with the use of target networks should reduce variance, a possible cause of overestimation bias.

The third technique is *target policy smoothing*. The smoothing is performed to avoid that incorrectly highly valued estimates are exploited by the agent. By adding noise to the target policy and averaging over mini-batches, variance in the target (caused by function approximation errors) can be smoothed. The reasoning behind this technique is that similar actions should have similar values and outliers are therefore probably incorrect.

Target policy smoothing is applied by injecting clipped noise to the target policy and then clipping the action that results from the target policy with added noise. The target action is then:

$$a'\left(s'\right) = \mathrm{clip}\left(\mu_{\mathrm{target}}\left(s', \boldsymbol{\theta}\right) + \mathrm{clip}(\epsilon, -c, c), a_{\mathrm{Low}}, a_{\mathrm{High}}\right), \quad \epsilon \sim \mathcal{N}(0,\sigma), \tag{2.28}$$

in which μ_{targ} is the target policy, c the maximum noise value and the action is subsequently clipped to ensure it lies within the valid action range $a_{Low} \leq a \leq a_{High}$.

Clipped double Q-learning is performed by constructing the target y for both action-value functions as

$$y = r + \gamma \min_{i=1,2} Q_{i,\text{targ}}\left(s', a'\left(s'\right), \mathbf{w}\right), \tag{2.29}$$

and then choosing the smallest target value of the two, to which both action-value functions are regressed.

Then, like in DDPG, Equation 2.27 is used for policy improvement, although less frequently than in DDPG, as prescribed by the delay technique. For this DDPG simply uses the first of the two action-value function approximations it has learned. Also similar to DDPG, noise in the behaviour policy is added to ensure exploration.

■

SAC: combining ideas from DDPG and stochastic policy-gradient methods

Soft Actor-Critic (SAC) is a state-of-the-art DRL algorithm for continuous action spaces (a version for discrete action spaces can also be constructed) that was developed by Haarnoja et al. [42] around the same time as TD3. Like TD3 it improves on its predecessors, although in a slightly different way. Similarities between TD3 and SAC are that both are off-policy actor-critic algorithms, that both incorporate clipped double Q-learning. Moreover, both TD3 and SAC use some form of target policy smoothing, although in a different way.

A unique aspect of SAC among off-policy methods is that it is based on the maximum-entropy reinforcement-learning framework, in which the actor aims to maximise both the expected reward and the *entropy*, a measure of randomness. This way of formulating a reinforcement learning problem increases robustness to model uncertainties, an advantageous asset if SAC were to be used to develop a flight control system for the Flying V based on a simulation model. Furthermore, maximum entropy policies improve exploration [41].

For the first version of SAC it proved hard to tune the temperature hyperparameter, so Haarnoja et al. [43] introduced an updated version which improved stability with respect to the hyperparameters and included a way of automatically tuning the hyperparameters. These updated features may open the door to online learning with DRL, which has previously proven to be hard, due to low sample efficiency and high sensitivity to hyperparameter tuning. Hyperparameter tuning may be possible during offline learning, but it prohibits online application. In [44] Haarnoja et al. used SAC to learn a quadrupedal robot to walk from scratch in the real world, without a model of the environment, in two hours.

The RL objective (normally the expected sum of future rewards) in the maximum-entropy reinforcement-learning framework can be constructed by adding an entropy term H multiplied with a temperature parameter α to the reward. α determines the relative importance of entropy an thereby the stochasticity of the optimal policy. The normal RL objective is a special case of the maximum-entropy objective for $\alpha \to 0$. The temperature parameter can be kept fixed as proposed in [42] or varied over training as proposed in [43].

Policy evaluation in the maximum-entropy framework can be formulated through an adaptation of Equation 2.9 as

$$\begin{aligned} Q^{\pi}(s,a) &= \mathop{\mathrm{E}}_{\substack{s'\sim P \\ a'\sim\pi}} \left[r\left(s,a,s'\right) + \gamma\left(Q^{\pi}\left(s',a'\right) + \alpha H\left(\pi\left(\cdot \mid s'\right)\right)\right)\right] \\ &= \mathop{\mathrm{E}}_{\substack{s'\sim P \\ a'\sim\pi}} \left[r\left(s,a,s'\right) + \gamma\left(Q^{\pi}\left(s',a'\right) - \alpha \log \pi\left(a' \mid s'\right)\right)\right], \end{aligned} \tag{2.30}$$

in which next states are drawn from the replay buffer and next actions are determined by the behaviour distribution (contrary to actions in TD3, which are determined by the target policy). An approximation of the expectation in Equation 2.30 can be made through

$$Q^{\pi}(s,a) \approx r + \gamma\left(Q^{\pi}\left(s',\tilde{a}'\right) - \alpha \log \pi\left(\tilde{a}' \mid s'\right)\right), \quad \tilde{a}' \sim \pi\left(\cdot \mid s'\right), \tag{2.31}$$

where $\tilde{a}'$ is used to accentuate that this action is determined by the behaviour distribution.

Like TD3, SAC uses clipped double Q-learning. The target y is constructed through

$$y = r + \gamma\left(\min_{j=1,2} Q_{\text{targ},j}\left(s',\tilde{a}',\mathbf{w}\right) - \alpha \log \pi\left(\tilde{a}' \mid s',\boldsymbol{\theta}\right)\right), \quad \tilde{a}' \sim \pi\left(\cdot \mid s',\mathbf{w}\right). \tag{2.32}$$

Policy improvement in SAC is performed through the following rule:

$$\max_{\theta} \mathop{\mathrm{E}}_{\substack{s\sim D \\ \xi\sim\mathcal{N}}} \left[\min_{j=1,2} Q_j\left(s,\tilde{a}(s,\xi,\boldsymbol{\theta}),\mathbf{w}\right) - \alpha \log \pi\left(\tilde{a}(s,\xi,\boldsymbol{\theta}),\boldsymbol{\theta} \mid s\right)\right], \quad \xi \sim \mathcal{N}(0,I) \tag{2.33}$$

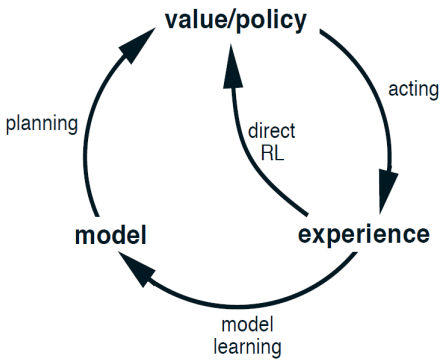


Figure 2.5: An illustration of the integration of direct RL (learning from direct interaction with the environment) and planning (basing updates on interaction with simulated experience from a learned model) in a single algorithm. From [82].

From this rule three differences with policy improvement in DDPG and TD3 can be deduced. Firstly, whereas TD3 simply uses the first of the two action-value function approximations it has learned in the policy improvement rule (Equation 2.27), SAC uses the minimum of the two action-value function approximations. Secondly, action selection in SAC is stochastic, with noise parameter ξ. And thirdly, policy improvement in SAC involves an entropy term.

■

This section has omitted the discussion of commonly used on-policy algorithms such as TRPO, PPO and A3C, as they are typically less sample efficient than off-policy methods. Until recently these on-policy algorithms were the best choice for many applications because off-policy algorithms suffered from stability and convergence issues if they were combined with high-dimensional non-linear function approximators. However, with the introduction of DDPG and its successors TD3 and SAC, the stability and convergence issues were largely solved.

2.9. Combing learning and planning

Section 2.3 explained that the dependency of DP methods on a perfect model of the environment limits their practicality. Section 2.4 then introduced TD as a way to completely remove the reliance on a model. In some cases it would however be beneficial to use some available knowledge of the environment, while not completely relying on it. The present section introduces the basic ideas behind methods that combine the ideas of model-based methods such as DP and model-free methods such as TD methods.

Model-based methods may use a model of the environment for *state-space planning*: searching the state space for an optimal policy. State-space planning methods use a model of the environment to simulate experience. They subsequently update value updates based on the simulated experience, which are in turn used to improve a policy. Planning is therefore similar to learning and in many cases the same RL algorithms that are used for learning can be used for planning. The difference is that planning methods base their updates on simulated experience, while learning methods base their updates on real experience (where *real* experience may in this case be experience gained in a simulated environment).

To allow online implementation of methods that combine learning and planning, planning may be done incrementally, to limit computational expense. Online RL methods that incorporate planning may not even require a model of the environment a priori, as they may learn the model online. These methods are model-free in the sense that they do not require a model given by the designer, but in this report they will be referred to as model-based methods, to make the difference with methods that do not plan clear. Model-based methods may combine learning (also known as direct RL), planning and acting in the real world in an arbitrary way, as illustrated in Figure 2.5, making a wide variety of RL methods that combine these steps possible.

A direct comparison between model-free RL and model-based RL is hard. However, a high-level comparison can be made based on the following distinctions. Firstly, model-free state-of-the-art methods are better understood than model-based methods, as they are more researched in the field of DRL. Secondly, model-free methods are simpler, as they do not require a model-learning nor planning algorithm, and are therefore easier to understand and implement. Thirdly, model-free methods may require less computation per step, as

they do not require computation for model-learning nor planning. This may be an important consideration in online applications or if computation power available for offline learning is limited. Fourthly, model-free methods do not suffer from bias in the learned model, which may arise when an RL agent learns in a simulated environment that is biased with respect to the real world.

On the other hand, model-based methods are often more sample efficient [35, 82], as they may learn from the knowledge gained by estimating a model of the environment. Brunnbauer et al. [16] argue that model-based RL also generalises better from simulation to the real world. However, they base their conclusion on empirical evidence of an RC car. More research is required to make a definite comparison between model-free and model-based methods in *sim-to-real* problems. In more complex environments such as 6-degrees-of-freedom aircraft and especially in environments that are not accurately modelled, conclusions on this comparison may well be different.

This report will from here onward focus on the model-free approach, because for this approach evidence exists of successful application on a fixed-wing passenger aircraft (see [24]), of which the dynamics are assumed to be reasonably similar to those of the Flying V. Furthermore, it is not known how good the existing simulation models of the Flying V are, as no full-scale model of the aircraft exists with which the model can be validated. It is therefore likely that the simulation model of the Flying V would introduce bias in the learned model, which may hinder performance if the trained agent were to be used in the real world at a later stage, on an iteration of the simulation model, or on a sub-scale model of the Flying V.

2.10. Comparison of state-of-the-art model-free DRL algorithms

This chapter started with the very basics of reinforcement learning and gradually built towards the latest developments, marked by the state-of-the-art model-free continuous-state-and-action DRL algorithms DDPG, TD3 and SAC (see section 2.11 for an overview of the evolution of model-free RL). However, as we are interested in the application of state-of-the-art RL to flight control of the Flying V, an important question remains (see research question 1): which of these algorithms is most promising?

Figure 2.6 gives a comparison of common model-free DRL algorithms, performed by Lazaridis et al. [51] on various continuous control tasks in the MuJoCo environment, a popular physics simulation platform. The figure shows that there is no one algorithm that clearly outperforms the others, but performance depends on the task. It is clear though that DDPG, TD3 and SAC all generally perform relatively well. In fact out of all these 7 MuJoCo tasks, DDPG took 2/7 wins, TD3 took 4/7 wins and SAC took 1/7 wins, suggesting that TD3 and DDPG outperform SAC.

However, a comparison based on the aforementioned results does not take all factors into account. As reported by Duan et al. [30] and Henderson et al. [45] DDPG requires careful tuning of its hyperparameters to attain satisfactory results and errors in policy evaluation often lead to divergent behaviour. To take the limited available time for the research proposed in this literature study into account and to leave the door to online implementation (to which high sensitivity to hyperparameter settings is a major obstacle) open, DDPG is discarded as a viable candidate. This leaves TD3 and SAC as the remaining candidates.

Figure 2.7 gives a comparison of TD3 and SAC for various MuJoCo tasks based on their learning curves, performed by Ball and Roberts [5]. This figure illustrates again that a comparison of state-of-the-art DRL algorithms is not straightforward, as performance is highly task-dependent. The author reports a statistically insignificant difference for the Ant and Walker2d tasks, a significantly better performance of SAC on the HalfCheetah task and a significantly better performance of TD3 on the Hopper task. In terms of sample efficiency TD3 is 1.5 times better on the Ant task.

To conclude, based on the results discussed in this section DDPG seems unpractical due to its relative sensitivity to hyperparameter settings and TD3 and SAC are the most favourable in terms of performance. TD3 does seem to slightly outperform SAC in the MuJoCo environment on average though, although this does not mean that the same would be true for a flight control task of the Flying V. Section 3.9 will further compare TD3 and SAC, although this time in terms of practicality for application in the present research.

2.11. A perspective on the research field of DRL

Figure 2.8 illustrates the evolution of model-free RL through the algorithms discussed in this chapter. Each algorithm represents a transformative step in the recent evolution of model-free RL. Through these steps the research field arrived at its current stage of development: one in which control of problems with continuous state and action spaces in many dimensions is possible.

However, to accelerate progress even more some aspects of the research field could be improved, two

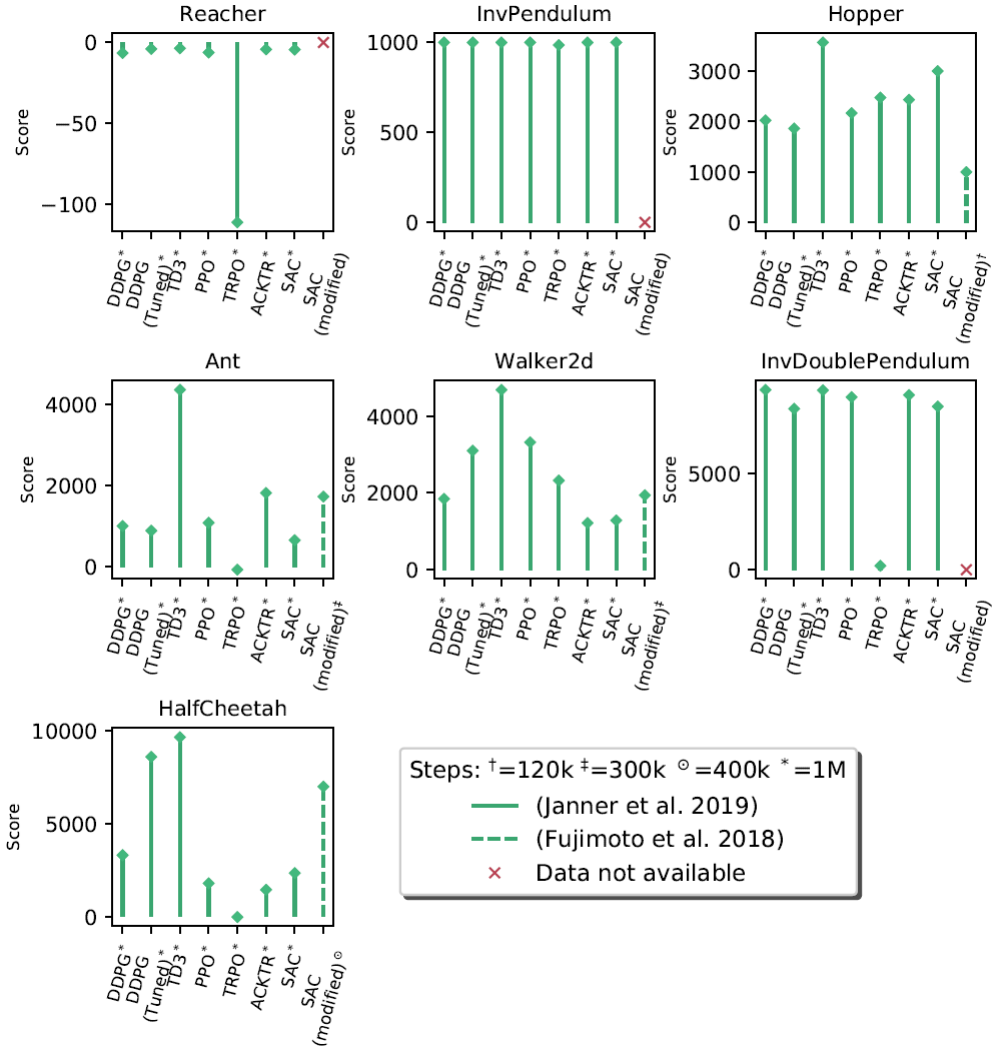


Figure 2.6: Scores of common model-free DRL algorithms on various MuJoCo tasks. From [51].

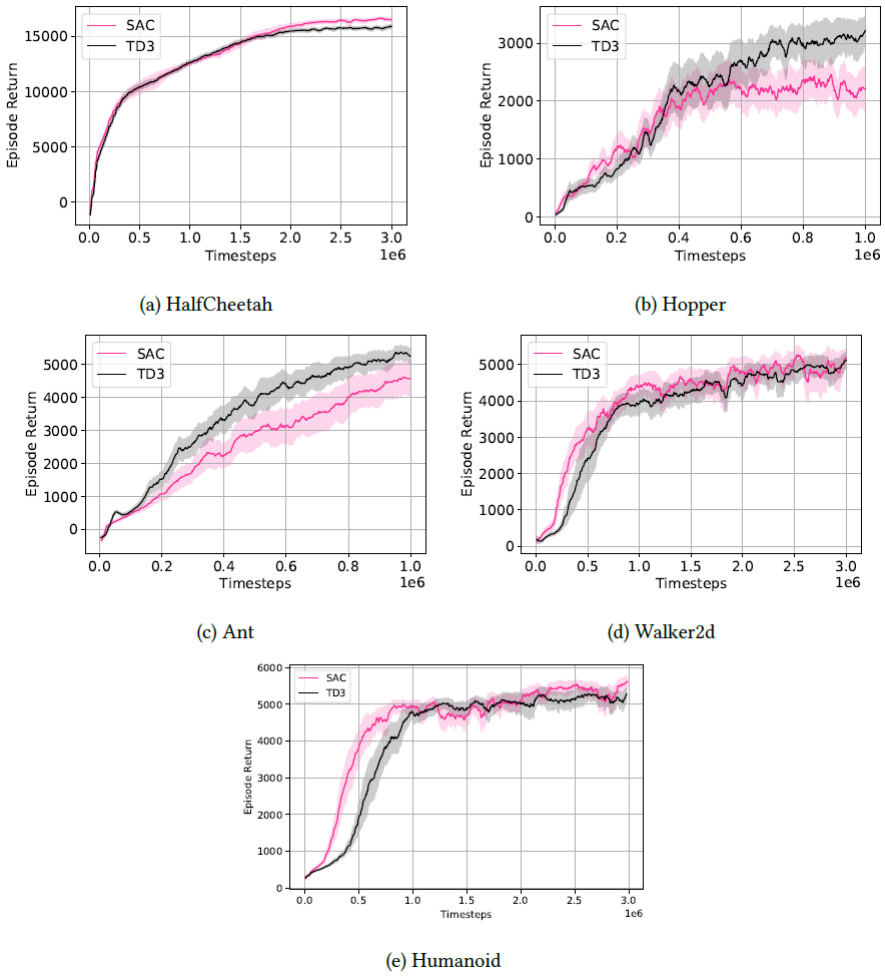


Figure 2.7: Learning curves for various MuJoCo tasks. A comparison of SAC and TD3. From [5].

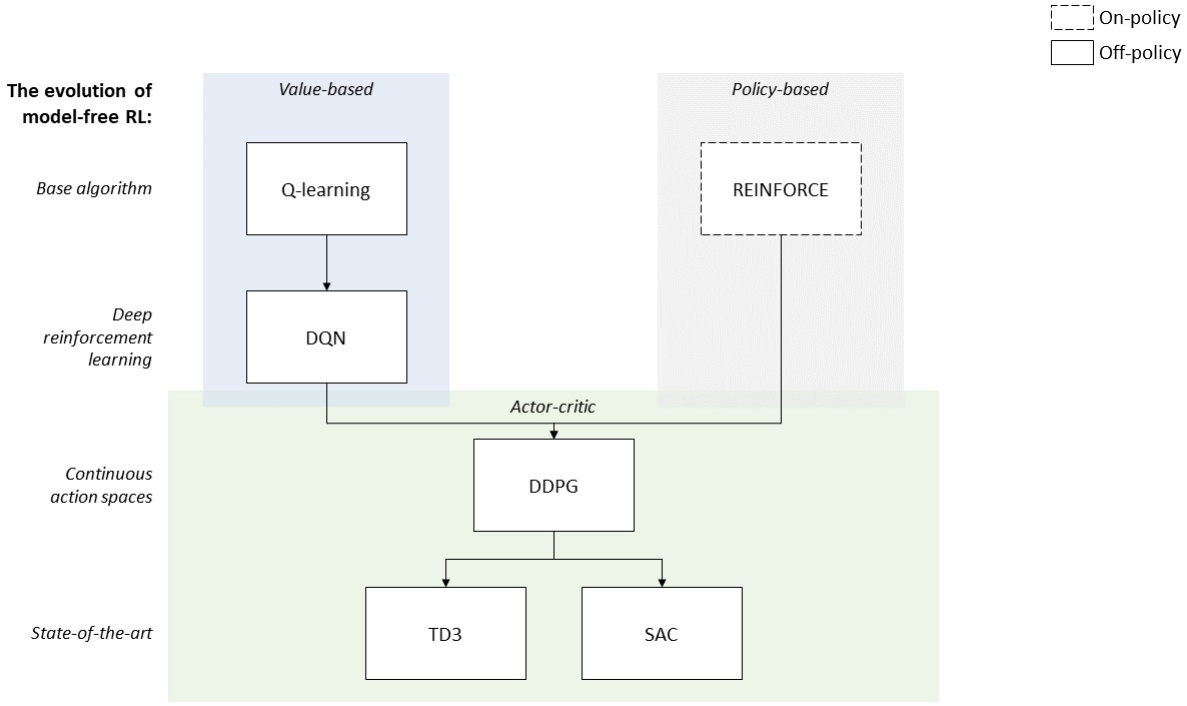


Figure 2.8: The evolution of model-free RL illustrated through the algorithms discussed in this chapter. Base algorithm Q-learning (section 2.5) could be extended to large state and action spaces through the combination with deep learning, leading to the introduction of DQN and thereby DRL (section 2.6). Ideas from DQN were later combined with those from REINFORCE - a base algorithm in the class of policy-based methods (section 2.7) - to arrive at DDPG (section 2.8), thereby enabling control of continuous action spaces. Most recently DDPG's shortcomings with respect to stability and convergence were addressed, leading to state-of-the-art algorithms TD3 and SAC (also section 2.8).

of which will be briefly discussed here. Firstly, as may have become clear from the comparison between TD3 and SAC in the previous section, comparing the performance of even quite similar RL algorithms is not straightforward. As there is no uniform view of the best way to benchmark RL algorithms, authors tend to cherry-pick tasks on which their algorithm performs well. For example, both the authors of TD3 and the authors of SAC claim (and demonstrate) that their algorithm outperforms the other. This lack of uniformity in benchmarking makes it harder for researchers in the field to make an objective choice between existing algorithms, which is clear from the fact that many research papers that use either TD3 or SAC in application state that that algorithm outperforms all competition. Specifically SAC outperforming TD3 is cited often, while the previous section of this report has shown that this is not necessarily the case.

Secondly, a lack of clear structure in the field of model-based RL makes it hard for a researcher to enter this field. As outlined in section 2.9 combining learning and planning can be done in a multitude of ways and there seems to be no unified view in the field of how these combinations should be formed. This is unfortunate because model-based RL seems to have the potential to improve sample efficiency over model-free methods. Nevertheless, a possible advantage of model-free methods is that they closest resemble how animals learn (e.g., a bird learns how to fly without a model of the world).

Apart from the aforementioned challenges many opportunities exist in the field of DRL. With the recent extension of RL to continuous state and action spaces many more real-world applications are possible. An especially exciting advantage of RL over other machine-learning methods is its potential to make increase autonomy of systems. While supervised learning relies on mimicking data provided and labelled by humans, RL can learn directly from experiences gathered in the real world. Although low sample efficiency has so far hindered many applications in the real world, the fast improvement of RL algorithms brings hope for the (near) future.

3

Application of reinforcement learning to flight control

Chapter 2 led from the fundamentals of reinforcement to the latest algorithms, of which especially TD3 and SAC seemed promising candidates for flight control of the Flying V. Knowing that these algorithms represent the state-of-the-art in DRL in theory does however not necessarily mean that they perform well when they are applied to a flight control problem. Therefore this chapter will explore the challenges and opportunities of applying state-of-the-art DRL methods to (flight) control problems, not only to assess whether these methods are suited for flight control of the Flying V but also to learn the best way to implement these methods.

Section 3.1 discusses common challenges in the application of RL to problems in general, whereas sections 3.2-3.9 analyse literature on RL applied specifically to flight control problems to discover additional issues and possible solutions. Hereby the focus is on literature that applies model-free DRL.

Specifically, section 3.2 assesses the feasibility of applying RL to problems with non-linear, coupled dynamics (see research question 1(e)). Section 3.3 assesses the robustness of RL to model uncertainties (see research question 1(f)). Thereafter section 3.4 aims to find a strategy for formulating states and actions (see research question 3(b)) and section 3.5 for a reward function (see research question 3(c)). Then section 3.6 discusses different ways of structuring RL agents (see research question 3(d)). Section 3.7 evaluates the choices between episodic and continuing learning and between setting up a learning curriculum and learning the target task directly (see research question 3(e)). Thereafter section 3.8 gives a preliminary indication of the computational resources required for the research proposed in this report (see research question 3(g)). Lastly, section 3.9 compares TD3 and SAC applied to flight control (see research question 1).

3.1. Common challenges

Although reinforcement learning algorithms have produced some impressive results on complex problems in recent years (see also section 2.6), examples of successful applications of RL in the real world are scarce. The reason for this scarcity is that the application of the current state-of-the-art in RL to real-world problems often proves to be hard. This section will briefly discuss the following three common challenges of applying RL, such that they can be taken into consideration and possibly be addressed in future research:

1. dealing with low sample efficiency,
2. trading off generalisation and overfitting and
3. ensuring safety of exploration.

The first and perhaps foremost problem in applying RL to real-world problems is its low sample efficiency. In real-world situations an RL agent usually requires many interactions with its environment to converge to a (nearly) optimal policy. This is especially a problem if the agent learns online, where interactions with the environment are usually expensive. However, also in an offline learning setting long training times hinder progress, as it can mean designers have to wait hours before it is clear if the program needs debugging or hyperparameter tuning.

The algorithms that the present report focuses on rely on non-linear function approximators which are likely not sample efficient enough to allow online learning. Research on online RL for flight control generally focuses on other methods, such as ADP methods in which a model of the environment is learned incrementally to increase sample efficiency. For a discussion of online RL for flight control using incremental ADP the reader is referred to, e.g., Zhou [102].

The second challenge of applying RL discussed here is a trade-off common to machine learning in general, between generalisation and overfitting. In an offline learning setting interactions with the (simulated) environment are relatively cheap and therefore many interactions may be gathered. However, training too long in the simulated environment may result in overfitting on an environment which is based on assumptions and does not represent the complete state space. Regarding flight simulation specifically, the dynamics at the extreme edges of the flight envelope are non-linear and are therefore usually not or inaccurately modelled. A common method to avoid overfitting is to stop training as soon as a certain reward threshold is reached. More sophisticated methods are mentioned in section 2.6.

A possible practical solution to avoid overfitting and thereby enhance sim-to-real transfer is to train the agent on a diverse set of manually designed environments. Capasso et al. [17] use this approach to train a DRL agent for autonomous driving on roundabouts, for which they create a diverse set of manually designed roundabouts. Another important advantage of this approach is that a separate validation environment can be created to test the robustness of the agent to a change in the environment. A similar approach could be adopted for the Flying V by manipulating aerodynamic coefficients to create different training environments (i.e., a different aircraft model) or a validation environment. A comparable method is *domain randomisation*, in which the environment is randomly manipulated for training, as is done in [2].

Thirdly a challenge that applies to online learning specifically is safety of exploration. Especially in the aerospace domain online exploratory actions may result in a vehicle moving outside the safe flight envelope. Two approaches are possible to ensure safe exploration: the aircraft may be placed in a safe environment (e.g. by placing a scale model in a wind tunnel or a UAV in an enclosed arena) or the RL algorithm may prevent unsafe exploratory actions. An example of the latter approach is given by Mannucci [57]. As the research proposed in this report will focus on offline learning, safety of exploration is not further discussed in detail. The following sections (3.2-3.9) will cover literature on RL applied to flight control to discover the more specific and practical challenges of applying RL to flight control and learn from existing solutions.

3.2. Feasibility

The Flying V's uncertain simulation model and non-linear, coupled flight dynamics most likely pose a challenge to standard PID-control methods (see also chapter 1). How and if a controller based on RL can solve the flight control problems of the Flying V, as posed in research question 1(e), will be answered in the main research phase that will follow this literature study. However, the following two research papers indicate a reasonable feasibility.

Clarke and Hwang [21] used model-free DRL algorithm Normalised Advantage Function, a variant of DQN, to train a controller for a fixed-wing aircraft offline. The resulting controller successfully demonstrated several aerobatic manoeuvres in real-time simulation. The manoeuvres involved flying at the extreme edges of the flight envelope and are therefore prime examples of flight control tasks with coupled, non-linear dynamics.

Another example of a successful application of model-free DRL to a flight control problem involving non-linear and coupled dynamics is given by Waldock et al. [95]. They trained a DQN agent offline on a simulation model of a variable-sweep UAV to perform a perched landing (a landing preceded by a rapid pitch-up movement to drastically reduce airspeed), by controlling the sweep angle and elevator deflection. The trained agent successfully demonstrated a perching trajectory at altitude in the real world. This section focused on flight control of a complex, though nominal, system. The next section will investigate how well RL performs on a system that differs from the nominal system.

3.3. Robustness

The previous two examples show that model-free DRL can indeed solve flight control problems that involve coupled, non-linear dynamics. However, a second important challenge for a flight-control system for the Flying V is that it will be developed with a simulation model that has not been validated on a real aircraft. Therefore the flight control system needs to be robust to model uncertainties (see research question 1(f)).

Wada et al. [94] tested the transfer of a model-free DRL (A3C [61]) agent trained in simulation to the real world. They used a simulation model of a fixed-wing UAV to train the agent on a pitch-angle-reference-tracking task and subsequently tested the agent on a real model of the UAV in a wind tunnel. For this experiment no specific measures were taken to reduce the reality gap, other than training on a range of time delays. The results showed that not simulating the right amount of time delay had a degrading effect on real-world performance. Moreover, friction around the pitching shaft (which was not modelled) resulted in unstable elevator and subsequently angle-of-attack behaviour. To conclude, this research shows the importance of considering the gap between simulation and the real world.

More recent DRL methods may be more robust due to enhanced exploration though (especially SAC, which maximises entropy to stimulate exploration, see section 2.8). Moreover, mitigation measures such as domain randomisation [66] and Long Short Term Memory (LSTM) layers in ANNs may improve sim-to-real transfer.

Besides facilitating the transfer from simulation to the real world, a robust flight control system may limit performance degradation when an aircraft experiences failures or atmospheric disturbances, which are hard to model due to their non-linearity. Bøhn et al. [14] showed that a tailless fixed-wing aircraft controlled by a model-free DRL agent (specifically PPO [73]) was robust to atmospheric disturbances (wind and turbulence) that it had not experienced in training, as illustrated by the results for wind and turbulence with a magnitude of 20 m/s displayed in Figure 3.1. The research also demonstrated that the DRL agent was more robust to turbulence than a PID controller. Figure 3.2 shows the comparison of the RL agent to a PID controller for attitude-angle-tracking tasks. The figure also shows that the RL agent is better capable of eliminating steady-state errors than the PID controller, which is most visible in the roll-angle plot.

Dally [24] showed that a flight control system based on state-of-the-art model-free DRL method SAC (see section 2.8) was not only robust to atmospheric disturbances, but also to failures, varying initial flight conditions and tracking task types, and biased sensor noise. He trained the agent using a high-fidelity simulation model of a CS-25 class aircraft (a business jet). The resulting controller was able to perform coordinated turns with a low tracking error and most notably was robust to severe failures (which it had not experienced in training), such as a jammed rudder at -15 degrees, a 70% less effective aileron and a reduced elevator range, as shown in Figure 3.3. The figure shows the performance of the SAC agent in an altitude-tracking task with reduced elevator range. The solid grey line indicates the time at which the failure occurs. Robustness of the controller to the reality gap between simulation and the real world was not tested, but it is likely that the demonstrated generalisation to unknown circumstances means that SAC and possibly similar algorithms can generalise relatively well from simulation to the real world.

Fault tolerance with the help of RL may also be accomplished by implementing a controller that adapts to changed system dynamics when a failure occurs. However, as discussed in chapter 1, model-free DRL algorithms are usually not sample efficient enough to learn the changed system dynamics online. An alternative way of using model-free DRL for adaptive control is to train the agent on failed system dynamics offline and then switching the adaptive controller on once a failure is detected by a separate failure detection system (see research question 6(b)). Sharma et al. [74] successfully applied this approach, using SAC to develop an adaptive control system for a single rotor failure of a quadcopter. Hover, landing and path following in 2D and 3D was possible with the failed system. Interestingly though, Dally [24] found that training an SAC agent on failed system dynamics did not necessarily improve performance over a robust controller which had not trained on the failed system dynamics. Figure 3.3 shows an example of this, where the agent that has trained on the elevator failure is active after the dashed grey line, whereas before the dashed grey line an agent is active that has never seen the failure before. Therefore, it is not clear whether an offline-trained model-free DRL approach to adaptive control is beneficial. In any case, an important downside of fault tolerance through offline-trained adaptive control is that failures are inherently unpredictable and that therefore the feasibility of training an adaptive controller offline, on all failures that are reasonably likely to occur, is questionable. Robust control does not rely on specific failure cases experienced offline and therefore - when properly designed - enables robustness to a more general set of failure cases, within a limited uncertainty range.

3.4. Formulating states and actions

The states and actions in the formulation of the reinforcement-learning problem of this research must be based on the states and control signals available in a simulation model of the Flying V, which will be discussed in section 4.3. However, before going into the specifics of the Flying V, past implementations that are similar to the present work can give an insight into what considerations must go into choosing the formulation of

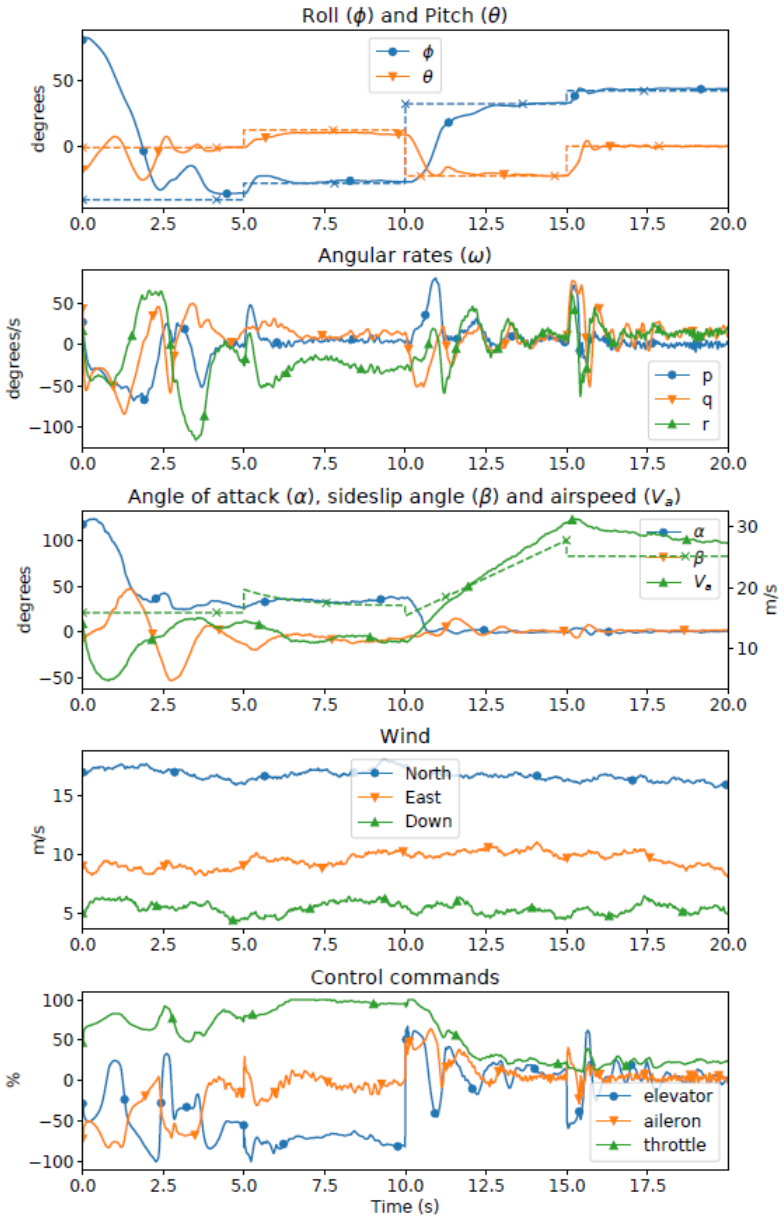


Figure 3.1: A PPO agent's performance on various tracking tasks for a fixed-wing UAV model, while subjected to wind and turbulence with a magnitude of 20 m/s. The agent had not experienced any wind nor turbulence during training. From [14].

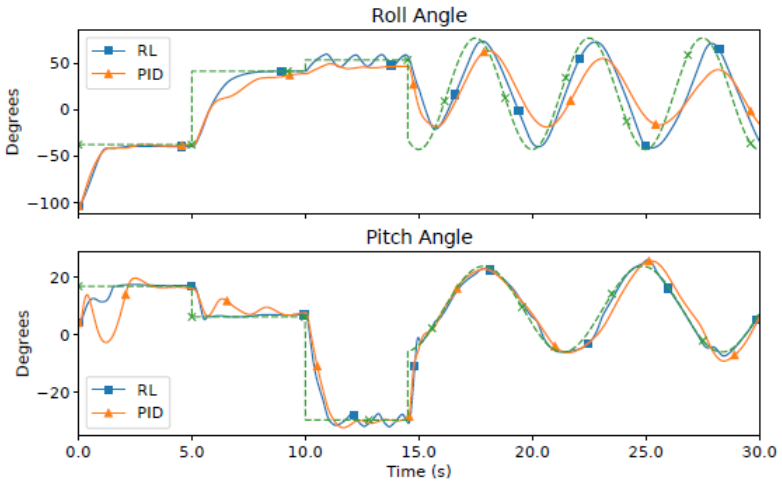


Figure 3.2: Comparison of an RL agent to a PID controller in tracking a reference attitude angle, displayed as a dashed green line. From [14].

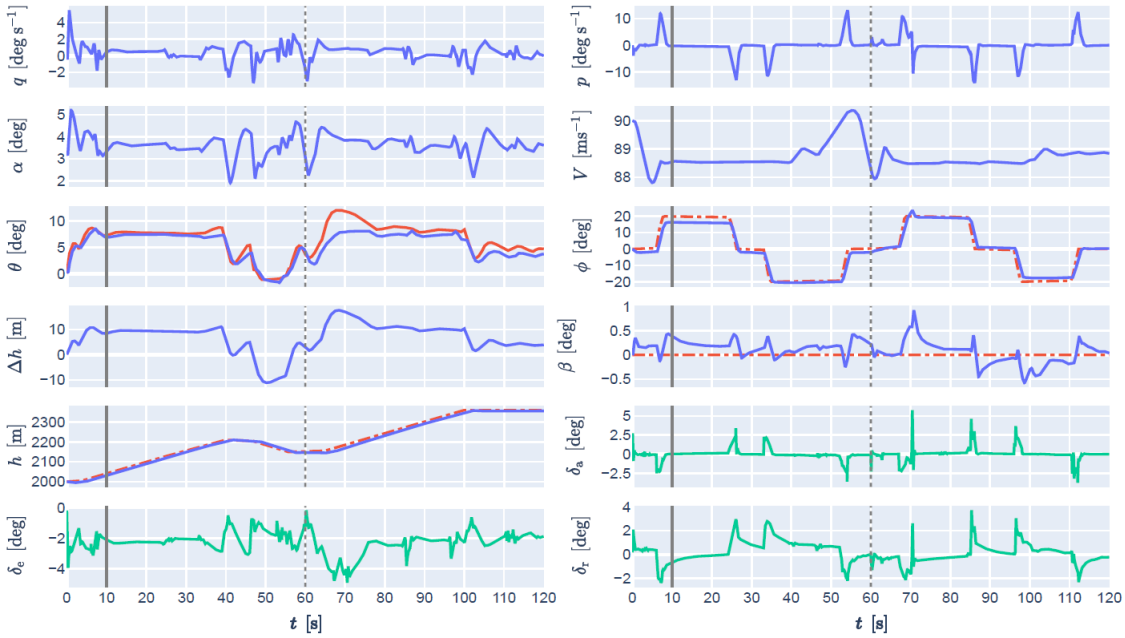


Figure 3.3: An SAC agent's performance on an altitude-tracking task for a fixed-wing business jet model, while experiencing a failure, causing a reduced elevator range. External reference signals are displayed as red dashed lines and self-generated reference signals as red solid lines. Control inputs are displayed as green solid lines. The solid grey lines indicate the time at which the failure occurred. To the left of the dashed grey line the agent that has not trained on the failure is active, while to the right of the dashed grey line the agent that has trained on the failure is active. From [24].

states and actions, thereby providing a partial answer to research question 3(b).

For the formulation of a state vector firstly its size must be considered carefully, as a smaller state vector generally speeds up learning [14], but too little information on the state of the environment can lead to unsatisfactory performance. Including information from previous time steps in the state vector may improve training time [14] and success rate, especially when latencies are involved [28]. Secondly the choice of specific states must be considered (often referred to as *observations*, as they do not necessarily correspond with the actual states in the Markovian sense). To avoid steady state errors, the error between the measured and the reference attitude angle may be used as a state, instead of the angle itself, whereas the angular rates may be used directly to attain a satisfactory transient response [24, 86]. Lastly normalising the state vector may speed up learning, as ANNs often converge faster when they do not need to learn how to scale their input features themselves [14].

Actions in an RL algorithm are usually defined in terms of a normal distribution and subsequently clipped (or squashed in case SAC is used) to a range of $[-1, 1]$. This range then needs to be scaled to meet the appropriate range for the corresponding control signal. This general formulation of the action range allows for more flexibility in mapping actions to control signals. As noted by Bøhn et al. [14] aggressive control signals may lead to actuator wear. Dally [24] uses this scaling to define actions in terms of control input increments - a hundredth of actuator limits - citing that direct control inputs resulted in unstable behaviour during tests.

Apart from preventing aggressive actuator deflections, using domain knowledge to define the action range may also help to speed up learning by limiting the action space. For example, Tsourdos et al. [86] used this tactic by limiting the range of control surfaces that were less relevant for a specific task (e.g., the rudder in a pitch control task). For the Flying V a similar approach may be adopted, although a trade-off must be made between allowing the RL agent to fully explore its action space and speeding up learning. As the Flying V possesses two pairs of elevons, which both influence roll and pitch, the size of the action space may be limited by imposing a control allocation scheme on the elevators. Limiting the ranges of actuators that are not relevant for a specific task may be explored as well.

3.5. Reward-function engineering

Besides adjusting the definition of actions, adjusting the reward function is another way of integrating domain knowledge in the formulation of the reinforcement learning problem (see also research question 3(c). Carlucho et al. [19] used this approach in their design of a reinforcement learning controller for an autonomous underwater vehicle. As their initial reward function resulted in thruster outputs too high to be implemented in the real vehicle, they incorporated two penalty terms for using the thrusters in the reward functions. The first term weighted the magnitude of thruster usage and the second term penalised sudden changes in the control signals to the thrusters. This approach to mitigating aggressive control signals shows an alternative to the approach used by Dally [24].

Regarding the mathematical formulation of the reward function several approaches can be found in literature, including clipping [14] and the L1 norm [24], a function of the squared error [86] and a Gaussian reward function [19]. Usually different components of the error used in a (negative) reward function are weighted empirically (e.g., Dally [24] gives the sideslip angle a relatively high weight as its magnitude is usually small). To the best knowledge of this author no systematic comparison of these different types of reward functions is available in the literature and therefore the most appropriate type for the present research may be found empirically.

3.6. Structuring RL agents

Research question 3(d) refers to another important consideration in developing a flight controller for the Flying V: the structure of the agents. Ideally the reinforcement learning problem is formulated such that an agent has direct control over control surface deflections and thereby tries to accomplish a high-level goal (analogous to the reference input for the outer loop of a standard flight control system), such as performing a coordinated turn. That way the problem formulation and the training procedure are kept simple. Clarke and Hwang [21] cite this way of formulating the problem as resulting in successful training of the RL agent to perform complex aerobatic manoeuvres. However, due to the increased complexity of the problem that the agent has to learn, this way of structuring may lead to slow learning or, as Dally [24] points out, to less robustness. Dally's results show that although performance loss was small, robustness suffered, as the aircraft could not be kept stable under severe failures, contrary to when the aircraft was controlled through a cascaded controller structure.

Using a cascaded controller structure on the basis of domain knowledge is a more complex but perhaps better option than direct control. For example, for an altitude tracking task two agents may be trained. The first agent senses as state the error between a reference altitude and the measured altitude and computes as action a (change in) reference pitch angle, which is in turn used as the state sensed by the second agent. The second agent can directly control the control surfaces of the aircraft through its actions. The dynamics the first agent faces in this setup are much slower than the dynamics the second agent faces. By uncoupling these distinct dynamics the problem the agent tries to solve becomes less complex.

3.7. Training strategy

The last aspect of implementation that will be discussed in this chapter is the training strategy (see research question 3(e)). Specifically, the choice between episodic or continuing learning and the choice between learning the target task directly or setting up a learning curriculum. Episodic learning has a distinct advantage over continuing learning because it allows control over exploration of the state space by using random initial conditions and reference attitudes (see, e.g., [14]). Moreover, episodic learning is the most popular choice in the literature on offline RL for flight control is therefore supported by more evidence than continuing learning. Therefore episodic learning is favoured over continuing learning for the present research.

Curriculum learning may speed up learning or allow the agent to learn tasks that are otherwise too complex. It is a bio-inspired approach, in which an agent is first trained on a simplified task and subsequently on more complex tasks that better resemble the target task. Tsourdos et al. [86] successfully applied this approach in training a DDPG agent for flight control of a 6 degree-of-freedom simulation model of a tube-and-wing aircraft. They designed a curriculum with 4 steps. The agent was subsequently trained to control pitch, roll and yaw and finally to control all attitude angles simultaneously.

The authors cite as an additional advantage of using this type of curriculum that problems in the learning process can be identified more easily and can be addressed without having to redo the whole learning process. Although this research demonstrates that curriculum learning in this form works, no comparison is made with a training setup in which the agent is directly trained to control all attitude angles. Therefore it is not clear whether curriculum learning sped up learning in this case.

3.8. Computational resources

This last section on lessons from past implementations tries to provide a preliminary answer to research question 3(g), "Are the available computational resources sufficient?", based on the literature. Hereby firstly literature on 6 degree-of-freedom coupled dynamics flight control tasks is considered, which is considered to have a level of complexity similar to that of the tasks considered for the present research.

A first observation is that computing resources found in literature are generally larger than the offline (i.e., laptop) resources at hand for the present research, which consist of an Intel i5-8265U CPU @ 1.60GHz - 1.80 GHz with 8.00 GB of RAM and no GPU. Clarke and Hwang [21] used an Intel i7-7700HQ CPU @2.80 Hz with 32 GB RAM to train a PPO agent in an "order of hours". Bøhn et al. [14] used an Intel i7-9700k CPU and an RTX 2070 GPU to train a DDPG agent in "about an hour". These training times highlight the lack of computational simplicity of these DRL algorithms that would be required for online learning. However, these computation times do indicate that the present research is feasible in terms of computation time if an offline learning setting is used. Due to the low computational resources at hand the use of cloud computing resources such as Google Cloud, Azure, and AWS will be considered.

3.9. Choice for TD3 over SAC

Section 2.10 showed that both TD3 and SAC are favourable candidates for the present research, as performance is comparable and the better performing algorithm differs per task. However, in terms of practicality for flight control a remarkable difference between the two algorithms can be observed. As SAC is based on the maximum entropy RL framework (see section 2.8) its exploratory steps during training are more random (randomness is rewarded in SAC). It is therefore common to convert the stochastic policy of an SAC agent to a deterministic one at test time, for which the mean of action probabilities is used. This procedure works well if training happens only offline, but if SAC were to be trained (partly) online in an aircraft, the high degree of randomness in its exploratory actions would increase the likelihood of performing unsafe actions. Moreover, the often highly oscillatory control signal that results from highly random actions would increase actuator wear.

Choosing a purely offline training procedure would solve the issues mentioned above, but from the results published in [28] another interesting consequence of the random action selection in the training of SAC can be observed. Figure 3.4 and Figure 3.5 show the time histories of actions of a TD3 agent an an SAC agent respectively for various initial conditions, both trained for attitude control of a flying wing in simulation using spanwise blowing for roll control. Although both agents learned a nearly optimal policy in approximately the same amount of time steps (the SAC controller showed a slightly larger overshoot though), SAC's policy notably results in more oscillatory behaviour. Whilst the oscillations might not increase tracking error, they would likely increase actuator wear and decrease passenger comfort.

In summary of the comparison between TD3 and SAC it is not clear which algorithm performs better in terms of speed of convergence (although empirical evidence seems to point towards TD3) and optimality of the final policy, as the latter is task dependent. In practical terms however TD3 seems preferable over SAC, which has the natural tendency to produce oscillatory policies because of its stimulation of entropy in training. Therefore, TD3 will be used to develop a flight controller for the Flying V in the present research.

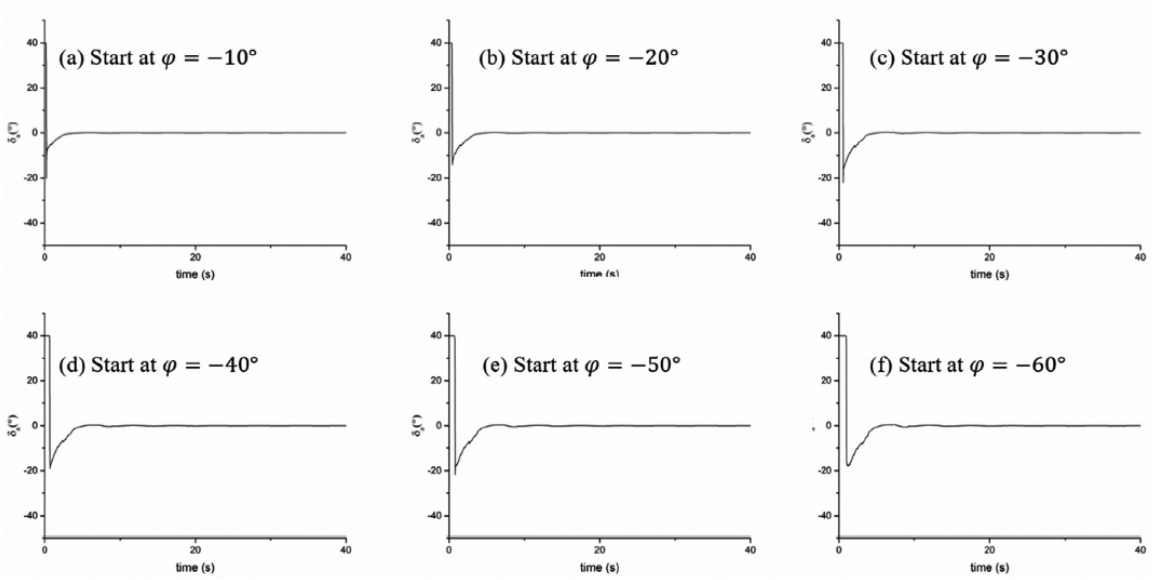


Figure 3.4: TD3: Test results with various initial conditions of the final agent trained for attitude control of a flying wing using spanwise blowing. The time histories of the agent's actions are depicted. From [28].

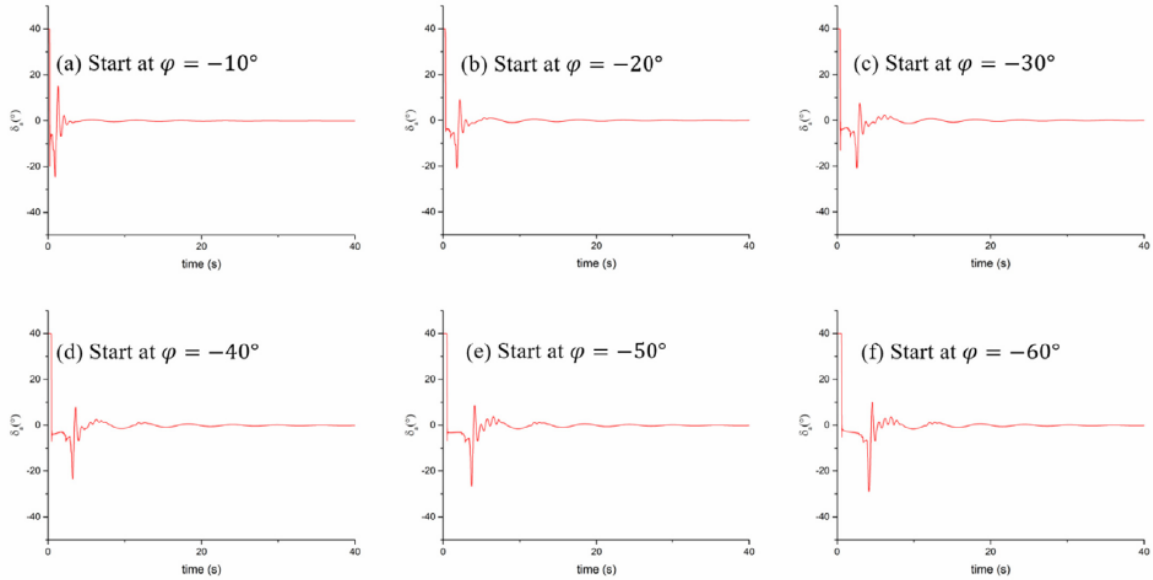


Figure 3.5: SAC: Test results with various initial conditions of the final agent trained for attitude control of a flying wing using spanwise blowing. The time histories of the agent's actions are depicted. From [28].

4

Flight simulation and control of the Flying V

To develop an AFCS that enhances the stability and control of the Flying V in a valuable way it is useful to know the current state of the stability-and-control research for the Flying V, which this chapter therefore summarises. The chapter also serves to become familiar with the to-be-controlled system, or the *environment* in the RL problem that will be formulated for this research, and to identify knowledge gaps. Lastly, from this chapter the selection of one of the available flight simulation models for the Flying V follows.

Firstly, section 4.1 gives an overview of past and present research on design and modelling of the Flying V, after which section 4.2 introduces its flight control system. Then section 4.3 presents the existing flight-simulation models and discusses which model is most appropriate for the present research. Section 4.4 gives a brief overview of existing automatic flight control systems for the Flying V. Lastly, section 4.5 summarises the main findings on the flying and handling qualities of the Flying V that followed from previous research.

4.1. Past and present research on design and modelling

The Flying V is a V-shaped flying wing with two pressurised cabins and two engines. The cabins are located in the wing's leading edge and the engines at the trailing edge, on top of the wing to reduce noise propagation to the ground. The Flying V design aims to outperform state-of-the-art commercial passenger aircraft designs similar to that of the Airbus A350-900, mainly in terms of lift-to-drag ratio and structural weight. The aircraft has a capacity of 314 passengers in a two-class configuration, a cruise speed of M=0.85, a range of 15,000 km and a wing span of 64.75 m, all similar to those of the Airbus A350-900. Benad [9] carried out the preliminary design of the Flying V with these requirements at TU Berlin, in collaboration with Airbus.

A subsequent aerodynamic design optimisation for cruise conditions by Faggiano et al. [32] resulted in an estimated maximum lift-to-drag ratio of 23.7, which is 25% higher than that of the NASA Common Research Model, a benchmark tube-and-wing aircraft model. The optimisation changed the proposal by Benad by increasing the sweep angle, adding vertical fins with integrated rudders and using one oval cabin per wing instead of two cylindrical ones.

To determine the Flying V's stability-and-control characteristics at lower speeds Palermo and Vos [63] tested a 4.6%-scale half-model in a low-speed wind tunnel. Their paper reports stability and control derivatives, a proposed centre of gravity, as well as the trimmed and untrimmed maximum lift coefficients.

A later study into the optimal engine location and its effect on the Flying V's aerodynamic characteristics was carried out by Pascual and Vos [65]. An engine location that caused a 10% loss of aerodynamic efficiency and limited the one-engine-inoperative yawing moment and thrust-induced pitching moment was selected. Van Empelen and Vos [87] conducted further research into the aerodynamic effects of engine integration on a 4.6%-scale model of the Flying V in a wind tunnel. The results revealed a significant effect of interference between the engine and the wing. The interference effect was found to significantly effect drag, lift coefficient and pitching moment coefficient.

Ruiz Garcia et al. [70] were the first to identify an aerodynamic model of the Flying V. They identified the longitudinal static stability derivatives from wind tunnel measurements on the 4.6%-scale model of the Flying V. Around the same time Cappuyns [18] developed an aerodynamic model of the Flying V in collaboration

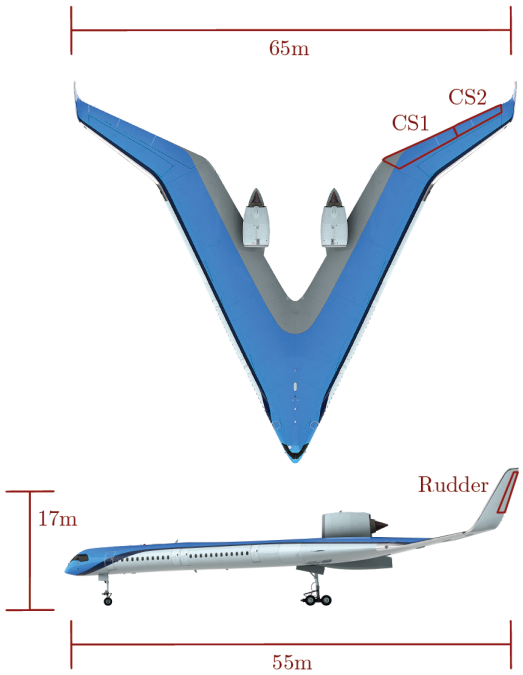


Figure 4.1: Illustration of the control surface layout and the outer dimensions of the Flying V. From [25] (this is an annotated version obtained from [92]).

with Airbus, based on numerical simulations for the full-scale model of the Flying V, using the vortex lattice method (VLM). He used the aerodynamic model for a handling-qualities analysis and concluded that lateral-directional controllability was insufficient for a one-engine-inoperative situation and Dutch roll damping was limited.

Currently four research projects on aerodynamic modelling and handling-qualities analysis of the Flying V are ongoing. Van Overeem et al. [92] are developing an aerodynamic simulation model that integrates the model based on VLM by Cappuyns with the model based on wind tunnel measurements by Garcia. Van Overeem et al. subsequently use the simulation model to asses flying and handling qualities of the Flying V for cruise and approach for various centre-of-gravity locations. Joosten [47] is performing both a theoretical and a piloted assessment of lateral-directional handling qualities of the Flying V, using the aerodynamic model developed by Cappuyns. Torelli [85] uses the same model for a theoretical and piloted assessment of handling qualities as well, focusing on longitudinal dynamics instead. Lastly, Siemonsma [75] is trying to identify an aerodynamic model of the Flying V based on flight tests performed with a radio-controlled scale model [26].

4.2. Flight control system

The flight control system of the Flying V that will be considered for the research proposed in this report has been sized by Cappuyns [18], who notes that control surface sizing has not been optimised yet and that further research is required. Control surface sizing is not an objective of the present research however, so the control surface dimensions proposed by Cappuyns will be used hereafter. Figure 4.1 shows that the flight control system layout consists of two elevons (CS1 and CS2), a rudder and an engine for each wing.

Both elevons may in principle be used for both pitch and roll control, although control allocation differs per research project. Cappuyns adopted a conventional approach and allocated the inboard elevons to pitch control (through symmetrical deflection), the outboard elevons to roll control (through asymmetrical deflection) and the rudders to yaw control (through deflection in the same direction around the aircraft body's vertical axis). Joosten thereafter optimised control allocation by using the generalised inverse method [27, 62].

For the present research several approaches to control allocation may be considered. As the controller will be based on RL, a trial-and-error learning method, the first option is to leave the RL agent to learn the optimal control allocation for each individual manoeuvre. This could be done by allowing the agent to deflect each

control surface (and possibly also adjust the thrust setting) independently. The second option is to adopt the optimised control allocation proposed by Joosten, which leaves less room for the RL agent to explore optimal use of each individual control surface, but may speed up learning as with predetermined control allocation the action space is smaller.

To allow the RL agent sufficient exploration of control allocation while still limiting the action space an intermediate option is proposed for the present research as a base case. Hereby the agent is allowed to deflect each control surface individually, but control surfaces that are less relevant to the manoeuvre that is performed during training are restricted (see also section 3.4). For example, if the Flying V were to learn a pure roll motion, the range of the inboard elevons may be restricted. If it is discovered that learning in this setup is unexpectedly slow or fast, the approach may be adjusted towards one of the first two options mentioned.

4.3. Simulation

To train a controller based on RL for the Flying V offline, as is the aim of the present research, a simulation model is required. At the basis of flight simulation are the equations of motion of the aircraft, which include the aircraft's stability-and-control derivatives. For the stability-and-control derivatives of the Flying V three sources are available, as introduced in section 4.1: fluid dynamics simulations around a full-scale numerical model based on VLM [18], wind tunnel tests with a 4.6%-scale half-model [70] and in-flight tests from a radio controlled 4.6%-scale model [75].

Three main types of simulation models of the Flying V are in use at Delft University of Technology at the time of writing of this report. The first is based only on the stability-and-control derivatives obtained from VLM [18]. The second expands the flight envelope of the first model by combining the derivatives obtained from VLM with the derivatives obtained from wind tunnel measurements [92]. And the third is based on the derivatives obtained from in-flight experimental data. A simulation model based on wind-tunnel measurements only is not obtainable because not all stability-and-control derivatives could be obtained from the wind-tunnel measurements [63].

For the present research one of the aforementioned simulation models will be used, as it does not fall within the scope to develop or improve a simulation model of the Flying V. The simulation model based on VLM only was chosen, for which the reasoning follows. In terms of modelling the largest region of the Flying V's flight envelope and thereby also capturing pitch break (see section 4.5), the model that combines VLM and wind-tunnel-based aerodynamic stability-and-control derivatives is favourable. However, as the control surface layout of the wind-tunnel model significantly differed from that of the VLM model (different dimensions and three instead of two elevons), the non-linear effects of pitch break on control surface effectiveness is not captured by the combined model, which uses the layout of the VLM model as a baseline. Although it would be relevant to develop an RL controller to augment stability of the Flying V when pitch break occurs, the limited fidelity of control surface effectiveness in this region of the flight envelope will likely result in a controller that would not translate well to the real world, thereby limiting the relevance of the results.

Moreover, the simulation model based on only VLM allows fully independent control surface deflection, whereas the combined model does not. As also discussed in section 4.2 RL has the potential to learn to effectively use the different control surfaces by trial and error and thereby potentially increase control effectiveness, which has proven to be a limiting factor for the Flying V [47].

From the simulation model of the Flying V 12 states can be obtained, describing the position, velocity, Euler angles and angular rates. The states can be manipulated through deflection of the three pairs of control surfaces (see section 4.2) and the thrust setting.

4.4. Existing automatic flight control systems

Three previous efforts to automate flight control of the Flying V, all focused on augmentation of stability or control, are described here. Firstly Torelli [85] developed a stability-and-control-augmentation system (SCAS) for longitudinal motions by designing a pitch-rate controller based on PID. He notes that the controller has not been extensively tuned and that it relies heavily on model accuracy. Moreover, he notes that control surface deflections with the implemented SCAS are "too large", but does not give further specifics.

Joosten [47] developed a lateral-directional SAS by designing a roll damper, a yaw damper and a sideslip feedback system. He notes that the SAS notably improves stability, but at the cost of manoeuvrability. Specifically the Dutch roll was successfully augmented, but none of the manoeuvrability requirements. He concludes that further optimisation of control allocation might increase manoeuvrability, but doubts if it will be sufficient, suggesting a resizing of the control surfaces.

Lastly Van Overeem et al. [91] developed an SCAS based on INDI. The SAS consists of an airspeed controller (based on PID) and angular rate controllers. The control augmentation system (CAS) consists of a roll angle, flight path angle and sideslip angle controller. Van Overeem et al. focused specifically on the approach flight condition and the most forward centre-of-gravity location.

4.5. Flying and handling qualities

Due to its tailless configuration, the Flying V naturally suffers from a relatively low longitudinal and lateral-directional dynamic stability. Due to the absence of a horizontal tailplane the short period and phugoid modes of the Flying V have a low damping coefficient [85, 92]. Similarly, due to the absence of a vertical tailplane the Dutch roll mode of the Flying V has a low damping coefficient [47]. As a result of the relatively low damping of the short-period, phugoid and Dutch roll modes of the Flying V would not meet current regulatory requirements in some crucial flight conditions and centre-of-gravity locations [47, 92] without a stability-augmentation system (SAS). However, Van Overeem et al. [92] showed that *with* an SAS (see section 4.4) the dynamic modes tested could be augmented such that they were stable and Joosten [47] showed that the augmented Dutch roll damping complied with regulations (specifically MIL-HDBK-1797), while the spiral complied both with and without an SAS.

Aside from the dynamic-stability issues mentioned above, wind-tunnel tests showed that the Flying V also becomes longitudinally statically unstable at angles of attack larger than 19 degrees (known as *pitch break*), due to vortex formation over the wing, shifting the aerodynamic centre forward [63]. However, the same tests showed that the Flying V's control surfaces remained effective up to the maximum lift coefficient, indicating that an SAS could solve this static-stability issue as well. It is hard to prove in simulation that stability augmentation can correct the Flying V's pitch break though, as modelling of this non-linear phenomenon will likely result in large model uncertainties.

The augmentation of an aircraft's stability and control requires a trade-off between stability and manoeuvrability. Joosten [47] found that lateral-directional manoeuvrability of the Flying V does not comply with regulations (specifically CS-25), both with and without the SAS that he designed, while other authors do not report manoeuvrability assessments. Therefore a knowledge gap seems to exist, as no existing SCAS of the Flying V properly addresses manoeuvrability requirements. Although the limiting factor for manoeuvrability may be the tuning of existing SCASs, the limiting factor may also be the undersizing of the Flying V's control surfaces.

5

Preliminary analysis: training TD3 in a simple environment

Before trying to train a TD3 agent to control the Flying V - a 6-degree-of-freedom environment with non-linear and coupled dynamics - a TD3 agent will be trained in a simpler environment. Training in a simpler environment first has two goals. Firstly, training in a simple environment facilitates testing of the effect of adjusting the hyperparameters of TD3, as training runs cost relatively little time, and thereby addresses research question 3(f). And secondly, training in a simple environment allows easy identification of implementation errors. A simple environment is thereby good practice ground for implementation of an RL algorithm.

This chapter starts with section 5.1, which introduces the simulated environment: a rotating rod. Thereafter section 5.2 introduces the formulation of the RL problem for this preliminary analysis. Section 5.3 then presents an empirical study of the function of four key hyperparameters of TD3, followed by section 5.4, which presents a validation of a trained agent.

5.1. Simulated environment

The environment that the TD3 agent will control for this preliminary analysis is a rotating uniform rod, hanging from a frictionless hinge that is located at its end, as displayed in Figure 5.1. The rod has mass m and length l. Its angle with the vertical is given by θ and the rod can be manipulated by applying a torque T at the location of the hinge.

The dynamics of the rotating rod can be simulated by performing Euler integration over its angular velocity, which can in turn be obtained from Euler integration over its angular acceleration, according to

$$\dot{\theta} = \dot{\theta} + \ddot{\theta}\Delta t, \tag{5.1}$$

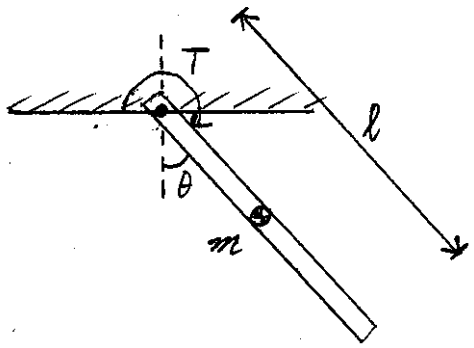


Figure 5.1: A schematic representation of the environment: a rotating uniform rod, hanging from a frictionless hinge that is located at its end.

$$\theta = \theta + \dot{\theta}\Delta t, \tag{5.2}$$

where $\ddot{\theta}$ and $\dot{\theta}$ are the first and second derivative of θ and Δt is a time step. $\dot{\theta}$ is clipped between $\dot{\theta}_{min} < \dot{\theta} < \dot{\theta}max$.

An expression for $\ddot{\theta}$, required for Equation 5.1, can be obtained by deriving the equations of motion of the rod. Starting from Newton's second law for rotational motion,

$$M = I\ddot{\theta},$$

where I is the moment of inertia of the rod with respect to its end, which is equal to $\frac{1}{3}ml^2$ and M is the total moment around the rod's end. The rod's weight mg, where g is the gravitational acceleration, has a component perpendicular to the rod equal to $mg\sin\theta$ and a moment arm equal to $\frac{1}{2}l$. Newton's second law can therefore be written as

$$\frac{1}{2}lmg\sin\theta + T = \frac{1}{3}ml^2\ddot{\theta},$$

which can be rearranged to

$$\ddot{\theta} = \frac{3g}{2l}\sin\theta + \frac{3}{ml^2}T. \tag{5.3}$$

So, in summary, with the current angle θ and the applied torque T the new angular velocity and new angle can be simulated by subsequently applying Equation 5.3, Equation 5.1 and Equation 5.2. Simulations were run with randomly initialised episodes with a length of 200 time steps.

5.2. RL problem

The goal of the TD3 agent for this preliminary analysis is to *swing up* the rod to the vertical position, equivalent to a zero angle θ. The accompanying reward function is defined as

$$r = -\left(\theta^2 + 0.1\dot{\theta}^2 + 0.001T^2\right), \tag{5.4}$$

in which θ is normalised to $[-\pi, \pi]$. The reward function includes penalties for a high rotational velocity and a high torque, to prevent aggressive control signals. With this definition of the reward function the maximum reward per transition transition is 0 and the minimum reward per transition is $-(\pi^2 + 0.1 \cdot 8^2 + 0.001 \cdot 2^2) \approx -16.27$.

The observation space available to the agent consists of the length-normalised x- and y-positions of the rod and its angular velocity. It is defined as $\left[\cos\theta, \sin\theta, \dot{\theta}\right]$. The agent has a one-dimensional continuous action space, which is clipped between a minimum and maximum torque value: $T_{min} < T < T_{max}$.

All constants that have been used to define the simulated environment and the RL problem for this preliminary analysis are given in Table 5.1.

Table 5.1: The constants used to define the simulated environment and the RL problem for this preliminary analysis.

Symbol	Description	Value
g	gravitational acceleration	$10\frac{m}{s^2}$
m	mass of rod	$1kg$
l	length of rod	$1m$
Δt	time step	$0.05s$
$\dot{\theta}_{max}$	max speed	$8rad/s$
T_{max}	max torque	$2Nm$

5.3. Hyperparameters

An important advantage of TD3 over its predecessor DDPG is that TD3 is less sensitive with respect to the settings of its hyperparameters. This brings TD3 closer to a universal RL algorithm, which facilitates its application to new domains, such as flight control of the Flying V. However, the hyperparameter settings of TD3

cited by its designers were tuned for a set of standard (MuJoCo) benchmark problems and were at the same time chosen to resemble the original hyperparameters of DDPG [77] to allow a fair comparison [36].

Improving TD3's hyperparameter settings for flight control of the Flying V may increase its sample efficiency and may allow the TD3 agent to, for example, escape a local optimum. Therefore, it is useful to study the function of TD3's hyperparameters. This section presents an empirical study of three of TD3's hyperparameters, specifically the

- learning rate,
- exploration noise,
- discount rate and
- batch size.

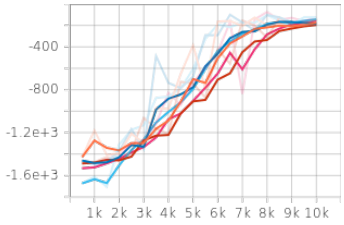
The rotating-rod problem introduced in the previous section facilitates a study of the effect of hyperparameter settings on learning, as a TD3 agent requires relatively little time to converge to a *good* reward for this problem and therefore several training runs may be performed in a short amount of time. Learning in a TD3 agent involves several random processes, such as exploration noise, target policy smoothing and random ANN initialisation. Moreover, episodes for this preliminary analysis were initialised randomly (to increase exploration of the state space). Therefore, multiple training runs are required to allow a useful analysis of each hyperparameter setting.

5 training runs of each 10,000 steps were performed for each hyperparameter setting. For each set of 5 runs a single hyperparameter was adjusted either downward or upward with respect to the *benchmark* hyperparameter, as presented by subfigures a-c of figures 5.2-5.5. Subfigures d of figures 5.2-5.5 show one representative run (chosen from the runs displayed in subfigures a-c) for each hyperparameter setting in a single plot and thereby allow a direct comparison of the three hyperparameter settings. The benchmark hyperparameter settings are the settings cited by Fujimoto et al. [36].

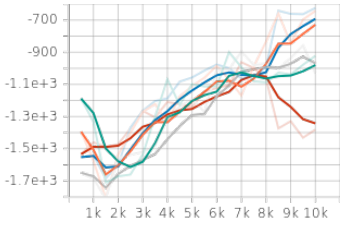
The mean reward was calculated by averaging the reward per episode over 10 episodes. The rewards were sampled in a separate evaluation environment, to exclude the effect of the exploration noise used during training. A mean reward of around -200, corresponding to an average error in the angle of the rod of 1 rad per time step, was considered as solving the problem. The mean-reward curves displayed by subfigures a-c of figures 5.2-5.5 have been smoothed to increase legibility. The actual mean-reward values are visualised with transparent lines. The mean-reward curves displayed by subfigures d of figures 5.2-5.5 have *not* been smoothed, to accentuate decreased learning stability caused by some hyperparameter settings.

The *learning rate* is a step-size parameter that controls how large the update of TD3's ANN parameters is with respect to their estimated error. A learning rate of 0.1 would mean that the ANN parameters are updated 10% of the estimated weight error at each weight update. For this analysis the same learning rate was used for the actor and the critic networks and the ANN parameters were optimised using Adam [49]. As illustrated by Figure 5.2 a small learning rate may result in stable, but slow learning. The mean reward obtained after 10,000 steps is much lower than for the benchmark runs, although for most runs the mean reward steadily progresses during training. However, as exemplified by the dark-red mean-reward curve in Figure 5.2b, a small learning rate may also result in learning getting stuck on a low reward value. A large learning rate, on the other hand, may lead to unstable learning. The mean-reward curves in Figure 5.2c show relatively large fluctuations and no visible progression over time. The large fluctuations may be explained by excessively large updates at each step. As illustrated by Figure 5.2d, both a small and a large learning rate generally result in slower learning than the benchmark learning rate.

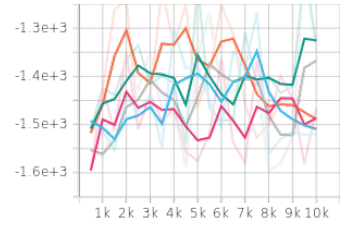
The *exploration noise* controls how much the TD3 agent explores by augmenting action selection with added noise. For this analysis unbiased Gaussian noise was used, as opposed to the Ornstein-Uhlenbeck noise used in DDPG (on which TD3 is based), as the Fujimoto et al. [36] found that Ornstein-Uhlenbeck noise does not offer an advantage. The noise was varied by adjusting the standard deviation of the Gaussian distribution. As illustrated by Figure 5.3 a small amount of exploration noise may lead to slow learning, as the agent has a lower chance of discovering good actions by chance early in the training process, when the policy is still far from optimal. Moreover, a small amount of exploration noise may cause an agent to get stuck in a local optimum, as exemplified by the orange learning curve in Figure 5.3b. A large amount of exploration noise, on the other hand, seems to slightly speed up learning in some training runs on the rotating-rod problem and may therefore be considered as a strategy for problems that involve hard-to-escape local optima. A large amount of exploration noise may provide an added benefit on non-stationary problems, such as an



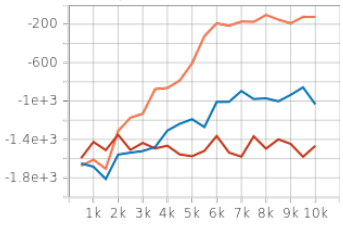
(a) Benchmark: learning rate = 0.001



(b) Small: learning rate = 0.0001



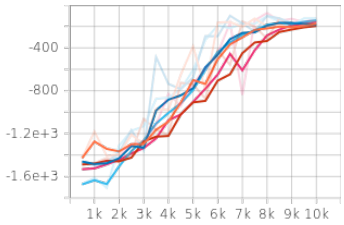
(c) Large: learning rate = 0.01



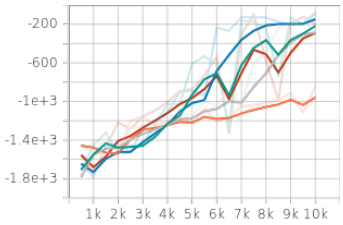
(d) Benchmark (orange), small (blue) and large (red) learning rates compared

Figure 5.2: The mean-reward curve for a TD3 agent on the rotating-rod problem, for various settings of the **learning rate**. x-axis: time steps, y-axis: mean reward.

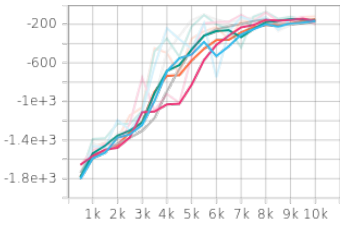
aircraft that experiences turbulence, by then stimulating the agent to search for better actions, outside of the learned policy. Figure 5.3d shows that a large amount of exploration noise does however lead to a noisier mean-reward curve, which should be taken into consideration if learning is done online.



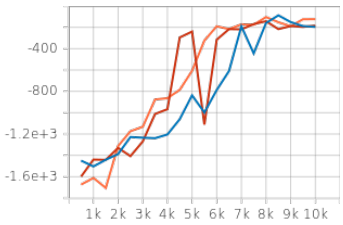
(a) Benchmark: standard deviation = 0.1



(b) Small: standard deviation = 0.001



(c) Large: standard deviation = 1

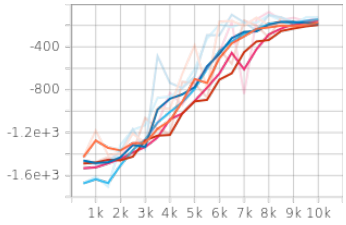


(d) Benchmark (orange), small (blue) and large (red) standard deviations compared

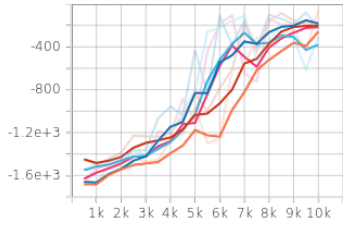
Figure 5.3: The mean-reward curve for a TD3 agent on the rotating-rod problem, for various settings of the **exploration noise**. x-axis: time steps, y-axis: mean reward.

The *discount rate* controls how far the TD3 agent looks in the future for estimating values. A small discount rate creates an agent that favours immediate rewards, whereas a large discount rate creates an agent that weighs rewards farther in the future more heavily, as discussed in section 2.2. As illustrated by Figure 5.4, a discount rate that is small may lead to slow learning and potentially to never reaching a high mean reward, as the agent does not take into account the long-term consequences of actions enough. A large discount rate, on the other hand, leads to fast learning to a high mean reward in some runs, but to noisy learning in others. The noisy runs may be explained by the fact that the agent relates rewards in the distant future to recent actions, while these recent actions may in reality have little influence on states and rewards in the distant future.

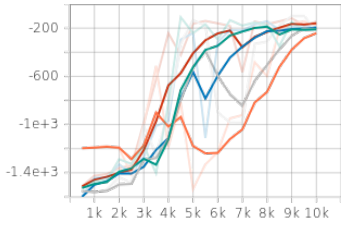
The *batch size* controls how many transitions (s, a, r, s') are sampled from the TD3 agent's replay buffer for each update. A small batch may not contain enough information to learn how to control the RL problem and may therefore lead to no, or, as illustrated in Figure 5.5, slow learning. The effect of a large batch size is more nuanced. The mean-reward curve displayed in Figure 5.5 for a large batch size does not show slow learning.



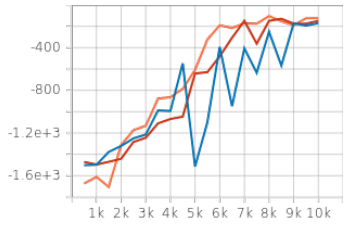
(a) Benchmark: discount rate = 0.99



(b) Small: discount rate = 0.95



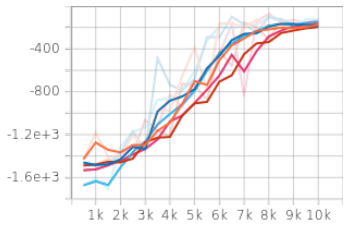
(c) Large: discount rate = 0.999



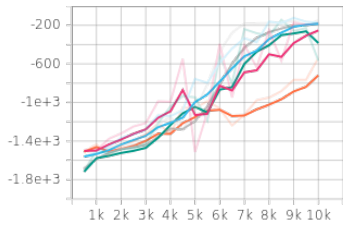
(d) Benchmark (orange), small (blue) and large (red) discount rates compared

Figure 5.4: The mean-reward curve for a TD3 agent on the rotating-rod problem, for various settings of the **discount rate**. x-axis: time steps, y-axis: mean reward.

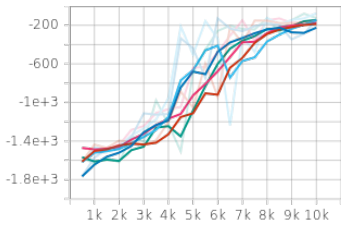
However, a large batch size has empirically been shown to lead to poor generalisation and should therefore be avoided [48].



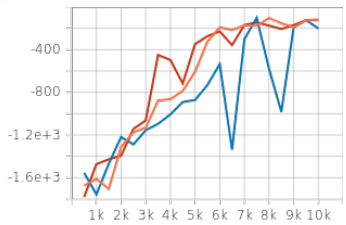
(a) Benchmark: batch size = 100



(b) Small: batch size = 32



(c) Large: batch size = 256



(d) Benchmark (orange), small (blue) and large (red) batch sizes compared

Figure 5.5: The mean-reward curve for a TD3 agent on the rotating-rod problem, for various settings of the **batch size**. x-axis: time steps, y-axis: mean reward.

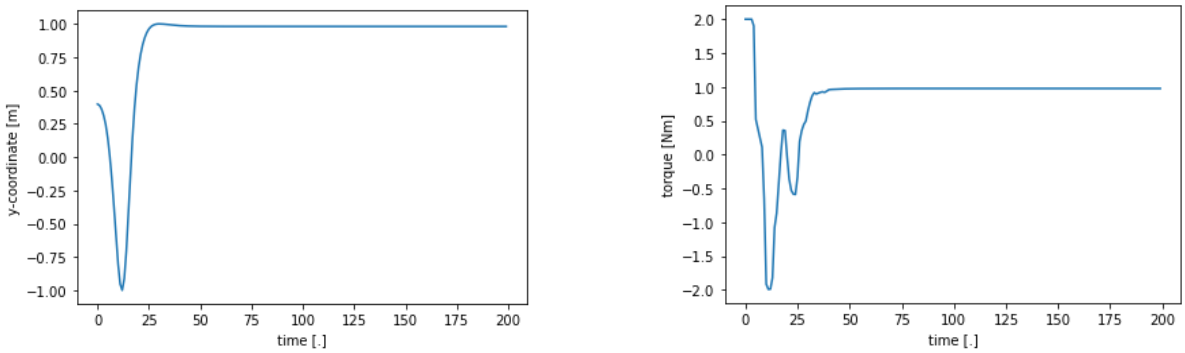
Although for the purpose of empirically studying the function of the key hyperparameters of TD3 the hyperparameters were held at a steady value, varying hyperparameters during training can be a way to speed up learning. The learning rate may be decreased during training, to stimulate fast learning at the beginning but decrease variance later, once the agent converges to a desirable reward. A similar rationale can be applied to the exploration noise.

5.4. Validation

To validate that the TD3 agent it should be tested in several test environments with random initialisations, to ensure the agent has not overfitted on the initialisations experienced in training. Moreover, in this way it can be validated that the agent actually learns a desirable policy at a mean reward of approximately -200 and the agent does not learn any additional undesirable behaviour.

Figure 5.6 shows a representative test run of a TD3 agent trained on the rotating-rod problem with a

final mean reward during training of approximately -200. The rod is vertical and upright if $y = 1$, which is achieved after approximately 30 seconds. In all validation runs the TD3 showed a similar solution, in which, after swinging the rod upright in one go, the rod was held at a slight angle by applying a small torque of approximately 1 Nm. The agent apparently finds that it can achieve a high reward by keeping the angular velocity, which is penalised through the reward function definition (Equation 5.4), close to zero, rather than letting the rod oscillate around the zero angle.



(a) y vs. time

(b) Torque vs. time

Figure 5.6: A simulation of a representative agent with a mean reward of approximately -200 (specifically -212) in a randomly initialised test environment. The agent eventually learns to hold the rod at a slight angle by applying a small torque.

The solution found by the TD3 agent demonstrates how RL agents may find surprising solutions, which can be both an advantage and a disadvantage. An advantage is that an RL agent may find solutions to problems for which the designer does not know the shape of a solution a priori. A disadvantage is that if the designer does not properly define the goal of an RL agent, the agent may find a solution with unforeseen undesirable consequences.

5.5. Conclusions

This chapter demonstrated that a TD3 agent can solve the rotating-rod problem in simulation by properly defining the goal of the agent through the reward function. The implementation of a TD3 agent on simulated dynamics of a real-world system offered practical experience that may be valuable for TD3's implementation on a simulation of the Flying V in the main phase of this thesis research. Moreover, the chapter showed how TD3's hyperparameters may be adjusted to increase learning speed or stability or to help the agent escape a local optimum. Finally, the solution found by the TD3 agent presented in this chapter demonstrated how an RL agent may find surprising solutions, which may be to the designer's advantage or disadvantage.

III

Additional results

6

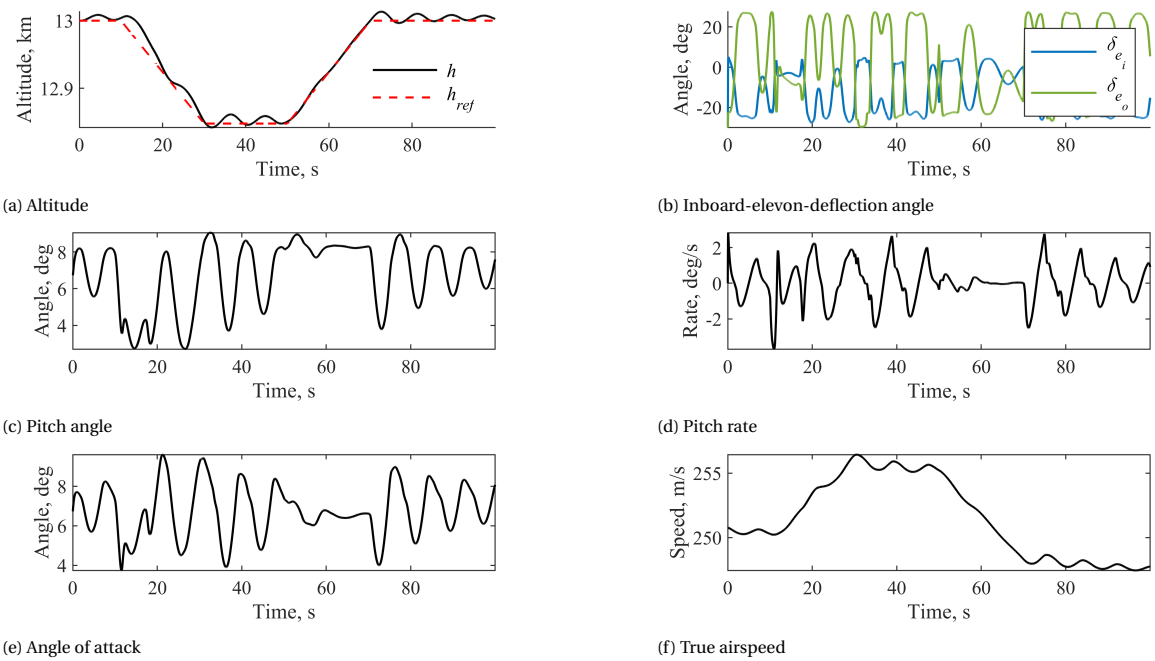
Two elevon pairs as two independent actions

The results presented in this chapter serve to explore how a TD3 agent learns to control altitude if it is given control over two actions: the inboard-elevon-angle δ_{e_i} and the outboard-elevon-angle δ_{e_o}, whereby the left and right elevon of each pair were deflected symmetrically.

In every training run performed for this research, the TD3 agent learned a control policy that resulted in behaviour similar to the behaviour shown in Fig. 6.1b, whereby the outboard elevons were deflected in the direction opposite to the direction of the inboard elevons. As Fig. 6.1a shows, deflecting the inboard and outboard elevons in opposite directions is a reasonably effective strategy for controlling altitude. The mean-absolute-reference-tracking error for the response shown in Fig. 6.1 was 4.4 m, which is 1.4 m higher than the error of the controller presented in the scientific paper in this report, for the same reference signal. However, the simulation model of the Flying V used to produce Fig. 6.1 simulated ideal actuators, unlike the simulation model used for the results in the scientific paper. Therefore, control for the controller presented in the present chapter was easier.

The main reason for pursuing the development of a controller that controlled only the inboard elevons was, however, that the control policy shown in Fig. 6.1 is aerodynamically less efficient, due to the large deflections of both elevon pairs. Altitude control with only the inboard elevons showed at least as good a tracking performance as control with both elevon pairs, with a higher aerodynamic efficiency. Moreover, as Fig. 6.1c shows, oscillations of the pitch angle are barely damped, which has a negative effect on passenger comfort.

Figure 6.2 shows a different policy learned by a TD3 agent that had control over both elevon pairs, again with the assumption of ideal actuators. Here the behaviour favours less oscillations in the elevon deflection (see Fig. 6.2b) than the policy shown in Fig. 6.1b. The pitch rate signal shown in Fig. 6.2d also has a smaller amplitude than the pitch rate shown in Fig. 6.1d. The less oscillatory policy was achieved by engineering the reward function of the TD3 agent. Hereby the agent received a reward of 2 if the absolute altitude error was smaller than 2 m, a reward of 5 if the absolute altitude error was smaller than 5 m, and a reward of 0 otherwise. The reason for learning a policy with relatively little oscillation may be that because the agent would receive zero reward for altitude errors larger than 5 m, it learned to avoid overshooting the reference signal, by using small control signals.



(a) Altitude

(b) Inboard-elevon-deflection angle

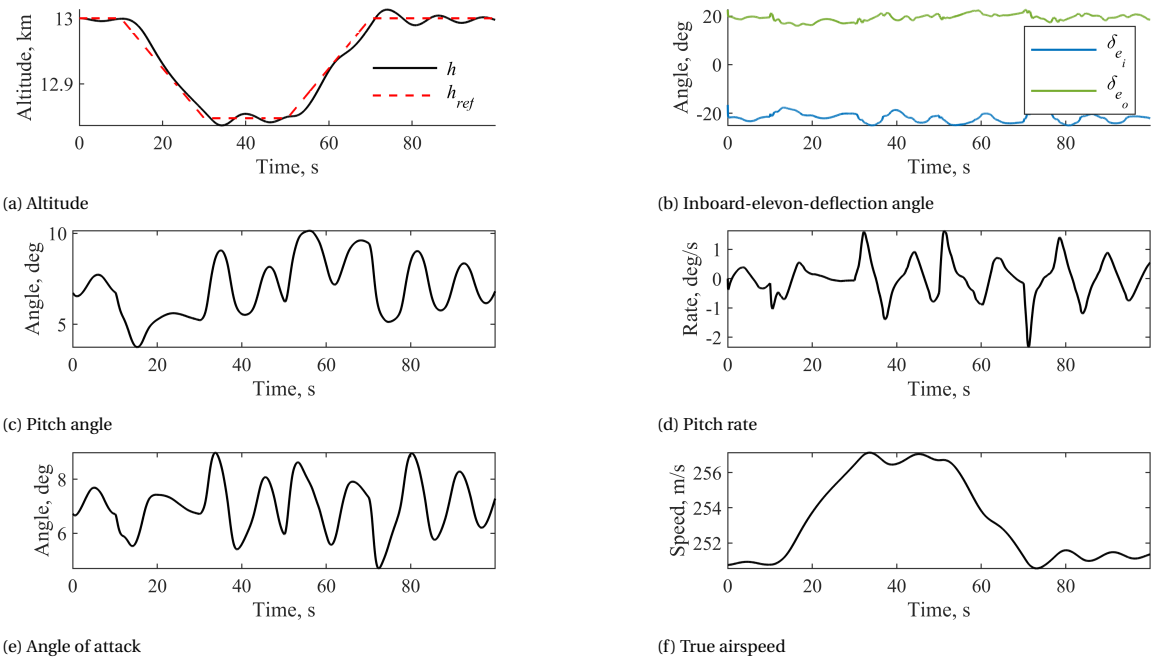
(c) Pitch angle

(d) Pitch rate

(e) Angle of attack

(f) True airspeed

Figure 6.1: Response of a single-loop altitude controller based on TD3, with direct control over the deflection angle of both elevon pairs of the Flying V.



(a) Altitude

(b) Inboard-elevon-deflection angle

(c) Pitch angle

(d) Pitch rate

(e) Angle of attack

(f) True airspeed

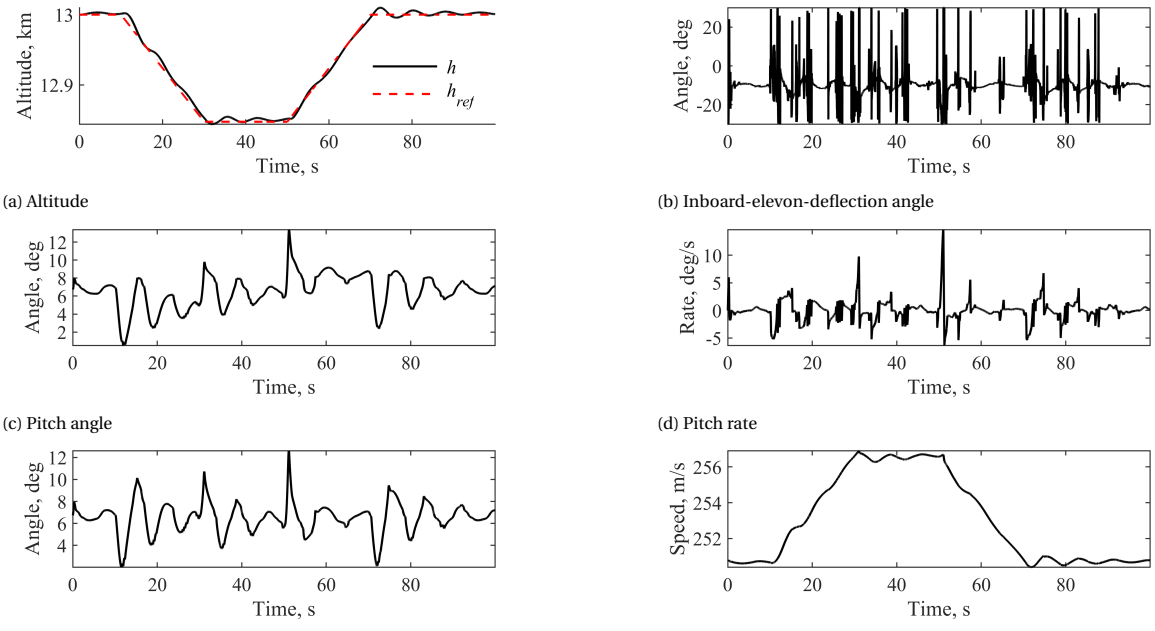
Figure 6.2: Response of a single-loop altitude controller based on TD3, with direct control over the deflection angle of both elevon pairs of the Flying V.

7

Direct control of the inboard-elevon angle

Figure 7.1 shows the response of a single-loop altitude control based on TD3, with the action defined as a direct command for inboard elevon angle. The agent was hereby able to track the altitude reference signal with a mean absolute error of 3.3 m. As Fig. 7.1b shows, direct control over the elevon angle allows the agent to change the elevon deflection very rapidly. All training runs performed for this research in which the agent was given direct control over the elevon angle, resulted in similarly high-gain control policies.

As rapid oscillations in the elevon deflection angle are not realistically possible, due to physical limitations of the elevon actuator, nor desirable due to the high actuator wear and low passenger comfort they would cause, several strategies for mitigating the oscillations in the policy were investigated for this research. Firstly, the elevon rate was calculated at each time step and a reward penalty was given if the rate exceeded certain limits. Secondly, a rate limiter was implemented in the actuator simulation model. Thirdly, several training reference signal shapes were tested, to investigate whether smoother reference signal shapes during training (such as a sinusoidal signal) would lower the level of oscillation in the policy learned. However, all the aforementioned strategies did not offer a solution to the oscillatory policy. Formulating the action as a *change* in elevon angle, rather than the direct angle, did however consistently result in less oscillatory policies. This action formulation was therefore used to develop the policy used to produce the results presented in the scientific paper included in this thesis report.



(a) Altitude

(b) Inboard-elevon-deflection angle

(c) Pitch angle

(d) Pitch rate

(e) Angle of attack

(f) True airspeed

Figure 7.1: Response of a single-loop altitude controller based on TD3, with direct control over the deflection angle of only the inboard-elevon pair of the Flying V.

IV

Closure

8

Conclusion

This final chapter summarises the most important conclusions of this report, which together address and in some cases fully answer the research questions. The introduction of this report firstly answers research question 1(a): "What are the problem characteristics that dictate the choice of method?" as follows. For novel aircraft such as the Flying V the model uncertainties are relatively large, the dynamics are non-linear and complex and the shape of a solution is not necessarily known a priori. Therefore, reinforcement learning is an appropriate method for improving the stability and control characteristics of a flying wing as passenger carrier. Thereafter the introduction continues by motivating the choice for offline learning and thereby answering research question 1(b): "Based on the identified problem characteristics, is learning online or offline more suitable?" as follows. Offline learning generally results in lower failure rates than online learning and offline learning allows more straightforward validation than online learning. Moreover, offline learning can handle more complex control tasks, especially if a deep reinforcement learning method is used. Lastly, offline learning does not involve the danger of trying unsafe control signals for online exploration.

The body of this report thereafter starts with a literature study and firstly answers research question 1(c): "What is the preliminary knowledge of reinforcement learning required to make an informed choice among existing reinforcement learning algorithms?" by explaining the agent-environment interaction and mathematically defining the reinforcement-learning problem as a Markov decision process. Thereafter the literature study progressively introduces more advanced techniques to solve MDPs to build up a set of tools with which relatively sample-efficient model-free reinforcement learning algorithms can be designed for large and optionally continuous state and action spaces.

With the fundamental knowledge covered, the literature study continues by addressing research question 1(d): "What are the most promising state-of-the-art algorithms for the learning setting (online or offline) determined from question 1(c)?" as follows. Model-free algorithms are arguably more promising for the present research and TD3 and SAC are both capable of obtaining high rewards in complex benchmark environments with relatively few samples. However, the stochastic nature of SAC makes it less suitable for flight control, as the stochasticity may result in oscillatory behaviour, even if training is performed offline.

After covering the choice of an algorithm, the literature study addresses research question 1(e): "How can these algorithms solve flight-control problems that involve non-linear, coupled dynamics?". Although challenges such as a low sample efficiency, avoiding overfitting and ensuring safety of exploration remain, it is feasible to apply model-free reinforcement learning to a non-linear flight control problem. Moreover, to answer research question 1(f): "How robust are these algorithms to uncertainties?", these algorithms are shown to be relatively robust to model uncertainties, even severe unmodelled failures, in previous research.

The literature study thereafter addresses research question 3(b): "How can the states and control signals defined in the simulation model of the Flying V be used to formulate the states and actions in a reinforcement-learning problem?" and concludes that the size of the state vector influences learning speed. Moreover, limiting the action space based on domain knowledge can speed up learning and appropriate formulation of the action can prevent aggressive control signals. A discussion of the reward function follows, by discussing research question 3(c): "How can the reward functions be defined?". The literature study concludes that penalty terms in the reward function offer an additional way to prevent aggressive control signals.

Training strategies are covered next, as the report concludes that an answer to research question 3(d):

"How can the controller be structured, i.e. cascaded or not, to exploit the ability of RL to learn a complex task with minimal input from the human expert?" is that single-controller structure is the best structure to investigate the ability of RL to learn flight control as autonomously as possible. Also, an answer to research question 3(e): "How can a training strategy be devised that increases sample efficiency and ease of implementation?" is that episodic learning offers a good way of controlling exploration of the state space and curriculum learning may optionally be pursued if learning the target task directly proves too challenging for the TD3 agent. Research question 3(g): "Are the available computational resources sufficient?" is subsequently briefly discussed. Computational requirements will likely be too high for online learning, but offline learning in terms of computational requirements seems feasible.

The literature study provided an answer to research question 3(a): "What existing simulation model of the Flying V is most suited for this research?" by selecting a simulation model based on the Vortex Lattice Method. The existing AFCSs for the Flying V are analysed to identify knowledge gaps and answer research question 2(a): "What are the flying and handling qualities of the Flying V with existing AFCSs?". Specifically, the analysis finds that manoeuvrability is not properly addressed in existing AFCSs for the Flying V.

Following the literature study a preliminary analysis is reported, in which four key hyperparameters of TD3 are analysed: the learning rate, the exploration noise, the discount rate and the batch size. Thereby research question 3(f): "How do the hyperparameter settings affect learning efficiency and stability?" is addressed, as four hyperparameters' settings are empirically studied in a simple environment.

By answering the aforementioned research questions the literature study aims to provide a solid foundation for the experimental research phase: the development of an automatic altitude controller based on TD3, using a simulation model of the Flying V that was obtained from the Vortex Lattice Method.

The experimental research phase is reported in the form of a scientific paper, which firstly answers research question 2(b): "How can the states and control signals defined in the simulation model of the Flying V be used to formulate the states and actions in a reinforcement-learning problem?" and 2(c): "How can the reward function be defined?" by specifying the observation vector, action, and reward function used to successfully develop an RL altitude controller, as presented in the scientific paper.

To assess how well the altitude controller performs on a nominal simulation model of the Flying V, the scientific paper firstly addresses research question 3(a): "What metrics are most suited to characterise performance?" by choosing the mean-absolute-altitude-tracking error for a reference signal that simulates representative climb and descent commands for aircraft with a flight envelope similar to the Flying V. The results show that the controller has a mean-absolute-reference-tracking error of 3.0 m. Moreover, the results indicate that the TD3 controller is robust to aerodynamic-model uncertainty of at least the same level for which incremental non-linear dynamic inversion (INDI) was shown to be robust in previous research. Thereby research question 3(b): "How does the flight control system compare to a flight control system that is based on a different method?" is addressed.

Thereafter, to assess the robustness of the presented altitude controller, the scientific paper firstly addresses research question 4(a): "How well does the nominal flight control system perform on the "real system", i.e., an altered version of the simulation model of the Flying V that is used to simulate a sim-to-real transfer?" by simulating aerodynamic-model uncertainty and showing results that indicate that the altitude controller is robust to aerodynamic-model uncertainty.

Secondly, to address research question 4(b): "How well does the nominal flight control system perform under measurement noise or bias?" the scientific paper shows that biased sensor noise has a minimal effect on altitude-reference-tracking error. Moreover, the paper shows that the altitude controller can follow various shapes of reference signals in the presence of biased sensor noise, thereby also addressing research question 4(c): "How well does the nominal flight control system perform in response to various shapes of altitude-reference signals?".

Thirdly, research question 4(d): "How well does the nominal flight control system perform if initial flight conditions deviate from the nominal (trimmed) initial flight conditions?" is answered by showing the altitude-reference-tracking error for deviations from the nominal (trimmed) initial flight condition. Thereby the results show that small deviations that would realistically be caused by atmospheric disturbances do not cause a large increase in tracking error nor cause the controller to force the Flying V into a dangerous flight condition. However, the results also show that a large increase in initial altitude - compared to the altitude simulated during training of the TD3 agent - can significantly worsen performance of the altitude controller.

To conclude, this contribution shows that a single-loop controller based on TD3 can successfully learn al-

titude control of a non-linear simulation model of the Flying in an offline setting. Hereby the controller only observes the altitude-tracking error, the pitch angle, the pitch rate, and the elevon-deflection angle. The results also show that the controller is robust to aerodynamic-model error, sensor noise, various shapes of the altitude-reference signal, and unfavourable initial flight conditions. Therefore, the research presented in this paper suggests that deep-reinforcement learning and in particular TD3 has the potential to be used for creating robust flight controllers. However, several questions remain open to investigation.

Recommendations for further research include the following. Firstly, to build on the research presented in this report it is suggested to investigate how robust a controller with the structure proposed in this research is to faults and atmospheric disturbances that were not simulated for this research. Furthermore, to increase the applicability of the controller to a wider variety of flight regimes it is recommended to investigate the addition of airspeed and height in the observation and to train at various altitudes and airspeeds, and with various centre-of-gravity locations. Thereby, to prevent the problem from becoming too inconsistent during training and too complex for the TD3 agent to find a successful policy, it is recommended to use a learning curriculum. The learning curriculum may include progressively more difficult initial conditions, atmospheric disturbances and a varying aerodynamic model to increase robustness of the learned policy. Lastly, it is recommended to investigate how the Flying V controlled by a TD3 agents performs with respect to regulatory requirements on flying and handling qualities.

Secondly, to start a research project on a different subject, similar to the subject presented in this report, the following suggestions are made. For example, the use of TD3 or similar RL algorithms for lateral-directional control or a combination of lateral-directional and longitudinal control may be investigated. Alternatively, an RL controller similar to the controller presented in the present report may be developed, based on a different (or several different) RL algorithm(s), to allow direct comparison of the performance between the algorithms. Moreover, adaptive control for the Flying V may be investigated with the use of a more sample-efficient algorithm than TD3. Hereby it is recommended to study available model-based RL algorithms. Lastly, another suggestion is to investigate a combination of offline and online learning, in which an RL agent is trained on simulated disturbances offline, but also allowed to adjust the offline-learned policy during flight, to allow adaptation to the real world.

Bibliography

[1] Airbus. Cities, airports & aircraft. Technical report, Airbus S.A.S., 2019. URL https://www.airbus.com/aircraft/market/global-market-forecast.html.

[2] Péter Almási, Róbert Moni, and Bálint Gyires-Tóth. Robust Reinforcement Learning-based Autonomous Driving Agent for Simulation and Real World. 9 2020. doi: 10.1109/IJCNN48605.2020.9207497. URL http://arxiv.org/abs/2009.11212http://dx.doi.org/10.1109/IJCNN48605.2020.9207497.

[3] Theodore Wilbur Anderson. *An Introduction to Multivariate Statistical Analysis*. Wiley, New York, 1962.

[4] John Andreae. Learning machines: a unified view. *Encyclopaedia of Linguistics, Information and Control*, pages 261–270, 1969.

[5] Philip Ball and Stephen Roberts. OffCon3: What is state-of-the-art anyway? *arXiv*, 2021. URL http://arxiv.org/abs/2101.11331.

[6] Andrew Barto, Charles Anderson, and Richard Sutton. Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics*, 43(3):175–185, 1982.

[7] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5):834–846, 1983. ISSN 21682909. doi: 10.1109/TSMC.1983.6313077.

[8] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[9] J Benad. The Flying V - A new Aircraft Configuration for Commercial Passenger Transport. Technical report, Airbus Operations GmbH and Berlin University of Technology, 2015.

[10] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–27, 2009.

[11] Dimitri Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, 3 edition, 2005.

[12] Dimitri Bertsekas. *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*, volume 2. Athena Scientific, Belmont, 4 edition, 2012.

[13] Shalabh Bhatnagar, Richard Sutton, Mohammad Ghavamzadeh, Mark Lee, and Richard S Sutton. Natural Actor-Critic Algorithms. *Automatica*, 45(11):10, 2009. doi: 10.1016/j.automatica.2009.07.008{\"{i}}. URL https://hal.inria.fr/hal-00840470.

[14] Eivind Bøhn, Erlend M. Coates, Signe Moe, and Tor Arne Johansen. Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533, 6 2019. doi: 10.1109/ICUAS.2019.8798254. URL http://arxiv.org/abs/1911.05478http://dx.doi.org/10.1109/ICUAS.2019.8798254.

[15] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. *arXiv*, 4 2016. URL https://arxiv.org/abs/1604.07316v1.

[16] Axel Brunnbauer, Luigi Berducci, Andreas Brandstätter, Mathias Lechner, Ramin Hasani, Daniela Rus, and Radu Grosu. Model-based versus Model-free Deep Reinforcement Learning for Autonomous Racing Cars. *ArXiv*, 3 2021. URL http://arxiv.org/abs/2103.04909.

[17] Alessandro Paolo Capasso, Giulio Bacchiani, and Alberto Broggi. From Simulation to Real World Maneuver Execution using Deep Reinforcement Learning. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1570–1575. IEEE, 10 2020. ISBN 978-1-7281-6673-5. doi: 10.1109/IV47402.2020.9304593. URL https://ieeexplore.ieee.org/document/9304593/.

[18] Thibaut Cappuyns. *Handling qualities of a Flying V configuration*. PhD thesis, TU Delft Aerospace Engineering, 12 2019.

[19] Ignacio Carlucho, Mariano De Paula, Sen Wang, Yvan Petillot, and Gerardo G. Acosta. Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning. *Robotics and Autonomous Systems*, 107:71–86, 9 2018. ISSN 09218890. doi: 10.1016/j.robot.2018.05.016.

[20] Pawe Cichosz. Truncating temporal differences: On the efficient implementation of TD(lambda) for reinforcement learning. *Journal of Artificial Intelligence Research*, 2:287–318, 1995.

[21] Shanelle G. Clarke and Inseok Hwang. Deep reinforcement learning control for aerobatic maneuvering of agile fixed-wing aircraft. In *AIAA Scitech 2020 Forum*, volume 1 PartF, pages 1–18. American Institute of Aeronautics and Astronautics Inc, AIAA, 2020. ISBN 9781624105951. doi: 10.2514/6.2020-0136.

[22] Corinna Cortes, Vladimir Vapnik, and Lorenza Saitta. Support-Vector Networks Editor. Technical report, AT&T Bell Labs, 1995.

[23] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[24] Killian Dally. Deep Reinforcement Learning for Flight Control. Technical report, Delft University of Technology, 2021. URL http://repository.tudelft.nl/.

[25] Delft University of Technology. Flying-V, . URL https://www.tudelft.nl/lr/flying-v.

[26] Delft University of Technology. Flying scale model, .

[27] Yann Denieul, Joël Bordeneuve, Daniel Alazard, Clément Toussaint, and Gilles Taquin. Multicontrol Surface Optimization for Blended Wing–Body Under Handling Quality Constraints. *Journal of Aircraft*, 55(2):638–651, 3 2018. ISSN 0021-8669. doi: 10.2514/1.C034268.

[28] Y. Dong, Z. Shi, K. Chen, and Z. Yao. Self-learned suppression of roll oscillations based on model-free reinforcement learning. *Aerospace Science and Technology*, 116, 2021. doi: 10.1016/j.ast.2021.106850.

[29] Dries Verstraete. *The Potential of Liquid Hydrogen for long range aircraft propulsion*. PhD thesis, Cranfield University, 2009.

[30] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *33rd International Conference on Machine Learning*, New York, 4 2016. URL http://arxiv.org/abs/1604.06778.

[31] Russell Enns and Jennie Si. Helicopter trimming and tracking control using direct neural dynamic programming. *IEEE Transactions on Neural Networks*, 14(4):929–939, 7 2003. ISSN 10459227. doi: 10.1109/TNN.2003.813839.

[32] Francesco Faggiano, Roelof Vos, Max Baan, and Reinier Van Dijk. Aerodynamic design of a flying V aircraft. In *17th AIAA Aviation Technology, Integration, and Operations Conference, 2017*. American Institute of Aeronautics and Astronautics Inc, AIAA, 2017. ISBN 9781624105081. doi: 10.2514/6.2017-3589.

[33] B Farley and W Clark. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4):76–84, 1954.

[34] Silvia Ferrari and Robert F. Stengel. Online adaptive critic flight control. *Journal of Guidance, Control, and Dynamics*, 27(5):777–786, 2004. ISSN 15333884. doi: 10.2514/1.12597.

[35] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4): 219–354, 12 2018. ISSN 19358245. doi: 10.1561/2200000071.

[36] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *35th International Conference on Machine Learning*, Stockholm, 2018. URL http://arxiv.org/abs/1802.09477.

[37] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to Fly by Crashing. In *International Conference on Intelligent Robots and Systems*, Vancouver, 2017. ISBN 978-1-5386-2682-5.

[38] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63 (1):3–42, 4 2006. ISSN 08856125. doi: 10.1007/s10994-006-6226-1.

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[40] Brandon Graver, Kevin Zhang, and Dan Rutherford. CO2 emissions from commercial aviation. Technical report, The International Council on Clean Transportation, 2019.

[41] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *34th International Conference on Machine Learning*, pages 1352–1362, 2017.

[42] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv*, 1 2018. URL http://arxiv.org/abs/1801.01290.

[43] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv*, 12 2018. ISSN 2331-8422. URL http://arxiv.org/abs/1812.05905.

[44] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to Walk via Deep Reinforcement Learning. *ArXiv*, 2019. URL http://arxiv.org/abs/1812.11103.

[45] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3207–3214. AAAI press, 2018. ISBN 9781577358008.

[46] Martin Hepperle. Electric Flight - Potential and Limitations. In *Energy Efficient Technologies and Concepts of Operation*, 10 2012. URL https://elib.dlr.de/78726/.

[47] Sjoerd Joosten. Piloted assessment of the lateral-directional handling qualities of the Flying-V. Technical report, Delft University of Technology, Delft, 2021.

[48] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. 9 2016. URL http://arxiv.org/abs/1609.04836.

[49] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *ArXiv*, 12 2014. URL http://arxiv.org/abs/1412.6980.

[50] Harry Klopf. Brain function and adaptive systems - A heterostatic theory. Technical report, Air Force Cambridge Research Laboratories, Bedford, 1972.

[51] Aristotelis Lazaridis, Anestis Fachantidis, and Ioannis Vlahavas. Deep Reinforcement Learning: A State-of-the-Art Walkthrough. *Journal of Artificial Intelligence Research*, 69:1421–1471, 12 2020.

[52] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning, 5 2015. ISSN 14764687.

[53] D. S. Lee, G. Pitari, V. Grewe, K. Gierens, J. E. Penner, A. Petzold, M. J. Prather, U. Schumann, A. Bais, T. Berntsen, D. Iachetti, L. L. Lim, and R. Sausen. Transport impacts on atmosphere and climate: Aviation. *Atmospheric Environment*, 44(37):4678–4734, 12 2010. ISSN 1352-2310. doi: 10.1016/J.ATMOSENV.2009.06.005.

[54] J.H. Lee and E.-J. Kampen. Online reinforcement learning for fixed-wing aircraft longitudinal control. In *AIAA Scitech 2021 Forum*, pages 1–20, 2021. ISBN 9781624106095.

[55] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research*, 17:1–40, 2016.

[56] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[57] Tommaso Mannucci. *Safe Online Robust Exploration for Reinforcement Learning Control of Unmanned Aerial Vehicles*. PhD thesis, Delft University of Technology, Delft, 2017. URL https://doi.org/10.4233/uuid:dbaf67cc-598c-4b26-b07f-5d781722ebfd.

[58] R. Martinez-Val, J. F. Palacin, and E. Perez. The evolution of jet airliners explained through the range equation. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 222(6):915–919, 2008. ISSN 09544100. doi: 10.1243/09544100JAERO338.

[59] Rodrigo Martinez-Val. Flying wings. An new paradigm for civil aviation? *Acta Polytecnica*, 47(1), 2007. URL http://ctn.cvut.cz/ap/.

[60] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

[61] V. Mnih, A.P. Badia, L. Mirza, A. Graves, T. Harley, T.P. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *33rd International Conference on Machine Learning, ICML 2016*, volume 4, pages 2850–2869, 2016. ISBN 9781510829008.

[62] Michael W. Oppenheimer, David B. Doman, and Michael A. Bolender. Control Allocation for Overactuated Systems. In *14th Mediterranean Conference on Control and Automation*, pages 1–6. IEEE, 6 2006. ISBN 0-9786720-1-1. doi: 10.1109/MED.2006.328750.

[63] Marco Palermo and Roelof Vos. Experimental aerodynamic analysis of a 4.6%-scale flying-v subsonic transport. In *AIAA Scitech 2020 Forum*, volume 1 PartF. American Institute of Aeronautics and Astronautics Inc, AIAA, 2020. ISBN 9781624105951. doi: 10.2514/6.2020-2228.

[64] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to Real Reinforcement Learning for Autonomous Driving. *British Machine Vision Conference 2017, BMVC 2017*, 4 2017. doi: 10.5244/c.31.11. URL https://arxiv.org/abs/1704.03952v4.

[65] B.R. Pascual and R. Vos. The effect of engine location on the aerodynamic efficiency of a flying-v aircraft. In *AIAA Scitech 2020 Forum*, volume 1 PartF, 2020. ISBN 9781624105951. doi: 10.2514/6.2020-1954.

[66] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *arXiv*, 10 2017. doi: 10.1109/ICRA.2018.8460528. URL http://arxiv.org/abs/1710.06537http://dx.doi.org/10.1109/ICRA.2018.8460528.

[67] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning. *arXiv*, 10 2017. ISSN 2331-8422. doi: 10.15607/rss.2018.xiv.008. URL http://arxiv.org/abs/1710.06542.

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

[69] Carl Edward Rasmussen. Gaussian Processes in Machine Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3176: 63–71, 2003. doi: 10.1007/978-3-540-28650-9{_}4. URL https://link-springer-com.tudelft.idm.oclc.org/chapter/10.1007/978-3-540-28650-9_4.

[70] Alberto Ruiz Garcia, Roelof Vos, and Coen de Visser. Aerodynamic model identification of the flying V from wind tunnel data. *AIAA AVIATION 2020 FORUM*, 1 PartF, 2020. doi: 10.2514/6.2020-2739. URL https://repository.tudelft.nl/islandora/object/uuid%3Ae1afcad2-7dab-4bab-ab72-587ce476fada.

[71] Arthur Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 11(6):601–617, 1959.

[72] J. Schmidhuber. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003.

[73] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv*, 7 2017. URL http://arxiv.org/abs/1707.06347.

[74] Paras Sharma, Prithvi Poddar, and P. B. Sujit. A Model-free Deep Reinforcement Learning Approach To Maneuver A Quadrotor Despite Single Rotor Failure. *arXiv*, 9 2021. URL http://arxiv.org/abs/2109.10488.

[75] Kevin Siemonsma. Flight Model Estimation of the Flying-V. Technical report, Delft University of Technology, Delft, 2021.

[76] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270.

[77] David Silver, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *31st International Conference on Machine Learning*, Beijing, 2014. URL http://proceedings.mlr.press/v32/silver14.pdf.

[78] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 1 2016. ISSN 14764687. doi: 10.1038/nature16961.

[79] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv*, 12 2017. ISSN 2331-8422. URL http://arxiv.org/abs/1712.01815.

[80] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

[81] Andrew Sutton. Learning to predict by method of temporal difference. *Machine Learning*, 3(1):9–44, 1988.

[82] R Sutton and A Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2 edition, 2018.

[83] Richard Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, Unoversity of Massachusetts Amherst, 1984. URL https://scholarworks.umass.edu/dissertations/AAI8410337.

[84] Richard Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Information Processing Systems 12*, pages 1057–1063, 1999.

[85] Riccardo Torelli. Piloted simulator evaluation of low speed handling qualities of the Flying-V. Technical report, Delft University of Technology, Delft, 2021.

[86] A. Tsourdos, I.A. Dharma Permana, D.H. Budiarti, H.-S. Shin, and C.-H. Lee. Developing flight control policy using deep deterministic policy gradient. In *Proceedings of the 2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology, ICARES 2019*, 2019. ISBN 9781728136769. doi: 10.1109/ICARES.2019.8914343.

[87] S.A. Van Empelen and R. Vos. Effect of engine integration on a 4.6%-scale flying-v subsonic transport. In *AIAA Scitech 2021 Forum*, pages 1–15, 2021. ISBN 9781624106095.

[88] Hado van Hasselt. Double Q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

[89] Hado van Hasselt. *van Hasselt, Hado Philip. Insights in reinforcement rearning: formal analysis and empirical evaluation of temporal-difference learning algorithms*. PhD thesis, Utrecht University, 2011.

[90] E. Van Kampen, Q.P. Chu, and J.A. Mulder. Online adaptive critic flight control using approximated plant dynamics. In *Proceedings of the 2006 International Conference on Machine Learning and Cybernetics*, volume 2006, pages 256–261, 2006. ISBN 9781424400614. doi: 10.1109/ICMLC.2006.258964.

[91] Simon Van Overeem, Xuerui Wang, and Erik-Jan Van Kampen. Handling Quality Improvements for the Flying-V Aircraft using Incremental Nonlinear Dynamic Inversion. 2021.

[92] Simon Van Overeem, Xuerui Wang, and Erik-Jan van Kampen. Modelling and Handling Quality Assessment of the Flying-V Aircraft. In *AIAA Scitech 2022 Forum*. Delft University of Technology, 2022.

[93] Harm van Seijen, A Rupam Mahmood, Patrick Pilarski, Marlos Machado, and Richard Sutton. True online temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):5057–5096, 2016.

[94] D. Wada, S.A. Araujo-Estrada, and S. Windsor. Unmanned aerial vehicle pitch control under delay using deep reinforcement learning with continuous action in wind tunnel test. *Aerospace*, 8(9), 2021. doi: 10.3390/aerospace8090258.

[95] Antony Waldock, Colin Greatwood, Francis Salama, and Thomas Richardson. Learning to Perform a Perched Landing on the Ground Using Deep Reinforcement Learning. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 92(3-4):685–704, 12 2018. ISSN 15730409. doi: 10.1007/s10846-017-0696-1.

[96] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989.

[97] Paul Werbos. Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence. *General Systems Yearbook*, 22, 1977.

[98] Paul Werbos. Building and understanding adaptive systems. *IEEE Transactions on Systems, Man and Cybernetics*, 17(1):7–20, 1987.

[99] Paul Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley and Sons, 1994.

[100] Richard M. Wood and Steven X.S. Bauer. Flying wings / flying fuselages. In *39th Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics Inc., 2001. doi: 10.2514/6.2001-311.

[101] Xinyi Sola Zheng and Dan Rutherford. Fuel burn of new commercial jet aircraft: 1960 to 2019. Technical report, The International Council on Clean Transportation, 9 2020.

[102] Zhou. *Online reinforcement learning control for aerospace systems*. PhD thesis, Delft University of Technology, Delft, 2018. URL `https://doi.org/10.4233/uuid:5b875915-2518-4ec8-a1a0-07ad057edab4`.

[103] 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):489–496, 2017. doi: 10.2514/1.G001762.

[104] 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, 4 2018. ISSN 09670661. doi: 10.1016/j.conengprac.2017.12.011.