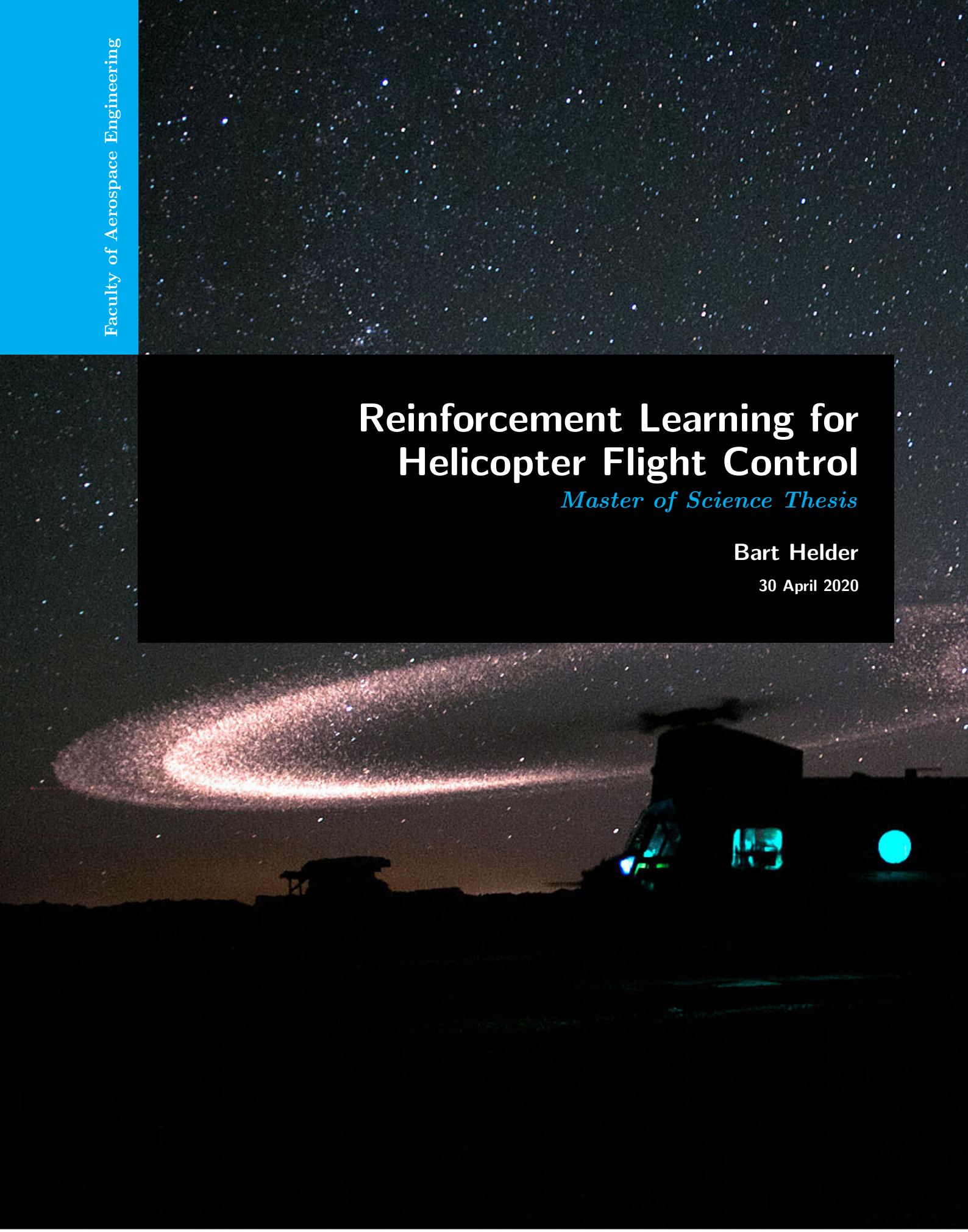
Faculty of Aerospace Engineering



Reinforcement Learning for Helicopter Flight Control

Master of Science Thesis

Bart Helder

30 April 2020

TU Delft Delft University of Technology

Challenge the future

Reinforcement Learning for Helicopter Flight Control

Master of Science Thesis

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

Bart Helder

30 April 2020

Cover Image [1]

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Reinforcement Learning for Helicopter Flight Control"** by **Bart Helder** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 30 April 2020

Readers:

dr.ir. E. van Kampen

dr.ir. M. D. Pavel

dr.ir. E. Mooij

Acronyms

ACD	Adaptive Critic Design
ADDHP	Action-Dependant Dual Heuristic Programming
ADGDHP	Action-Dependant Globalized Dual Heuristic Programming
ADHDP	Action-Dependant Heuristic Dynamic Programming
ADP	Approximate Dynamic Programming
CNN	Convolutional Neural Network
DHP	Dual Heuristic Programming
DP	Dynamic Programming
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ER	Experience Replay
FCS	Flight Control System
GAE	Generalized Advantage Estimator
GDHP	Globalized Dual Heuristic Programming
GPI	Generalized Policy Iteration
HDP	Heuristic Dynamic Programming
HER	Hindsight Experience Replay
IBS	Incremental Backstepping
IDHP	Incremental Dual Heuristic Programming
IHDP	Incremental Heuristic Dynamic Programming
INDI	Incremental Nonlinear Dynamic Inversion
MC	Monte Carlo
MDP	Markov Decision Process
PER	Prioritized Experience Replay
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
TD	Temporal Difference

TD3 Twin Delayed Deep Deterministic Policy Gradient
TRPO Trust Region Policy Optimization

List of Symbols

Greek Symbols

α	Learning rate
δ_t^V	Temporal Difference error
γ	Discount factor
$\nabla_w f(w)$	Vector of partial derivatives of $f(w)$ wrt w
$\pi(s)$	Policy (control law)
$\pi(a\|s)$	Probability of taking action a from state s under policy π
$\pi(a\|s,\theta)$	Policy parameterized with weight vector θ
σ_w	Initial variance of neural network weights

Roman Symbols

$\mathcal{A}$	Set of all actions
A_t	Action, as a random variable
G_t	The return following timestep t
H_∞	The infinite horizon control paradigm
K_β	Flap hinge spring constant
$q_\pi(s,a)$	Exact action-value function corresponding to policy π
R_{t+1}	Reward
$\mathcal{S}$	Set of all states
s	State
S_t	State, as a random variable
$v_\pi(s)$	Exact value function corresponding to policy π
$V(S_t)$	Estimated value function, evaluated at state St
$V(s\|w)$	Value function parameterized with weight vector w

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Large-scale helicopters have unique characteristics of maneuverability and low-speed performance compared to fixed-wing aircraft. They can take off and land vertically, hover in place for extended periods of time, and move in all six directions, making them occupy important niches in both military and civil aviation. However, these advantages come at a cost: helicopters are inherently unstable with complicated dynamics, and generally more unsafe than commercial air travel. The fatality rate of non-military helicopters is about 1.44 per 100,000 flight hours [9]. This high number is partially explained by the more risky nature of helicopter missions, but is still considerably high compared to the fatality rate of commercial aviation in general.

When an airliner loses all engine power, it can glide a horizontal distance proportional to its altitude and glide ratio, and ven a rather large stall can generally be recovered from. Helicopters have a way to safely land without engine power called *autorotation*, but this is not possible from every point in the flight envelope, has little room for error, and must be started within seconds of engine failure [10]. This is why helicopter pilots learn this maneuver early in their training and must practice it very often. Nevertheless, loss of control in-flight (LOC-I) is the largest key risk area for offshore helicopter operations [11], and engine failure is the cause of 75.9% of helicopter accidents in the category Systems Failure [9].

The difficult handling of helicopters combined with risky flight profiles make them particular good candidates for advanced flight control systems [12]. Traditionally, Flight Control Systems (FCSs) rely on classical control methods and use gain scheduling to switch between different controllers for each flight regime. Creating such controllers is a labor-intensive, precise task for fixed-wing aircraft, and even more so for the wide variety of flight regimes of a helicopter [12]. Besides that, such control systems turn off the autopilot and warn the pilot in case of a situation that falls outside the capabilities of the autopilot. The main task of the pilot then is to monitor the automation system, which is what humans are notoriously bad at [13]. An *adaptive flight control* system, one that could react to changing conditions, would therefore be the next logical step. In case of an engine failure, an adaptive control system could significantly improve the survival rate of the occupants. Techniques such as nonlinear dynamic inversion (NDI) and backstepping (BS) have successfully been applied to deal

with system nonlinearities [14, 15]. However, such techniques require a high quality model of the aircraft throughout its flight envelope. More recently, incremental variations of these techniques (Incremental Nonlinear Dynamic Inversion (INDI) and Incremental Backstepping (IBS)) have been implemented on real fixed-wing aircraft [16, 17] as well as high-fidelity models of rotorcraft [18]. INDI and IBS have shown promising results and a reduced model dependency. However, in return these methods require fast and accurate acceleration measurements, and while these methods have improved fault tolerance over traditional methods, they still require aircraft models a priori.

A more promising avenue of adaptive flight control seeks to use Reinforcement Learning (RL): a field of machine learning in which agents learn what actions to take by interacting with the environment. The agent is not told explicitly what good and bad actions are, but must discover this themselves by trying them out [4], while good results are reinforced by numerical rewards. The advantage of this approach is that a policy can be acquired online and purely from experience, without any knowledge of plant dynamics. Traditionally, RL was only formulated for discrete state and action spaces, in which results could be kept in a tabular format. With the introduction of function approximators, RL methods called Adaptive Critic Designs (ACDs) have been successfully applied for adaptive flight control of missiles [19], helicopters [20], business jets [21] and military aircraft [22]. More recently, two main directions in RL seem especially promising. Firstly, general methods leveraged with massive computing power have yielded world-class results in playing the games Go [23], Chess and Shogi [24], and Dota 2 [25] - problems previously deemed essentially intractable [26]. Secondly, the use of the incremental model techniques (as shown in INDI and IBS) in novel ACDs has yielded methods that learn an incremental model in real time and therefore do not require an offline learning phase. Recently, one of these methods, called Incremental Dual Heuristic Programming (IDHP) was applied for the first time to a a high-fidelity, nonlinear, six-degrees-of-freedom model of a Cessna 550 Citation II aircraft [27, 28]. However, it is not known if or how this method is applicable to rotorcraft control.

The highly nonlinear, coupled dynamics of full-scale helicopters provide unique challenges for novel flight control systems. This also provides the unique opportunity to attempt to create a single controller that can operate under normal flying conditions and perform aggressive maneuvers. As engine failure is one of the most common failure scenarios in helicopters, the adaptability of a novel flight control system can also be tested by performing a one-engine-inoperative landing. This research will take the first step in filling this knowledge gap by developing and demonstrating an online reinforcement learning controller for a high-fidelity full-scale helicopter model.

1-1 Research Objective and Questions

The design of a complete adaptive, fault-tolerant flight controller on a real helicopter is not feasible in the scope of a master's thesis research. Therefore, a research objective is defined to limit the scope to be more realistic and achievable.

Research objective: Develop an online, nonlinear, model-free, adaptive flight control system for full-scale helicopters by investigating the applicability of novel reinforcement learning frameworks on a high-fidelity helicopter model, with the purpose of designing a system capable of both normal flight and one-engine inoperative flight.

The research objective is accomplished by answering the accompanying set of five research questions, as well as sub-questions that break them down into specific and measurable pieces.

RQ1 What is the state-of-the-art of reinforcement learning for helicopter flight control?

(a) What is the current state-of-the-art in helicopter flight control?

(b) What is the current state-of-the-art in continuous RL control?

RQ2 What is the proposed baseline framework for adaptive, online flight control for a full-scale helicopter model?

(a) Which RL framework is most suitable for helicopter flight control?

(b) What should the flight control architecture of the adaptive controller be?

(c) What is the performance of the proposed system implemented on a simplified helicopter model?

RQ3 How does scaling up to controlling more complicated helicopter models affect the adaptive control system?

RQ4 How can the proposed RL framework be modified to improve controller adaptability and learning stability?

RQ5 What is the overall performance of the RL flight controller design?

(a) How quickly does the learning process of the adaptive controller converge?

(b) How does the RL controller generalize to different flight regimes?

(c) What are the online fault-tolerant capabilities of the system, for both normal flight and one-engine inoperative flight?

1-2 Structure

This report consists of four parts. In part I, the conducted research is presented in the form of a scientific paper. Part II contains the preliminary thesis which forms the groundwork of this research[1]. The preliminary thesis is structured as follows. First, a literature survey is presented in Chapter 2, which outlines the fundamentals and state-of-the-art of both reinforcement learning and helicopter flight control, as well as a more in-depth analysis of publications relevant to the research objective. This chapter concludes with a proposed baseline RL framework. Then, Chapter 3 contains a preliminary analysis of this framework applied to helicopter control. This is done by applying a simplified version of the proposed framework to a simplified helicopter model. In Part III, additional results are presented. Finally, in Part IV, the conclusions and recommendations of this thesis are presented.

[1]Part II was graded as part of the course AE4020 Literature Study

Part I

Scientific paper

Online Adaptive Helicopter Control Using Incremental Dual Heuristic Programming

Bart Helder*
Delft University of Technology, The Netherlands

Reinforcement learning is an appealing approach for adaptive, fault-tolerant flight control, but is generally plagued by its need for accurate system models and lengthy offline training phases. The novel Incremental Dual Heuristic Programming (IDHP) method removes these dependencies by using an online-identified local system model. A recent implementation has shown to be capable of reliably learning near-optimal control policies for a fixed-wing aircraft in cruise by using outer loop PID and inner-loop IDHP rate controllers. However, fixed-wing aircraft are inherently stable, enabling a trade-off between learning speed and learning stability which is not trivially extended to a physically unstable system. This paper presents an implementation of IDHP for control of a non-linear, six-degree-of-freedom simulation of an MBB Bo-105 helicopter. The proposed system uses two separate IDHP controllers for direct pitch angle and altitude control combined with outer loop and lateral PID controllers. After a short online training phase, the agent is shown to be able to fly a modified ADS-33 acceleration-deceleration manoeuvre as well as a one-engine-inoperative continued landing with high success rates.

Nomenclature

a_t	=	Action taken based on the information of state s_t
e_a, e_c	=	Actor and critic error
F_t, G_t	=	State and control matrix of recursive least squares estimator
L_A, L_C	=	Actor and critic loss
n, m	=	Number of states, number of actions
$\hat{P}_t$	=	Covariance matrix of recursive least squares estimator
P, Q	=	State-selection and weighting matrices
p, q, r	=	Aircraft body rotational rates
r_t	=	Reward obtained from transition to state s_t
s_t, s_t^r	=	State and reference state
u, v, w	=	Aircraft body velocities
$V(s)$	=	State value function
X_t	=	Combined state and action vector
x, y, z	=	Location of aircraft center of gravity in Earth coordinates
γ	=	Discount factor
$\delta_{col}, \delta_{lon}, \delta_{lat}, \delta_{ped}$	=	Helicopter inputs: collective, longitudinal cyclic, lateral cyclic, and pedal
ϵ_t	=	Prediction error or innovation
η^{col}, η^{lon}	=	Collective and longitudinal learning rates
η_a, η_c	=	Actor and critic learning rates
Φ_t	=	Parameter matrix of recursive least squares estimator
ϕ, θ, ψ	=	Aircraft Euler angles: roll, pitch, yaw
κ	=	Recursive least squares forgetting factor
$\lambda_{0_{mr}}, \lambda_{0_{tr}}$	=	Main- and tail rotor normalized uniform inflow velocity
$\lambda(s)$	=	Critic: state value function derivative
$\lambda(s_t, s_t^r, w_c), \lambda'(s_t, s_t^r, w_{c'})$	=	Critic, target critic
$\pi(s)$	=	Policy: state to action mapping

*MSc. Student, Control & Simulation, Faculty of Aerospace Engineering, Delft University of Technology

σ_w	=	Standard deviation of neural network weight kernel initializer
τ	=	Target critic mixing factor

I. Introduction

Large-scale helicopters have unique characteristics of maneuverability and low-speed performance compared to fixed-wing aircraft. They can take off and land vertically, hover in place for extended periods of time, and move in all six directions, making them occupy important niches in both military and civil aviation. However, compared to fixed-wing aircraft, these advantages come at a cost: helicopters are inherently unstable with complicated dynamics, and generally less safe than commercial air travel [1]. Although this is partly due to being used for risky missions in the first place, loss of control in-flight (LOC-I) is the largest key risk area for helicopter operations [2], and engine failure is the cause of 75.9% of helicopter accidents in the category systems failure [1].

The difficult handling of helicopters combined with their risky flight profiles make them especially good candidates for advanced flight control systems [3]. Traditionally, flight control systems rely on classical control methods and use gain scheduling to switch between different controllers for each flight regime. Creating these controllers is a labor-intensive, precise task for fixed-wing aircraft, and even more so for the wide variety of flight regimes of a helicopter [3]. Furthermore, the accompanying strategy for situations that fall outside what the autopilot can handle is to turn the autopilot off and give a warning to the pilot, whose main task is then to monitor the automation system, which is what humans are notoriously bad at [4]. An *adaptive flight control* system, one that could react to changing conditions, would therefore be the next logical step. Techniques such as nonlinear dynamic inversion and backstepping have successfully been applied to deal with system non-linearities [5, 6]. However, these techniques require a high quality model of the aircraft throughout its flight envelope. More recently, incremental variations of these techniques (incremental nonlinear dynamic inversion [7] and incremental backstepping [8]) have been implemented on real fixed-wing aircraft [9, 10] as well as high-fidelity models of rotorcraft [11], and have shown promising results and a reduced model dependency. However, in return for that reduced model dependency, these methods require fast and accurate acceleration measurements, and while these methods have improved fault tolerance over traditional methods, they still require aircraft models a priori.

Another promising avenue of adaptive flight control seeks to use Reinforcement Learning (RL), a field of machine learning where agents learn what actions to take by interacting with the environment. The agent is not told explicitly what good and bad actions are, but must discover this themselves by means of trial and error [12]. Good results are reinforced by numerical rewards. This has as an advantage that a policy can be learned online and purely from experience, without any knowledge of plant dynamics. Traditionally, RL was only formulated for discrete state and action spaces, where results could be kept in a tabular format. With the introduction of function approximators, RL methods called Adaptive Critic Designs (ACDs) have been successfully applied for adaptive flight control of missiles [13], helicopters [14–16], business jets [17] and military aircraft [18]. However, these methods often need hundreds to thousands of offline training episodes, both to approximate a global nonlinear system model as well as to train the controllers themselves, which requires a high-quality simulation model of the controlled system [19–21].

It becomes clear that neither incremental control techniques nor traditional ACDs will lead to true model-free control. Based on a synthesis between the advancements in incremental model techniques and RL, two novel frameworks called Incremental Heuristic Dynamic Programming (IHDP) [22] and Incremental Dual Heuristic Programming (IDHP) [23] have been proposed. These frameworks learn an incremental model in real time and therefore do not require an offline learning phase. The feasibility of this approach was demonstrated in [24], where it was shown that the IDHP framework could be used for near-optimal control of a CS-25 class fixed-wing research aircraft without prior knowledge of the system dynamics or an offline learning phase. Compared to small, fixed-wing aircraft in cruise, rotorcraft have relatively slow control responses and are unstable or marginally stable in almost all flight regimes. One design choice in [24] traded speed of convergence away for increased learning stability. In online adaptive control of rotorcraft, this trade-off is non-trivial, as there is the possibility of the system itself diverging before the controller has learned to control it.

The contribution of this paper is the applicability of the IDHP framework for online adaptive control of a nonlinear, six-degree-of-freedom simulation model of a MBB Bo-105 helicopter. To reduce the scope of the research, only the collective and longitudinal cyclic were controlled by RL agents. A control system is proposed containing two separate IDHP agents to control the collective and longitudinal cyclic, directly tracking a reference altitude and pitch angle, respectively. Outer loop control and the lateral motions are controlled by conventional PID controllers.

The remainder of this paper is structured as follows. Section II explains the working principles behind basic RL and follows up with the IDHP algorithm. In Section III, the simulation model is discussed and the integration

of IDHP in a complete control system is shown. Next, Section IV describes the experiments performed to find the optimal hyperparameters for this set-up and to test the performance of the resulting control system. The results of these experiments are discussed in Section V. Finally, Section VI concludes the paper.

II. Reinforcement Learning Framework

In this section, flight control is reformulated as a reinforcement learning problem. Afterwards, the IDHP framework is described in terms of both architecture and update rules.

A. RL problem formulation

In RL, the interaction between agent and environment is generally modeled as a Markov Decision Process (MDP), and the internal mechanics of the environment are completely hidden from the agent [12]. In this paper, a modified MDP framework is used where the reward function is a separate entity whose structure is known to the agent. This formulation is often used in ADP literature [13–18].

Flight control can be described as the process of minimizing the difference between the actual state of an aircraft s_t and a variable reference s_t^r. Reformulated as an MDP, this can be described as follows. At each timestep t, the agent chooses an action a_t based on the state s_t, reference state s_t^r, and the current policy π, as shown in Eq. (1). The environment then provides a scalar reward r_{t+1} and new state s_{t+1}. The goal of the agent is to learn a parameterized, deterministic policy, mapping state to action, that maximizes the cumulative sum of future discounted rewards, also known as the return. The mapping of state to expected return is known as the (state-)value function and is described in Eq. (2). Here, the parameter $\gamma \in [0, 1]$ is called the discount factor.

$$a_t = \pi(s_t, s_t^R) \tag{1}$$

$$V(s_t) = \mathbb{E}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots\right\} = \mathbb{E}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1}\right\} \tag{2}$$

In Approximate Dynamic Programming (ADP), the control theory perspective on the reinforcement learning problem, the most common approach is the use of actor-critic methods, also called Adaptive Critic Designs (ACDs). In ACDs, the tasks of action selection and state evaluation are handled by separate structures called Actor and Critic, respectively.

B. Incremental Dual Heuristic Programming

The structure of the IDHP agent is based on that first described in [23] and expanded upon in [24], and contains four main parametric structures: actor π, critic λ, target critic λ', and an incremental model of the plant. Fig. 1 shows how the components of the agent interact with the environment in a single timestep. The colored blocks represent parts of the agent while the white blocks are part of the environment. This section gives an overview of the different parts visible in this model.

The original IDHP algorithm works forward in time, which requires predicting the states one timestep ahead. This has as an advantage that multiple updates can be done in every timestep, but also means that the accuracy is entirely dependent on the quality of the prediction. In this paper, a backward-in-time approach such as in [15, 24] is chosen. Though this means only a single update can take place every timestep, the high update frequency assumed for the incremental model as described in section II.B.1 and reduced reliance on forward prediction outweigh this loss.

1. Actor and critic neural networks

In this paper, neural networks are chosen as the function approximator for the actor, critic, and target critic. Specifically, single-hidden-layer fully-connected multi-layered perceptrons (MLPs) are the structure of choice. They are easy to use with RL libraries such as Tensorflow and PyTorch, widely used in RL-for-flight-control literature [13–24], can theoretically approximate any function arbitrarily well [25], and are differentiable. The critic estimates the partial derivative of the state-value function with respect to the states, while the actor provides a direct mapping between the current state and the action to take. Their structures are shown in Fig. 2. The input of both network types is the same: a combination of their respective input states and tracking errors, which are further elaborated on in Section III.B. Both networks use a single hidden layer with ten neurons, using hyperbolic tangent activation functions. The output of the

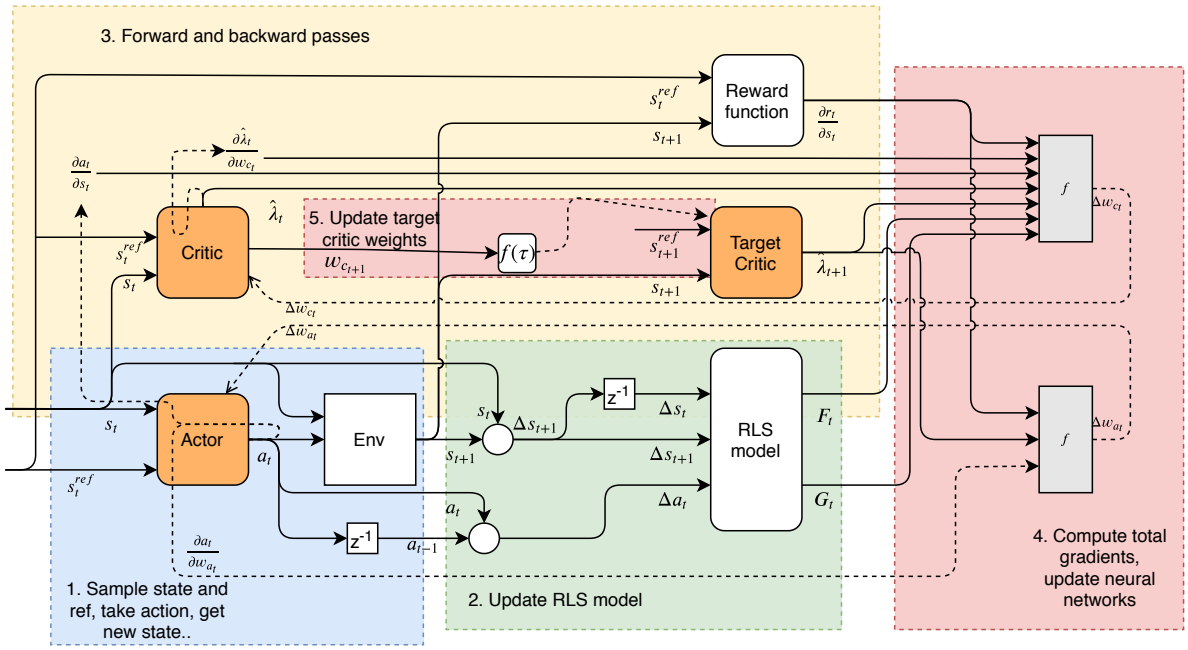


Fig. 1 Flowchart of the information during a single complete time-step of the IDHP learning framework. Solid lines feed-forward information, while dashed lines indicate feed-back update paths.

critic and target critic networks consists of a linear output layer with the same dimension as the input. The actor network has a single sigmoid output neuron, corresponding to the required input range of the helicopter model used as explained in Section III.A.

2. Target critic

A separate target network as first introduced in [26, 27] is often used to stabilize learning in a Deep RL context by decoupling action selection and evaluation. This idea was successfully applied to IDHP in [24], where it was shown that a separate target critic λ' with weight vector $w_{c'}$ increased the stability of the learning process at the cost of learning speed.

3. Incremental model

In contrast to older ADP methods, IDHP uses online estimation of an instantaneous linear model identified through Taylor expansion. Consider a discrete-time, nonlinear system $s_{t+1} = f(s_t, a_t)$. A Taylor expansion of this system around t_0 yields Eq. (3). By choosing the operating point to be $t_0 = t - 1$, a number of new definitions can be made. Defining the partial derivatives of the state-transition function to be $F_t = \frac{\partial f(s_t, a_t)}{\partial s_t}$ (the system matrix) and $G_t = \frac{\partial f(s_t, a_t)}{\partial a_t}$ (the control matrix), as well as defining the state and control *increments* to be $\Delta s_t = (s_t - s_{t-1})$ and $\Delta a_t = (a_t - a_{t-1})$, results in the incremental form of the Taylor expansion shown in Eq. (4). Given the assumption of a high sampling rate and slow dynamics, the incremental model form provides a valid linearized, time-varying approximation to the real nonlinear system [28].

$$s_{t+1} \approx f(s_{t_0}, a_{t_0}) + \frac{\partial f(s,a)}{\partial s}|_{s_{t_0},a_{t_0}}(s_t - s_{t_0}) + \frac{\partial f(s,a)}{\partial a}|_{s_{t_0},a_{t_0}}(a_t - a_{t_0}) \tag{3}$$

$$\begin{aligned} s_{t+1} &\approx s_t + F_{t-1}(s_t - s_{t-1}) + G_{t-1}(a_t - a_{t-1}) \\ \Delta s_{t+1} &= F_{t-1}\Delta s_t + G_{t-1}\Delta a_t \end{aligned} \tag{4}$$

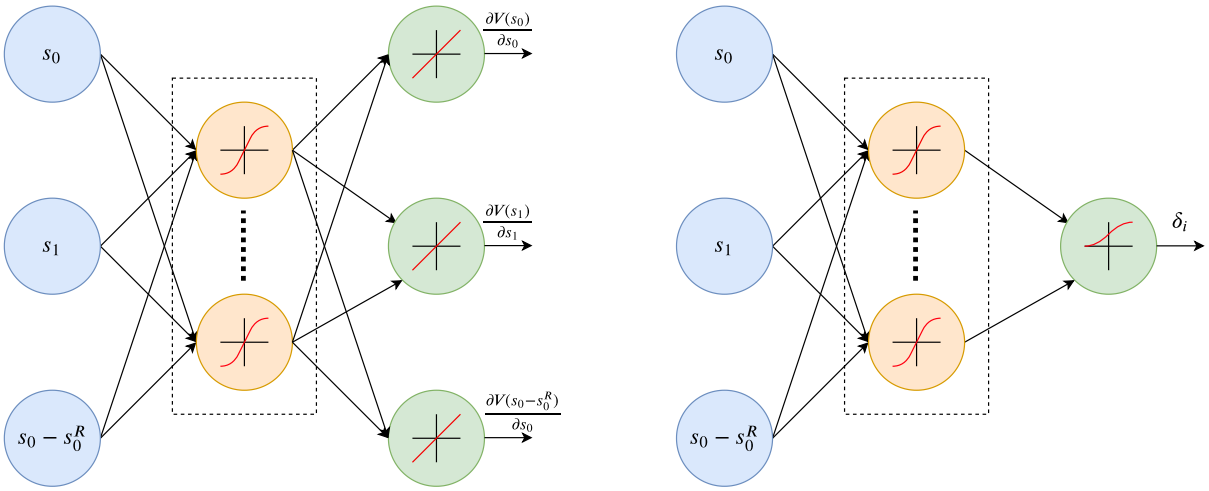


Fig. 2 Structure of the actor and critic neural networks. Both networks have a single hidden layer with ten neurons.

4. Reward function

In this paper, the reward is designed to be a negative, weighted, squared difference between the reference state and the new actual state, as defined in Eq. (5), with $P \in \mathbb{R}^{p \times n}$ the (Boolean) state selection matrix and $Q \in \mathbb{R}^{p \times p}$ the state weighting matrix. This definition provides the agent with a rich, informative reward signal at each timestep and is inherently differentiable, which is a requirement for the IDHP algorithm explained in section II.B. The derivative of the reward function with respect to the state is given in Eq. (6).

$$r_{t+1} = -P\left(s_{t+1} - s_t^r\right)^T QP\left(s_{t+1} - s_t^r\right) \tag{5}$$

$$\frac{\partial r_{t+1}}{\partial s_{t+1}} = -2P\left(s_{t+1} - s_t^r\right)^T QP \tag{6}$$

C. Update rules

In this section, the update rules for the four parametric structures in the IDHP agent are described in the same order as in which they are updated in each timestep. For brevity, the reference state s_t^r, actor weight $w_a(t)$, critic weight $w_c(t)$, and target critic weight $w_{c'}(t)$ are not explicitly shown. Therefore, the four entities in Eq. (7) are interchangeable.

$$\lambda(s_t, s_t^R, w_c(t)) = \lambda(s_t) \qquad \lambda'(s_t, s_t^R, w_{c'}(t)) = \lambda'(s_t) \qquad \pi(s_t, s_t^R, w_a(t)) = \pi(s_t) \tag{7}$$

1. Incremental model

The incremental model is identified through Recursive Least Squares (RLS) estimation. RLS is similar to a Kalman filter, making it very efficient in both computational and memory cost, and avoiding any potential problems with matrix inversions. The current state and control matrix are estimated together in one parameter matrix $\hat{\Theta}_t$, as shown in Eq. (8).

$$\hat{\Theta}_{t-1} = \begin{bmatrix} \hat{F}_{t-1}^T \\ \hat{G}_{t-1}^T \end{bmatrix} \tag{8}$$

The parameter matrix is accompanied by a covariance matrix P_t, which provides an indication of the reliability of the parameter estimates. For the update process, first, a prediction of the next state increment, $\Delta\hat{s}_{t+1}$, is made using the current state and action increments as well as the current parameter matrix, as shown in Eqs. (9) and (10).

$$X_t = \begin{bmatrix} \Delta s_t \\ \Delta a_t \end{bmatrix} \tag{9}$$

$$\Delta\hat{s}_{t+1} = \left(X_t^T \hat{\Theta}_{t-1}\right)^T \tag{10}$$

This prediction is then compared with the actual state increment, and the prediction error, also known as innovation, is computed as $\epsilon_t = (\Delta s_{t+1} - \Delta\hat{s}_{t+1})^T$. Finally, the parameter and covariance matrices are updated according to Eqs. (11) and (12), respectively, where $\kappa \in [0, 1]$ is the scalar forgetting factor, which exponentially decays the importance of older measurements.

$$\hat{\Theta}_t = \hat{\Theta}_{t-1} + \frac{\hat{P}_{t-1} X_t \epsilon_t}{\kappa + X_t^T \hat{P}_{t-1} X_t} \tag{11}$$

$$\hat{P}_t = \frac{1}{\kappa}\left[\hat{P}_{t-1} - \frac{\hat{P}_{t-1} X_t X_t^T \hat{P}_{t-1}}{\kappa + X_t^T \hat{P}_{t-1} X_t}\right] \tag{12}$$

2. Critic

The critic in IDHP estimates the partial derivative of the state-value function with respect to the states: $\lambda(s_t, s_t^R) = \frac{\partial V(s_t, s_t^R)}{\partial s_t}$. The critic is updated by means of a one-step temporal difference (TD) backup operation that minimizes the mean-squared error of the critic error: $L_C = \frac{1}{2} e_c^2$. The critic error is the partial derivative of the one-step TD error with respect to the state vector as defined in Eq. (13).

$$\begin{aligned} e_c &= -\frac{\partial\left[r(s_{t+1}, s_t^R) + \gamma V(s_{t+1}) - V(s_t)\right]}{\partial s_t} \\ &= -\left[\frac{\partial r(s_{t+1}, s_t^R)}{\partial s_{t+1}} + \gamma\lambda'(s_{t+1}, s_{t+1}^R)\right]\frac{\partial s_{t+1}}{\partial s_t} + \lambda(s_t, s_t^R) \end{aligned} \tag{13}$$

The value of the next state s_{t+1} is dependent on both the previous state and the action. Therefore, its derivative with respect to the previous state, which is the final term in Eq. (13), must be expanded to contain both these pathways. The approximations to the new derivative terms, obtained previously from the RLS model, as well as the backpropagation result of the state through the actor network, can then be substituted to yield Eq. (14).

$$\begin{aligned} \frac{\partial s_{t+1}}{\partial s_t} &= \frac{\partial f(s_t, a_t)}{\partial s_t} + \frac{\partial f(s_t, a_t)}{\partial a_t}\frac{\partial a_t}{\partial s_t} \\ &= \hat{F}_{t-1} + \hat{G}_{t-1}\frac{\partial \pi(s_t, s_t^R, w_c)}{\partial s_t} \end{aligned} \tag{14}$$

Finally, the critic weights are updated through gradient descent on the critic loss, with learning rate η_c, as shown in Eqs. (15) and (16).

$$w_c(t+1) = w_c(t) + \Delta w_c(t) \tag{15}$$

$$\begin{aligned} \Delta w_c(t) &= -\eta_c \frac{\partial L_C}{\partial w_c} = -\eta_c \frac{\partial L_C}{\partial \lambda(s_t, s_t^R, w_c(t))}\frac{\partial \lambda(s_t, s_t^R, w_c(t))}{\partial w_c(t)} \\ &= -\eta_c e_c(t)\frac{\partial \lambda(s_t, s_t^R, w_c(t))}{\partial w_c(t)} \end{aligned} \tag{16}$$

3. Actor

The goal of the actor is to find a policy which maximizes the state-value function: the optimal policy π^*. Consequently, the optimal action a^* is defined as:

$$a_t^* = \pi^*(s_t, s_t^R, w_a(t)) = arg\max_{a_t} V(s_t, s_t^R) \tag{17}$$

Because the update takes place after a state transition, the TD(0) expansion of this value can be maximized instead. Consequently, the loss function to minimize with gradient descent is the negative TD(0) target, as shown in Eq. (18).

$$L_A = -V(s_t, s_t^R) = -\left[r(s_{t+1}, s_t^R) + \gamma V(s_{t+1}, s_{t+1}^R)\right] \tag{18}$$

The state-value function is not directly dependent on the weights of the actor. Therefore, the update path takes place through the critic, reward function and environment model instead, as shown in Eq. (19).

$$\begin{aligned}\frac{\partial L_A}{\partial w_a} &= -\frac{\partial \left[r(s_{t+1}, s_t^R) + \gamma V(s_{t+1}, s_{t+1}^R)\right]}{\partial a_t}\frac{\partial a_t}{\partial w_a} \\ &= -\left[\frac{\partial r_t}{\partial s_{t+1}} + \gamma\frac{\partial V(s_{t+1})}{\partial s_{t+1}}\right]\frac{\partial s_{t+1}}{\partial a_t}\frac{\partial a_t}{\partial w_a} \\ &= -\left[\frac{\partial r_t}{\partial s_{t+1}} + \gamma\lambda'(s_{t+1})\right]G_{t-1}\frac{\partial \pi(s_t)}{\partial w_a}\end{aligned} \tag{19}$$

As with the critic, the actor weights are updated through gradient descent with learning rate η_a, as shown in Eq. (20).

$$w_a(t+1) = w_a(t) + \Delta w_a(t) = w_a(t) - \eta_a\frac{\partial L_A}{\partial w_a} \tag{20}$$

4. Target critic

Finally, the target critic is updated towards the critic using soft updates [27], also known as Polyak averaging [29], as shown in Eq. (21), where τ indicates the (usually small) mixing factor.

$$w_{c'}(t+1) = \tau w_c(t+1) + (1-\tau)w_{c'}(t) \tag{21}$$

III. Controller design

In this section, the design of the flight controller is discussed. First, the helicopter model used in the experiments is introduced. Afterwards, the proposed flight control architecture used to control this model is presented, and the most important hyperparameters are given.

A. Helicopter model

The helicopter model used is a nonlinear, six-degrees-of-freedom model of the Messerschmitt-Bölkow-Blohm (MBB) Bo 105 which was developed at the TU Delft [30, 31] and subsequently modified with engine and rotor speed dynamics [32]. The main rotor inflow is assumed to be uniform, with analytical blade element equations used for the forces and moments. The main rotor speed is dependant on the interplay between four sources of drag and the engine, the dynamics of which in turn are based on [33] with reaction time modeled as a first-order lag. The tail rotor is modeled as an actuator disk, and linear aerodynamics are used to model the horizontal and vertical tails as well as the fuselage. The simulation model is run at 100Hz, assuming synchronous clean measurements and no turbulence. The state and action space of the model are given in Eq. (22) and Eq. (23).

$$s = \begin{bmatrix} u & v & w & p & q & r & \phi & \theta & \psi & x & y & z & \lambda_{0_{mr}} & \lambda_{0_{tr}} & \Omega & P_{req} & P_{av}\end{bmatrix}^T \tag{22}$$

$$a = \begin{bmatrix}\delta_{col} & \delta_{lon} & \delta_{lat} & \delta_{ped}\end{bmatrix}^T \tag{23}$$

Control inputs are given in percentages corresponding to how far that control input is between its minimum and maximum angle, with maximum being up or right, depending on the control channel. Because of phase lag inherent in rotorcraft, these control inputs do not correspond 1:1 with their respective control angles, as a certain degree of mixing occurs in the swashplate [33]. The saturation limits of these control angles corresponding to 0% and 100% control input are given in Table 1.

B. Flight controller

The complete control system consists of a mixture of RL as well as conventional PID, and is shown in Fig. 3. Only the longitudinal channels (collective and longitudinal cyclic) are controlled by the adaptive controllers, while the lateral channels (lateral cyclic and pedal) are controlled by conventional PID controllers. It must be noted that there is a strong degree of coupling between the different channels, and a unified controller would likely permit taking advantage of this in a way that multiple separate controllers cannot [15]. On the other hand, this set-up would also inevitably lead to

Table 1 Saturation limits of the control angles of the Bo 105 simulation model [34]

Control channel	Symbol	Associated control angle	Saturation limits
Collective	δ_{col}	θ_0	[2, 18] deg
Longitudinal cyclic	δ_{lon}	θ_{1s}	[10, -5.5] deg
Lateral cyclic	δ_{lat}	θ_{1c}	[-6, 4] deg
Pedal	δ_{ped}	$\theta_{0_{tr}}$	[18, -6] deg

slower learning, as it becomes inherently harder to learn the desired behavior of multiple coupled states from a scalar reward signal. Although there is a strong degree of coupling between the different channels, the controllers have distinct enough tasks to allow for decoupling of the controllers. The resulting agent input states and action vectors are shown in Eq. (24). The state-selection and weight matrices required to extract these states from the complete state are given in Eqs. (25) and (26). A lower value for Q^{col} is chosen to bring the magnitudes of the rewards of both agents more in line with each other, allowing for easier tuning.

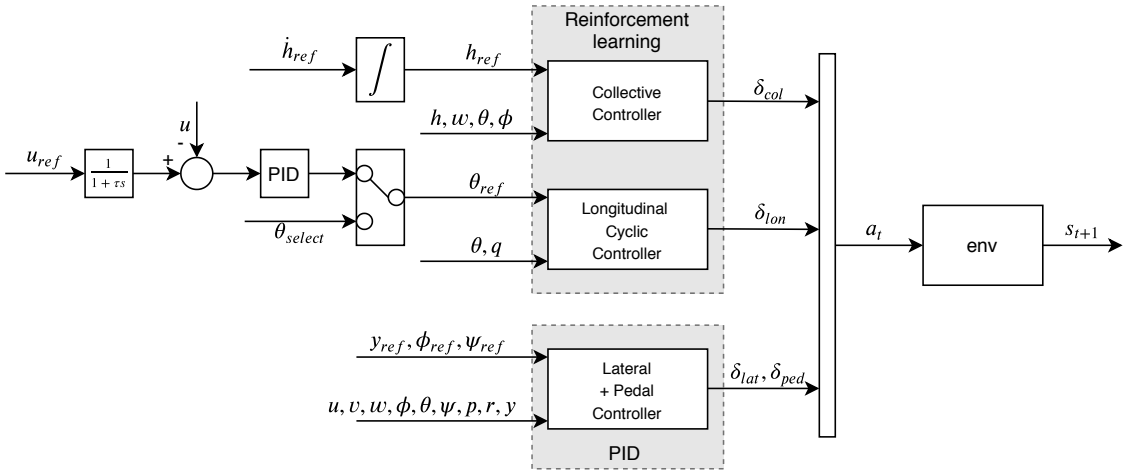


Fig. 3 High-level overview of the complete flight control system

$$s^{col} = \begin{bmatrix} z & w & (z - z_{ref}) \end{bmatrix}^T \qquad s^{lon} = \begin{bmatrix} \theta & q & (\theta - \theta_{ref}) \end{bmatrix}^T \tag{24}$$

$$P^{col} = P^{lon} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{25}$$

$$Q^{col} = \begin{bmatrix} 0.1 \end{bmatrix} \quad Q^{lon} = \begin{bmatrix} 1 \end{bmatrix} \tag{26}$$

C. Hyperparameters

The hyperparameters used by this implementation are shown in Table 2. A number of these states have their values listed as *: these were determined empirically at first, but later fine-tuned by means a grid-search over multiple hyperparameter combinations as explained in Section IV.A. The others were determined empirically. The discount factor γ is a measure of the importance of future rewards with respect to more immediate ones. A relatively large value of γ is assumed in order to place a larger weight on the slow dynamics of the helicopter.

IV. Experiment

As described in Section III, only the longitudinal motions are controlled by reinforcement learning agents. This section explains the experiments performed to test the proposed control system in the longitudinal plane, while keeping

Table 2 Hyperparameters for the IDHP agents. The values indicated with an asterix were fine-tuned in the first part of the experiment

Parameter	Description	Value
γ	Discount factor	0.8*
σ_w	NN weight initialization standard deviation	0.1*
τ	Target critic mixing factor	0.01*
η_a^{lon} η_c^{lon}	Longitudinal agent actor and critic learning rates	5*
η_a^{col} η_c^{col}	Collective agent actor and critic learning rates	0.1*
κ	RLS estimator forgetting factor	0.999
$\hat{F}_0, \hat{G}_0, \hat{P}_0$	Initial RLS matrices	$I, 0, I \cdot 10^8$

lateral motions to a minimum. Throughout all phases, the lateral motion controller had the task of keeping the deviation in lateral path distance and heading angle at a minimum. This was done by supplying the lateral PID controller with the references shown in Eq. (27).

$$y_{ref} = 0 \qquad \psi_{ref} = 0 \qquad \phi_{ref} = \phi_{trim} \tag{27}$$

The experiment consisted of two phases: an (online) training phase and a test phase. The training phase consisted of two parts. First, a training scenario was designed that could reliably allow the reinforcement learning controller to converge while containing sufficiently aggressive maneuvers for real scenarios. Next, the optimal training hyperparameters for training were determined by means of a grid search over various hyperparameter combinations and random seeds. For the test phase, two maneuvers were designed to push the limits of the newly trained controller.

A. Training phase

In the training phase, the RL controller is asked to perform basic control tasks of increasing difficulty in order to achieve a certain baseline performance. To that end, the cyclic and collective agents were trained semi-separated from each other for 120 seconds in total. The environment is initialized at level, low-speed cruise, $V_{tas} = 15\text{m/s}$. For the first 60 seconds, the cyclic is training while the collective is controlled by a PID controller that is tasked with keeping a constant altitude. In the second minute, the collective is actively trained while the cyclic fine-tunes.

The reference signal is as follows. The cyclic controller follows a reference pitch angle of format shown in Eq. (28), while the collective controller follows a reference altitude created by numerically integrating a given vertical velocity. The complete flight profile is shown in Fig. 4.

$$\theta_{ref} = A_{ref} \frac{\pi}{180} \cdot \sin\left(\frac{2\pi \cdot t}{10}\right) \tag{28}$$

It can be seen that the pitch angle reference signal amplitude A_{ref} increases in steps over the first minute of training time, starting at 10°, increasing to 15° after 20 seconds, and 20° after 40 seconds. This approach was found to lower the chance of the agent overshooting the reference significantly when provided with a very large tracking error early in training. After 60 seconds, the learning rate of the cyclic agent is reduced by 90%, the collective PID controller is switched off, and the collective RL agent starts training. The learning task of the collective agent is a steady climb with a climb rate of 2m/s for 30 seconds, followed by an altitude hold for 30 seconds. Meanwhile, the cyclic is asked for a constant pitch angle $\theta_{ref} = 2.5°$ to steadily reduce forward airspeed. At the end of the training run, the helicopter should be at steady altitude and approximately zero airspeed.

To help with the incremental model identification, an exponentially decaying sinusoidal excitation is applied to both inputs in the first eight seconds. A sine signal was preferred over other common excitation patterns such as 3211 or doublet because it exposes the model to different action increments as well as different state increments each timestep, allowing for more rapid convergence.

The optimal hyperparameters for training were found by means of a grid search. Each combination of parameters was tested for 100 trials, and the success rate and final performance of each experiment was measured. The final performance is measured in the root-mean-squared error (RMSE) of the final 10 seconds of the experiment. Those

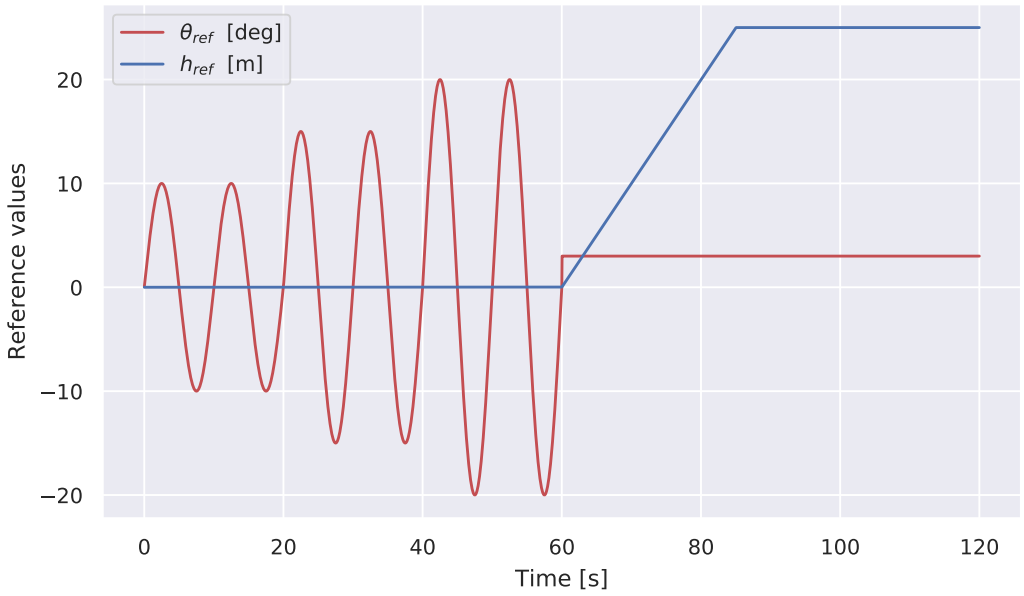


Fig. 4 Reference pitch angle and altitude during the training scenario

hyperparameters with the best combination of success rate and final performance were used to train one agent whose weights were then saved and used as a starting point for the maneuvers in the test phase.

B. Test phase

The test phase consisted of two maneuvers aimed at pushing different parts of the longitudinal envelope. Both maneuvers are initialized from a pre-trained agent, but with 90% reduced learning rates with respect to the training scenario.

The first manoeuvre was a modified ADS-33 [35] acceleration-deceleration. Its objective is to check the heave and pitch axis for aggressive maneuvering near the rotorcraft limits of performance, undesirable longitudinal-lateral coupling, and harmony between the pitch and heave controls. The desired performance characteristics are as follows. From hover, the aircraft accelerates to 25m/s using maximum power while maintaining altitude and lateral track deviations below 15 and 3 meters, respectively. After attaining the target speed, an immediate deceleration takes place, achieving at least 30° pitch-up attitude and less than 5% engine power.

The second manoeuvre was a one-engine inoperative landing based on [32]. This manoeuvre checked the controller for the ability to quickly adapt to a new trim point, perform steady flight for a while, followed by an immediate aggressive manoeuvre under reduced engine power. A single engine failure occurred at low altitude, after which the helicopter no longer had enough power to perform a balked landing. Therefore a continuous landing with flare manoeuvre was performed. The safe limits of forward and downward velocity during touchdown, u_{max} and w_{max}, were assumed to be 4.5m/s and 1.5m/s, respectively [36].

V. Results

As described before, the experiment consisted of two phases, with three parts in total: the design of a training scenario, a grid search over the best hyperparameters, and the maneuvers flown by the resulting controller. Although the training scenario design took place first, the resulting parameters came out of the settings found with the grid search. Therefore, this section presents the results of the three different experiment phases described in Section IV in a slightly different order, as shown in Fig. 5.

First, the results of the hyperparameter search are presented. Those hyperparameters with the best performance were then used as a starting point for the training phase. Finally, the controllers that resulted from the training phase were saved, and the two test maneuvers were performed by starting from the training save point.

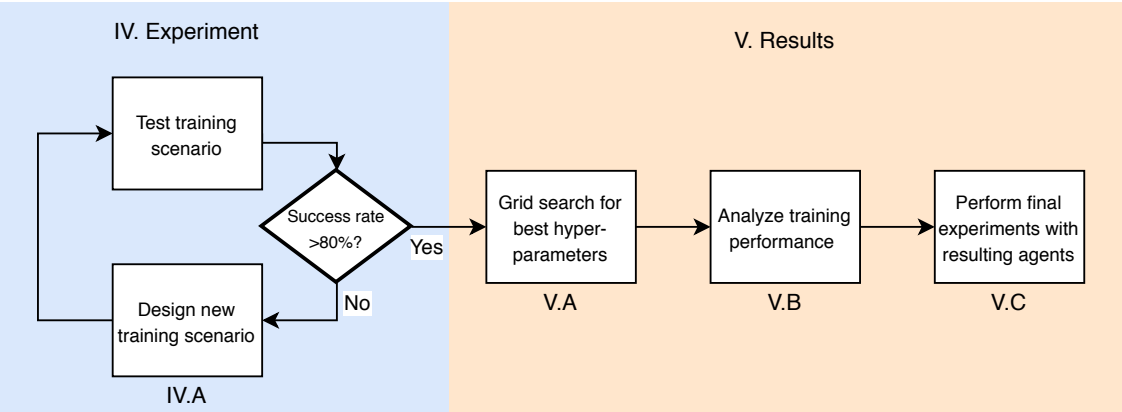
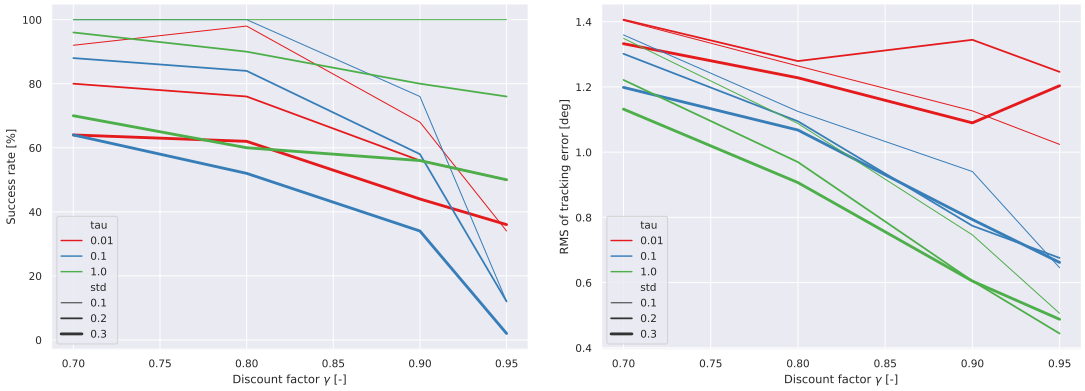


Fig. 5 Flowchart of the steps taken in the design and execution of the experiment, as well as the subsections where these parts are discussed

A. Hyperparameter search

The grid search over the hyperparameters yielded two sets of results. Firstly, the success rate, defined as the percentage of trials that did not end prematurely, is shown in Fig. 6a. Trials could end prematurely in two ways: either by breaking the performance limits of ±90° in pitch and roll, or by "parameter explosion", which is numerical overflow of the neural network weights. Secondly, the final performance of the successful runs is shown in Fig. 6b. The performance is measured in RMSE of the tracking error in the final ten seconds of successful trials.



(a) Success rates

(b) Final performance

Fig. 6 Aggregated success rates and final performance of different combinations of training hyperparameters

Inspecting these figures, a few trends become visible. First, it can be seen that the discount factor γ is simultaneously correlated with a lower success rate as well as higher final performance. This result exposes the "dual role" of the discount factor γ. As a discount factor explicitly weighs future rewards more or less strongly, it also implicitly serves as a variance reduction parameter, weighing the importance of the critic's estimate of value derivatives in the actor update. This makes a set-up with a low value of γ less prone to the parameter explosion failure mode often found in ADP. On the other hand, a high value of γ leads to stronger weighting of future rewards, which is important in tasks with relatively slow controls such as helicopter control. Secondly, in this implementation, higher values of τ were found to be correlated with both higher success rates and better final performance, all the way up to $\tau = 1$. At $\tau = 1$ the target critic immediately tracks the critic, and such is not used at all. This can be attributed to the fact that in this online learning scenario, the slow learning in presence of a target critic can actually cause the unstable helicopter environment to diverge more quickly than using a less robust but faster implementation without a target critic can. This was verified

by comparing the failure modes with and without target critic: it was found that the majority of failures with a target critic in place were loss-of-control failures, while this shifted to numerical overflow failures when no target critic was in place. Thirdly, lower values of σ_w were generally associated with higher success rates, but lower final performance. Based on these results, the hyperparameters used for training are shown in Fig. 3. This combination showed a 100% success rate in training over 100 random seeds while also having near-optimal final performance.

Table 3 Hyperparameters of the IDHP agents used in training

Hyperparameter	Description	Value
γ	Discount factor	0.95
σ_w	Weight initialization standard deviation	0.1
τ	Target critic mixing factor	1.0
η_a^{lon} η_c^{lon}	Longitudinal agent actor and critic learning rates	5
η_a^{col} η_c^{col}	Collective agent learning rates	0.1

B. Training phase

A sample training episode is shown in Fig. 7. It can be seen that both cyclic and collective are able to follow the reference signal after approximately ten seconds, after which the performance is slowly improved over the next 50 seconds. Around $t = 90$s, some yawing motion appears as the result of tail rotor saturation, which in return is the result of the collective approaching 90% in low-speed flight. In the collective controller, it can be seen that manages to achieve two different steady-state (trim) points: even though the controller was initialized at the trim for 15m/s, it has no problem holding altitude at flight speeds as low as 2m/s.

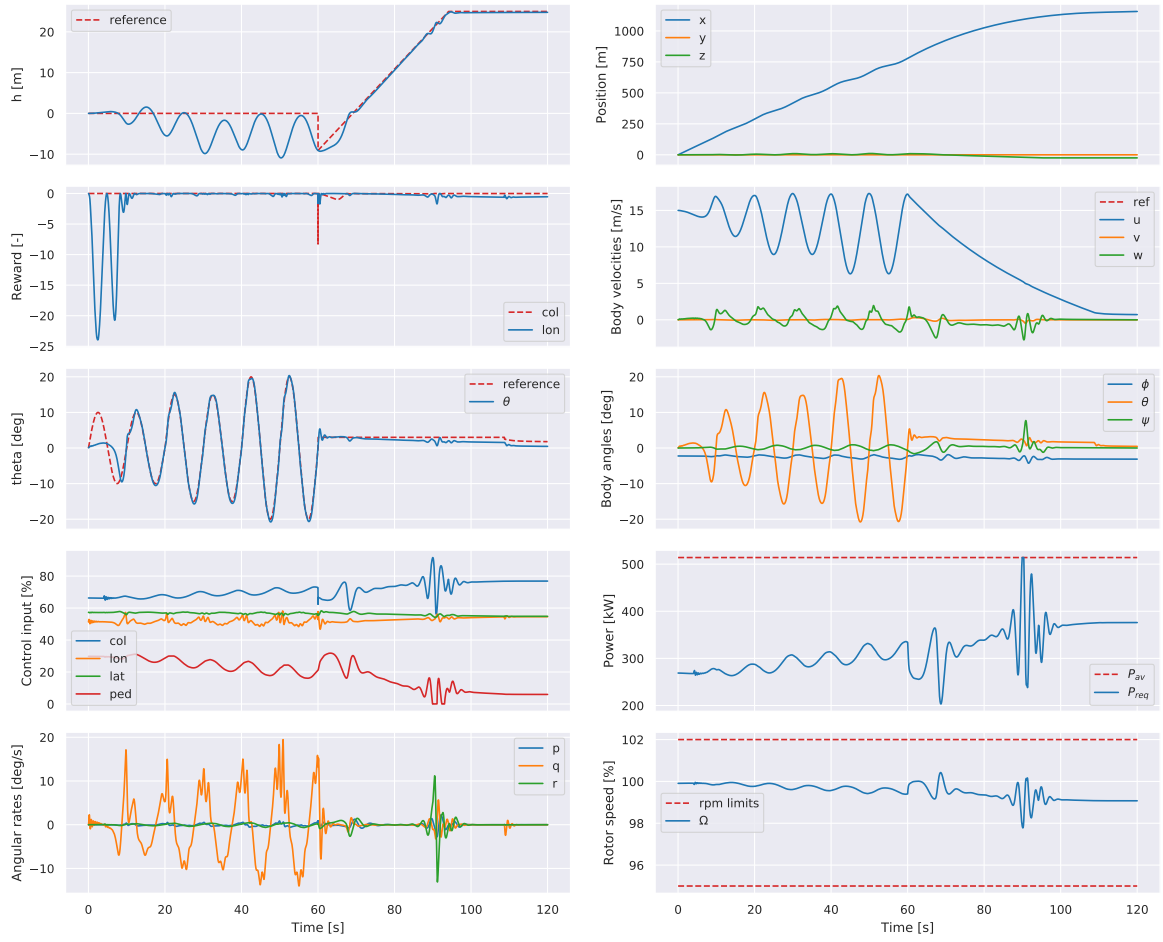
In Fig. 8, the learned parameters of the online estimated state and input matrices as well as the weights of both the actor and critic are shown. The top two plots in each subfigure show the parameters of the RLS model, and it can be seen that the parameters of both the state and input matrix in the incremental model converge within seconds, providing critical information to the actor and critic updates. The parameters of the cyclic converge immediately upon start of the scenario, while the parameters corresponding to the collective only start reacting after eight seconds. This corresponds to the timing of the input excitation signals, which are spaced eight seconds apart, as can be seen in Fig. 7. The bottom two plots in each subfigure show the actor and critic parameters, respectively. Both the cyclic and the collective agents are shown to reach their approximate final weight distributions ten seconds after starting the learning process. Afterwards, as the reward signal becomes small, and there is further fine-tuning towards the global optimum. The cyclic actor weights continue growing until the twenty second mark, after which they slowly start decreasing again. Interestingly, this corresponds to the first increase in reference signal amplitude, indicating this might be prevent overtraining from occuring. After 60 seconds, two things happen: the learning rate of the cyclic agent is reduced, reducing the oscillations in the weights, and the learning of the collective agent starts. It is also interesting to note that both sets of actor weights converge towards smaller values than their original estimates. This leads to less aggressive policies, even though the cyclic reference signal actually increases in magnitude. On the other hand, the cyclic critic weights slowly grow throughout the episode. Finally, a sharp jump in the collective critic weights can be observed at the 90 second mark, when the reference signal changes from a steady climb to an altitude hold.

C. Test phase

1. Acceleration-deceleration

Fig. 9 shows the results of a successful episode from the proposed acceleration-deceleration test. In the figure, the different flight phases are indicated with numbered dashed vertical lines. In phase 1, the helicopter is in hover. Some minor control excitation can be seen in the first second of the episode, as the environment in initialized in a slightly different state from saved agent.

In phase 2, the helicopter starts accelerating from hover to $u = 25$m/s. It can be seen that the actual forward velocity lags behind the commanded value quite significantly. The pitch angle reference is followed sharply in the sharp transient part of this phase, but is slightly above the reference during the second half of this phase. A slight altitude gain is also



(a) Tracking performance, control inputs and angular rates

(b) Remaining state variables

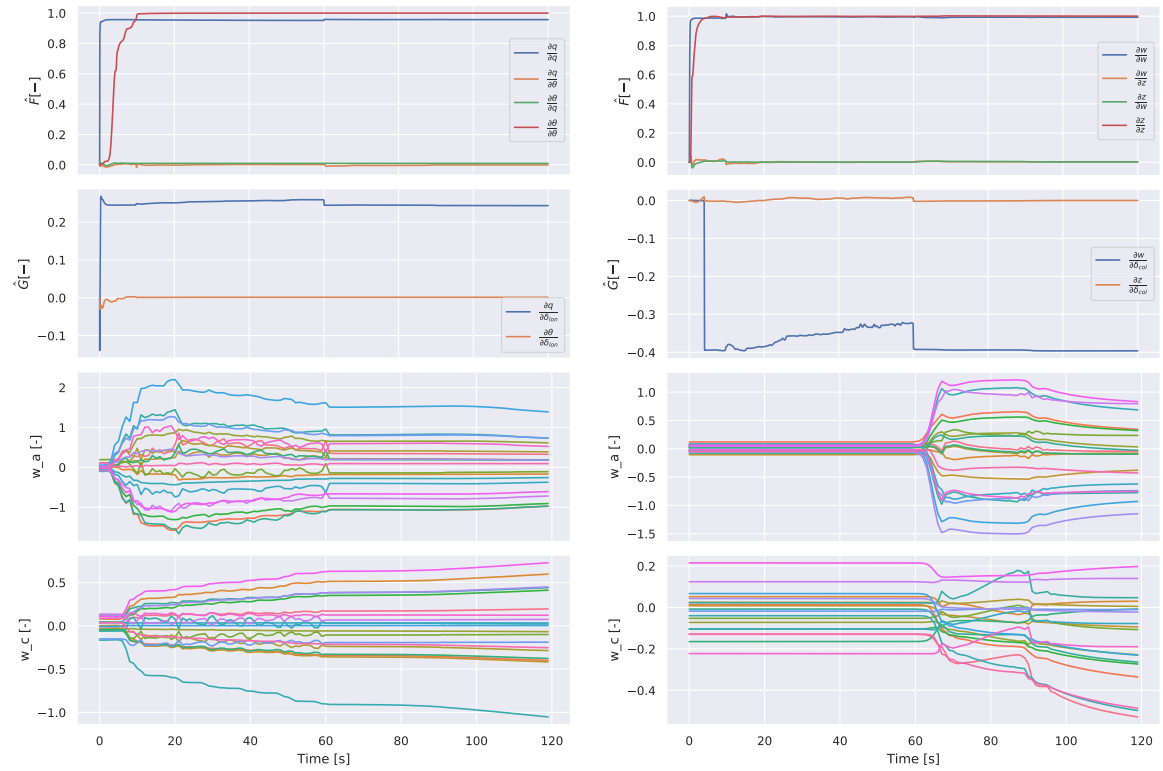
Fig. 7 Tracking performance and uncontrolled states during a typical training episode, showing stable behavior while following reference signals.

observed, though this is well within the limits of 15m. As a result of the high collective required for the acceleration, the pedal control saturates, leading to a slight yawing motion around $t = 8$s, which was the limiting factor in the aggressiveness of the manoeuvre.

As soon at the required velocity is achieved, phase 3 starts and an aggressive pitch-up is performed to decelerate the helicopter. The leads to a sudden increase in altitude due to two reasons. Firstly, as the fuselage rotates from slightly pitched-down through the neutral position, the increases the vertical component of the thrust vector. Secondly, part of the lost kinetic energy of the helicopter is transferred to the main rotor, increasing the rotor speed to slightly below the maximum safe speed, which in turn increases the total thrust force. As a reaction to this, the collective is reduced significantly and the engine power is reduced to less than 5%. The control system remains stable even with pitch rates over 40°/s. The maximum pitch-up attitude of 24° is reached two seconds into the third phase.

Throughout the episode, it can be observed that the reference pitch angle is smoothly tracked. The pitch angle does not reach the required 30° in the deceleration phase, as this would have led to large heading deviations due to pedal saturation. The maximum deviations in altitude, lateral track, and heading angle remain well within the performance bounds.

The acceleration-deceleration did not meet all the desired performance standards of the ADS-33, as it was largely held back by collective-pedal coupling. This is likely a result of deficiencies in the helicopter model, which is rather



(a) Cyclic agent

(b) Collective agent

Fig. 8 Estimates of the online identified incremental model, actor weights and critic weights during a sample training trial

simple. Similar trends were observed in [31]. When compared to real-life test data, the trends of the control inputs were similar but the magnitude of the collective required was significantly smaller.

2. Continued landing

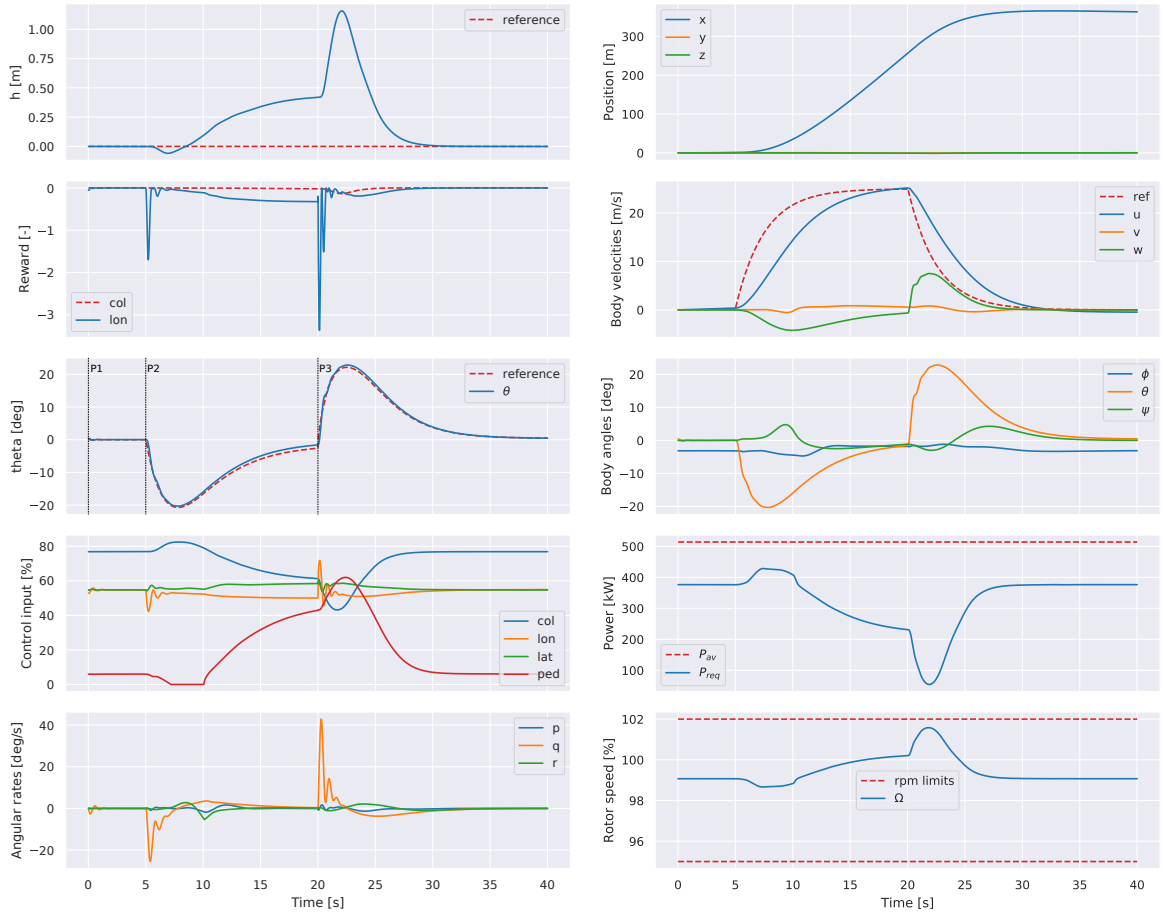
The results of the continued landing test are shown in Fig. 10.

In phase 1, a steady descending flight at a total airspeed of $V_{tas} = 18\text{m/s}$ and a flight path angle $\gamma = -6°$ is performed. The agent, which was saved at approximately hover conditions, thus has to quickly establish a new approximate trim point.

Phase 2 starts at $t = 10\text{s}$ when a single engine failure occurs, reducing the power available from the two-engine continuous limit $P_{aeo} = 514\text{kW}$ to the single-engine transient limit $P_{oei,tr} = 327\text{kW}$. With a single engine the helicopter no longer has enough power to stop in a hover, requiring the performance of a continuous landing. However, in this steady descent phase, the engine is not performing at its limits, so nothing happens yet.

In phase 3, a flare is performed to reduce the forward airspeed as much as possible. The pitch angle reaches 17.6° when approximately 5m above the ground. To prevent the helicopter from losing too much altitude, the collective is also slightly increased while the forward airspeed is decreasing. The collective is then lowered and immediately increased to set in a slow descent with a reference vertical speed of $\dot{h}_{ref} = -0.5\text{m/s}$, while the cyclic is reduced to level out the airframe. As the forward airspeed decreases, the helicopter ends up in more and more of its own downwash, decreasing the efficiency and requiring more collective to keep the descent vertical speed stable.

Phase 4 starts when the engine power reaches its maximum. As the engine power is insufficient to keep the descent steady, the collective is increased even further to trade rotor rotational speed for kinetic energy and cushion the last part of the landing. As a result, the helicopter touches down with forward and downward speeds of $u = 3.11\text{m/s}$ and $w = 0.82\text{m/s}$, which are both well within the safe values of 4.5m/s and 1.5m/s, respectively. As the fourth phase lasted

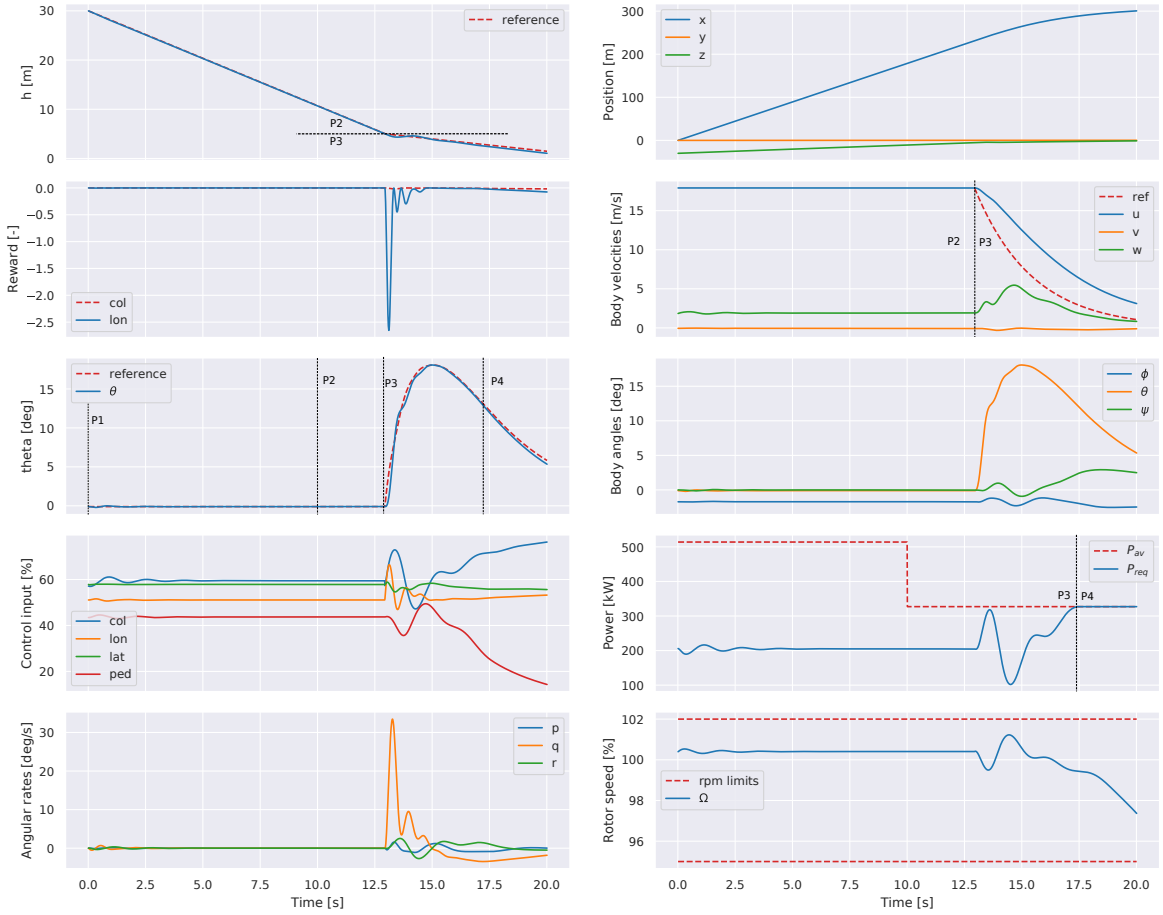


(a) Tracking performance, control inputs and angular rates

(b) Remaining state variables

Fig. 9 Results of a sample episode in the acceleration-deceleration test. Three different phases are indicated: hover, acceleration, and deceleration.

under five seconds, the choice for transient max power is warranted. Interestingly, it should be noted that the available power is likely underestimated as the ground effect is not taken into account in the simulation model. This would have increased the power available at these low speeds and altitudes, reducing the amount of rotor energy needed to cushion the landing.



(a) Tracking performance, control inputs and angular rates

(b) Remaining state variables

Fig. 10 Results of a sample episode in the one-engine inoperative landing test. Four different phases are indicated: steady descent, engine failure, flare, and rotor deceleration.

VI. Conclusion and Recommendations

The design and analysis of an IDHP-based flight controller for a Bo-105 helicopter are presented. The controller was shown to be able to reliably learn to directly control the pitch angle and altitude without an offline learning phase. Furthermore, the controller does not depend on any prior knowledge of the controlled system, and could therefore adapt to changes online. Results from [24], indicating that the addition of a target critic is a valuable addition to the IDHP framework, were shown not to necessarily apply in all situations. Though a target critic adds learning stability, this is offset by the reduced learning speed, leading to frequent loss-of-control due to the inherent dynamic instability of rotorcraft. After a 120 second online training phase, the resulting controller was shown to be able to perform two different, aggressive maneuvers when provided with a proper reference signal. It can be concluded that the proposed framework is a first step towards a helicopter flight control system based on online reinforcement learning.

To further advance this field, further research is recommended. Firstly, the control system should be expanded to control all four axes instead of only the longitudinal motions. Although it was not found in this research, it is also speculated that a modification of the current setup where multiple control inputs are controlled by one agent could improve the performance of the control system. The desired performance characteristics of the ADS-33 acceleration-deceleration manoeuvre could not be achieved because of unwanted collective-pedal coupling effects which are suggested to be due to modeling deficiencies. Therefore, the use of a higher-fidelity helicopter model with more degrees of freedom is suggested. Finally, the assumptions of clean measurements and no turbulence done in this research are not realistic.

Future research should work on quantifying the effects of these two disturbances.

References

[1] IHSTI-CIS, "Helicopter accidents: statistics, trends and causes," Tech. rep., International Helicopter Safety Team, Louisville, Kentucky, USA, 2016.

[2] Ky, P., "EASA Annual Safety Review," Tech. rep., European Aviation Safety Agency, 2018.

[3] Hu, J., and Gu, H., "Survey on Flight Control Technology for Large-Scale Helicopter," *International Journal of Aerospace Engineering*, Vol. 2017, No. 1, 2017, pp. 1–14.

[4] Bainbridge, L., "Ironies of automation," *Automatica*, Vol. 19, No. 6, 1983, pp. 775–779.

[5] Lane, S. H., and Stengel, R. F., "Flight Control Design using Nonlinear Inverse Dynamics," *1986 American Control Conference*, IEEE, 1986, pp. 587–596.

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

[7] Sieberling, S., Chu, Q. P., and Mulder, J. A., "Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 6, 2010.

[8] Acquatella, P., Van Kampen, E.-J., and Chu, Q. P., "Incremental Backstepping for Robust Nonlinear Flight Control," *Proceedings of the EuroGNC*, Delft, The Netherlands, 2013.

[9] Pollack, T., Looye, G., and Van der Linden, F., "Design and flight testing of flight control laws integrating incremental nonlinear dynamic inversion and servo current control," *AIAA Scitech Forum*, American Institute of Aeronautics and Astronautics, Reston, Virginia, 2019.

[10] Keijzer, T., Looye, G., Chu, Q., and Van Kampen, E.-J., "Flight Testing of Incremental Backstepping based Control Laws with Angular Accelerometer Feedback," *AIAA Scitech Forum*, San Diego, California, 2019.

[11] Simplício, P., van Kampen, E., and Chu, Q., "An acceleration measurements-based approach for helicopter nonlinear flight control using Incremental Nonlinear Dynamic Inversion," *Control Engineering Practice*, Vol. 21, No. 8, 2013, pp. 1065–1077.

[12] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, 2nd ed., The MIT Press, Cambridge, Massachusetts, 2018.

[13] Bertsekas, D. P., Homer, M. L., Logan, D. A., Patek, S. D., and Sandell, N. R., "Missile Defense and Interceptor Allocation by Neuro-Dynamic Programming," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, Vol. 30, No. 1, 2000.

[14] Enns, R., and Si, J., "Apache Helicopter Stabilization Using Neural Dynamic Programming," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, 2002, pp. 19–25.

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

[16] Enns, R., and Si, J., "Helicopter flight-control reconfiguration for main rotor actuator failures," *Journal of Guidance, Control, and Dynamics*, Vol. 26, No. 4, 2003, pp. 572–584.

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

[18] van Kampen, E.-J., Chu, Q., and Mulder, J., "Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics," *AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, Keystone, CO, 2006.

[19] Prokhorov, D. V., Santiago, R. A., and Wunsch, D. C., "Adaptive Critic Designs: A Case Study for Neurocontrol," *Neural Networks*, Vol. 8, No. 9, 1995, pp. 1367–1372.

[20] Balakrishnan, S. N., and Biega, V., "Adaptive-Critic-Based Neural Networks for Aircraft Optimal Control," *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 4, 1996.

[21] Prokhorov, D. V., and Wunsch, D. C., "Adaptive Critic Designs," *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, 1997, pp. 997–1007.

[22] Zhou, Y., Van Kampen, E.-J., and Chu, Q., "Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control," *Proceedings of the international micro air vehicles conference and competition (IMAV) 2016*, 2016.

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

[24] Heyer, S., Kroezen, D., and Van Kampen, E.-J., "Online Adaptive Incremental Reinforcement Learning Flight Control for a CS-25 Class Aircraft," *AIAA SciTech Forum*, 2020.

[25] Hornik, K., Stinchcombe, M., and White, H., "Multilayer feedforward networks are universal approximators," *Neural Networks*, Vol. 2, No. 5, 1989, pp. 359–366.

[26] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, 2015, pp. 529–533.

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

[28] Zhou, Y., van Kampen, E.-J., and Chu, Q., "Nonlinear Adaptive Flight Control Using Incremental Approximate Dynamic Programming and Output Feedback," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2016, pp. 493–500.

[29] Lee, D., and He, N., "Target-Based Temporal-Difference Learning," *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, CA, 2019.

[30] Pavel, M. D., "Six degrees of freedom linear model for helicopter trim and stability calculation," Tech. rep., Delft University of Technology, December 1996.

[31] Van Der Vorst, J., "Advanced pilot model for helicopter manoeuvres," Master's thesis, Delft University of Technology, 1998.

[32] Gille, M., "Flight Mechanics Exercise, Simulation of a One-Engine-Inoperative helicopter landing," Tech. rep., Delft University of Technology, 2006.

[33] Padfield, G. D., *Helicopter Flight Dynamics: The Theory and Application of Flying Qualities and Simulation Modeling*, AIAA education series, American Institute of Aeronautics and Astronautics, 1996.

[34] European Aviation Safety Agency, "Type Certificate Data Sheet R.011 BO105," Tech. rep., 2009.

[35] ADS-33E-PRF, "Aeronautical Design Standard, Performance Specification, Handling Qualities Requirements for Military Rotorcraft," Tech. rep., United States Army Aviation and Missile Command, Redstone Arsenal, Alabama, USA, 2000.

[36] Chen, R. T. N., and Zhao, Y., "Optimal Trajectories for the Helicopter in One-Engine-Inoperative Terminal-Area Operations," Tech. rep., NASA Ames Research Center, may 1996.

Part II

Literature survey and preliminary analysis

Chapter 2

Literature Review

This chapter contains a literature survey of the cross-section of topics required to apply a novel reinforcement learning controller to a full-scale helicopter. First, the fundamentals of helicopter flight control are presented in section 2-1, followed by the basics of reinforcement learning in section 2-2. With the basics established, the state-of-the-art of Reinforcement Learning is discussed: first in general in section 2-3, followed by the most influential research on its application to (helicopter) flight control in section 2-4. Finally, the overall conclusions of the literature research are presented in chapter 2-5.

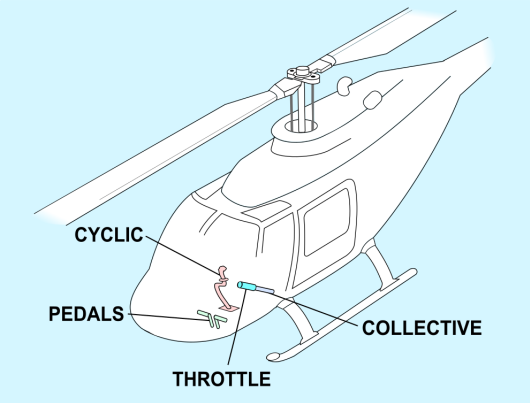
2-1 Helicopter Flight Control Fundamentals

This section contains an introduction to helicopter flight control by providing an overview of the most important concepts in rotorcraft mechanics and design. Although different configurations are possible, this text will focus on the conventional setup, consisting of a single main rotor (with anywhere from 2 to 8 blades) and a smaller vertical tail rotor mounted at the end of a tail-boom.

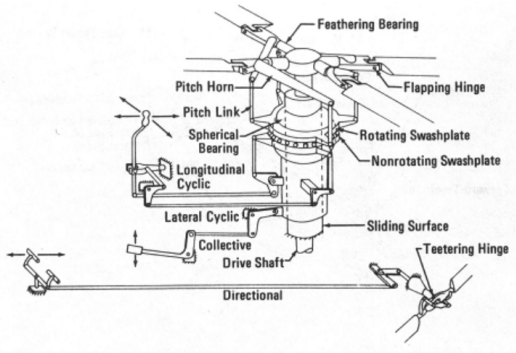
2-1-1 Flight Controls

At the heart of the control system sits the *swashplate* mechanism, which controls the pitch angle of the rotor blades. A swashplate consists of a non-rotating part, which the pilot can move, and a rotating part, which connects to the rotor blades. Both the main and the tail rotor are equipped with a swashplate. To provide position and attitude control over all three body axes, a typical helicopter uses a set of three independent controls that influence the swashplates in different ways: collective, pedal, and cyclic. Smaller helicopters also have a manual throttle to keep the rotor speed in the optimal range, though this is done by a governor in turbine helicopters. By moving the *collective* lever, the pilot moves the main rotor swashplate up or down. This adjusts the pitch angle of all rotor blades at the same time, increasing or decreasing the total rotor thrust. A similar system is used to adjust the

thrust provided by the tail rotor: this is operated by means of the anti-torque *pedals* to provide directional control. The *cyclic* lever controls the tilt of the main rotor swashplate, changing the pitch angle of the rotor blades cyclically depending on their position in a revolution around the rotor hub. This in turn tilts the rotor disk and thereby the direction of the thrust vector, allowing longitudinal and lateral control. In smaller helicopters that lack a governor to automatically adjust the rotor speed, the throttle is located at the end of the collective lever in the form of a motorcycle-style twist grip. The most important mechanical parts of a swashplate-based control system are shown in Figure 2-1.



(a) The location of helicopter flight controls in the cockpit. Image from [3]



(b) The swashplate mechanism. Image from [2]

Figure 2-1: Helicopter flight controls

2-1-2 Coupling Effects

The three controls of a helicopter do not allow for independent movement over all three axes: cross-coupling effects are typical of helicopter control. This fact is well-illustrated by examining the case where a pilot enters forward flight from hover by pushing the cyclic stick forward. By doing this, the vertical component of the main rotor thrust force is decreasing and thus the pilot has to first raise the collective lever and then lower it in order to maintain height. By lowering the collective, the engine torque decreases and thus right pedal has to be applied in order to prevent the helicopter from yawing. Applying the right pedal means a decrease in the pitch angle of the tail rotor blades and thus in the tail rotor thrust. The equilibrium of forces in the lateral plane is disturbed and thus the pilot has to apply cyclic stick to the left in order to keep its attitude. The primary and secondary impacts of each control input on the vehicle motion is summarized in Table 2-1.

2-1-3 Rotor Hub Design

In forward flight, asymmetry of the flow around the rotor disc will occur, inducing large forces and moments on a rigid rotor hub, where the rotor blades are attached to the shaft. The solution to this problem is to allow the blades to flap up and down freely while rotating. Different strategies that accommodate for this exist, with the main ones being as follows:

Table 2-1: The main input-output coupling effects of a single main rotor helicopter, derived from [8]. TR stands for tail rotor.

Input Axis	Response			
	Vertical speed (w)	Roll (ϕ)	Pitch (θ)	Yaw (ψ)
Main rotor collective (θ_0)	Main response	Due to lateral flapping and sideslip	Due to longitudinal flapping	Power change varies requirement for TR thrust
Main rotor lateral cyclic (θ_{1c})	Descent with roll angle	Main response	Due to longitudinal flapping	Undesired
Main rotor longitudinal cyclic (θ_{1s})	Desired in forward flight	Due to lateral flapping	Main response	Negligible
Tail rotor collective ($\theta_{0_{TR}}$)	Undesired, due to power changes in hover	Due to TR thrust and sideslip	Negligible	Main response

1. In a fully articulated rotor system, each blade is connected to the hub by mechanical or flexible hinges and is free to flap, feather, and lead or lag independently of others. A variation on this uses elastomeric bearings much to the same effect.

2. In a teetering or semi-rigid rotor, two opposite blades are interconnected rigidly, but are free to tilt with respect to the shaft.

3. In a hingeless or rigid rotor, the blades are flexibly attached to the hub. Instead of using hinges, each blade moves about flexible sections of the blade

The choice for a rotor hub type is not only a structural decision: the choice of hub greatly influences a rotorcraft's control response characteristics [7]. The main rotor types are shown in Figure 2-2.

2-1-4 Autorotation

Unpowered helicopter flight, better known as autorotation, is a condition which is analogous to gliding for a fixed-wing aircraft. During autorotation, the main rotor is not driven by a running engine but by air flowing upwards through the rotor disk, while the helicopter is descending. In this case, the power required to keep the rotor spinning is obtained from the vehicles potential and kinetic energy, and the task during an autorotative flight becomes one of energy management [8]. The most common reason to enter autorotation is engine failure, but the maneuver may also be performed in the event of a tail rotor failure, since autorotation hardly produces any torque, or to recover from a dangerous flight condition called vortex ring state [8]. For this purpose, a freewheeling unit is located between the gearbox and the main rotor drive shaft, disconnecting the rotor from the engine whenever the rotor rpm is higher than that of the engine. Unfortunately, autorotation maneuvers are known to be difficult to

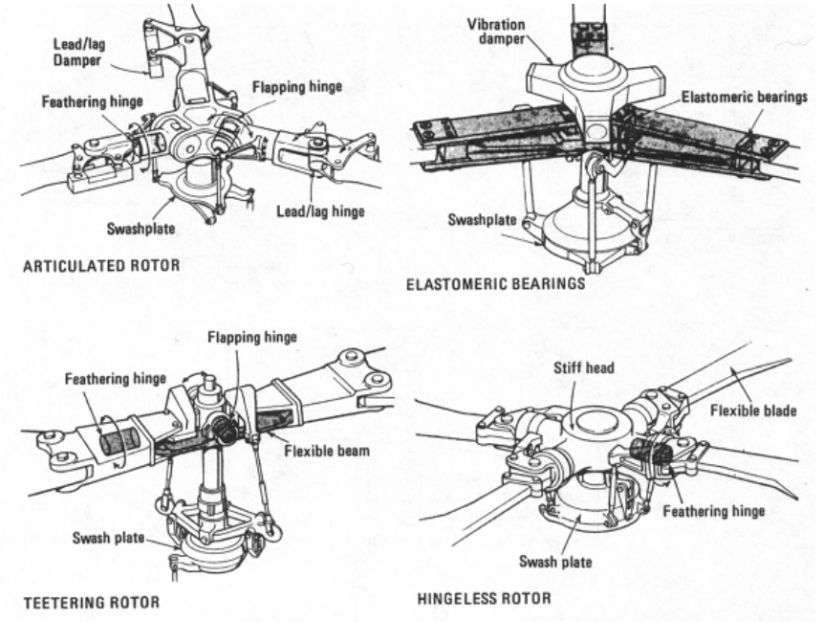


Figure 2-2: The four main rotor head configurations. Image from [2]

perform, and highly risky [3]. From a flight maneuver standpoint, a complete autorotation generally contains the following phases, illustrated by Figure 2-3.

Entry
When the decision to enter autorotation is made (position 1), the collective is immediately lowered in order to keep the rotor rpm within the operating limits. The tail rotor thrust should also be lowered because the engine is no longer providing torque. Next, the cyclic is used to pitch the nose down and gain forward airspeed, as this lowers the minimum sink rate.

Steady autorotation
In steady autorotation (position 2), a constant balance of inputs on all control channels is made to ensure the rotor rpm remains constant. The aerodynamics of this flight regime are such that some parts of the rotor blade absorb power from the air while others consume it, yielding a net negative just sufficient to make up for the losses in the transmission and gear systems.

Flare and landing
At the right landing altitude, a flare is initiated to dump forward airspeed and further decrease the descent speed (position 3). At the minimum forward airspeed, the body is leveled (position 4), and finally collective is increased to exchange rotor kinetic energy for an increase in lift, lowering descent speed and making a soft landing possible (position 5).

2-1-5 Flight Control Systems

The unique capabilities of helicopters lead to them being used in extreme conditions, placing strong demands on their precise control characteristics. This is particularly true for flight in bad weather conditions and in military roles. FCSs therefore play a major role in their

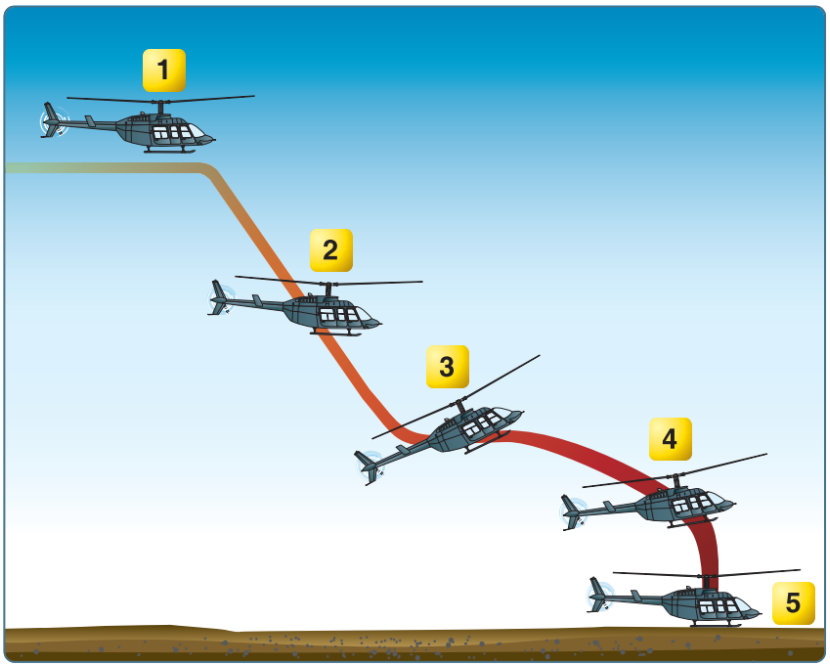


Figure 2-3: An autorotation being performed from forward flight. Image from [3]

use, to make flying easier so that the pilot can concentrate on fulfilling the primary mission [29]. Nowadays, automatic stabilization systems are considered an essential component of most military helicopters and larger civil helicopters[29]. For example, the UH-60 Blackhawk and the AH-64 Apache have conventional mechanical control systems and use low-bandwidth stabilization systems with limited authority [30]. This limited control authority, in the order of 10-20%, is typical for helicopter FCSs, though extensive research into full-authority digital [31] and optical [32] control on either of these platforms has also been performed. In general, currently implemented helicopter FCSs is designed either using gain-scheduled, PID-based feedback techniques [31, 32, 29] or with model following architecture [33]. Modern control implementations that have been tested through extensive engineering simulations include the use of fuzzy logic [34], INDI on an Apache AH-64D [18], and the H_∞ control paradigm [12].

2-2 Reinforcement Learning Fundamentals

Reinforcement learning (RL) is the study of agents and how they learn by trial and error. It formalizes the idea that rewarding or punishing an agent for its behavior makes it more likely to repeat or forego that behavior in the future. Historically, this field has different origins that have merged over the years [4], the most important of which are optimal control theory, which uses an engineering perspective, and artificial intelligence, which uses a computer science perspective. Consequently, many different terms exist that describe the same thing. For example, where a control engineer might use the terms controller, control signal, and plant, the bulk of the RL literature uses the terms agent, action and environment to mean the same things. In this text, the RL terminology is followed.

The behaviour of an agent is defined by interaction with the environment, which is modeled

as a finite Markov Decision Process (MDP). MDPs are an idealized form of reinforcement learning where, at every discrete time step, the agent makes a (partial) observation of the environment state $S_t \in \mathcal{S}$, and then decides on a new action $A_t \in \mathcal{A}$ to take. As a consequence of its action, the agent receives a scalar reward R_{t+1}, which tells it how good or bad the current state is, and ends up in a new state S_{t+1}. This is schematically represented in Figure 2-4, yielding a *trajectory* $\tau = (S_0, A_0, R_1, S_1, A_1, S_2, \ldots)$. The environment can be fully defined by the state transition probability function p, as shown in Eq. (2-1).

$$p(s', r|s, a) = Pr\left\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right\} \tag{2-1}$$

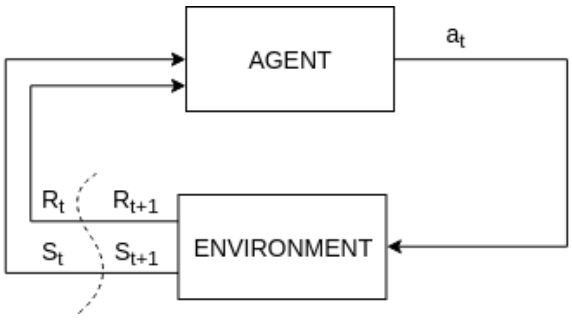


Figure 2-4: The agent-environment interaction loop in a Markov Decision Process

An important assumption in MDPs is the Markov property. A stochastic process has the Markov property if future states depend only on the current state and not on a history of states that preceded it. For a discrete process this can be written as:

$$Pr\left\{S_{t+1} = s_{t+1}|S_0 = s_0, S_1 = s_1, \ldots, S_t = s_t\right\} = Pr\left\{S_{t+1} = s_{t+1}|S_t = s_t\right\} \tag{2-2}$$

In the reinforcement learning framework, an agent's goal is to maximize the reward it receives. Generally this is the cumulative future reward (also known as return) discounted over time, as described by Eq. (2-3). Here, the parameter $\gamma \in [0, 1]$ is called the discount rate. This parameter makes returns k steps into the future less valuable than if it were received immediately, weighed by γ^k.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t_3} + \ldots = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2-3}$$

An agent's *policy* $\pi(s)$ is the behavioural strategy it follows. More formally, $\pi(a|s)$ is a mapping of the current state s to a probability distribution over a finite, discrete set of actions to take. Many reinforcement learning algorithms involve estimating a value function, which tells you how good the current state is in terms of the expected return. The policy implicitly follows from the value function by selecting an action with the highest value in each state. As the expected return depends on the current policy, it follows that policy and value function are interdependent. Different definitions of a value function exist. The state-value function is defined as $v_\pi(s) \doteq \mathbb{E}\big[G_t|S_t = s\big]$, yielding the estimated reward when starting from state $S_t = s$ while following policy $\pi(s)$. The action-value function, on the other hand, is the

expected reward when first taking action $A_t = a$ and thereafter following policy $\pi(s)$, and is defined as $q_\pi(s,a) \doteq \mathbb{E}\big[G_t | S_t = s, A_t = a\big]$.
As mentioned before, the agent's goal is to maximize the obtained reward. Solving a reinforcement learning problem, then, means finding a policy that has a higher expected return for all states than any other policy - the *optimal* policy π^*. The optimal policy is not necessarily unique, but for a given problem all optimal policies share the same optimal value function, shown in Eq. (2-4), and optimal action-value function, shown in Eq. (2-5).

$$v^*(s) = \max_\pi v_\pi(s) \tag{2-4}$$

$$\begin{aligned} q^*(s,a) &= \max_\pi q_\pi(s,a) \\ &= \mathbb{E}\left[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a\right] \end{aligned} \tag{2-5}$$

The relationship between action-value and state-value function described in Eq. (2-5) is the basis of many important reinforcement learning algorithms. Another important relationship are the recursive forms of the value function, known as the Bellman equations and shown in Eq. (2-6). Defining the value of a state in terms of possible successor states is the basis of the most important reinforcement learning algorithms.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}\left[G_t | S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{2-6}$$

2-2-1 Obtaining a Value Function

Dynamic Programming (DP) is a family of algorithms that compute the optimal value function and policy using complete knowledge about the environment. The main idea in DP is Generalized Policy Iteration (GPI), which is a method to iteratively improve the value function and policy function by means of the Bellman equations mentioned above in (2-6). This is done by alternatively performing policy evaluation and policy improvement. In policy evaluation (Eq (2-7)), the value function corresponding to the current policy is calculated. Then, during policy improvement (Eq. (2-8), a better policy is sought using the current value function.

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}\left[R_{t+1} + \gamma v_k | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \left[r + \gamma v_k(s')\right] \end{aligned} \tag{2-7}$$

$$\begin{aligned} \pi'(s) &= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma v_k | S_t = s, A_t = a\right] \\ &= \arg\max_a \sum_{s',r} p(s', r | s, a) \left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{2-8}$$

It is important to note that the complete state transition probability function $p(s', r|s, a)$ is generally not known. Two main methods exist that can learn purely from experience, real or simulated, without any knowledge of the environment: Monte Carlo (MC) methods, and Temporal Difference (TD) methods. In both methods, this is done by updating the current estimate towards a *target*, as shown in Equation (2-9), with α the update step-size, often called the *learning rate*. The main difference between MC and TD methods is the update target. Furthermore, the use of a value function estimate is indicated by use of the notation $V(S_t)$ instead of $v_\pi(s)$.

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate] \tag{2-9}$$

MC methods are a family of methods where the main point is that the value function is learned by averaging sample returns in the form of complete trajectories. Therefore, these methods are only well-defined for tasks that are episodic in nature. At the end of every episode, the true return following a visited state, G_t, is used as the update target, as shown in Eq. (2-10). Therefore, MC methods provide an unbiased estimate of the true return, but since many steps can follow the visit to a particular state, MC updates generally have high variance.

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{2-10}$$

TD methods [35], on the other hand, can update the value function on every transition by sampling the expectation and *bootstrapping* - computing an estimate based on a previous estimate. The simplest variant, one-step TD or TD(0), is shown in Eq. (2-11). Here, the full return G_t is replaced by the one-step estimate $R_t + \gamma V(S_{t+1})$. This feature allows TD methods to be used in continuing tasks and, since updates are made on every transition, TD methods tend to converge faster than MC methods. Since less random variables appear in any single update, TD updates generally have lower variance than MC methods. However, the step updates are biased towards the initial conditions of the learning parameters, and the methods themselves are more complex to analyze.

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_t + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2-11}$$

Within the many different RL algorithms have a set of dimensions among they can vary, a set of implementation choices to make. The three most important of these choices are:

Update Depth

The update depth is the different between the current timestep and that of the update target. On one extreme there are MC methods, which use the true return of an entire episode as update targets. On the other, we have one-step TD methods which only use a single transition. In between, there is a whole spectrum of valid update targets, shown in Figure 2-5. Keeping in mind the definition of the complete return in Eq. (2-3), the n-step return shown in Eq. (2-12) is defined by truncating the true return after timestep n, replacing the rest of the future rewards with the discounted estimated value:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t_3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{2-12}$$

As any of these n-step returns is a valid update target for a TD method, so is any average of any of these n-step returns. An update that averages simpler component updates is called a

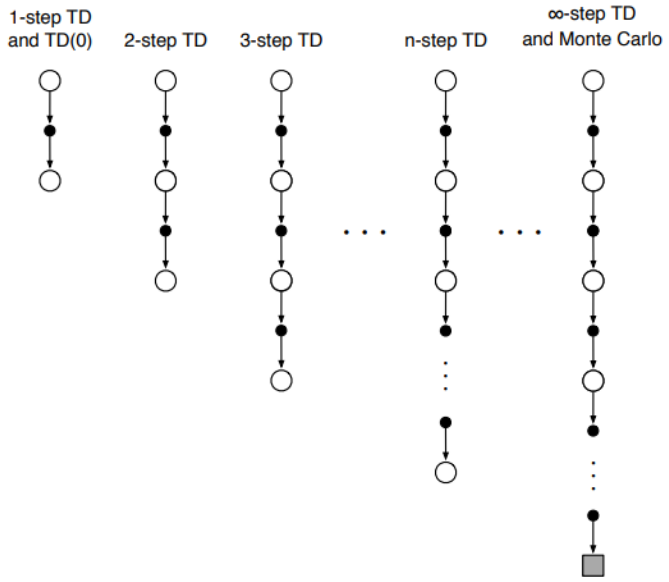


Figure 2-5: A schematic of the spectrum of n-step TD methods, ranging from TD(0) to Monte Carlo. Image from [4].

compound update, the most used of which is the λ-return, defined in Eq. (2-13). This return averages all n-step returns, weighted by λ^{n-1} and normalized by $(1-\lambda)$ to ensure all weights sum to 1.

$$G_t^{\lambda} \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t+t+n} \tag{2-13}$$

By using lambda-returns as the update target, a family of methods called TD(λ) is produced that has Monte Carlo methods at one end by setting $\lambda = 1$, and one-step TD methods at the other for $\lambda = 0$. As these methods provide a history of states and how they contributed to the current reward, they are called eligibility traces.

On-policy vs. Off-policy

In order to maximize reward, one should take the action of highest value at every timestep, but in order to discover actions and states that are potentially even better, a sub-optimal action must be taken. Reinforcement learning therefore involves a fundamental choice: exploration versus exploitation.
In general, two options are available. In on-policy learning, a value function is learned not for the optimal policy, but a near-optimal one that still explores - the most well-known example of this is Sarsa [36], shown in Eq. (2-14). Essentially, Sarsa is the action-value equivalent of TD(0) described earlier.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{2-14}$$

On the other hand, it is also possible to follow one policy (the *behaviour policy*) while learning the value function for a different *target policy* - this is called off-policy learning. The most

well-known example of off-policy learning is Q-learning [37], which approximates the optimal action-value function directly, independent of the policy followed.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{2-15}$$

Off-policy methods often use *importance sampling*, a technique to estimate expected values under one distribution given samples from another. On-policy methods are generally simpler and converge more quickly, but the complexity of off-policy methods makes them more powerful and general [4].

Model Use

In the sections above, it is assumed that the agent has no knowledge about the environment and must learn all of its behaviour by trial and error. Should this knowledge be available, it would allow an agent to plan and think ahead, resulting in a significant increase in sample efficiency: a famous example of this is the AlphaZero breakthrough [24]. However, as discussed earlier, a good model of the environment is often not available and this is exactly the reason why one would want to use RL in the first place. One option would be to learn a model from experience or data, if available. Model learning comes with its own set of complexities, the most problematic of which is that the agent might exploit any bias in the learned model, but this is nevertheless a subject that is actively researched. Methods which do not use a model at all are henceforth called model-free methods, while those that use some kind of inferred on learned model are called model-based methods. A third category, model-dependant methods, contains those that require a fixed and accurate environment model, such as dynamic programming: these are not considered here.

2-2-2 Function Approximation

In the simplest RL problems, the value function can be represented by a simple table of value estimates. However, when the number of valid states and actions grows large, the amount of state/action permutations quickly becomes intractable: this is called the *curse of dimensionality* [38]. Most interesting real-world applications also have continuous state and action spaces: using a fine discretization necessary for smooth control then leads to the same problems as above [39]. A certain degree of generalization, in the form of function approximation, becomes necessary. This function approximator could have many forms, from a simple linear-in-the-parameters model to the most complicated nonlinear ones. In practice, neural networks are often chosen because they are flexible, scalable, and can approximate any function arbitrarily well [40].

The notation $V(s|w)$ indicates a value function parameterized with the generic weight vector w. In RL literature, the symbols ϕ and θ are often used to denote the weights of a parameterized value function and policy, respectively. Whereas the value of a state can directly be adjusted in tabular RL, this is not possible when using function approximation. Instead, stochastic gradient descent methods iteratively adjust the weight vector by a small amount in the direction of the gradient of the error, which is the direction that would most reduce a certain measure of error on a transition. An often-used measure of error is the mean-squared

error, defined as $\text{MSE}(\hat{Y}_i) = \frac{1}{n}\sum_{i=0}^{n}\left(Y_i - \hat{Y}_i\right)^2$. Using this to update a value function estimate leads to the scheme shown in Eq. (2-16), where $\nabla_w f(w)$ denotes the partial derivatives of f with respect to the weights.

$$\begin{aligned} w_{t+1} &= w_t - \frac{1}{2}\alpha\nabla_w \left[v_\pi(S_t) - V(S_t|w)\right]^2 \\ &= w_t + \alpha\left[v_\pi(S_t) - V(S_t|w)\right]\nabla_w V(S_t|w) \end{aligned} \tag{2-16}$$

2-2-3 Obtaining a Policy

There are two main ways of obtaining a policy. The first is to follow the GPI scheme of greedily selecting actions from an (action-)value function as described in Section 2-2-1. Therefore, these methods are only well-defined in discrete action spaces. The second option is to directly learn a parameterized policy $\pi(a|s,\theta)$ that can select actions without consulting a value function [4]. Additionally, this makes it possible for reinforcement learning to be implemented in continuous action spaces. *Policy gradient methods* directly maximize a certain performance measure $J(\theta)$ with respect to the policy parameters via gradient descent. The policy gradient theorem [41] proves that it is possible to express the gradient of this performance measure analytically with respect to the policy parameters and is defined for the episodic case in Eq. (2-17). Here, $\mu(s)$ is a weighting of states by how often they occur under the target policy π.

$$\begin{aligned} \nabla_\theta J(\theta) &\propto \sum_s \mu(s)\sum_a q_\pi(s,a)\nabla_\theta\pi(a|s,\theta) \\ &= \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a)\nabla_\theta\pi(a|S_t,\theta)\right] \end{aligned} \tag{2-17}$$

The simplest actual implementation of this is Monte Carlo policy gradient, better known as REINFORCE [42]. Here, the sum over actions is replaced by a sample trajectory in the policy, yielding Eq. (2-18).

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_\pi\left[\sum_a \pi(a|S_t,\theta)q_\pi(S_t,a)\frac{\nabla\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)}\right] \\ &= \mathbb{E}_\pi\left[G_t\nabla\ln\pi(A_t|S_t,\theta)\right] \end{aligned} \tag{2-18}$$

As discussed in Section 2-2-1, MC methods inherently have high variance. One way to reduce variance and increase stability is subtracting the cumulative reward by a baseline: $G_t \rightarrow (G_t - b(s))$. This is allowed as long as that baseline does not change the expected value: in other words, the baseline must not depend on the action taken. This leads to the general form of policy gradient methods in Eq. (2-19), where G_t is replaced by Φ_t, which can be any of a number of expressions such that $\mathbb{E}_\pi[\Phi_t] \approx \mathbb{E}_\pi[G_t]$.

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_\pi\left[(G_t - b(s))\nabla\ln\pi(A_t|S_t,\theta)\right] \\ &\approx \mathbb{E}_\pi\left[\Phi_t\nabla\ln\pi(A_t|S_t,\theta)\right] \end{aligned} \tag{2-19}$$

Whichever formulation of the policy performance measure $J(\theta)$ is used, the parameter vector θ can directly be optimized by means of stochastic gradient descent:

$$\theta_{t+1} = \theta_t + \alpha\nabla_\theta\hat{J}(\theta_t) \tag{2-20}$$

2-2-4 Actor-Critic Methods

From the knowledge that a baseline can be freely chosen as long as certain conditions are met, a logical next step for policy gradient methods is to attack their other weakness: the inability of MC methods to be performed on-line. This can be done by replacing the full return with baseline from Eq. (2-19) by the TD estimate, such as in Eqs (2-11), (2-14), and (2-15), yielding the parameter update scheme in Eq. (2-21).

$$\theta_{t+1} = \theta_t + \alpha \left[R_t + \gamma V(S_{t+1}) - V(S_t) \right] \nabla \ln \pi(A_t|S_t, \theta) \tag{2-21}$$

As discussed in Section 2-2-1, using the TD estimate introduces some bias but significantly reduces variance and accelerates learning. Doing this requires learning both an explicit policy (which carries out actions) as well as a value function (which estimates how good those actions are, providing criticism). Consequently, methods that employ both of these are called *actor-critic methods*. Generally, both the actor and the critic are parameterized with separate neural networks, $\pi(a|s, \theta)$ and $V(s|\phi)$ respectively, which are updated by some kind of gradient descent: Figure 2-6 shows the flow of information and the update paths in a general actor-critic design. When implemented well, actor-critic methods are the most powerful reinforcement learning methods, offsetting the drawbacks of value-based and policy-based methods with each other. Consequently, most state-of-the-art RL algorithms are actor-critic methods [43].

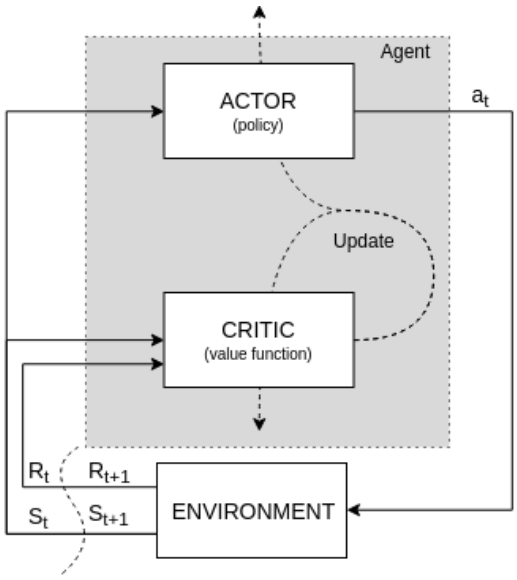


Figure 2-6: The flow of information and updates in an actor-critic design

2-2-5 Synopsis

Most reinforcement learning algorithms use the techniques mentioned in this segment in one of three ways. First, value-based methods were described - these try to find or approximate an optimal value function, and implicitly extract a policy from that. This is generally done by means of TD updates, which means they have low variance and can be done online, but have no strong convergence guarantees and cannot handle continuous action spaces. Secondly, there

are policy-based methods, which learn a parameterized policy directly without consulting a value function. The policy gradient theorem guarantees smooth convergence to a local optimum and a parameterized policy allows for continuous action spaces, but these methods are episodic in nature, and due to their Monte Carlo nature suffer from sample inefficiency and high variance. Finally, actor-critics learn both a value function and a policy directly, offsetting the drawbacks from either of these methods in return for being more complicated to implement.

2-3 Reinforcement Learning State-of-the-Art

The current state-of-the-art in reinforcement learning is studied in two different fields: Machine Learning and Adaptive Optimal Control. Consequently, this yields two perspectives on the same kind of problem. The first is Deep Reinforcement Learning (DRL), described in Section 2-3-1, which reasons from a computer science / AI perspective. The second is Approximate Dynamic Programming (ADP), which uses a control theory perspective and is discussed in Section 2-3-2. While both the research focus and the notation may differ between the two of them, there is also a lot of overlap.

2-3-1 Deep Reinforcement Learning

Deep learning is the extension of machine learning with neural networks to use complex networks with multiple hidden layers. Using these deep networks with a variety of layers such as convolution, max-pooling, recurrent, and dense layers has made it possible for neural networks to process increasingly more complex input formats: an example of such a network is shown in Figure 2-7.

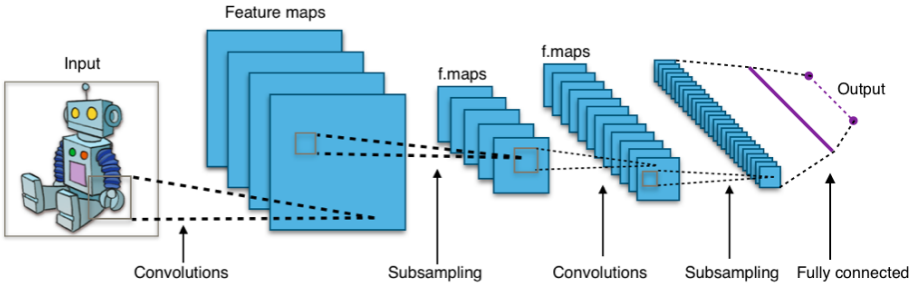


Figure 2-7: Typical architecture of a CNN, widely used in DRL. Alternating convolutional and subsampling layers allow the network to construct and detect features in images while keeping the number of free parameters relatively low. Image from [5]

The combination of deep learning with RL has made it possible to learn in high-dimensional state-spaces that were previously deemed intractable [43]. The field of DRL was launched in 2013 with the Deep Q-Network (DQN) algorithm [44]. In this research, superhuman performance on a set of 29 different classic Atari-2600 video games was achieved using raw pixels as input and scores as reward.
The main purpose of deep neural networks is to automate feature extraction: therefore, DRL focuses on extremely high-dimensional problems, such as learning locomotion on a simulated

humanoid from raw pixels: this is called *end-to-end learning*. The nature of these problems is that often thousands to millions of trials are necessary before good solutions arise, which means it is often performed in simulation only [43]. This is strongly contrasted by the control theoretical approach described in Section 2-3-2, where pre-processed states are usually available. This makes very deep networks superfluous to a control task akin to flight control. Nevertheless, a lot of breakthroughs have focused on sample efficiency and quick convergence, and might provide good improvements to the ADP frameworks if implemented there.

Since the objective of DRL is quite different from that of this thesis, the focus of this section will be on the novel techniques introduced into the field, rather than on individual papers. The most influential improvements to DRL fall in one of the following categories.

Continuous and Deterministic Policies

In the basis of RL, a policy is always assumed to be a probability distribution over a discrete number of actions. Many interesting physical tasks have continuous and high-dimensional action spaces, where finding the action that maximizes the value cannot be applied straightforwardly. For a stochastic policy, this can be tackled by learning a standard deviation and normal distribution over the actions instead of direct action probabilities, from which the action is then sampled [42]. In [45], it was proven that the policy gradient also exists for deterministic policies, which are more efficient to estimate than stochastic policies at the cost of requiring more measures to ensure enough exploration is being done. This idea was extended to DRL in [46], where the problem of exploration was solved by adding temporally correlated Ornstein-Uhlenbeck noise [47] to the action during the learning phase.

Experience Replay

As described in Section 2-2-1, off-policy methods make it possible to learn one policy while following another. Because the experience used does not have to come from the current policy, an efficient way to do this is to re-use past experience in the learning process. Experience Replay (ER) is a mechanism for storing past transitions $(S_t, A_t, R_{t+1}, S_{t+1})$ in a cyclic buffer from which the agent can sample during learning. This has two main benefits: the re-use of past transitions increases the sample efficiency, and random sampling of the transitions decreases temporal correlation, stabilizing the learning process. ER was first introduced in [48], but was popularized because of its prominent use in the breakthrough paper [23]: since then, all value-based DRL methods have incorporated ER. Two major improvements have since been done. Prioritized Experience Replay (PER)[49], is a mechanism where more important transitions are replayed more frequently, and Hindsight Experience Replay (HER) [50] improves learning from sparse rewards. The authors of [50] note that these developments are orthogonal and could theoretically be combined. In [51], PER was shown to be one of the most important additions to DQN yet.

Target and Double Networks

A straight-forward implementation of Q-learning such as in Eq. (2-15) with a neural network can have unwanted side-effects due to the combination of off-policy learning, function ap-

proximation, and bootstrapping: this is called the deadly triad [52]. Over the years, several improvements to the straight-forward Q-learning algorithm have been proposed.

First, the same Q-network that is used to calculate the value of the TD target is also the one that is being trained, which changes the target. This means the network is constantly updating towards a moving target, which can lead to instability in the learning process. In [23], this was solved by keeping two separate Q-networks: one for calculating values, and a second network $\hat{Q}$ to generate the targets. Every fixed-number of timesteps, the most recent network is copied and used as a new target network. In [46], this was improved upon by using soft target updates, where the weights of these target networks are updated by having them slowly track the learned networks:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \qquad \tau << 1$$

A second problem is that in certain situations, Q-learning performs poorly because they overestimate action-values, which causes the policy to fail as it learns to exploit errors in the Q function. These errors are introduced because the algorithm uses the maximum Q-value $max_a Q_t\left(S_{t+1}, a\right)$ as an approximation for the maximum *expected* Q-value. Double Q-learning [53] tackles this learning two separate Q-networks that are each updated with a value estimate from the other and by assigning experience randomly to either network. This idea was extended to DRL in [54], which uses the target network mentioned above for the value estimate while using the current network to select the actions. A further improvement to this concept called Clipped Double-Q was presented as part of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm in [55], which proposed to keep the two networks, but always take the minimum value estimate of the two networks to calculate the target used to train both of them. This removes any overestimation the target could introduce.

Generalized Advantage Estimation

In Section 2-2-1, multi-step returns and eligibility traces were introduced as an effective technique to accelerate reinforcement learning by smoothly assigning credit to recently visited states. Their implementation is incompatible with off-policy algorithms that utilize ER, as states are no longer processed in the order in which they are visited. For on-policy algorithms such as policy gradient methods, however, they can offer good benefit. In Section 2-2-3 a general form of policy gradient methods was described in Eq. (2-19). One powerful variant that can be used to reduce variance is the advantage function:

$$\Phi_t \leftarrow A(s,a) \doteq Q(s,a) - V(s) \tag{2-22}$$

The advantage function is a measure of how much better or worse an action is than the policy's default behaviour. In [56] the Generalized Advantage Estimator (GAE) was proposed which applies the ideas of TD(λ) to small batches of advantage estimates, using the exponentially-weighted average of multiple TD errors, as shown in Eq. (2-23). Here, δ_t^V is the TD error: $R_t + \gamma V(S_{t+1}) - V(S_t)$.

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{n=0}^{\infty} (\gamma\lambda)^n \delta_{t+n}^V \tag{2-23}$$

This idea was subsequently applied to actor-critic methods in [57], and was found to be the most crucial component in an ablation study of recent advances in DRL [51]. It has since become a staple for on-policy DRL methods [58, 59, 60].

Advanced Policy Gradients

Policy gradients work by taking a small step in parameter space, which does not necessarily translate to a small change in policy: a single step can collapse performance. Natural gradients methods are second-order methods that take a small step in policy space instead of directly in parameter space. In [61], the natural policy gradient was shown to guarantee performance increase on every step, and in [62] this approach was introduced for actor-critic methods. However, the performance benefits were still modest. The first breakthrough second-order approach was the Trust Region Policy Optimization (TRPO) algorithm, introduced in [63]. TRPO suggest the use of a trust region to prevent large changes in the policy by penalizing the KL divergence, a measure for differences in two probability distributions, as shown in Eq. (2-3-1).

$$\begin{aligned} &\nabla_{w_a} V(s) = \nabla_{w_a} \left[\frac{\pi(s|w_a)}{\pi(s|w_{a_{old}})} A(s,a) \right] \quad s.t. \\ &\mathbb{E}\left[KL\left(\pi(s|w_a), \pi(s|w_{a_{old}}) \right) \right] \leq \delta \end{aligned}$$

In a survey of the state-of-the-art of DRL in 2016, [64] found that TRPO was the best performing algorithm at that time. However, TRPO is computationally expensive as it requires the calculation of a matrix of complex second-order derivatives at every timestep. The Proximal Policy Optimization (PPO) paper [58] proposed an alternative: clip the probability ratio of the two policies and use a simple first-order algorithm to make updates, as shown in Eq. (2-3-1). This dramatically reduces the calculation cost while achieving the same performance.

$$\nabla_{w_a} V(s) = \nabla_{w_a} \log \left[min \left(\frac{\pi(s|w_a)}{\pi(s|w_{a_{old}})} A(s,a), clip \left(\frac{\pi(s|w_a)}{\pi(s|w_{a_{old}})}, 1-\epsilon, 1+\epsilon \right) A(s,a) \right) \right]$$

2-3-2 Approximate Dynamic Programming

ADP is a subfield of adaptive optimal control which provides the control theory perspective on actor-critic methods (section 2-2-4), which in this field are often called Adaptive Critic Designs (ACDs). The focus of ADP is on quick convergence and stability in the face of changing systems, while being presented with filtered and pre-processed states and often a rich reward signal. Several different ACD architectures exist, but they can be divided into three main categories: Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP), and Globalized Dual Heuristic Programming (GDHP) [65]. In each of these, the critic approximates some variant of the state-value function $V(s)$. Furthermore, each has an action-dependant (AD-) variant, which are called Action-Dependant Heuristic Dynamic Programming (ADHDP), Action-Dependant Dual Heuristic Programming (ADDHP), and Action-Dependant Globalized Dual Heuristic Programming (ADGDHP), respectively. As opposed to their action-independent variants, here the critic also takes the action as an input, and correspondingly approximates some measure of the action-value function $Q(s,a)$. Two main attributes distinguish each of these variants: the function of the critic and the requirement of an environment/plant model. These are summarized in table 2-2, and explained in more detail in the rest of this section.

The updates of the actor and critic require the gradients of an error measure with respect to the actor and critic weights respectively. For the critic, this is generally the TD error, while

for the actor this is the difference between the current value and a goal value. In both cases, the loss function to minimize is the mean squared error (MSE), yielding the following general update format shown in Eqs (2-24) and (2-25), where V is used as a general parameter for the parameterized critic output. The exact update path then depends strongly on the specific ACD used.

$$\begin{aligned} E_c(t) &= \frac{1}{2}\left[r_t + \gamma V(S_{t+1}) - V(S_t)\right]^2 \\ \phi_{t+1} &= \phi_t + \alpha \frac{\partial E_c(t)}{\partial \phi} = \phi_t + \alpha \frac{\partial E_c}{\partial V}\frac{\partial V}{\partial \phi} \end{aligned} \tag{2-24}$$

$$\begin{aligned} E_a(t) &= \frac{1}{2}\left[V(S_t) - V^*(S_t)\right]^2 \\ \theta_{t+1} &= \theta_t + \alpha \frac{\partial E_a(t)}{\partial \theta} = \theta_t + \alpha \frac{\partial E_a}{\partial V}\frac{\partial V}{\partial \theta} \end{aligned} \tag{2-25}$$

The simplest ACD variant is HDP. Here, the critic estimates the state-value function. This means the value $\frac{\partial V}{\partial \phi}$, which is required for the critic update, can be obtained from the critic itself through backpropagation. On the other hand, the value $\frac{\partial V}{\partial \theta}$ required for the actor update is not directly available, as the value function does not depend on the action taken. Therefore, HDP requires an environment transition model to close the backpropagation path for the actor update. In the action-dependant variant ADHDP the critic approximates the action-value function, from which a derivative with respect to the action is readily available. Therefore, ADHDP does not require any plant model.
In DHP, the output of the critic is not the value function itself but the derivative of the value function with respect to the state, which can directly be used to update the actor without requiring backpropagation. However, in DHP both the actor and the critic updates require a plant model. In the action-dependant variant ADDHP the critic calculates the derivatives of the action-value function with respect to both the state and action. Like in ADHDP, this relieves the actor update of plant model requirements.
The third category GDHP combines the ideas of HDP and DHP by having the critic estimate both the value function as well as its derivative directly.

Studies have shown that DHP reliably outperforms HDP [66, 67], and that this is attributed to the accuracy of the critic's estimate of the value function derivatives [68]. GDHP, while more complex in implementation and computation, offers a negligible performance increase with respect to DHP [65]. The action-dependant variants are generally easier to implement in return for lower performance [22].

2-3-3 Online Reinforcement Learning Control for Aerospace Systems

The studies mentioned above show that ACDs are capable of successfully controlling complex systems. However, in general they require at least some form of a priori knowledge of the controlled system in the form of a simulation model for an offline learning phase. More recently, the ADP framework was redesigned to use incremental control techniques. Inspired by INDI [69] and IBS [70], the learned plant model is replaced by a local, linear, time-varying

incremental approximation of the real system, estimated on-line by a recursive least squares estimator.

The first method, incremental Approximate Dynamic Programming (iADP) [71] estimates a quadratic cost function online which is then used in an optimal control scheme. Effectively, this is a critic-only method, as the actor is an explicit function of the critic, but it was nevertheless shown to be able to perform well in a nonlinear disturbance rejection control task. Two new actor-critic methods were also presented: Incremental Heuristic Dynamic Programming (IHDP) and Incremental Dual Heuristic Programming (IDHP) [72, 73]. Both learned an incremental approximation to a nonlinear aircraft model online. Both IHDP and IDHP showed faster convergence and better tracking performance than their non-incremental variants. Additionally, IDHP was shown to be able to control an unknown, unstable system before divergence occurred where DHP failed to do so. These results were shown to hold for more complex systems too when the IDHP framework was used to learn a near-optimal control policy for a high-fidelity, six-degree-of-freedom simulation of the Cessna 550 Citation II PH-LAB research aircraft [28, 27].

Table 2-2: Overview of the main variants of Adaptive Critic Designs

Variant	Critic		Plant model required	
	Input	Output	Actor	Critic
HDP	s	$V(s)$	yes	no
DHP	s	$\frac{\partial V}{\partial s}$	yes	yes
GDHP	s	$[V(s), \frac{\partial V}{\partial s}]$	yes	yes
ADHDP	$[s, a]$	$Q(s, a)$	no	no
ADDHP	$[s, a]$	$[\frac{\partial Q}{\partial s}, \frac{\partial Q}{\partial a}]$	no	yes
ADGDHP	$[s, a]$	$[Q(s, a), \frac{\partial Q}{\partial s}, \frac{\partial Q}{\partial a}]$	no	yes
IHDP	s	$V(s)$	no	no
IDHP	s	$\frac{\partial V}{\partial s}$	no	no

2-4 Reinforcement Learning for Helicopter Flight Control

In this section, three of the most influential research projects on reinforcement learning for helicopter flight control are discussed.

2-4-1 Autonomous Helicopter Control using Policy Search Methods

One of the first applications of reinforcement learning to helicopter flight was presented in [74]. In this research, policy search methods were used to autonomously hover a model helicopter. To do this, the longitudinal and lateral motions were decoupled, and two simple neural networks with five free parameters each were via Monte Carlo search on a deterministic simulation model. The authors then tested the controller on a Yamaha R-50 model helicopter, and achieved similar performance to an expert human pilot on a hover task in windy conditions.

2-4-2 Direct Neural Dynamic Programming

The second major application was presented in a series of papers in 2002 and 2003. These papers discussed three different applications of reinforcement learning to an industrial AH-64 Apache Helicopter model: stabilization [75], fault-tolerance [76], and reference tracking [20]. Based upon the earlier work by the same authors [68], the Direct Neural Dynamic Programming framework used is architecturally similar to ADHDP, which was presented in Section 2-3-2. According to the authors, it was one of the first systematic applications of the ADP framework to a complex, nonlinear MIMO system. Two ideas made this framework especially successful: the use of a pre-trained trim network, and a cascaded actor structure. This study demonstrates that ADHDP is capable of controlling a nonlinear, coupled, high-dimensional system under varying conditions and failure cases where conventional PID controllers failed. However, it should be noted that a large amount of offline training with an accurate simulation model was necessary in all cases, with more trials necessary for more aggressive maneuvers.

2-4-3 Stanford Autonomous Helicopter Project

From 2004 to 2010, the Stanford Autonomous Helicopter Project focused on aerobatic flight for autonomous model helicopters. Compared to full-scale helicopters, model helicopters have a higher thrust-to-weight ratio, higher body stiffness, and faster control responses. While each of these focuses on different maneuvers, the general gist of each is as follows. First, a dynamic model of the helicopter was obtained by performing system identification on a set of flight data obtained by recording an expert pilot. A controller for a specific maneuver was then created by using imitation learning on another set of recorded data, this time of the expert pilot performing the maneuver, by treating the recorded data as off-policy data points. Together with the learned dynamic model, this allowed for Monte Carlo methods to update the controller. In this way, the authors were able to successfully have a model helicopter perform autonomous hover [6], inverted hover [77], autorotation [78], and several extremely advanced maneuvers [79, 80]. One example of the performance of these methods is shown in Figure 2-8, which compares the performance of a controller for autonomous hover (in blue) to that of an expert human pilot (red). It can be seen that the controller performance is significantly more stable than that of the pilot, with maximum x-velocities an order of magnitude in difference.

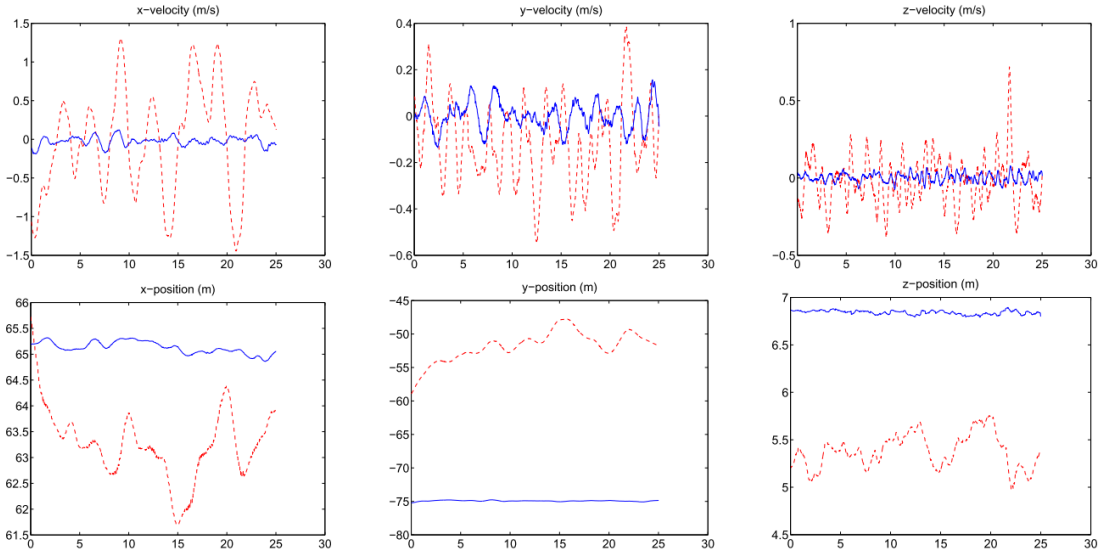


Figure 2-8: Hover performance of the Stanford Heli controller (blue) compared to an expert human pilot (red). From [6]

2-5 Conclusion

The goal of this literature survey was to get acquainted with the state-of-the-art in reinforcement learning for helicopter flight control. First, the basics of helicopter flight control itself were discussed, and the mechanics of autorotation were explored in more detail. Next, the most important concepts in reinforcement learning were described: value functions, policies, and the Markov Decision Process. Actor-critic methods learn both a value function and a policy directly, offsetting the drawbacks from either of these methods in return for being more complicated to implement. These methods are studied in two different perspectives on the reinforcement learning problem: Deep Reinforcement Learning, which uses a computer science perspective, and Approximate Dynamic Programming, which reasons from a control theory framework. In Deep Reinforcement Learning, the focus is on complex end-to-end methods, such as learning locomotion directly from raw pixel inputs. Four ideas have shown to be especially prominent: multi-step advantage estimates increase the speed of convergence, the use of target and double networks to provide learning stability, trust-region methods prevent catastrophic policy performance collapse, and experience replay increases the sample efficiency of off-policy methods by allowing one transition to be used multiple times. However, all Deep Reinforcement Learning methods require many off-line trials before a good policy is learned. Out of the three main variants of Adaptive Critic Designs, DHP achieves the best balance between complexity and performance. Recently, incremental methods popularized in INDI and IBS were combined with DHP, yielding the novel IDHP algorithm which does not require an offline learning phase.

Only a few applications of reinforcement learning to rotorcraft control exist so far. Of the three most prominent studies, only one focused on full-scale helicopters. Significantly more work has been done for model helicopters: while the platforms are similar, model helicopters have a significantly higher thrust-to-weight ratio and faster control responses. Furthermore,

no work on online learning for helicopters has been performed yet. Although IDHP has not been applied to rotorcraft control before, both ADP and incremental methods themselves have succesfully been used for this purpose. Therefore, the novel IDHP algorithm is seen as the best baseline for true online, fault-tolerant control for helicopters. To improve the learning stability and convergence speed of the algorithm, the novel ideas from DRL presented here are seen as promising additions to IDHP.

Chapter 3

Preliminary Analysis

In the literature study, Incremental Dual Heuristic Programming (IDHP) [73] was identified as the most promising approach for the online flight control of a full-scale helicopter system. As an advanced ACD from the ADP perspective of reinforcement learning, the implementation of this algorithm is not trivial. In this chapter, a preliminary analysis is performed by applying a simpler ACD, heuristic dynamic programming (HDP) [65], to a simplified helicopter control task. This preliminary analysis has three main goals: gain experience with the implementation of ACDs, test their feasibility for a control task, and compare the framework to more straightforward control methods.

The structure of this chapter is as follows. In Section 3-1, the flight task to be performed is stated. Next, in Section 3-2 two simple controllers are established as a baseline for performance. In Section 3-3, the HDP agent is described in more detail. The results of the experiments are then described in Section 3-4. Finally, conclusions drawn from these results are given in Section 3-5.

3-1 Problem Formulation

The controlled environment is a simplified helicopter model based on [7]. The model approximates the longitudinal transition dynamics from hover to forward flight, taking into account some basic flapping dynamics, of different types of helicopters. It consists of two states the pitch rate q and the blade flapping angle a_1, and the model input is the swashplate angle θ_s. A free body diagram of the situation is given in Figure 3-1, and the dynamics are according to Eq. (3-1).

$$\begin{aligned} \dot{q} &= -\frac{\left(mgh + \frac{3}{2}K_\beta\right)}{I_y}\left(\theta_s - a_1\right) \\ \dot{a_1} &= -\frac{1}{\tau}\left(a_1 + \frac{16q}{\gamma\Omega}\right) \end{aligned} \tag{3-1}$$

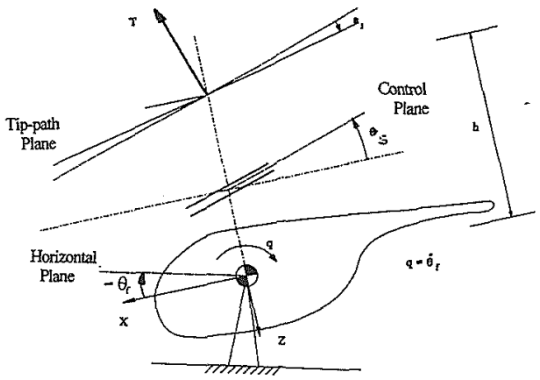


Figure 3-1: Free body diagram of the simplified helicopter pitch model, from [7]

Two parameters are varied in this experiment. First, the hinge spring stiffness K_β can be adjusted to represent different rotor heads: zero stiffness for a teetering rotor, a relatively large stiffness for hingeless systems, and an intermediate value for the articulated rotor with hinge-offset. These systems differ in their response to a step input signal: as the stiffness grows, the system becomes more responsive. A teetering hub with $K_\beta = 0$ has a typical response for acceleration control, while a hingeless system with $K_\beta = 46000$Nm is more typical for velocity control. Second, the flapping dynamics time constant τ affects the tilting motion of the rotor disk, and increasing this value increases the overshoot and oscillatory behaviour when applying a step input to the system. However, this is only significant for high values of K_β. The other parameters in Eq. (3-1) are fixed, and the values used for the experiment (directly adapted from [7]) are given in table 3-1.

Table 3-1: The aircraft parameters for the dynamic system in Eq. (3-1)

Parameter	Description	Value
m	Aircraft mass	2200 [kg]
g	Acceleration due to gravity	9.81 [m/s^2]
h	Rotorshaft height	1 [m]
Ω	Rotor angular velocity	27.32 [rad/s]
γ	Lock number	6 [-]
I_y	Moment of inertia	10625 [kg m^2]
τ	Flapping dynamics time constant	variable
K_β	Rotor hinge stiffness	variable

For this experiment, a teetering rotor hub was chosen, with K_β=0 and τ=0.25. This corresponds to aceleration control, which requires more anticipation from a pilot than velocity control does. As a comparing case, the same hyperparameters are also used to train a controller for the hingeless case. The goal of the agent will be a pitch rate tracking task with an episode length of 120 seconds and the following properties:

$$q_{ref} = q_{max} \sin\left(\frac{2\pi t}{T_{ref}}\right) \tag{3-2}$$

After every timestep, the agent receives a negative reward, which is defined as the quadratic difference of the goal state and current state, according to Eq. (3-3).

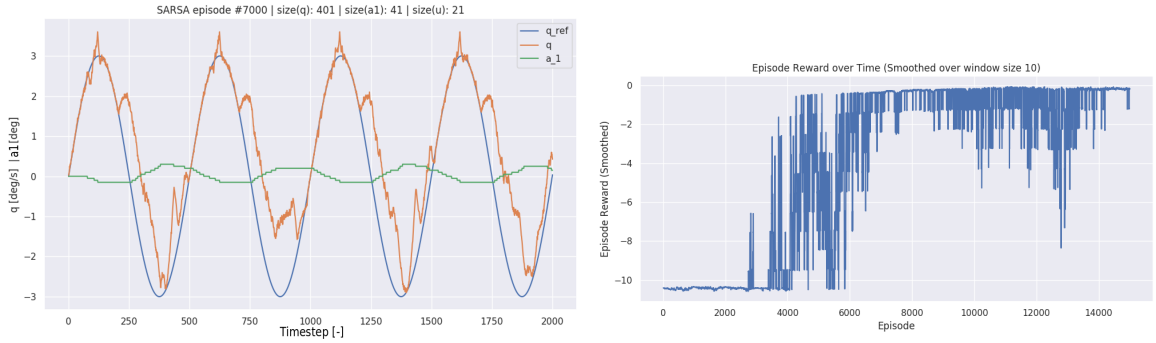
$$r_t = -\frac{1}{2}\left(\frac{q - q_{ref}}{q_{max}}\right)^2 \tag{3-3}$$

3-2 Baseline controller

As a first experiment, a SARSA(0) controller was implemented on a discretized version of the environment, following a pitch rate task with an amplitude $q_{max} = 3$ deg and a period of 20 seconds. The state available to the controller was augmented with the pitch rate error $q_e = (q - q_{ref})$, which provides more useful information than only the reference pitch rate. The coarsest discretization which still represented the real system was found to have steps of 0.01 degree for the pitch rate, 0.1 degree for the flapping angle, and 0.1 degree for the input. Even with a relatively coarse discretization, the Q-table used to store action values already had 138 million entries. This meant that for a large amount of time, almost every step ended up in a state-action pair that had never been visited before. Furthermore, the table-based approach meant no generalization between similar states was possible. All in all, convergence turned out to be very slow. Figure 3-2a shows the performance of this system after 7000 training episodes. It can be seen that the agent follows the reference signal closely at first, when the system passes states it has seen many times before. However, several unstable regions are still present. The first to come to light is the top of each peak in the reference signal, which the agent has not yet learned to track smoothly. The second unstable region happens around timestep 200, where a input into the wrong direction causes te system to lose track of the reference, before regaining strong tracking performance at the beginning of the next cycle. This is proof of the poor generalization of table-based methods: in the eyes of this system, every state is unique and there are no trends. Figure 3-2b gives more insight into the convergence of this agent by showing the smoothed reward per episode over the entire training run. It can be seen that the peak performance of the system happens around episode 8000, and that average performance slowly gets worse again after that. However, from episode 14000 and on performance once again reaches peak level, averaging out around zero reward for hundreds of episodes in a row. All in all, this baseline experiment showcases the major reasons why tabular RL is not suited for online control problems. It becomes clear that even for such an example problem, a more powerful system is necessary for smooth control performance.

3-3 Heuristic Dynamic Programming

Next, the HDP framework is used because of its power relative to the low complexity. To further reduce the complexity an action independent and model based solution is used. This means the critic is only a function of the state and the model network is replaced with the model used by the environment.



(a) (b)

Figure 3-2: (a) Performance of the Sarsa(0) controller after 7000 training episodes. (b) Sliding-window average reward per episode over the entire length of the training run

3-3-1 Network structure

The actor and critic are modelled using artificial neural networks with simple layouts as shown in Figure 3-3: both take the same augmented state as in Section 3-2 as input, fully connected six hidden units with hyperbolic tangent activation functions, followed by a single output unit. For the critic, this output is linear (no activation function), whilst for the actor the output has a hyperbolic tangent activation scaled with an actuator limits of ± 15 degrees. Both the actor and critic are updated by stochastic gradient descent, as described in Section 2-2-2. Here, the actor and critic are parameterized with weight vectors w_a and w_c, respectively.

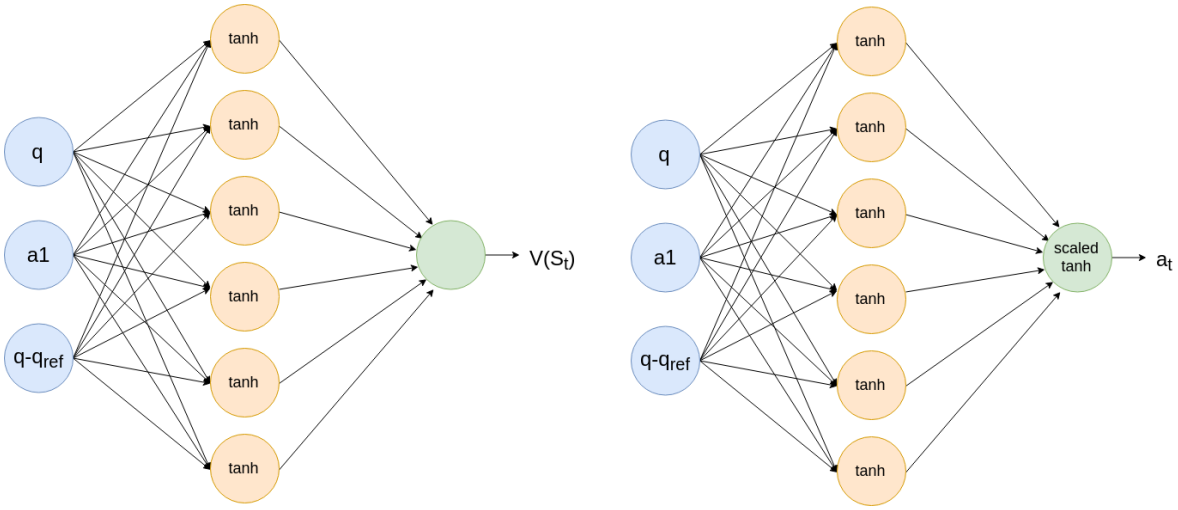


Figure 3-3: Neural network layouts for the actor and critic of the HDP agent

3-3-2 Update rules

A schematic layout of the flow of (gradient) information through the actor-critic network is given in Figure 3-4. The goal of the critic is to minimize the mean squared temporal difference

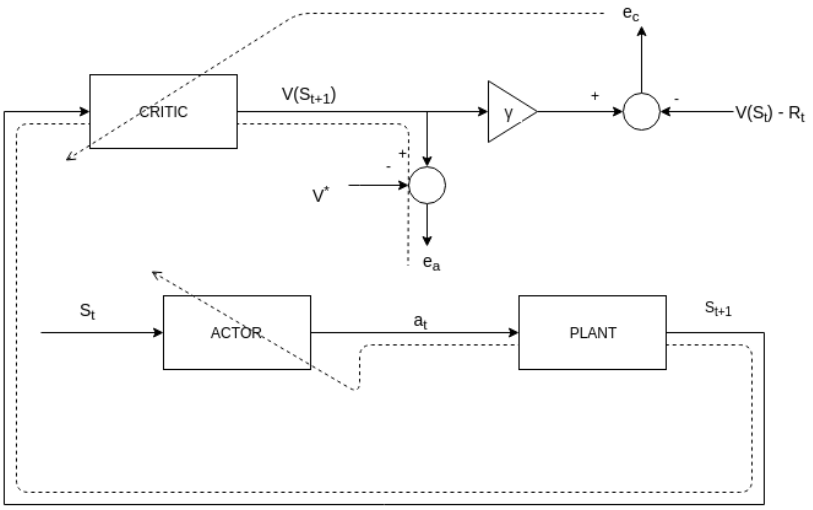


Figure 3-4: Schematic of the backpropagation of actor and critic errors through the network

error $\mathrm{E}_c(t)$:

$$\begin{aligned} e_c(t) &= r_t + \gamma V(S_{t+1}) - V(S_t) \\ E_c(t) &= \frac{1}{2} e_c^2(t) \end{aligned} \tag{3-4}$$

The update path is then:

$$\frac{\partial E_c}{\partial w_c} = \frac{\partial E_c}{\partial e_c} \frac{\partial e_c}{\partial V(S_t|w_c)} \frac{\partial V(S_t|w_c)}{\partial w_c} = -e_c \nabla_{w_c} V(S_t|w_c) \tag{3-5}$$

The goal of the actor is to find the policy which minimizes the difference between the value function and a goal value, which in our case is zero:

$$\begin{aligned} e_a(t) &= V(S_t|w_c) - V^*(S_t) = V(S_t|w_c) \\ E_a(t) &= \frac{1}{2} e_a^2(t) \end{aligned} \tag{3-6}$$

In this case, the actor loss does not depend directly on the actor weights. Therefore, the backpropagation path is taken through the critic and environment model by means of the chain rule as follows:

$$\frac{\partial E_a}{\partial w_a} = \frac{\partial E_a}{\partial e_a} \frac{\partial e_a}{\partial V(S_t)} \frac{\partial V(S_t)}{\partial S_t} \frac{\partial S_t}{\partial A_t} \frac{\partial A_t}{\partial w_a} \tag{3-7}$$

The first two partial derivatives are obtained by differentiation of the loss function of the actor, conveniently resulting in the critic output of the current state. The term $\frac{\partial V(S_t)}{\partial S_t}$ is obtained by backpropagation, differentiating through the critic network with respect to the input, and the term $\frac{\partial A_t}{\partial w_a}$ by differentiating through the actor network with respect to the weights. The final term $\frac{\partial S_t}{\partial A_t}$ is a direct function of the environment and is normally unknown.

However, because we have a known environment model this can be obtained directly from the system dynamics: differentiating Eq. (3-1) yields the expression in Eq. (3-8).

$$\frac{\partial S_t}{\partial A_t} = [-k, 0, -k]^T, \quad where$$
$$k = \frac{mgh + \frac{3}{2}K_\beta}{I_y} \tag{3-8}$$

3-3-3 Algorithm

The hyperparameters[1] used in the first experiment are given in table 3-2.

Table 3-2: Hyperparameters used for the HDP agent in the preliminary experiment

Hyperparameter	Description	Value
γ	Discount factor	0.95
α	Learning rate	0.2
σ_w	Initial variance of neural network weights	0.1
$N_{updates}$	Update cycles per timestep	5

The algorithm starts with the set-up of the neural networks and environment. The training loop is then started, which calculates the actor action for the current environment state, after which a step in the environment is taken. Using the next observation and reward, the TD target is calculated with the current critic. The actor and critic are then repeatedly updated each time step using the update rules presented above, alternating between a critic and actor update. To present the neural networks with a stable target, the TD target is only calculated once per timestep, so that the only changing variable each iteration within a timestep is the value estimate of the current state.

3-4 Results and Discussion

The first experiment consisted of 100 episodes for the agent with the teetering rotor ($K_\beta = 0$) to evaluate the base performance. Next, a sensitivity analysis of the two most important hyperparameters, the learning rate and the weight initialization standard deviation, was performed. Finally, the performance of the teetering rotor agent was compared with the hingeless case.

3-4-1 Performance

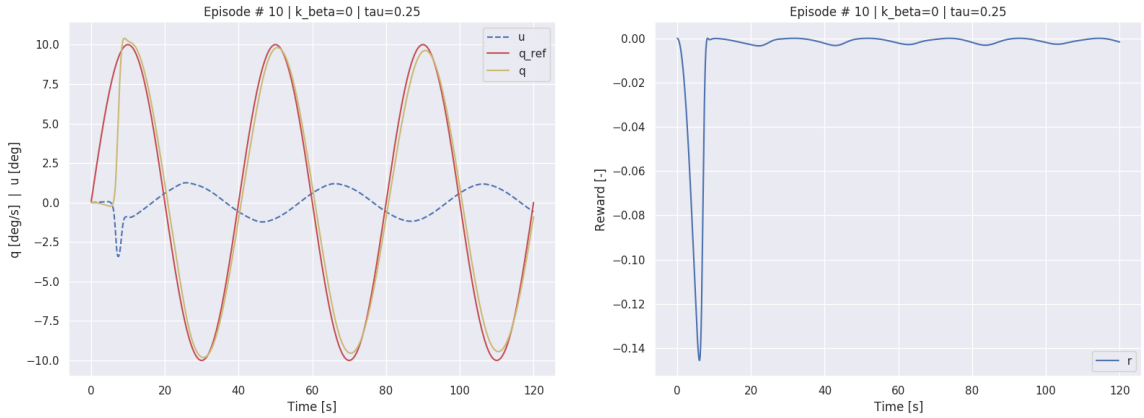
All 100 episodes resulted in acceptable performance, with none diverging.

Figures 3-5a, 3-5b, and 3-6 show the tracking performance, reward per timestep, and neural network weights of the best episode out of the 100. It can be seen that the agent converges

[1] Parameters whose value is set before learning starts

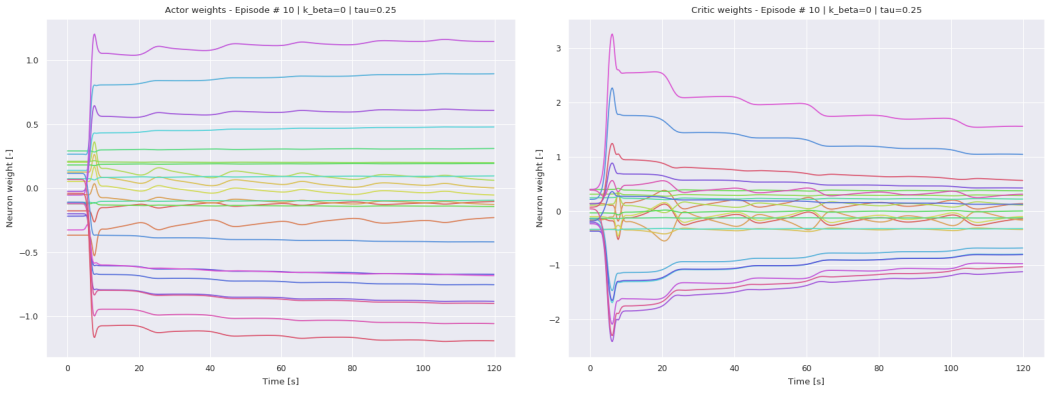
nearly perfectly within ten seconds: this can be seen both from the tracking performance, as well as the stability of the actor weights. The reward per timestep also oscillates slowly around a near-zero value. The critic weights are still converging after the ten second mark, but they do so symmetrically. This is to be expected as not all states actually contribute to the reward, only the tracking error does.

Similarly, Figures 3-7a, 3-7b and 3-8 show this for the worst performing run out of the series. It becomes clear that the agent manages to follow the general pattern of the reference signal, but it does not achieve good tracking, especially in the peaks of the reference signal. This behavior is also visible in the actor weights, which do not approach a steady value and even seem to diverge. Nevertheless, it can be seen that even the worst performing run does not completely diverge, but seems to get stuck in a local minimum and is unable to improve its performance.



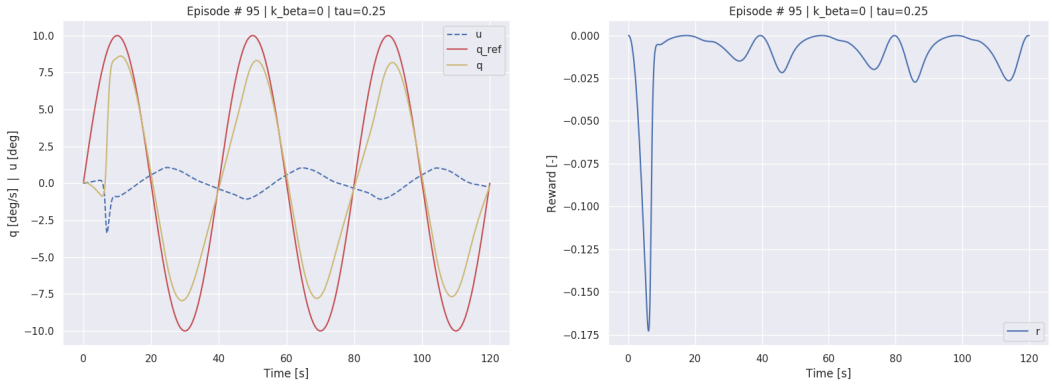
(a) Tracking performance **(b)** Reward per timestep

Figure 3-5: Tracking performance and reward per timestep of the best agent in the preliminary experiment



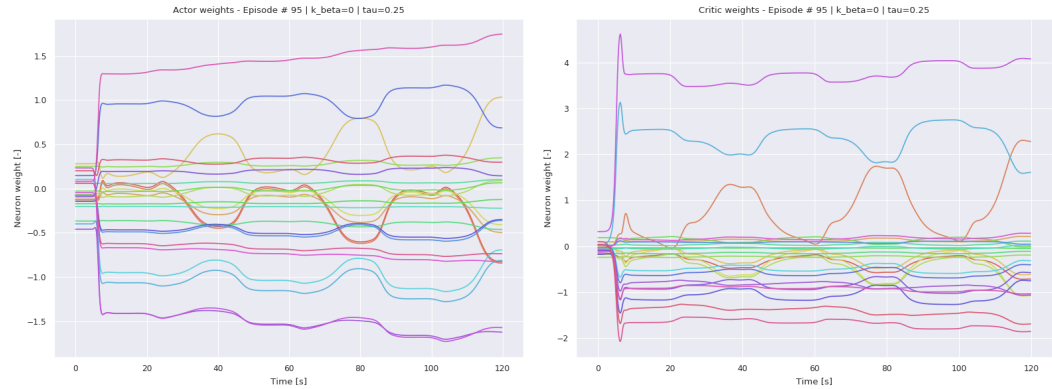
(a) Actor weights **(b)** Critic weights

Figure 3-6: Neural network weights of the actor and critic of the best performing HDP agent



(a) Tracking performance **(b)** Reward per timestep

Figure 3-7: Tracking performance and reward per timestep of the worst agent in the preliminary experiment



(a) Actor weights **(b)** Critic weights

Figure 3-8: Neural network weights of the actor and critic of the worst performing HDP agent

3-4-2 Sensitivity analysis of the learning parameters

Two hyperparameters are especially important to learning performance: the learning rate α, and the standard deviation of the neural network weights initialization σ_w. The learning rate governs the step size that is taken in gradient descent: a large value allows for quick convergence initially, but can make it impossible to find the exact optimum; a value that is too low will generally result in better performance, though this will take longer, and comes with the risk of getting stuck in a local optimum. The effect of σ_w is similar: aggresively initialized weights lead to more rapid convergence because of larger initial gradients, but can also cause instability.

To find out the exact effect of the hyperparameters on the convergence rate and speed, 196 training runs of 100 episodes each were performed with different combinations of α and σ_w. The average result and confidence bound of each run is given in figure 3-9. It can be seen that both parameters strongly influence the performance. If both are low, then the network is unable to learn, but performance is also degraded if both have a high value. The best average performance is achieved at intermediate values: $\alpha = 0.6, \sigma_w = 0.2$.

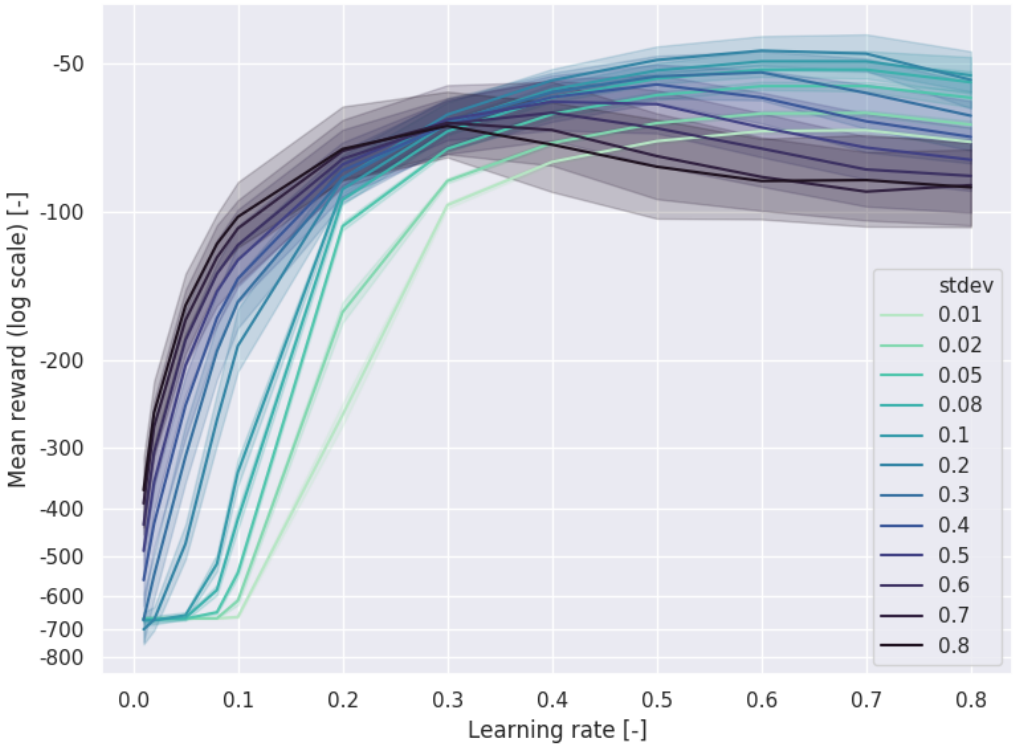


Figure 3-9: Sensitivity analysis of the learning rate and weight initialization standard deviation. Shaded areas are 95% confidence bounds

Finally, the best hyperparameters found in Figure 3-9 were used for a comparison run between helicopters with different rotor hubs: one with the original values $K_\beta = 0, \tau = 0.25$, and one representing a hingeless hub: $K_\beta = 46000, \tau = 0.25$ Both agents were trained for 100 episodes using the hyperparameters given above: the spread of these results are shown in the left

two masses of Figure 3-10. It can be seen that the spread in performance for the hingeless system is much larger than that for the teetering hub, even though the hingeless system is supposed to be easier to control. It can be concluded that the optimal hyperparameters for the teetering hub are too aggressive for the simpler hingeless rotor. To test this hypothesis, a final experiment was performed with the hingeless hub using less aggressive parameters: this was also tested with a run of 100 episodes, this time with $\alpha = 0.4, \sigma_w = 0.2$. This significantly increased performance, and brought both the average and maximum reward higher than the best original system, as shown in the final mass of Figure 3-10. The performance of the best agent of the hingeless system with the original and modified hyperparameters is compared in Figure 3-11, where it can be seen that the modified variant yields much more stable learning of the critic.

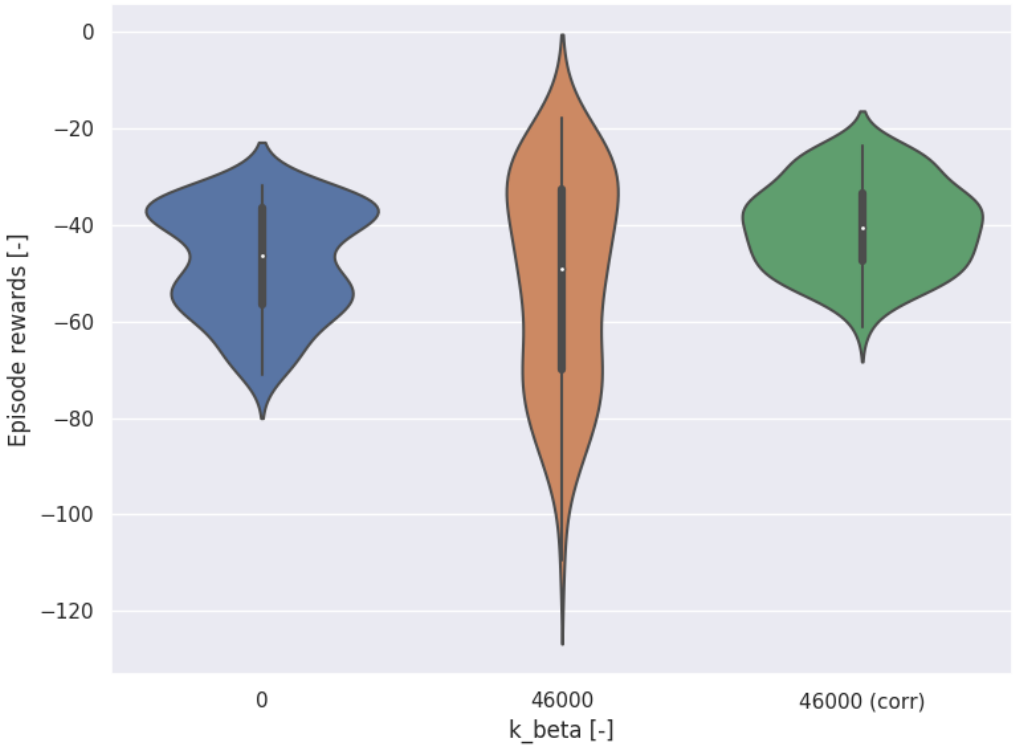
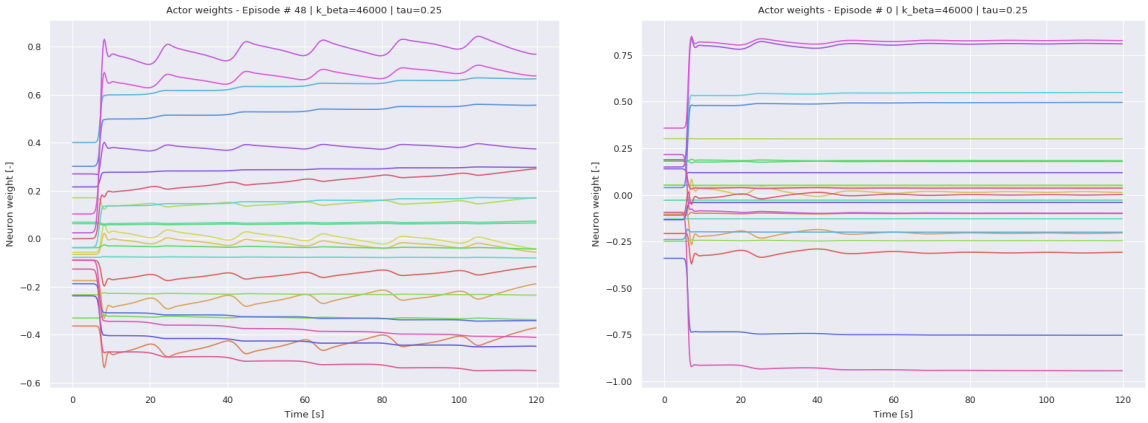


Figure 3-10: Performance density plots of different agents



(a) Original **(b)** Modified

Figure 3-11: Comparison of the best agents using a hingeless hub, using the original and modified learning hyperparameters

3-5 Conclusion

The online reinforcement learning controller based on the HDP algorithm is able to follow a pitch rate reference signal for a two state short period 3e helicopter model starting from random initialization. The tracking performance is significantly better than one based on discrete, tabular RL. The system was shown to perform adequately in all 100 runs. The best episodes converged almost perfectly within ten seconds, while the worst performing episodes showed a maximum pitch rate errors of 2.5deg/s. A sensitivity analysis of the learning hyperparameters showed that these strongly influence the convergence rate and speed: the best performance was observed for $\alpha = 0.6, \sigma_w = 0.2$. These values did not translate to a hingeless rotor hub, which has more direct control: here these values turned out to be too aggressive.

Part III

Additional results

Chapter 4

Scaling up in online adaptive flight control

In the preliminary research, a one-degree-of-freedom (1DOF) system was controlled by a model-based HDP controller, while the final experiments consisted of a nonlinear 6DOF model controlled by a complex combination of IDHP and PID controllers. This transition was not discrete, as in between those two, additional research into the scaling-up of these kinds of control systems was performed. In this chapter, the work in scaling up from work in the preliminary thesis towards the work in the paper is described. Scaling up in the context of this research takes place in two dimensions: controller complexity and model complexity. In order of increasing complexity, the controllers used were model-based heuristic dynamic programming (MBHDP), incremental HDP (IHDP), and incremental dual heuristic programming (IDHP). Likewise, on the modeling side, three different models used included 1DOF, 3DOF, and 6DOF. The path followed throughout the different combinations of helicopter model and controller complexity is shown in red in Fig. 4-1.

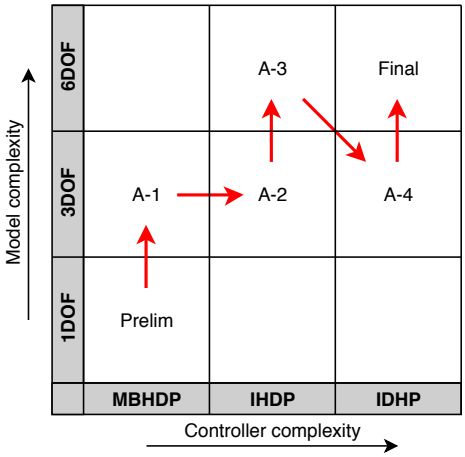


Figure 4-1: The steps taken in scaling up from the preliminary research to the final experiment

4-1 Three-degree-of-freedom helicopter model

As a first step in scaling up, the 1DOF model used in the preliminary research was abandoned in favor of longitudinal 3DOF model based on [2]. The model used is a nonlinear, three-degree-of-freedom, quasi-dynamic inflow approximation to the longitudinal motions of a generic full-scale helicopter. It has two inputs and seven states, which are shown in Eq. (4-1).

$$s = \begin{bmatrix} x & z & u & w & \theta & q & \lambda_i \end{bmatrix} \qquad a = \begin{bmatrix} \theta_0 & \theta_c \end{bmatrix} \tag{4-1}$$

With the addition of translational movement, new states arise: the earth-coordinate positions $[x, z]$, the body-frame velocities $[u, w]$, and the non-dimensional induced velocity λ_i. Furthermore, there are now two control inputs: the collective θ_0 and cyclic θ_c. With this model, the effect of the interaction between two coupled control channels on different control architectures could be studied.

4-2 Model-based heuristic dynamic programming - three degrees of freedom

In this section, the different experiments concerning model-based HDP applied to the 3DOF helicopter model are described. This corresponds to phase A-1 in Fig. 4-1.

4-2-1 Straightforward application of MDHDP

The first attempt to control the 3DOF model was with the same architecture as in the 1DOF case, applied to this new model with minimal changes. One required change is the move from one to two actions, significantly complicating the backpropagation procedure through the actor network, which now has weight matrices of sizes (7 x nhidden) and (nhidden x 2). The actor update requires the derivative $\frac{\partial a}{\partial w_a}$, but since a is now a vector, it is hard to determine the required shape of $\frac{\partial a}{\partial w_a}$. It was decided to treat the two update paths as separate and sum their respective products as follows:

$$\frac{\partial E_a}{\partial w_{a^1}} = \frac{\partial E_a}{\partial a_1}\frac{\partial a_1}{\partial w_{a^1}} + \frac{\partial E_a}{\partial a_2}\frac{\partial a_2}{\partial w_{a^1}} \tag{4-2}$$

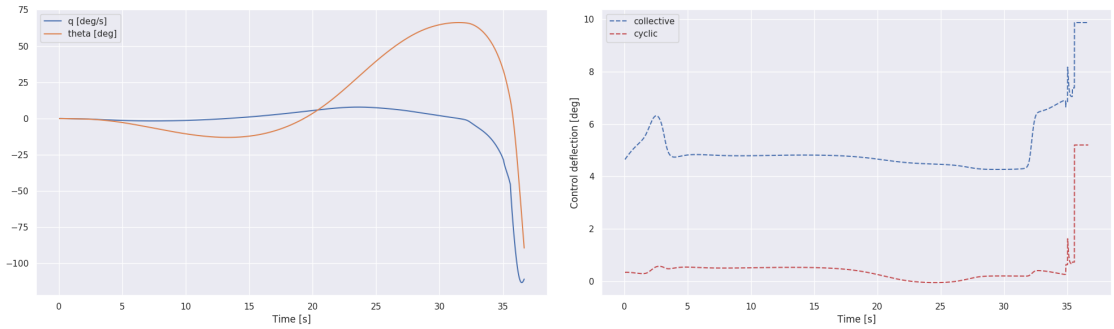
Furthermore, a more generic reward function was introduced that could easily be tweaked for different goals. This is shown in Eq. (4-3). Here, P is the state selection matrix that is defined such that the product Ps_{t+1} contains the same states as s_t^{ref}, and Q is a weight matrix used to define the relative importance of each state in the reward.

$$r_{t+1} = -\left(Ps_{t+1} - s_t^{ref}\right)^T Q \left(Ps_{t+1} - s_t^{ref}\right) \tag{4-3}$$

The first test was to see if this architecture could immediately be used for a realistic goal: keeping the pitch angle constant. Therefore, the following values were to define the reward function:

$$\begin{aligned} s_t^{ref} &= \theta^{ref} = 0 \\ P &= \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \\ Q &= \begin{bmatrix} 1 \end{bmatrix} \end{aligned} \tag{4-4}$$

This control architecture was not found to be successful for any random seed and a variety of hyperparameters. A sample run is shown in Fig. 4-2. It can be seen that both the collective and cyclic react similarly, although they are offset by their respective trim points. A consequence of using one integrated controller for two control channels is that it can be hard to discern which control action lead to a change in reward. This is especially obvious from $t = 30$s onwards: the large deviations of the pitch angle from its reference value cause large changes in both control channels, further destabilizing the situation. From this, it was concluded that a straight-forward application of the previously-used control architecture to a higher-order model was not trivial, and that further simplifications should first be made before additional complexity was added.

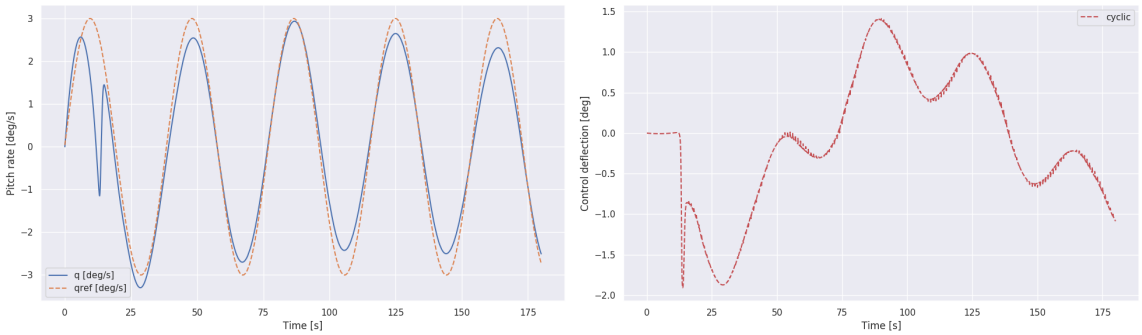


(a) Controlled states **(b)** Actions

Figure 4-2: MDHDP - 3DOF - learning cyclic rate control with a single agent

4-2-2 Learning pitch rate control with fixed-value collective

The second attempt simplified the control architecture to be more in line with what worked in the 1DOF case. Control was limited to the cyclic, which followed a sinusoidal pitch rate reference signal. The collective was kept fixed at the trim value, in order not to interfere with the learning process of the cyclic. The results of this task are shown in Fig. 4-3. It can be seen that these results are very similar to those obtained in the 1DOF case, falling somewhere in between the performance of the best and worst agents as presented in Figs. 3-5 and 3-7, respectively. The agent is able to track the reference signal with 0.5°/s accuracy after approximately 15 seconds. These results are not ideal, but do show that it is possible for HDP to learn to control the pitch rate of the 3DOF model in a moderate time span if the influence of coupling is ignored.

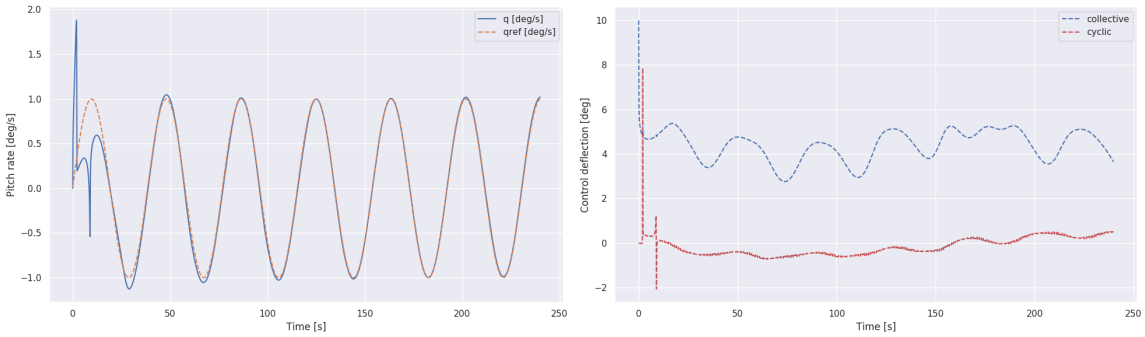


(a) Tracking performance and rewards **(b)** Control signals

Figure 4-3: MDHDP - 3DOF - learning rate control cyclic with fixed collective

4-2-3 Adding PID control for the collective

In the previous subsection, the collective dynamics were almost completely ignored. The next step was therefore to add a separate controller for the collective, and find out how the coupling between the two control channels influences the learning process. The results are shown in Fig. 4-4. It can be seen that the addition of a controller for the collective channel positively influences the learning of the cyclic. Although the time it takes for the reference to be followed approximately remains constant at 20 seconds, the final performance is significantly better than the previous effort. It is likely that the influence of the collective PID exposes the cyclic agent to a wider range of state transitions. As the actions of the collective are effectively hidden to the RL agent, they act as a source of temporally correlated noise, which has been shown in Deep RL applications to positively influence learning for deterministic RL algorithms [46].

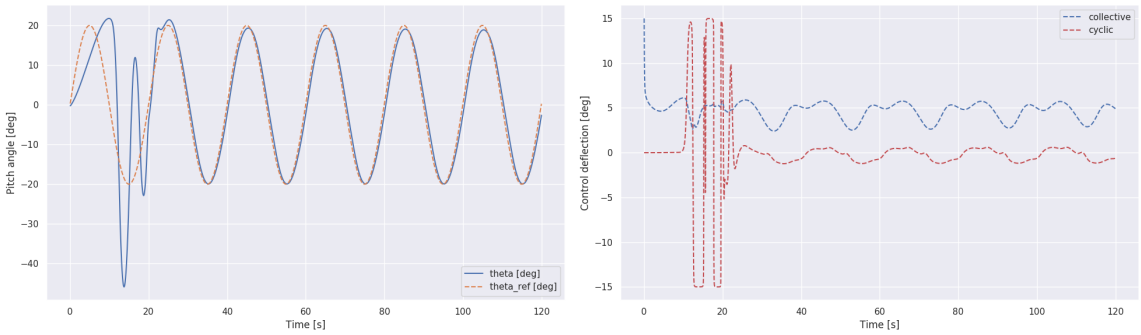


(a) Tracking performance and rewards **(b)** Control signals

Figure 4-4: MDHDP - 3DOF - learning rate control cyclic with collective PID

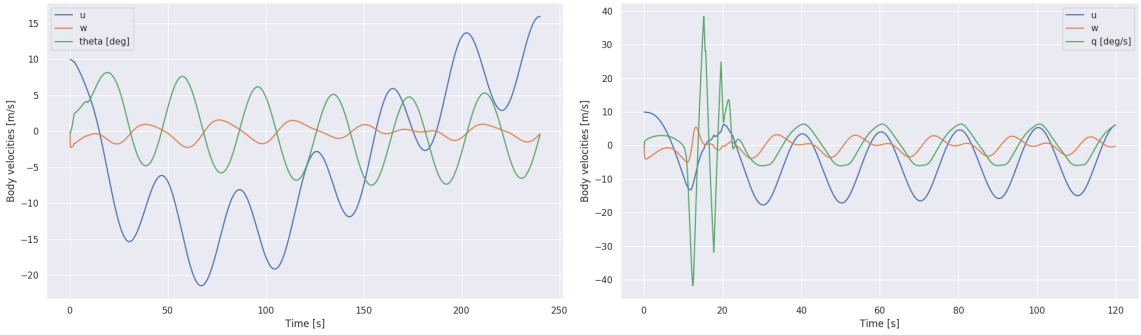
After tuning the controller for pitch rate control, the same setup was used for direct pitch angle control. These results are shown in Fig. 4-5. It should be noted that the period of the reference signal is half that of the previous tasks, and that it takes the agent the same amount of time to achieve good tracking performance. However, the final performance is not as good as the rate tracking task: a slight phase delay can be observed, as well as an amplitude error

of approximately 1°. Comparing Fig 4-4b to 4-5b reveals two major differences. First, direct attitude control takes longer to start learning than rate control does: it takes approximately 8 seconds before the first input on the cyclic is given, compared to 5 seconds for the rate controller. Second, there is more oscillation in the learning phase of the attitude controller than with the rate controller. The most probable cause for this is the smaller direct dynamic relation between cyclic input and pitch angle. When an input is applied to the cyclic, the pitch rate immediately increases, but its effect on the pitch angle only becomes apparent one timestep later when the pitch rate is integrated. The HDP algorithm, which relies on the first order derivative of the states with respect to the actions, seems to find it hard to capture these kinds of relations.



(a) Tracking performance and rewards **(b)** Control signals

Figure 4-5: MDHDP - 3DOF - learning attitude control cyclic with collective PID



(a) Rate control **(b)** Attitude control

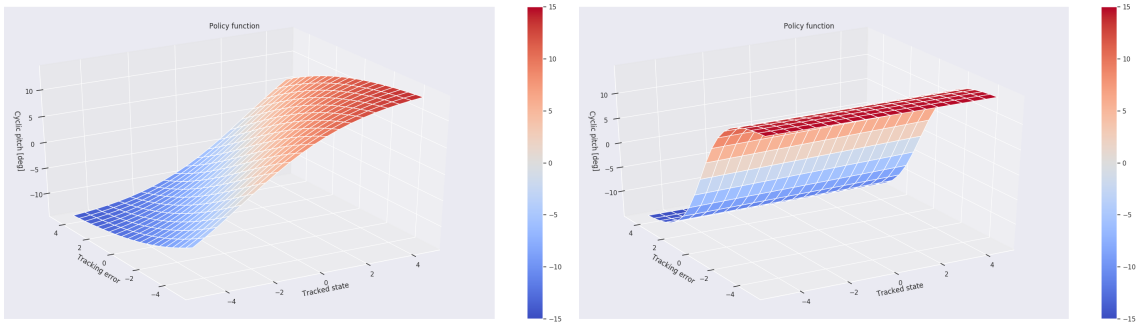
Figure 4-6: MDHDP - 3DOF - Comparing the non-tracked states of the rate and attitude controllers

To complete the comparison of rate control versus attitude control, the remaining states of both experiments are shown in Fig. 4-6. Although the reference signal in both cases is a constant sine wave, significantly more variance over time is observed in the case of rate control. This could lead to unstable training even in the case of perfect tracking performance when the effect of un-tracked states spirals out of control, as can be seen in the body velocity u in Fig 4-5a. In contrast, after a slightly slower and more chaotic start, the attitude controller remains within a well-defined flight envelope for the remainder of this training. Furthermore, this setup is simpler, as no additional controllers are needed to translate the required pitch

angles to pitch rates, and training is more predictable. From this, it was concluded that direct attitude control is in principle more desirable than rate control if the time to convergence is similar.

4-2-4 Countering overfitting with more complex reference signals

In the 3DOF case, the input space is of low enough dimensionality that the resulting policy of a training episode can be visualized in three dimensions. As an example, the policy resulting from the training episode shown in Fig. 4-5 is shown in Fig. 4-7a for different values of θ and $\left[\theta - \theta^{ref}\right]$. It can be seen that there is a strong dependance on the actual pitch angle where a weak dependance would be expected. This is an indication that the controller is overfitting to the current training scenario. To test this hypothesis, a more complex reference signal based on a sum of sinusoids was created. These kinds of compound sinusoids are often used in manual control experiments because they are deterministic but hard to predict. The results of a training episode following this signal is shown in Fig. 4-8, and the resulting policy of this training episode is shown in Fig. 4-7b. It can be seen that, although the convergence takes a similar time, the final tracking performance has significantly improved with respect to the previous scenario. Furthermore, a certain dependance on the actual pitch angle can still be seen, but it is significantly smaller than the first case. A certain degree of dependance is to be expected, because the attitude of the helicopter affects its dynamic response. It was concluded that a more complex reference signal can improve training performance, by reducing the effect of overfitting to the training scenario.



(a) (b)

Figure 4-7: Policies resulting from different reference signals, showing a large degree on overfitting on the left

4-2-5 Adding reinforcement learning control for the collective

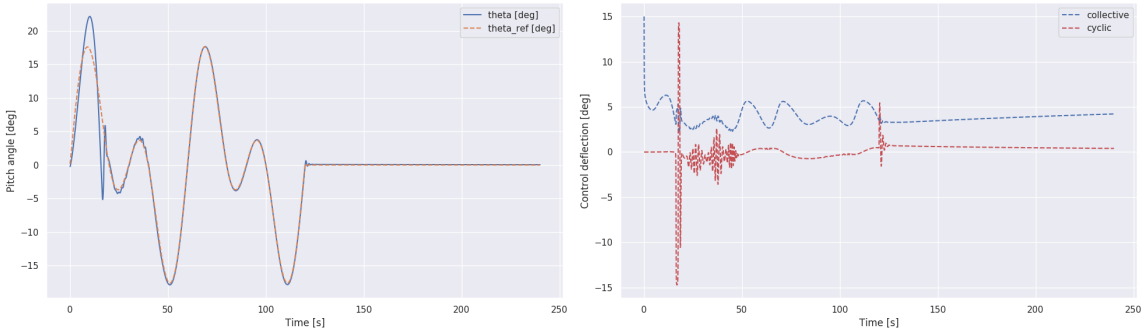
The next step about the learning process of the cyclic, the focus was shifted to training a RL controller for the collective. The first approach was to take the pre-trained cyclic actor and have it follow the same reference signal as used in Fig. 4-8, and replace the collective PID by a RL agent. The tracking task for the collective agent was to keep constant altitude in face of large variations of the pitch angle. Since altitude variations (in meters) encountered in any sample manoeuvre are generally of higher magnitude than angle deviations (in radians), a

scaling factor of 0.1 was added to the reward function to keep the reward in the same order of magnitude as encountered in the cyclic experiments. This yielded the following parameters for the reward function:

$$\begin{aligned} s_t^{ref} &= h^{ref} = 0 \\ P &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ Q &= \begin{bmatrix} 0.1 \end{bmatrix} \end{aligned} \tag{4-5}$$

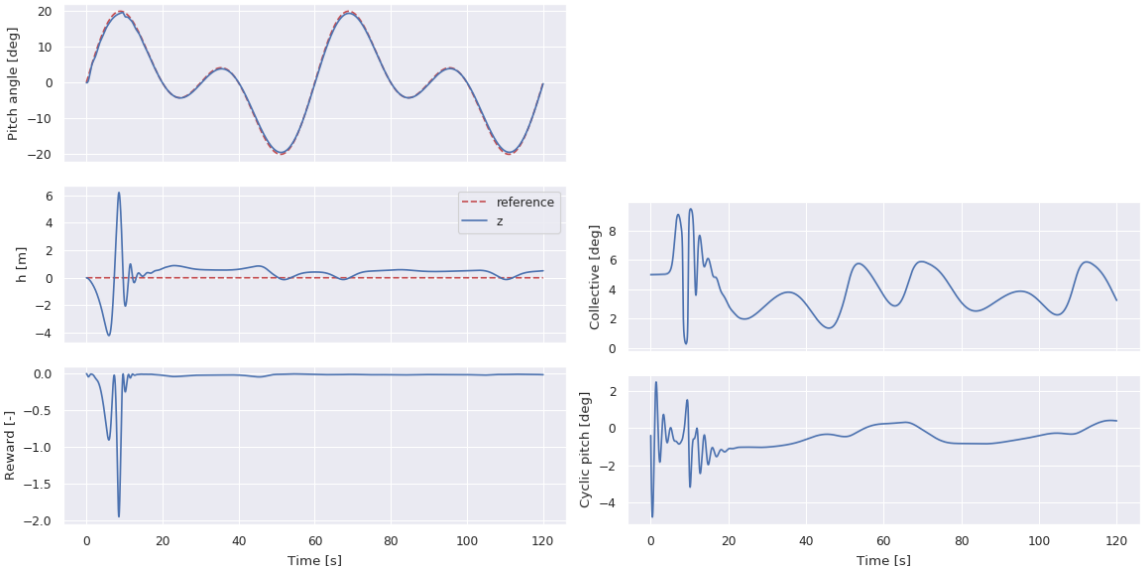
The results of the first experiment are shown in Fig. 4-9. It can be seen that the collective controller, like the cyclic, converges in approximately 20 seconds. During pitch angle variations of $\pm 20°$, the trained controller manages to keep altitude variations below 1m. To fully judge the efficacy of the resulting controller, the altitude profile it produces is compared to that of the PID controller used in the training of the cyclic in Fig. 4-10. It can be seen that the resulting RL controller has significantly smaller altitude deviations than the PID controller used for training the cyclic.

The final part of this phase in the research was to train both controllers concurrently, without stopping the episode in between. This was not found to be possible with the current architecture, which used a single, shared critic: all episodes would diverge. To improve the performance, the controller was split up into two separate agents with their own actor and critic, yielding an architecture similar to that described in Part I, Sec II-A. A sample episode with this setup is shown in Fig. 4-11. It can be seen that the addition of a separate critic for each actor allows both agents to train concurrently, and that the performance of this setup is comparable to the previous scenarios where both agents were trained separately.



(a) Tracking performance and rewards **(b)** Control signals

Figure 4-8: MDHDP - 3DOF - learning attitude control cyclic with collective PID and a more complex reference signal.



(a) Tracking performance and rewards **(b)** Control signals

Figure 4-9: MDHDP - 3DOF - learning altitude control collective with a fixed, learned cyclic

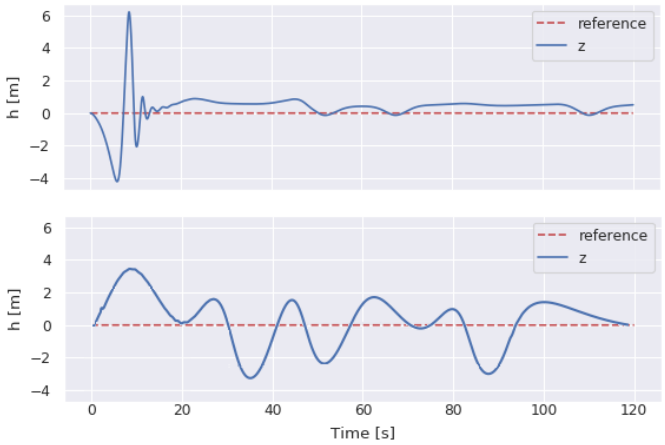
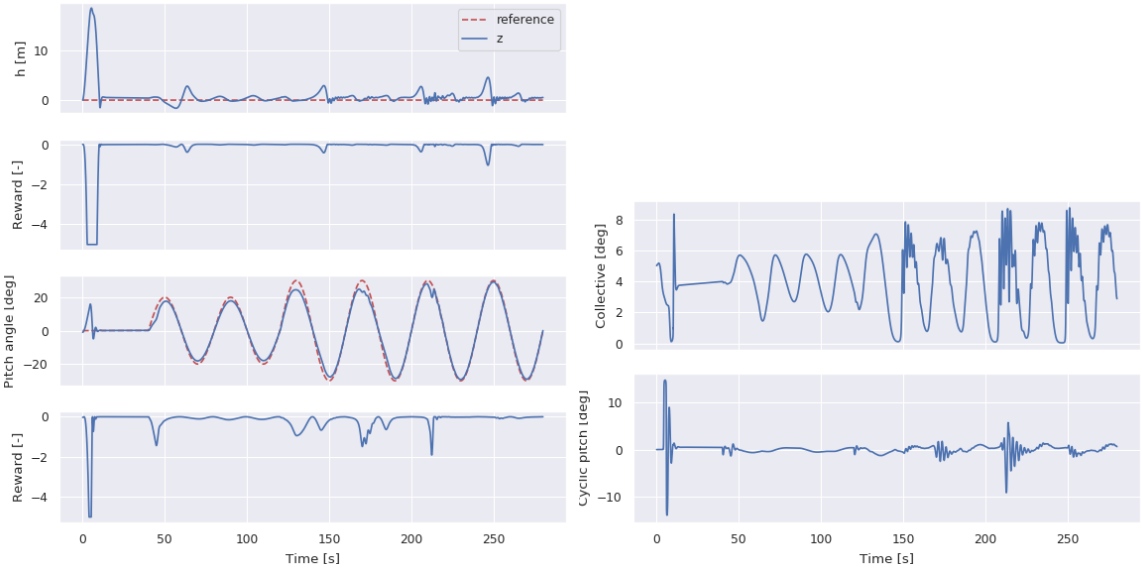


Figure 4-10: Comparison of the altitude tracking performance of the RL controller (top) and the previously used PID controller (bottom)



(a) Tracking performance and rewards **(b)** Control signals

Figure 4-11: MDHDP - 3DOF - training both channels concurrently

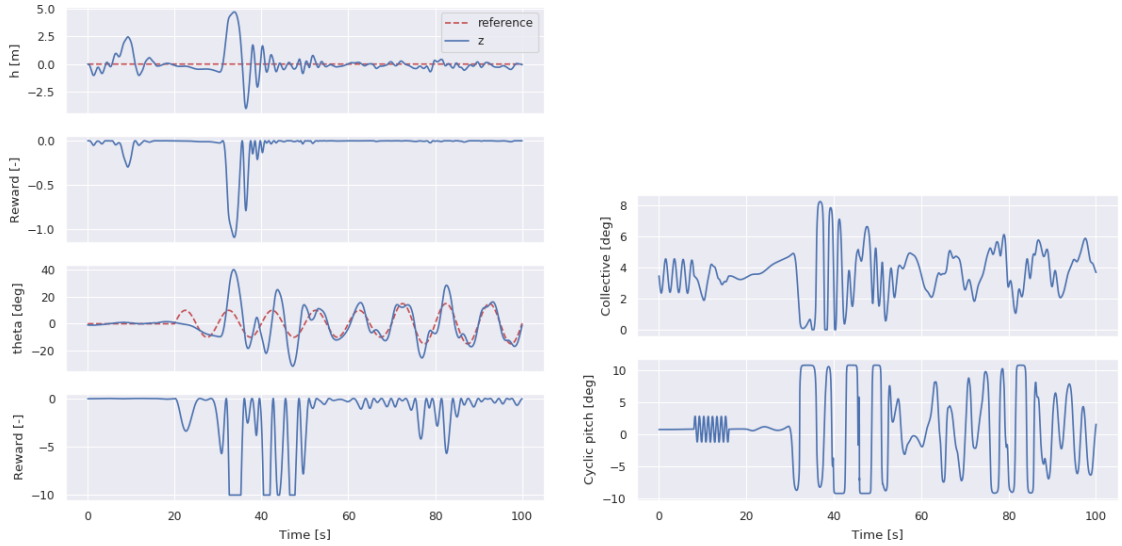
4-3 Replacing model gradients with online estimates

In the previous section, model-based HDP was shown to be able to learn to control a 3DOF helicopter model online. The next step in the research was to remove the dependence on the known model. This corresponds to Phase A-2 in Fig. 4-1. An incremental model was estimated online via recursive least squares following the method described in Part I, Sec II-C. The pre-calculated model gradients used previously were the replaced by appropriate rows and columns of the state- and control matrices of the incremental model, yielding the IHDP algorithm [72]. This adds an extra layer on complexity to the controller, which was assumed to negatively affect its performance, but enables online learning without any model knowledge.

4-3-1 Incremental heuristic dynamic programming

In Fig. 4-12, the results of a training episode using the IHDP controller for attitude control is shown. It can be seen that the collective performs reasonable, but the cyclic does not converge to the same performance seen in the previous experiment. This performance decay with respect to the model-based variant described in the previous section is attributed to the difficulty in estimating the model gradients. To aid in this, an input excitation signal was added to both control channels in the early stages of the episode. In this experiment, the excitation was a triangular wave. In Fig. 4-13, the corresponding control effectiveness estimates are shown, and it can be seen that this shape of input excitation does not lead to stable estimates of the control effectiveness. The estimate of $\frac{\partial z}{\partial \theta_0}$ flips sign multiple times. This was hypothesized to be caused by a lack of variety for the incremental signal, as in a triangular wave the action increment is equal in every timestep. It was found that a sinusoidal, exponentially decaying input excitation did lead to stable estimates, as this signal exposed

the RLS system to a larger variety of state-action pairs. This is shown in Fig. 4-14, where it can be seen that with the new excitation, both the cyclic and collective gradients are quickly estimated and remain stable thereafter.



(a) Tracking performance and rewards

(b) Control signals

Figure 4-12: Incremental HDP with triangular input excitation applied to the 3DOF helicopter model

4-4 Scaling up to six-degrees-of-freedom

The next step was to assess the performance of the IHDP system on the 6DOF helicopter model that would be used for the final experiments and was presented in Part I Sec III-A, corresponding to Phase A-3 as shown in Fig. 4-1. To reduce the complexity, the cyclic would once again have a pitch rate reference signal. None of the experiments with this setup were successful; a sample episode is shown in Fig. 4-15. Either the cyclic or the collective would fail in every experiment as a result of the cross-coupling effects that appeared in 6DOF. It was concluded that IHDP is not powerful enough to control a 6DOF helicopter model online, and that the more powerful IDHP architecture was required.

4-4-1 Incremental Dual Heuristic Programming

The final phase before the experiment was to implement the IDHP framework described in Part I, Section II-A. IDHP was shown to be more powerful than the IHDP framework used in the sections above. The implementation was first verified in 3DOF, the results of which are shown in Fig. 4-16.

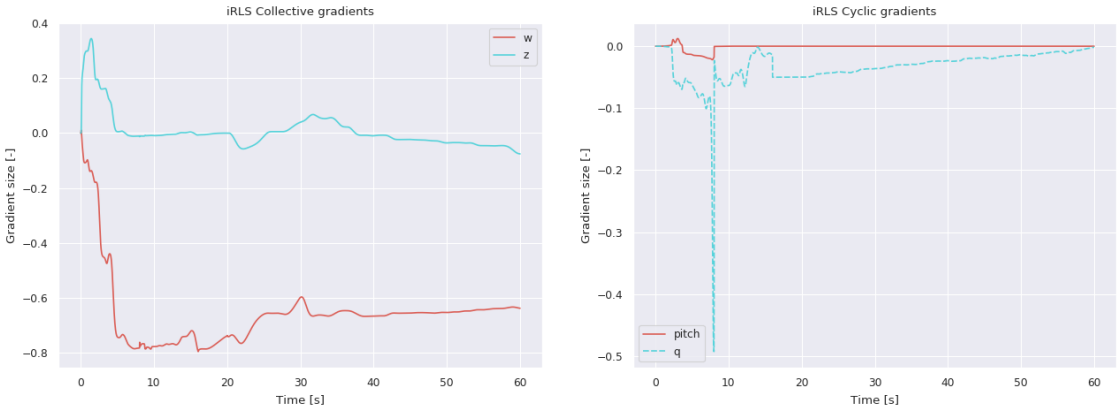
Finally, using the same setup, a rate tracking experiment similar to that in Section 4-4 was performed. A sample episode is shown in Fig. 4-17. It can be seen that, with the new IDHP framework, the controller was able to reliably learn to control the 6DOF helicopter model.

4-5 Conclusions

In this chapter, the steps taken in scaling up from the 1DOF experiment in the prelim to the 6DOF experiment in the paper were shown. From these steps, a number of conclusions were drawn. First, direct attitude control was found to be more desirable than a rate controller with added outer loop control. Second, the use of a PID controller for the collective while the cyclic was training was found to significantly improve the learning speed and final performance. Third, when using HDP, a decoupled architecture where the cyclic and collective controllers had their own critics was able to control the more complicated models where a unified approach could not. Fourth, the use of an exponentially decaying sinusoidal input excitation early in training significantly improves the performance of the incremental RLS estimator used in the IHDP and IDHP frameworks. Finally, IDHP was shown to be able to control the 6DOF model in a simple training scenario where IHDP could not, confirming that IDHP is the more powerful algorithm. This knowledge was used in the design of the training scenario and experiments presented in Part I.

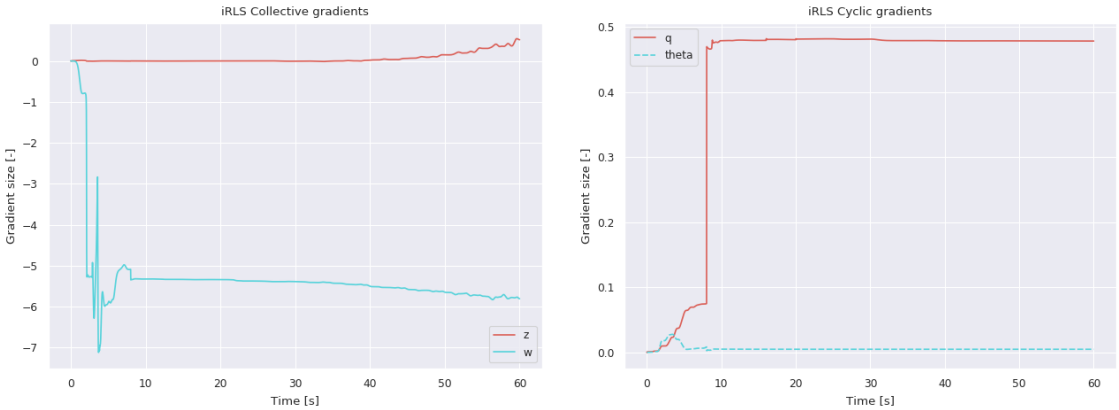
However, some discussion on these conclusions is warranted. In the 3DOF case, a trade-off was made between having two separate controllers for the two control channels, or a single unified controller. Based on the results of applying HDP to the 3DOF model, the separate controller strategy was found to be more reliable, and therefore this setup was also used on the 6DOF case. This approach was however not validated when switching to IDHP. Therefore it is possible that there is still performance gain available when trying a more unified setup.

Overall, in retrospect, the whole model-based HDP controller should have been dropped much earlier, and IDHP should have been implemented much sooner. A lot of time was lost researching the intricacies of HDP and getting it to work on borderline-stable cases, but that knowledge was not useful for the final experiments. A more useful timeline would have been to implement and compare IDHP on the 1DOF case, and then move up in helicopter models. Figure 4-18 shows the suggested work order for future research in this topic.



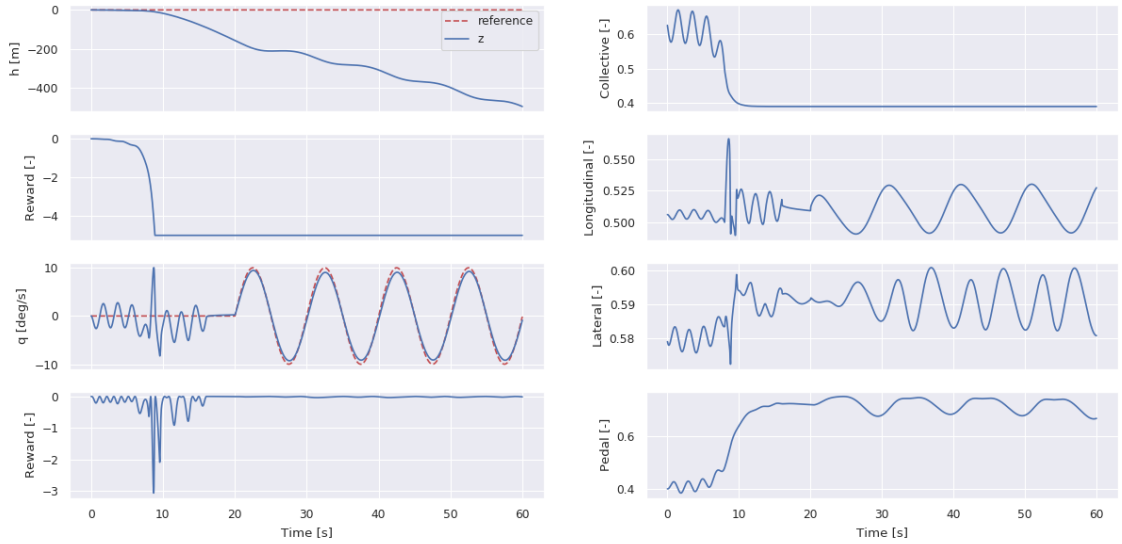
(a) Collective **(b)** Cyclic

Figure 4-13: Online estimates of the control effectiveness of the cyclic and collective using a triangular excitation signal



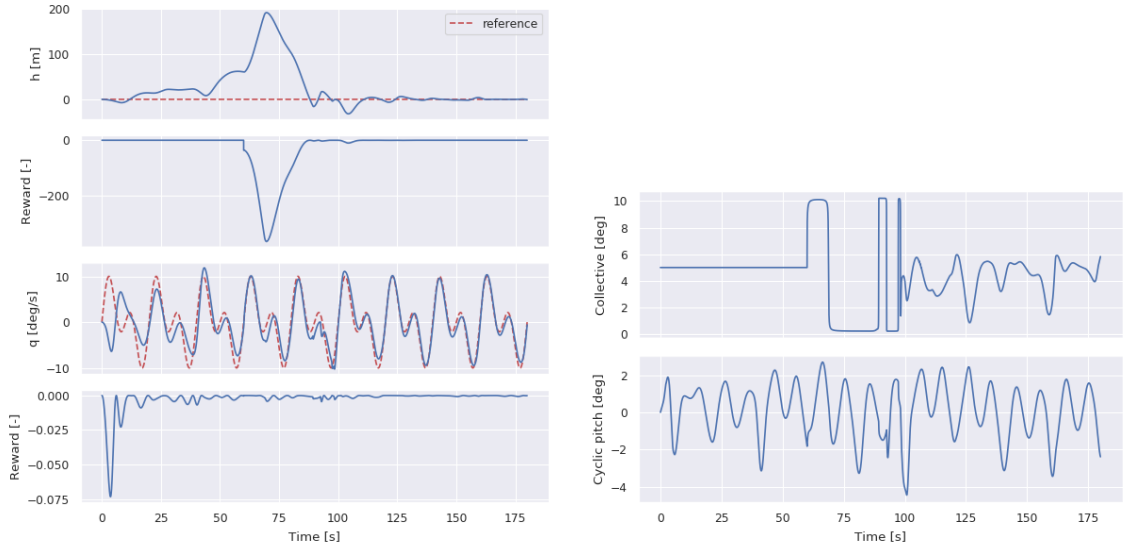
(a) Collective **(b)** Cyclic

Figure 4-14: Online estimates of the control effectiveness of the cyclic and collective using a exponentially decaying sinusoidal excitation signal



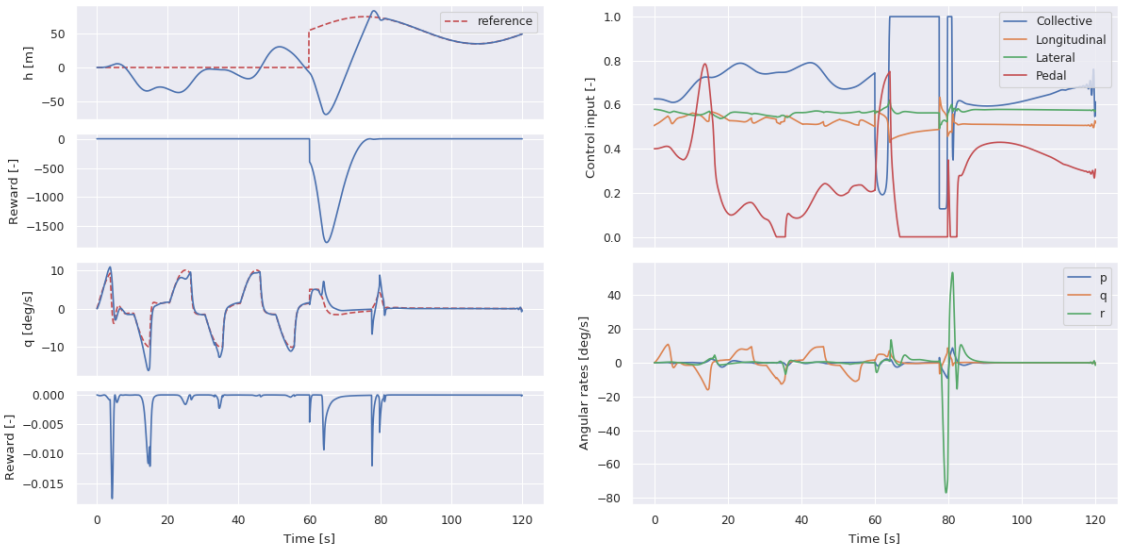
(a) Tracking performance and rewards **(b)** Control signals

Figure 4-15: Incremental HDP was not able to follow both reference signals in six degrees of freedom



(a) **(b)**

Figure 4-16: Incremental DHP applied to the 3DOF helicopter model



(a) (b)

Figure 4-17: Incremental DHP applied to the 6DOF helicopter model

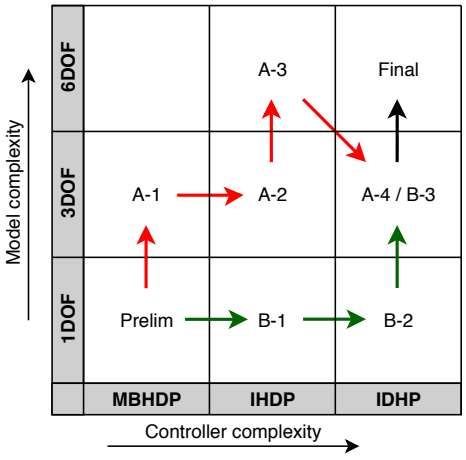


Figure 4-18: The steps suggested for future researchers in this field are shown in green, contrasted with the actual steps taken in red.

Chapter 5

Additional hyperparameter search

In the experiments presented in Part I, the learning rates of both agents were set heuristically. Afterwards, it was deemed that a more thorough grid search into the validity of this choice was necessary. In Figs. 5-1 and 5-2, the hyperparameter grid search is repeated for nine combinations of learning rates for the actor and the critic of the cyclic agent, for 100 random seeds each. In turn, each of Figs 5a and 5b from Part I is now replaced by a grid of three by three plots with the same experiment for a certain combination of learning rates. In each figure, the average performance metric is denoted by solid lines, while the shaded areas indicate the standard deviation of the data. Increasing values of target critic mixing factor τ are indicated by darker colors, while increasing values for the neural network weight initialization standard deviation σ_w are indicated by thicker lines.

In Fig 5-1, the training success rate is plotted against discount factor γ. The same general trends from Part I can still be observed. Higher values of τ are correlated with higher success rates for all combinations of learning rates, indicating that the use of a target critic is not advised in this implementation. Furthermore, higher discount factors and weight initializations decrease the success rate, although most implementations without target critic remain stable up until $\gamma = 0.9$.

In Fig 5-2, the final training performance, denoted in the root-mean-square of the tracking error in the last 10 seconds of training, is plotted against discount factor γ. Except for the extreme cases with $\eta_c = 10$, there is a strong general trend showing higher discount factors increase final performance. It should be noted that failed runs automatically had a final 'performance' of zero, so all entries that end up on the x-axis should be disregarded as these had zero successful runs.

When comparing the two sets of figures, it can be seen that the original choice of $[\eta_a = \eta_c = 5, \gamma = 0.95, \sigma_w = 0.01]$ has the combination of 100% success rate and top-tier rms of approximately 0.009. The only combination of parameters with significantly better performance is $[]\eta_a = 5, \eta_c = 2, \gamma = 0.99]$. This combination showed a single failure in training its average final performance rms is slightly lower at 0.006. Ultimately, these results show that the original choice for hyperparameters was done judiciously, and the results that follow from this choice remain valid.

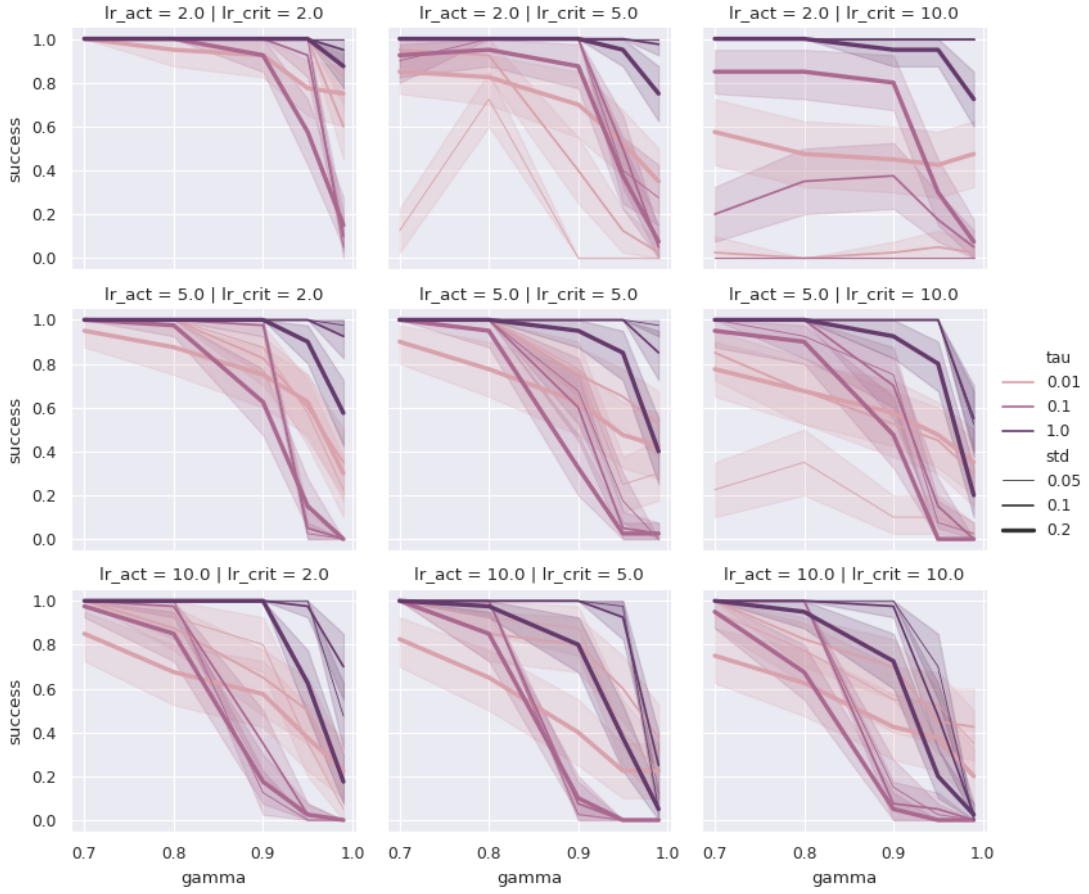


Figure 5-1: Success rate versus discount factor for a variety of hyperparameters

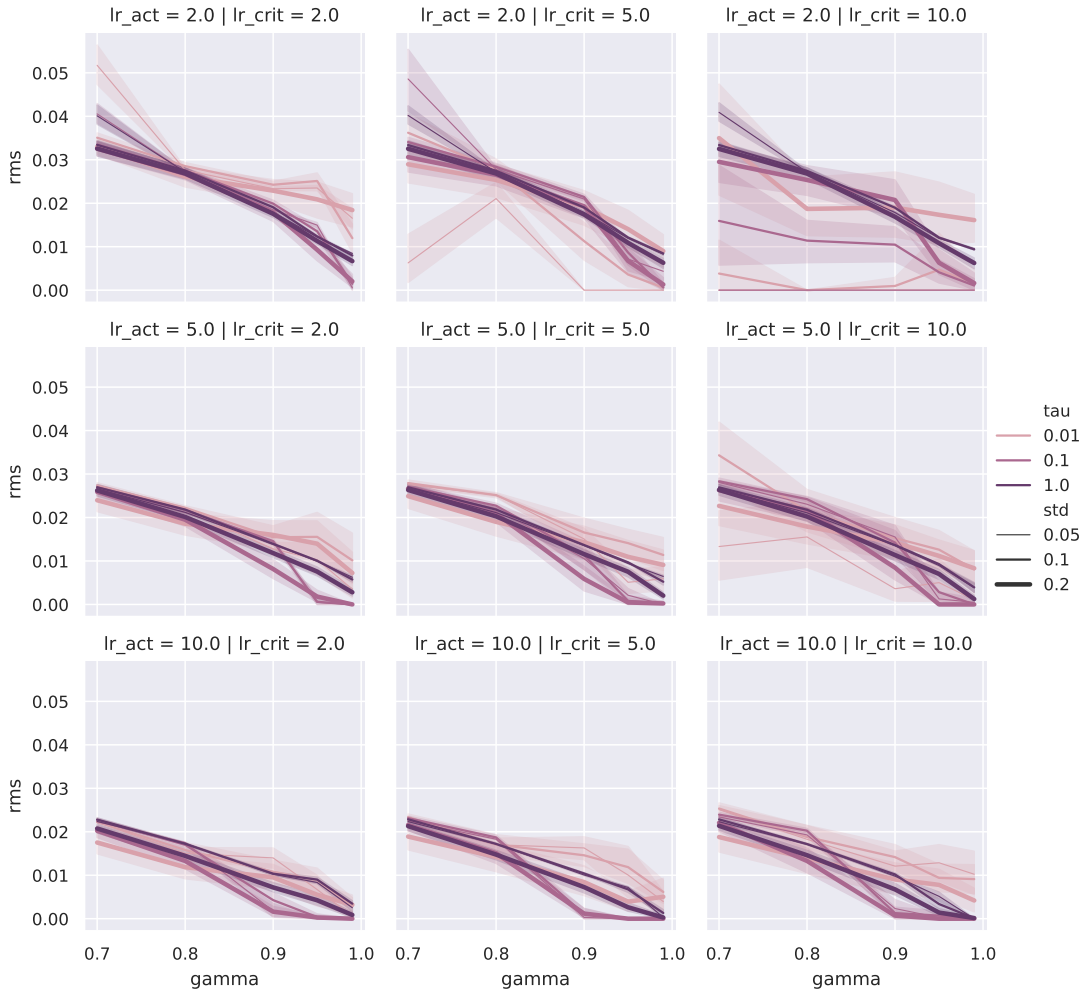


Figure 5-2: Final tracking performance versus discount factor for a variety of hyperparameters

Part IV

Conclusions and recommendations

Chapter 6

Conclusions

Reinforcement learning has shown to be a promising avenue for adaptive flight control. The novel IDHP framework has shown to be capable of learning a near-optimal control policy for a nonlinear, six-degree-of-freedom fixed wing aircraft model. Compared to fixed-wing aircraft, helicopters are inherently unstable with more coupled, complex dynamics. Therefore, applying IDHP to rotorcraft seemingly has a greater risk of aircraft loss-of-control before the adaptive controller manages to learn a control policy. This thesis research investigates whether online reinforcement learning control is possible for full-scale rotorcraft, and how best to implement such a controller. The formal research objective that has been stated in Chapter 1 is repeated here.

Research objective: Develop an online, nonlinear, model-free, adaptive flight control system for full-scale helicopters by investigating the applicability of novel reinforcement learning frameworks on a high-fidelity helicopter model, with the purpose of designing a system capable of both normal flight and one-engine inoperative flight.

To fulfil this objective, five research questions have been stated which have been answered throughout the preliminary and main thesis phases. The final conclusions based on the research questions are stated below.

RQ1: *What is the state-of-the-art of reinforcement learning for helicopter flight control?*

a What is the current state-of-the-art in helicopter flight control?

b What is the current state-of-the-art in continuous RL control?

First, the basics of helicopter control and maneuvers were explained, after which a closer look was taken at automatic flight control. Currently, most helicopter FCSs are designed with gain-scheduling and PID loops. However, a novel approach based on Incremental Nonlinear Dynamic Inversion has shown great results [18], and can be seen as the current state-of-the-art in helicopter control. This answer RQ1-a. Next, the most important concepts in reinforcement learning were explained. This led up to the state-of-the-art in RL, which is studied in two perspectives: Deep Reinforcement Learning (DRL) and Approximate Dynamic Programming

(ADP). The DRL perspective focuses on modeling complex end-to-end relations, such as locomotion from raw pixel input. This often requires thousands of trials, making DRL not suitable for online flight control. Nevertheless, the field has yielded significant innovations that can be adapted for an online system. The ADP perspective focuses more on online control and stability, and the novel incremental ADP frameworks have removed the need for an offline training phase. Finally, a review of some literature on RL for helicopter flight was done. For the purpose of this research, IDHP is the state-of-the-art algorithm in online RL for flight control. It is unknown if the highly coupled and non-linear dynamics of a full-scale helicopter can be reliably learned online without loss of control. To conclude, the improvements in both DRL and ADP make for the answer to **RQ1**-b.

RQ2 *What is the proposed baseline framework for adaptive, online flight control for a full-scale helicopter model?*

a Which RL framework is most suited for helicopter flight control?

b What should the flight control architecture of the adaptive controller be?

c What is the performance of the proposed system implemented on a simplified helicopter model?

IDHP, together with the results of the literature review, is chosen as the baseline framework for this task (RQ2-a). A simpler variant called HDP has been used for the preliminary research as the IDHP algorithm is more complex to implement and validate than other ADP algorithms. Since HDP is known to be a weaker control algorithm than DHP, it is hypothesized that the succesful application of HDP is a strong indicator DHP is also suited for the task. The framework controlled the body rates of the helicopter in order to reduce the complexity of the learning task and increase convergence speed (RQ2-b). The HDP controller was shown to be able to learn to control a simple helicopter model online in 100 out of 100 episodes, answering RQ2-c.

As a closing remark in answering RQ2, IDHP provides a good balance between complexity and performance, and a reduced need for model knowledge of the controlled system. To improve the learning stability and convergence speed of the algorithm, the novel ideas from DRL presented here are seen as promising additions to IDHP.

RQ3 *How does scaling up to controlling more complicated helicopter models affect the adaptive control system?*

The complete six degree of freedom helicopter model has a number of pitfalls compared to the simple system used for the preliminary research. Firstly, significant coupling exists between the four control channels. Small steps in increasing controller complexity and helicopter model degrees of freedom were taken to combat this without losing view of the end goal. Lessons learned in the 3DOF case were successfully used to make online control of the 6DOF helicopter model possible. Second, there are numerous different control architectures that place the learning system in various locations in the complete control architecture. The simplest implementation uses IDHP only for inner-loop rate control and PID controllers for all outer loop work. On the other extreme, all controllers could be learned, although this is not likely to be feasible in the context of online adaptive control.

RQ4 *How can the proposed RL framework be modified to improve controller adaptability and learning stability?*

The baseline framework introduced in [81] has two novel features. Firsly, a target critic, originally a DRL innovation, was used to improve learning stability at the cost of learning speed. Secondly, RL was only used for inner-loop rate control, and was combined with outer loop PID control for navigation. In this implementation, direct pitch angle and altitude control was used. This was found to perform equally well as rate control with PID attitude control, but with lower complexity. Furthermore, the target critic was found to decrease both the final performance as well as the success rate in training in the current implementation. More in-depth analysis has shown that the main reason for this is the increase in learning stability was outweighed by the decrease in learning speed. Concretely, most failures in the scenario with target critic have been caused be excessive roll or yaw angles. Most failures without target have been caused by parameter explosion. This leads to the conclusion that any new additions to an existing framework should be carefully weighed before applying them to novel situations. More specifically, in any application of the novel IDHP framework, the target critic mixing factor τ should remain an important optimization hyperparameter.

RQ5 *What is the overall performance of the RL flight controller design?*

1. How quickly does the learning process of the controller converge?
2. How does the RL controller generalize to different flight regimes?
3. What are the online fault-tolerant capabilities of the system, both for normal flight and one-engine inoperative flight?

The proposed setup allows for reliable convergence in a relatively short training phase of 120 seconds. The first minute of this is focussed on the longitudinal cyclic agent, and the second minute is focussed on the collective agent. The appropriate hyperparameters were found through an extensive grid-search. Using these hyperparameters, both agents start tracking the reference signal after approximately 10 seconds each, and take the rest of their respective training periods for fine-tuning. This approach yielded a 100% success rate in training, answering RQ5a. Two maneuvers have been designed to test the performance of the controller: a modified ADS-33 acceleration-deceleration and a one-engine inoperative landing. The acceleration-deceleration was successful in 84% of cases. Of the 16 failures, two were due to excessive altitude gain. The other fourteen were attributed to excessive collective-pedal coupling. This leads to pedal saturation causing yaw angles in excess of 30°. In reference literature on the used helicopter model, a similar trend was observed in the same manoeuvre. It was found that the modeled collective input is significantly higher than the input of comparable real-life flight test data. As a result, it is not clear whether this failure can be purely attributed to controller failure. The one-engine inoperative landing was successful is 100% of the simulations, showing very similar results throughout different random seeds. From this it can be concluded that the system is capable of both normal and one-engine-inoperative flight with relatively little trouble.

In this thesis, the application of the IDHP framework for control of a full-scale helicopter model was presented. The presented controller converges quickly and reliably, and is able to perform aggressive maneuvers in the longitudinal plane after a short training. Ultimately, it

can be concluded that IDHP is a viable framework for helicopter flight control, capable of both aggressive maneuvering and one-engine-inoperative flight.

Chapter 7

Recommendations for future work

A number of lessons were learned during this research. This chapter contains the recommendations for future research in this novel field.

Inclusion of the lateral control channels
In this thesis, only the longitudinal cyclic and collective were controlled by RL agents, while the lateral cyclic and pedal were controlled by PID controllers. Therefore, maneuvers were chosen that minimized intentional lateral motion. The next step in this research would be to extend the controller to all four input channels, and perform both longitudinal and lateral maneuvers. One maneuvers especially suited for testing a novel integrated controller would be the ADS-33 pirouette.

Controller architecture
Separate agents were used for the longitudinal cyclic and collective, but this approach was based on lessons learned for a simpler control system applied to a simpler model. Future implementations should re-evaluate these choices, and, where possible combine and harmonize controllers by having one agent control more than one input axis. In this way, a control system could potentially make use of control couplings in a way that multiple loose controllers could not.

Addition of stochastic elements
Three important assumptions were made in this research which are not true in real life. Firstly, higher order dynamics were largely ignored in the helicopter model used. The addition of flapping and inflow dynamics might invalidate some of the results, even while the Bo 105 is known to be a relatively responsive helicopter. Secondly, measurements were assumed to be clean and synchronous. The addition of sensor dynamics and measurement noise is expected to affect the RLS model especially. The extent of this effect could be investigated as the IDHP system strongly depends on the accuracy of the RLS estimates. Finally, turbulence is not taken into account. Removing this assumption might actually benefit the system. In [20] it was shown that the system noise introduced by turbulence improved the convergence rate and chance by de-correllating sequential measured states.

Adaptive learning rates
In this implementation, the learning rates were kept constant throughout training. They

were reduced once with a constant factor during the test phase. Adaptive learning rates have been widely shown to improve the performance of reinforcement learning tasks both in ADP and DRL, but many different implementations exist. A comparison of different methods for setting adaptive learning rates applied to flight control would benefit this area of research.

More advanced reward function

One problem with a reward function based around setpoint tracking is that the controller will constantly attempt to "catch up" to the new setpoint in almost any scenario. This results in a constant (though small) negative reward, even when a very accurate trajectory is flown. This leads to progressively more aggressive policies over time. In this thesis, this problem was reduced slightly by passing reference velocities through a low-pass filter before calculating tracking errors. However, as a result of this, the performance depends highly on the value of the low-pass filter time constant. Another solution to this problem, first presented in [6], is to base the reward not on the actual tracking deviation, but on the deviation from an idealized, desired trajectory.

Bibliography

[1] U.S. Army photo by Sgt. Michael J. MacLeod, "Night ride." https://www.flickr.com/photos/soldiersmediacenter/7257740090/, 2012.

[2] T. van Holten, "Lecture notes on AE4-213 Helicopter Performance, Stability and Control." Faculty of Aerospace Engineering, Delft University of Technology, The Netherlands, 2002.

[3] Federal Aviation Administration and U.S. Department of Transportation, *Rotorcraft Flying Handbook*. Skyhorse Publishing, 2012.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 2 ed., 2018.

[5] T. F. E. Wikipedia, "Typical CNN architecture." Retrieved from `https://en.wikipedia.org/wiki/Convolutional_neural_network` (2019/09/09).

[6] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, "Autonomous Helicopter Flight via Reinforcement Learning," in *Advances in Neural Information Processing Systems 16* (S. Thrun, L. K. Saul, and B. Schölkopf, eds.), pp. 799–806, MIT Press, 2004.

[7] M. Pavel and T. van Holten, "On the Prediction of the Necessary Dynamics For Helicopter Flight Simulation," in *23rd European Rotorcraft Forum*, (Dresden), 1997.

[8] S. Taamalllah, *Small-Scale Helicopter Automatic Autorotation*. PhD thesis, Delft University of Technology, 2015.

[9] IHSTI-CIS, "Helicopter accidents: statistics, trends and causes," tech. rep., International Helicopter Safety Team, Louisville, Kentucky, USA, 2016.

[10] S. C. Coyle, *Little Book of Autorotations*. Oregonia, OH: Eagle Eye Solutions LLC, first ed., 2012.

[11] P. Ky, "EASA Annual Safety Review," tech. rep., European Aviation Safety Agency, 2018.

[12] J. Hu and H. Gu, "Survey on Flight Control Technology for Large-Scale Helicopter," *International Journal of Aerospace Engineering*, vol. 2017, no. 1, pp. 1–14, 2017.

[13] L. Bainbridge, "Ironies of automation," *Automatica*, vol. 19, no. 6, pp. 775–779, 1983.

[14] S. H. Lane and R. F. Stengel, "Flight Control Design using Nonlinear Inverse Dynamics," in *1986 American Control Conference*, pp. 587–596, IEEE, jun 1986.

[15] L. Sonneveldt, E. van Oort, Q. Chu, C. de Visser, J. Mulder, and J. Breeman, "Lyapunov-based Fault Tolerant Flight Control Designs for a Modern Fighter Aircraft Model," in *AIAA Guidance, Navigation, and Control Conference*, (Reston, Virigina), American Institute of Aeronautics and Astronautics, aug 2009.

[16] T. Pollack, G. Looye, and F. Van der Linden, "Design and flight testing of flight control laws integrating incremental nonlinear dynamic inversion and servo current control," in *AIAA Scitech Forum*, (Reston, Virginia), American Institute of Aeronautics and Astronautics, jan 2019.

[17] T. Keijzer, G. Looye, Q. Chu, and E.-J. Van Kampen, "Flight Testing of Incremental Backstepping based Control Laws with Angular Accelerometer Feedback," in *AIAA Scitech Forum*, (San Diego, California), 2019.

[18] P. Simplício, E. van Kampen, and Q. Chu, "An acceleration measurements-based approach for helicopter nonlinear flight control using Incremental Nonlinear Dynamic Inversion," *Control Engineering Practice*, vol. 21, no. 8, pp. 1065–1077, 2013.

[19] D. P. Bertsekas, M. L. Homer, D. A. Logan, S. D. Patek, N. R. Sandell, and S. Member, "Missile Defense and Interceptor Allocation by Neuro-Dynamic Programming," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 30, no. 1, 2000.

[20] R. Enns and J. Si, "Helicopter Trimming and Tracking Control Using Direct Neural Dynamic Programming," *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 929–939, 2003.

[21] S. Ferrari and R. F. Stengel, "Online Adaptive Critic Flight Control," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 5, 2004.

[22] E.-J. van Kampen, Q. Chu, and J. Mulder, "Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics," in *AIAA Guidance, Navigation, and Control Conference*, (Keystone, CO), American Institute of Aeronautics and Astronautics, aug 2006.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[24] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[25] K. Wiggers, "Openai's dota 2 bot defeated 99.4% of players in public matches." Retrieved from `https://qz.com/1348177/why-are-ai-researchers-so-obsessed-with-games/` (2019-08-07).

[26] D. Lichtenstein and M. Sipser, "GO is Polynominal-Space Hard," *Journal of the Association for Computing Machinery*, vol. 27, no. 2, pp. 393–401, 1980.

[27] D. Kroezen, E.-J. van Kampen, and Q. Chu, "Online Reinforcement Learning for Flight Control," Master's thesis, Delft University of Technology, 2019.

[28] S. Heyer, E.-J. van Kampen, and Q. Chu, "Reinforcement Learning for Flight Control," Master's thesis, Delft University of Technology, 2019.

[29] H. Huber and P. Hamel, "Helicopter Flight Control System: State of the Art and Future Directions," in *Ninteenth European Rotorcraft Forum*, (Cernobbio, Italy), 1993.

[30] H. C. Curtiss, "Rotorcraft Stability and Control: Past, Present, and Future," *Journal of the American Helicopter Society*, vol. 48, no. 1, pp. 3–11, 2003.

[31] M. B. Tischler, "Digital Control of Highly Augmented Combat Rotorcraft," Tech. Rep. March, NASA, 1987.

[32] M. B. Tischler, J. W. Fletcher, P. M. Morris, and G. T. Tucker, "Applications of Flight Control System Methods to an Advanced Combat Rotorcraft," tech. rep., NASA, Moffett Field, 1989.

[33] T. Erturul, M. Arif Adli, and M. U. Salamci, "Model Reference Adaptive Control Design For Helicopters Using Gain Scheduled Reference Models," in *17th International Carpathian Control Conference (ICCC)*, (Tatranska Lomnica, Slovakia), 2016.

[34] T. G. B. Amaral and M. M. Cridstomo, "Automatic Helicopter Motion Control using Fuzzy Logic," in *The 10th IEEE International Conference on Fuzzy Systems*, (Melbourne, Australia), pp. 860–863, 2001.

[35] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[36] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," tech. rep., CUED/F-INFENG/TR 166., Engineering Department, Cambridge University, 1994.

[37] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[38] R. E. Bellman, *Dynamic Programming*. Rand Corporation research study, Princeton, NJ: Princeton University Press, 1957.

[39] K. Doya, "Reinforcement Learning In Continuous Time and Space," *Neural Computation*, vol. 12, no. 1, pp. 219–245, 2000.

[40] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, jan 1989.

[41] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063, 2000.

[42] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[43] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv e-prints*, 2013.

[45] D. Silver, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, (Beijing, China), 2014.

[46] 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 *International Conference on Learning Representations (ICLR)*, 2016.

[47] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of Brownian Motion," *Physical Review*, vol. 36, no. 5, pp. 823–841, 1930.

[48] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, may 1992.

[49] T. Schaul, J. Quan, I. Antonoglou, D. Silver, and G. Deepmind, "Prioritized Experience Replay," in *Proceedings of the 34th International Conference on Learning Representations (ICLR)*, 2016.

[50] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight Experience Replay," in *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, (Long Beach, CA, USA), 2017.

[51] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, oct 2017.

[52] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep Reinforcement Learning and the Deadly Triad," in *Deep RL Workshop NeurIPS*, (Montreal, Canada), 2018.

[53] H. V. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems 23* (J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds.), pp. 2613–2621, Curran Associates, Inc., 2010.

[54] H. V. Hasselt, A. Guez, and D. Silver, "Double DQN.pdf," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, pp. 2094–2100, 2016.

[55] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*, (Stockholm, Sweden), 2018.

[56] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional Continuous Control Using Generalized Advantage Estimation," in *International Conference on Learning Representations (ICLR)*, 2016.

[57] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of the 33rd International Conference on Machine Learning*, (New York, NY, USA), 2016.

[58] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv e-prints*, 2017.

[59] B. Dai, A. Shaw, N. He, L. Li, and L. Song, "Boosting the Actor with Dual Critic," in *International Conference on Learning Representations (ICLR)*, 2018.

[60] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," 2019.

[61] S. Kakade, "A Natural Policy Gradient," in *Advances in Neural Information Processing Systems*, (Vancouver, Canada), pp. 1531–1538, 2001.

[62] I. Grondman, L. Busoniu, G. Lopes, R. Babuska, L. Bus, G. A. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 42, no. 6, pp. 1291–1307, 2012.

[63] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, (Lille, France), pp. 1889–1897, 2015.

[64] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking Deep Reinforcement Learning for Continuous Control," in *Proceedings of the 33rd International Conference on Machine Learning*, (New York, NY, USA), 2016.

[65] D. V. Prokhorov and D. C. Wunsch, "Adaptive Critic Designs," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.

[66] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, "Adaptive Critic Designs: A Case Study for Neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367–1372, 1995.

[67] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Comparison of Heuristic Dynamic Programming and Dual Heuristic Programming Adaptive Critics for Neurocontrol of a Turbogenerator," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, 2002.

[68] J. Si, S. Member, and Y.-T. Wang, "On-Line Learning Control by Association and Reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, 2001.

[69] S. Sieberling, Q. P. Chu, and J. A. Mulder, "Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction," *Journal of Guidance, Control, and Dynamics*, vol. 33, pp. 1732–1742, nov 2010.

[70] P. Acquatella, E.-J. Van Kampen, Q. P. ·. Chu, and P. Chu, "Incremental Backstepping for Robust Nonlinear Flight Control," in *Proceedings of the EuroGNC*, (Delft, The Netherlands), 2013.

[71] Y. Zhou, E.-J. van Kampen, and Q. Chu, "Nonlinear Adaptive Flight Control Using Incremental Approximate Dynamic Programming and Output Feedback," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 493–500, 2016.

[72] Y. Zhou, E.-J. Van Kampen, and Q. Chu, "Incremental Model Based Heuristic Dynamic Programming for Nonlinear Adaptive Flight Control," *Proceedings of the international micro air vehicles conference and competition (IMAV) 2016*, 2016.

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

[74] J. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, (Seoul, South Korea), pp. 1615–1620, IEEE, 2001.

[75] R. Enns and J. Si, "Apache Helicopter Stabilization Using Neural Dynamic Programming," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 19–25, 2002.

[76] R. Enns and J. Si, "Helicopter flight-control reconfiguration for main rotor actuator failures," *Journal of Guidance, Control, and Dynamics*, vol. 26, no. 4, pp. 572–584, 2003.

[77] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *International Symposium on Experimental Robotics*, (Singapore), 2004.

[78] P. Abbeel, A. Coates, T. Hunter, and A. Y. Ng, "Autonomous Autorotation of an RC Helicopter," in *International Symposium on Robotics*, (Seoul, South Korea), 2008.

[79] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An Application of Reinforcement Learning to Aerobatic Helicopter Flight," in *Advances in Neural Information Processing Systems 19* (B. Schölkopf, J. C. Platt, and T. Hoffman, eds.), pp. 1–8, Van: MIT Press, 2007.

[80] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for Control from Multiple Demonstrations," in *Proceedings of the 25th International Conference on Machine Learning*, (Helsinki, Finland), 2008.

[81] S. Heyer, D. Kroezen, and E.-J. Van Kampen, "Online Adaptive Incremental Reinforcement Learning Flight Control for a CS-25 Class Aircraft," in *AIAA SciTech Forum*, no. January, 2020.